Estratégia Concursos

Aula 03

Banco do Brasil (Escriturário - Agente de Tecnologia) Desenvolvimento de Software - 2023 (Pós-Edital)

Autor:
Raphael Henrique Lacerda, Paolla Ramos e Silva

07 de Janeiro de 2023

Índice

APRESENTAÇÃO DA AULA

Pessoal, o tema da nossa aula é Java – Parte I. Pessoal, esse é um assunto um pouco assustador para quem não conhece a linguagem Java, mas não fiquem com medo, meu trabalho aqui é fazer com que o assunto seja didático, leve e simples.

Pessoal, o que eu como professora posso dizer a vocês? Infelizmente a banca não define especificamente o que vai solicitar em prova, e dizem "por aí" que TI é um "MUNDO" ... talvez TI seja um OCEANO... Devemos ter noção do básico e fazer o máximo possível de questões para conhecer o perfil da banca. Estar preparado para questões mais elaboradas ou novidades nunca cobradas! Como professora, tenho a obrigação de passar os conceitos a cada questão. Então vamos lá!

Java é a linguagem das possibilidades! Java está impulsionando a inovação por trás do nosso mundo digital. Há muitos anos dizia-se que Java era uma linguagem lenta, porém isso já mudou MUITO! Vamos ver isso e muito mais em nossa aula!

Bora, bora, pessoal, porque quem já chegou até aqui está com garra e vontade de ser aprovado! O pensamento do estudante (concurseiro) que quer ser aprovado deve ser sempre este: fazer o melhor que puder nas condições que possui! No contexto da nossa aula, o ponto mais importante não é tornar você um programador, e sim, fazer você acertar questões em prova! Portanto, foque nisso!

Para ajudá-los a realizar o sonho de ser aprovado, o time de TI do ESTRATÉGIA criou jogos bem legais para vocês fixarem o conteúdo das aulas! Aaaahhhh, o Estratégia é o melhorrr! 🥰 também acho!

Link para os GAMES 🎮 : *https://gamesdeti.herokuapp.com/*

Brincadeiras à parte, pessoal, durante minha preparação, eu sempre busquei estudar o máximo possível, pois eu pensava assim: Poxa, o professor já teve um trabalhão de criar a aula, escrever o PDF, gravar os vídeos, comentar questões. Eu só preciso ESTUDAR! Só isso! É o mínimo que eu tenho que fazer. Portanto, deem o máximo que vocês podem dar, lembrem-se é por VOCÊ, é pelos seus SONHOS!

JAVA

Introdução

INCIDÊNCIA EM PROVA: MÉDIA

Vamos falar agora sobre uma das linguagens mais famosas do mundo! O Java é a principal linguagem de programação e plataforma de desenvolvimento. Reduz custos e prazos de desenvolvimento, impulsiona a inovação e aprimora os serviços de aplicativos. Com milhões de desenvolvedores executando **mais de 51 bilhões de Java Virtual Machines em todo o mundo**, o Java continua sendo a plataforma de desenvolvimento preferida de empresas e desenvolvedores.

É isso ai galera! **Java 17**, a versão mais recente da plataforma de desenvolvimento e linguagem de programação que é **número um do mundo**! **Java é uma linguagem de programação orientada a objetos, multiplataforma, robusta, portável, segura, extensível, concorrente e distribuída**

Java é uma linguagem de programação portável, tendo em vista que basta compilar a aplicação apenas uma vez para que esta possa ser executada em qualquer plataforma que possua máquina virtual Java.

O Java só suporta **herança única**, na qual **cada classe é derivada exatamente de uma superclasse direta**. Ao contrário de C++, **o Java não suporta herança múltipla**, que ocorre quando uma classe é derivada de mais de uma superclasse direta.

A linguagem Java foi projetada tendo em vista os seguintes objetivos:

- Orientação a objetos - Baseado no modelo de Smalltalk e Simula67;
- Portabilidade - Independência de plataforma - "escreva uma vez, execute em qualquer lugar" ("write once, run anywhere");
- Recursos de Rede - Possui extensa biblioteca de rotinas que facilitam a cooperação com protocolos TCP/IP, como HTTP e FTP;
- Segurança - Pode executar programas via rede com restrições de execução.
- Além disso, podem-se destacar outras vantagens apresentadas pela linguagem:

Sintaxe similar a C/C++

- Facilidades de Internacionalização - Suporta nativamente caracteres Unicode;
- Simplicidade na especificação, tanto da linguagem como do "ambiente" de execução (JVM);
- É distribuída com um vasto conjunto de bibliotecas (ou APIs);
- Possui facilidades para criação de programas distribuídos e multitarefa (múltiplas linhas de execução num mesmo programa);
- Desalocação de memória automática por processo de coletor de lixo;

- Carga Dinâmica de Código - Programas em Java são formados por uma coleção de classes armazenadas independentemente e que podem ser carregadas no momento de utilização.

Um **resumo** das **características do Java**:

CARACTERÍSTICA	DESCRIÇÃO
CONCISA E SIMPLES	Não contém redundâncias e é fácil de entender, implementar e usar. Parecida com C++ para facilitar compreensão por grande parte de programadores. É uma evolução de C++: não suporta aritmética de ponteiros, registros, coercions, etc.
ORIENTADA A OBJETOS	Suporta os principais conceitos de orientação a objetos. Favorece reusabilidade.
ROBUSTA	Altamente tipada. Programas são confiáveis. Reduz imprevistos em tempo de execução: variáveis são automaticamente inicializadas, uso disciplinado de ponteiros, rotinas devem ser chamadas corretamente, etc.
PORTÁVEL	Completamente especificada. Não contém aspectos dependentes da implementação: o tamanho dos tipos é fixo para qualquer implementação, etc.
SEGURA	Restrições de acesso a arquivos, manipulação de ponteiros, etc. Implica que não é útil para desenvolver certas aplicaçoes como `device drivers'.
CONCORRENTE	Suporta aplicações concorrentes: multithreads, monitores, execução atômica.
INDEPENDENTE DE PLATAFORMA	Geração de bytecode que pode ser interpretado para qualquer arquitetura e sistema operacional tendo o sistema Java. Facilita distribuição de software.
INTERPRETADA	Facilita desenvolvimento exploratório.
COMPILADA	Utilizando compiladores, bytecodes podem ser traduzidos em tempo de execução para código de máquina.

Em 1991, na Sun Microsystems, foi iniciado o Green Project, o berço do Java. Em 2008 a Oracle Corporation adquire a empresa responsável pela linguagem Java, a Sun Microsystems, por US$ 7,4 bilhões, com o objetivo de levar o Java e outros produtos da Sun ao dispor dos consumidores.

Há anos falava-se que **Java era lento**, porém houve **melhorias na JVM** incluindo o compilador **Just-In-Time (JIT)** que é um componente do Java™ Runtime Environment que melhora o desempenho de aplicativos Java no tempo de execução. Dessa forma temos que o limite de compilação JIT ajuda a JVM a ser iniciada rapidamente e ainda assim ter desempenho melhorado.

Java é uma linguagem **WORA**, que significa Write once, run anywhere ou "**Escreva uma vez, execute em qualquer lugar**", é o slogan da Sun MicroSystems para exemplificar os benefícios multiplataforma da linguagem Java. Idealmente, isso significa que Java pode ser desenvolvido em qualquer dispositivo, compilado em um bytecode padrão e espera-se que seja executado em **qualquer dispositivo equipado com uma máquina virtual Java (JVM)**.

Java utiliza o compilador Java para converter o código-fonte Java em bytecodes que representam as tarefas a serem executadas na fase de execução. Diferentemente das instruções em linguagem de máquina, que são **dependentes de plataforma** (isto é, de hardware específico de computador), **instruções bytecode são independentes de plataforma**. Portanto, os **bytecodes do Java são portáveis** — sem recompilar o código-fonte, as mesmas instruções em bytecodes podem ser executadas em qualquer plataforma contendo uma JVM que entende a versão do Java na qual os bytecodes foram compilados. A JVM é invocada pelo comando **java**. Exemplo: **java Welcome.**

Atualmente, a plataforma de desenvolvimento moderna mais conhecida do mundo, o **Java SE**, é a plataforma de desenvolvimento moderna **mais popular do mundo**, reduz custos, reduz o tempo de desenvolvimento, impulsiona a inovação e melhora os serviços de aplicativos como a linguagem de programação preferida para aplicativos corporativos.

Quando estudarmos o ambiente de desenvolvimento Java típico vocês vão ver que a JVM analisa os bytecodes à medida que eles são interpretados, procurando hot spots (pontos ativos) — partes dos bytecodes que executam com frequência. Para essas partes, um compilador just in time (JIT), como o compilador Java HotSpot™ da Oracle, traduz os bytecodes para a linguagem de máquina do computador subjacente.

Quando a JVM encontra de novo essas partes compiladas, o código de linguagem de máquina mais rápido é executado. Portanto, os programas Java realmente passam por duas fases de compilação: uma em que o código-fonte é traduzido em bytecodes (para a portabilidade entre JVMs em diferentes plataformas de computador) e outra em que, durante a execução, os bytecodes são traduzidos em linguagem de máquina para o computador real no qual o programa é executado.

Quando dizemos que duas tarefas operam concorrentemente, queremos dizer que ambas progridem ao mesmo tempo. Desde sua criação, o Java suporta a concorrência. Quando dizemos que duas tarefas operam em paralelo, queremos dizer que elas são executadas simultaneamente.

Nesse sentido, o **paralelismo** é um subconjunto da **concorrência**. O corpo humano realiza uma grande variedade de operações paralelas. Respiração, circulação sanguínea, digestão, pensar e

caminhar, por exemplo, podem ocorrer em paralelo, assim como todos os sentidos — visão, audição, tato, olfato e paladar. Acredita-se que esse paralelismo é possível porque se acredita que **o cérebro humano contém bilhões de "processadores".** Computadores multiprocessados de hoje em dia têm múltiplos processadores que podem realizar tarefas em paralelo.

Entre os atrativos de Java está a facilidade que essa linguagem oferece para desenvolver aplicações para execução em sistemas distribuídos. Já em sua primeira versão, Java oferecia facilidades para o desenvolvimento de aplicações cliente-servidor usando os mecanismos da Internet, tais como os protocolos TCP/IP e UDP.

Além disso, **Java realiza o encapsulamento e ocultamento de informações** para melhoria da segurança. Classes (e seus objetos) encapsulam, isto é, **contêm seus atributos e métodos**. Os atributos e métodos de uma classe (e de seu objeto) estão **intimamente relacionados**. Os objetos podem se comunicar entre si, mas eles em geral não sabem como outros objetos são implementados — os **detalhes de implementação permanecem ocultos** dentro dos próprios objetos. Esse ocultamento de informações, como veremos, é crucial à boa engenharia de software.

Para fins de segurança, um compilador Java compila programas fonte em bytecodes e um compilador confiável assegura que o código fonte não viola as regras de segurança. Um fragmento de código pode ser obtido da rede. Em tempo de execução, Java não confia no código recebido e o sujeita a uma série de testes por meio de um **verificador de bytecode**. O verificador de bytecode possui um **mini provador de teoremas** que verifica o respeito às regras básicas da linguagem para que o código garanta:

- Não fraudar ponteiros
- Não violar as restrições de acesso
- Ter acesso a objetos no estado que eles estejam
- Chamar os métodos com os argumentos apropriados e com os tipos apropriados
- Não causar o trasbordamento de pilhas

O bytecode Java contém mais informações de tipos do que seria estritamente necessário para o interpretador. A pilha de operandos armazena informações sobre tipos. Cada tipo primitivo de dados possui operações especializadas feitas para operar sobre operandos do tipo apropriado. A coleção de tipos de informações, isto é, todos os registros da pilha de operandos e todas as variáveis locais constituem o chamado estado de tipos do quadro de execução.

Por fim, devido ao fato de integrar várias bibliotecas, integrar vários pacotes e possibilitar a criação de novos tipos de classe conforme necessário; o Java é conhecido como uma **linguagem extensível.**

Java é uma linguagem de programação amplamente usada para codificar aplicações Web. Ela tem sido uma escolha popular entre os desenvolvedores há mais de duas décadas, com milhões de

aplicações Java em uso hoje. Java é uma linguagem multiplataforma, orientada a objetos e centrada em rede que pode ser usada como uma plataforma em si. É uma linguagem de programação rápida, segura e confiável para codificar tudo, desde aplicações móveis e software empresarial até aplicações de big data e tecnologias do servidor.

Como o Java é uma linguagem de uso **gratuito e versátil**, ele cria software **local e distribuído**. Alguns usos comuns de Java incluem:

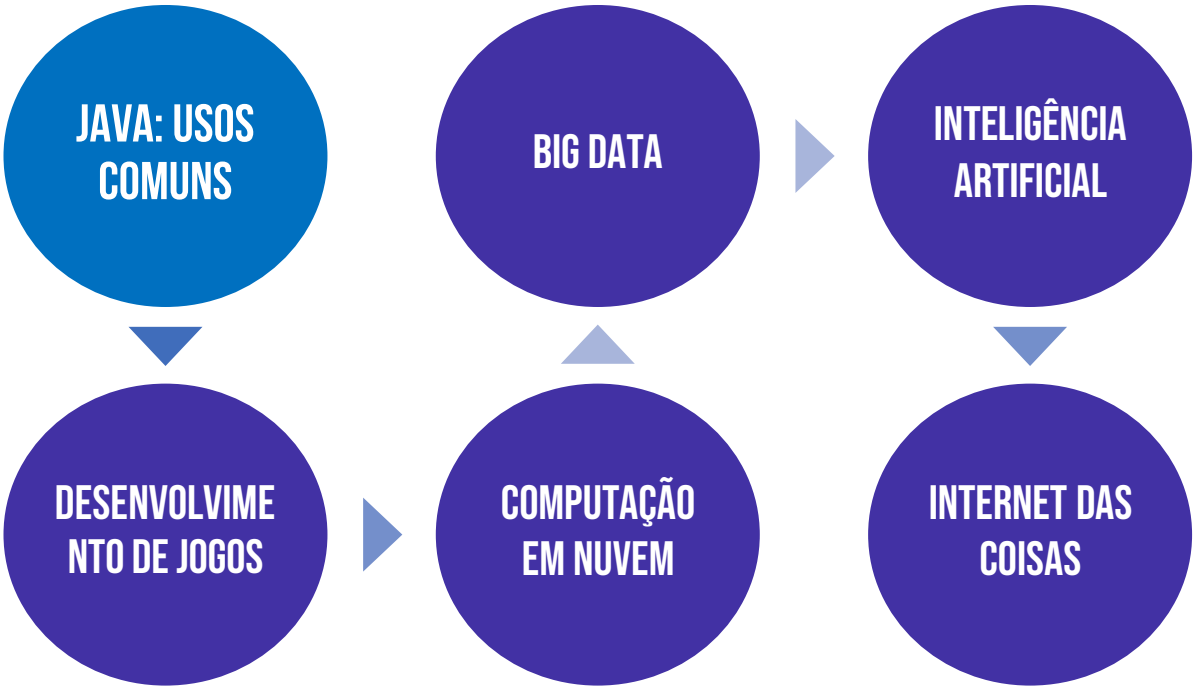
Desenvolvimento de jogos: muitos jogos populares para dispositivos móveis, computadores e videogames são criados em Java. Mesmo os jogos modernos que integram tecnologia avançada, como machine learning ou realidade virtual, são criados com tecnologia Java.

Computação em nuvem: a linguagem Java é muitas vezes chamada de Write Once and Run Anywhere (WORA – Escreva uma vez e execute em qualquer lugar), tornando-a perfeita para aplicações baseadas na nuvem descentralizadas. Os provedores de serviços de nuvem escolhem a linguagem Java para executar programas em uma ampla variedade de plataformas subjacentes.

Big data: a linguagem Java é usada para mecanismos de processamento de dados que podem trabalhar com conjuntos de dados complexos e grandes quantidades de dados em tempo real.

Inteligência artificial: Java é uma potência de bibliotecas de machine learning. Sua estabilidade e velocidade o tornam perfeito para o desenvolvimento de aplicações de inteligência artificial, como processamento de linguagem natural e aprendizado profundo.

Internet das Coisas: Java tem sido usado para programar sensores e hardware em dispositivos de borda que podem se conectar de forma independente à Internet.

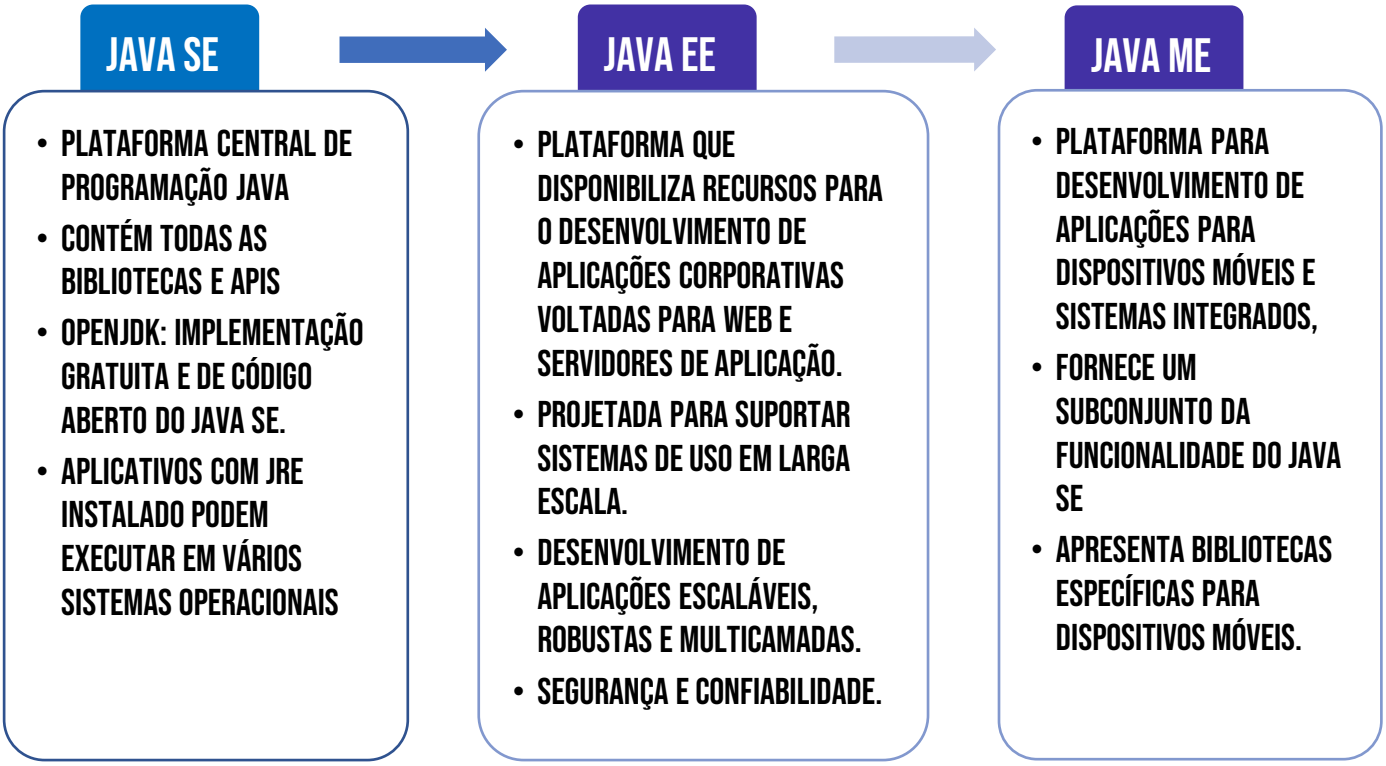


Vejamos agora as **quatro principais edições Java**:

- **Java Standard Edition (Java SE):** O Java Standard Edition é a plataforma central de programação Java. Ele contém todas as bibliotecas e APIs que qualquer programador precisa para o desenvolvimento java. O Open Java Development Kit (OpenJDK) é a implementação gratuita e de código aberto do Java SE. É voltada para criação de applets e desenvolvimento de softwares para desktop, destinados a computadores pessoais, notebooks ou outras arquiteturas com maior capacidade de processamento e memória. Os aplicativos podem ser executados em Windows, Mac OS, Linux, Solaris ou outros sistemas operacionais, contanto que estes tenham instalado o ambiente de execução JRE (Java Runtime Environment).

- **Java Enterprise Edition (Java EE):** O Java EE, ou Java Platform, Enterprise Edition, é a plataforma que disponibiliza recursos para o desenvolvimento de aplicações corporativas voltadas para web e servidores de aplicação. Java EE foi projetada para suportar sistemas de uso em larga escala, ou seja, para uma quantidade significativa de usuários, possibilitando o desenvolvimento de aplicações escaláveis, robustas e multicamadas. Toda essa estrutura incorpora características como segurança e confiabilidade, muitas vezes consideradas difíceis de serem implementadas. Com o objetivo de amenizar as dificuldades de implementação dessas características, o Java EE fornece um conjunto de tecnologias que reduz significativamente o custo e a complexidade de desenvolvimento. Em uma aplicação multicamadas, por exemplo, tem-se a estrutura do aplicativo separada em camadas, onde cada uma possui uma responsabilidade específica. Atualmente é possível classificar essa estrutura em duas partes: arquitetura lógica e arquitetura física.

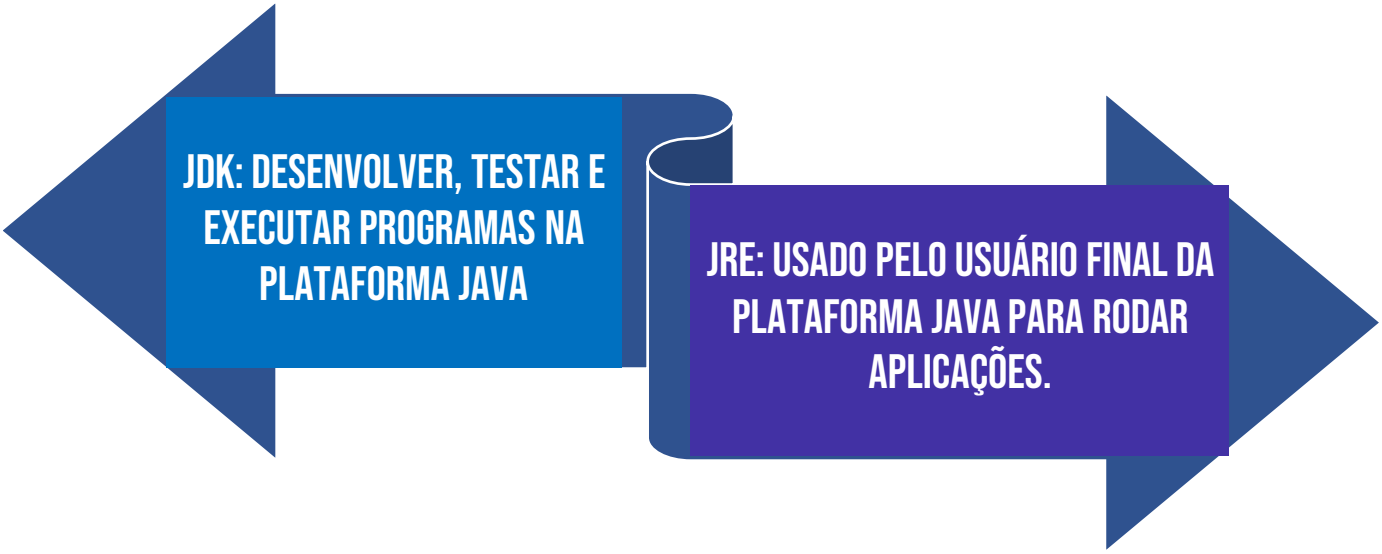
- **Java Micro Edition (Java ME):** O Java Micro Edition é a plataforma para desenvolvimento de aplicações para dispositivos móveis e sistemas integrados, como decodificadores de sinais. O Java ME fornece um subconjunto da funcionalidade do Java SE, mas também apresenta bibliotecas específicas para dispositivos móveis.



O usuário final da plataforma Java usa JRE para rodar aplicações Java. A Plataforma Java Standard Edition Runtime Environment (JRE) destina-se a **desenvolvedores e fornecedores de software** para uso com seus aplicativos.

O Java SE Runtime Environment contém a máquina virtual Java, bibliotecas de classes de tempo de execução e o ativador de aplicativos Java que são necessários para executar programas escritos na linguagem de programação Java. **Não é um ambiente de desenvolvimento e não contém ferramentas de desenvolvimento como compiladores ou depuradores.**

Fica a informação que o JDK inclui ferramentas para **desenvolver e testar programas** escritos na linguagem de programação Java e executados na plataforma Java. Já, o **JRE** é usado pelo **usuário final** da plataforma Java para rodar aplicações.



(CESPE / PGDF – 2021) A extensão das classes Java compiladas é *.java.

_____

Comentários: Pessoal, na verdade, a extensão é .jdk (Gabarito: Errado).

(CESPE / TCE-PA – 2016) Diferentemente do que ocorre com o JavaScript, as aplicações Java são executadas em uma máquina virtual ou em um browser.

_____

Comentários: É exatamente isso pessoal! As aplicações Java são executadas em uma máquina virtual! (Gabarito: Correto).

Compilação e Interpretação

Vejamos os passos para criar e executar um aplicativo Java. Normalmente, existem cinco fases: **editar, compilar, carregar, verificar e executar**.

Vamos falar sobre Compilação e Interpretação. Compilar é uma fase mais elaborada. Consiste em converter os programas de linguagem de alto nível em linguagem de máquina. Linguagens de alto nível permitem aos programadores escrever instruções que se pareçam com o inglês cotidiano e contenham notações matemáticas comumente utilizadas. Do ponto de vista do programador, as

linguagens de alto nível são preferíveis às de máquina. Java é uma das linguagens de programação de alto nível mais amplamente usadas.

Compilar um programa grande de linguagem de alto nível em linguagem de máquina pode levar tempo considerável de computador. Os programas interpretadores, desenvolvidos para executar diretamente programas de linguagem de alto nível, evitam o tempo de espera da compilação, embora sejam mais lentos do que programas compilados.

Com os interpretadores, cada instrução de código de alto nível é interpretada em código de máquina **em tempo real**. As instruções escritas são **executadas imediatamente** pelo hardware antes de considerar a próxima instrução.

Já utilizando compiladores, o programa completo é escrito em sintaxe natural semelhante ao inglês com compiladores, e **a linguagem compila (ou converte) todo o código** em código de máquina. O código compilado é então executado no hardware.

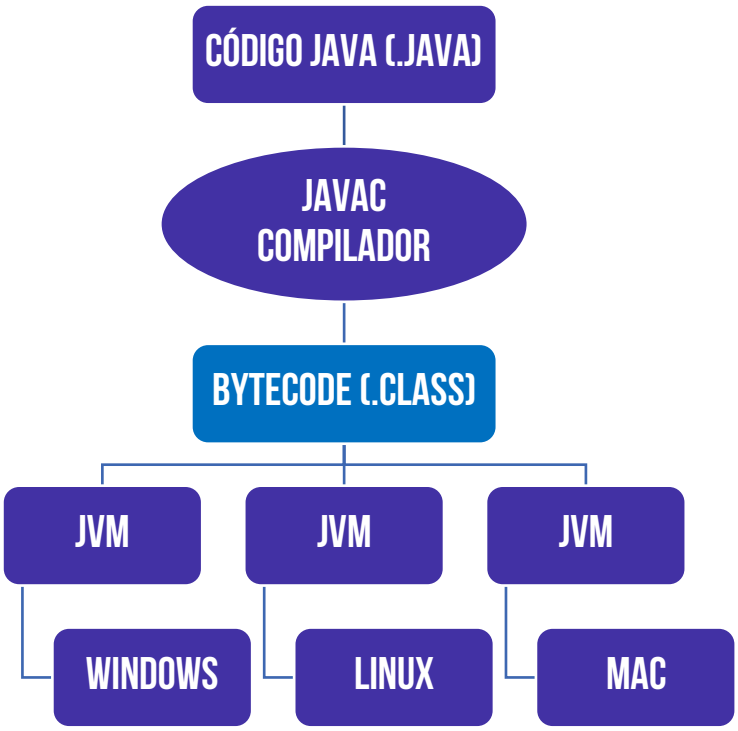
O programa Java foi a primeira linguagem a combinar os dois métodos acima (Compilação e Interpretação) usando uma **máquina virtual Java (JVM).** O compilador de código Java é chamado **máquina virtual Java**. Qualquer arquivo Java é compilado primeiro em bytecode. O bytecode Java só pode ser executado na JVM. A JVM então interpreta o bytecode para executá-lo na plataforma de hardware subjacente. Portanto, se a aplicação estiver sendo executada em uma máquina Windows, a JVM a interpretará para o Windows. Mas se estiver sendo executada em uma plataforma de código aberto como Linux, a JVM vai interpretá-la para o Linux.

Ta-dah!!! Essa é a mágica do Java! Compilar e interpretar códigos.

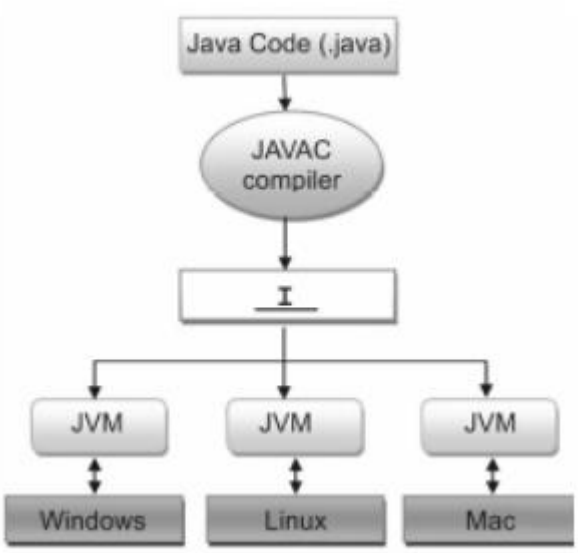
Então quando Java compila (ou converte) todo o código .java transforma-o em um arquivo .class.


```
public class JavaHelloWorldProgram {

      public static void main(String args[]){
            System.out.println("Hello World");
      }
}
```



(FCC / PGE MT – 2016) Considere a imagem abaixo, que mostra o esquema de funcionamento da Java Virtual Machine – JVM.



O produto resultante do processo de compilação de classes Java, que deve preencher a lacuna I, é conhecido como,

a) hashcode.
b) bytecode.
c) CIL (Common Intermediate).
d) assembly.
e) classcode.

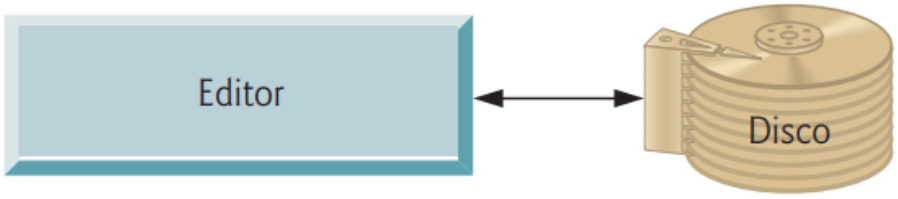
_____

Comentários: Pessoal, JAVA utiliza uma JVM para "entender" bytecodes. Logo, se existe um compilador que seja capaz de transformar o código-fonte em uma bytecode seguindo as especificações corretamente, o bytecode poderá ser interpretado por uma JVM. A questão pergunta qual o produto resultante do processo de compilação de classes Java que é exatamente o bytecode(Gabarito: Letra B).

Ambiente típico de desenvolvimento Java — fase de edição.

Fase 1: criando um programa

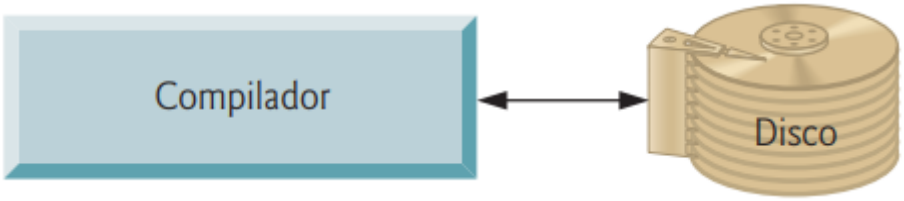
A Fase 1 consiste em editar um arquivo com um programa editor, muitas vezes conhecido simplesmente como um editor. Você digita um programa Java (em geral referido como código-fonte) utilizando o editor, faz quaisquer correções necessárias e salva o programa em um dispositivo de armazenamento secundário, como sua unidade de disco. Arquivos de código-fonte Java recebem um nome que termina com a extensão .java, que indica um arquivo contendo código-fonte Java.



Inicialmente, o programa é criado em um editor e armazenado em disco em um arquivo cujo nome termina com .java.

Fase 2: compilando um programa Java em bytecodes

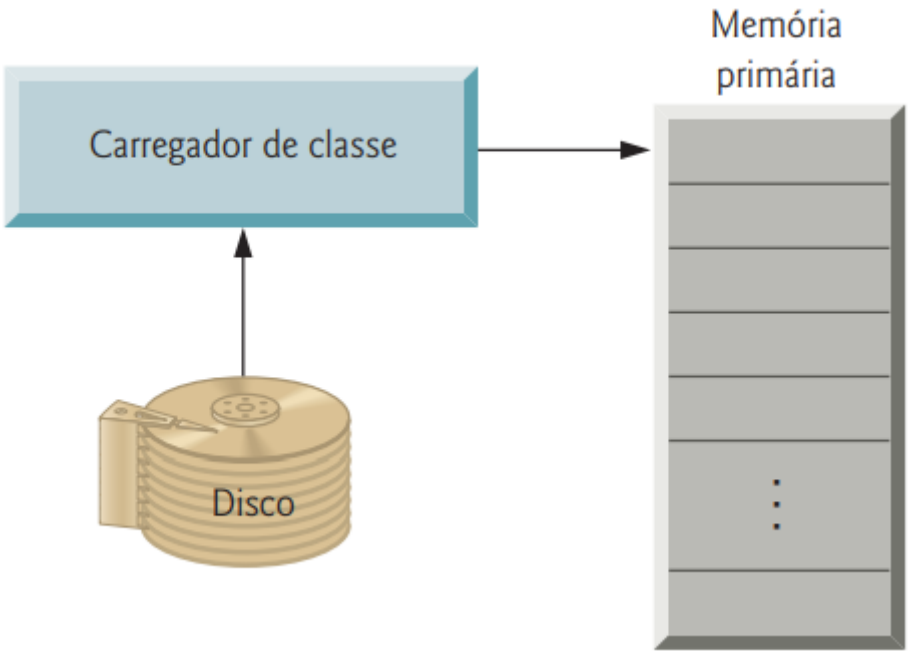
Na Fase 2, utilize o comando javac (o compilador Java) para compilar um programa. Por exemplo, a fim de compilar um programa chamado Welcome.java, você digitaria javac Welcome.java



O compilador cria bytecodes e os armazena em disco em um arquivo cujo nome termina com .class.

Fase 3: carregando um programa na memória

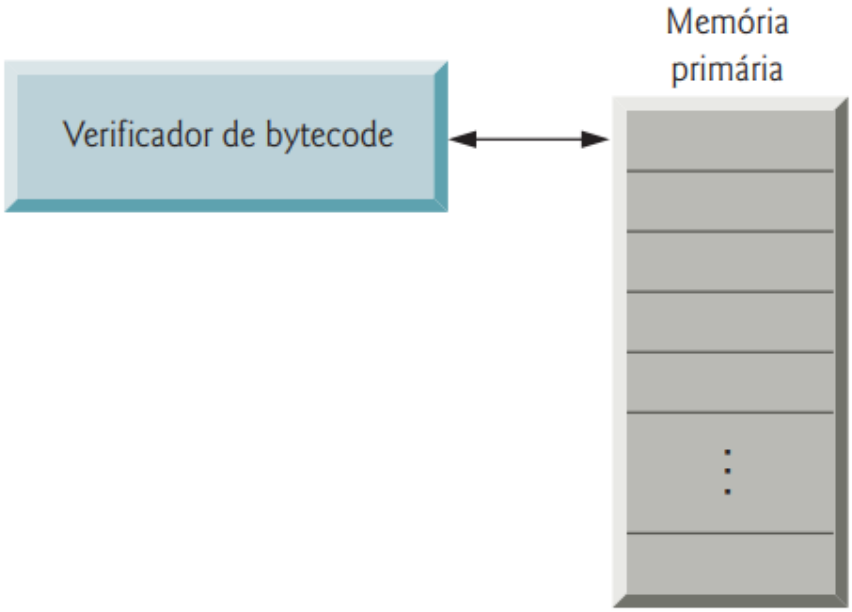
Na Fase 3, a JVM armazena o programa na memória para executá-lo — isso é conhecido como carregamento. O carregador de classe da JVM pega os arquivos .class que contêm os bytecodes do programa e os transfere para a memória primária. Ele também carrega qualquer um dos arquivos .class fornecidos pelo Java que seu programa usa. Os arquivos .class podem ser carregados a partir de um disco em seu sistema ou em uma rede (por exemplo, sua faculdade local ou rede corporativa ou a internet).



O carregador de classe lê os arquivos .class que contêm bytecodes a partir do disco e coloca esses bytecodes na memória.

Fase 4: verificação de bytecode

Na Fase 4, enquanto as classes são carregadas, o verificador de bytecode examina seus bytecodes para assegurar que eles são válidos e não violam restrições de segurança do Java. O Java impõe uma forte segurança para certificar-se de que os programas Java que chegam pela rede não danificam os arquivos ou o sistema (como vírus e worms de computador).



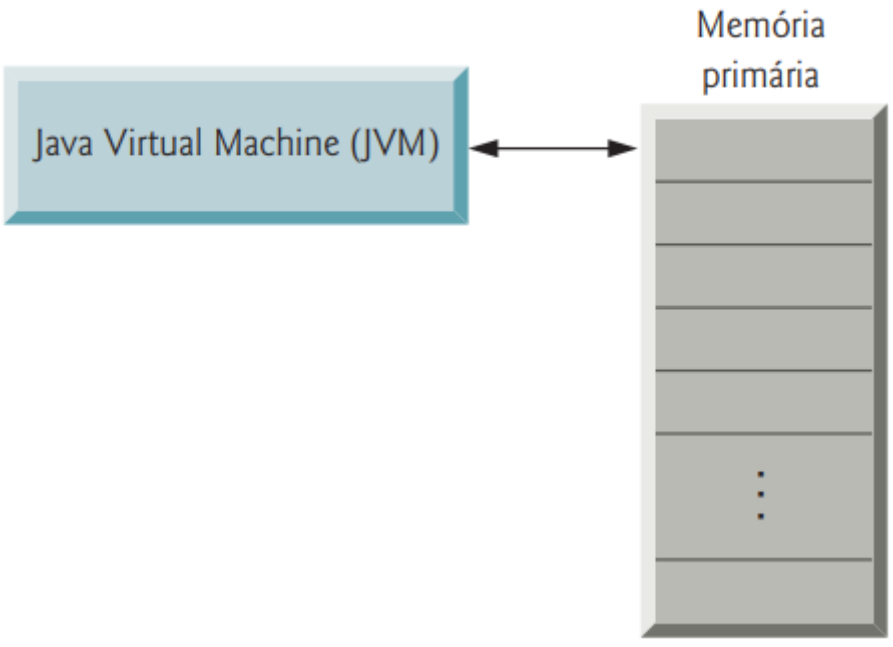
O verificador de bytecode confirma que todos os bytecodes são válidos e não violam restrições de segurança do Java.

Fase 5: execução

Na Fase 5, a JVM executa os bytecodes do programa, realizando, assim, as ações especificadas por ele (Figura 1.10). Nas primeiras versões do Java, a JVM era simplesmente um interpretador para bytecodes. A maioria dos programas Java executava lentamente, porque a JVM interpretava e executava um bytecode de cada vez. Algumas arquiteturas modernas de computador podem executar várias instruções em paralelo.

Em geral, as JVMs atuais executam bytecodes utilizando uma combinação de interpretação e a chamada compilação just in time (JIT). Nesse processo, a JVM analisa os bytecodes à medida que eles são interpretados, procurando hot spots (pontos ativos) — partes dos bytecodes que executam com frequência. Para essas partes, um compilador just in time (JIT), como o compilador Java HotSpot™ da Oracle, traduz os bytecodes para a linguagem de máquina do computador subjacente. Quando a JVM encontra de novo essas partes compiladas, o código de linguagem de máquina mais rápido é executado.

Portanto, os programas Java realmente passam por duas fases de compilação: uma em que o código-fonte é traduzido em bytecodes (para a portabilidade entre JVMs em diferentes plataformas de computador) e outra em que, durante a execução, os bytecodes são traduzidos em linguagem de máquina para o computador real no qual o programa é executado



Para executar o programa, a JVM lê os bytecodes e os compila (isto é, traduz) no momento certo (ou Just-In-Time — JIT) para uma linguagem que o computador possa entender. Como o programa existe na memória ele pode armazenar valores de dados na memória principal.

(CESPE / SERPRO – 2021) A compilação de um programa JAVA para ambiente Windows gera um programa com extensão EXE, o qual é executado pelo sistema operacional.

_____

Comentários: Pessoal, na verdade, não há a extensão EXE no ambiente citado. Nas 5 fases há.jdk, .class, java. Mas não há .exe (Gabarito: Errado).

Java Virtual Machine (JVM)

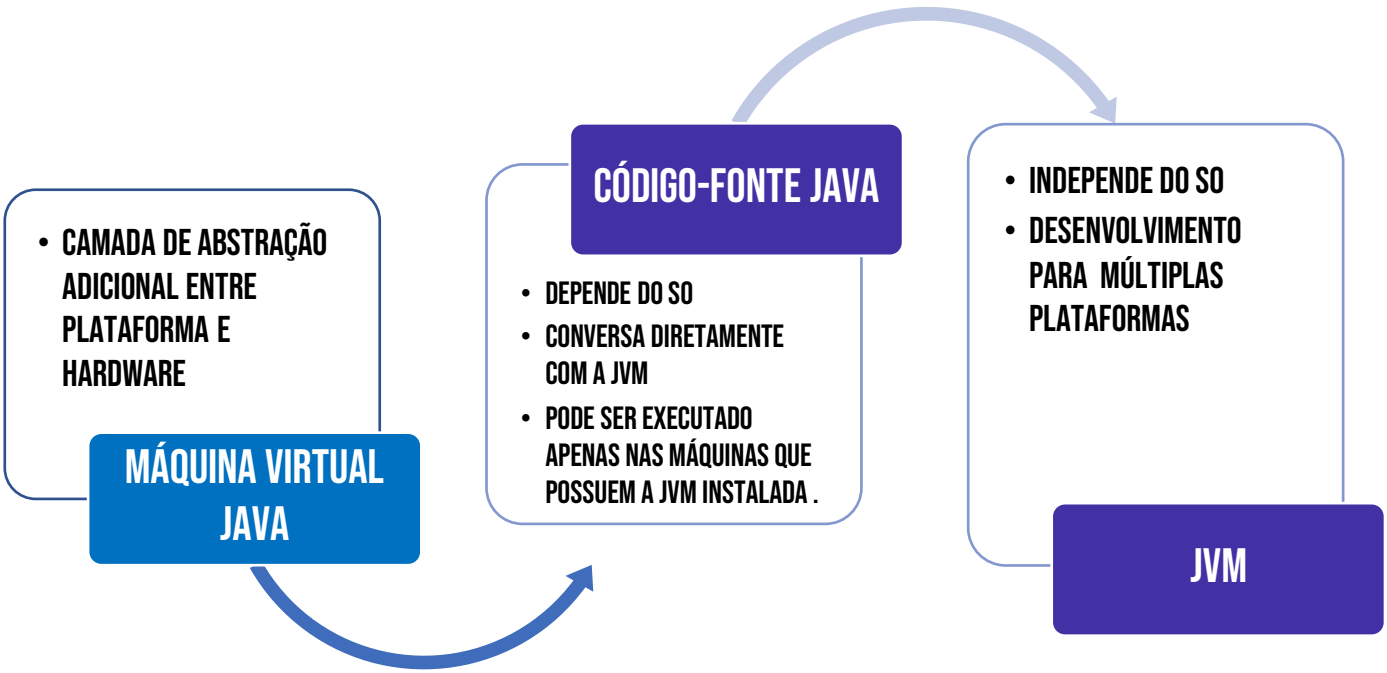
A **máquina virtual Java** atua como uma camada de abstração adicional entre a plataforma Java e o hardware da máquina subjacente. O código-fonte Java pode ser executado apenas nas máquinas que possuem a JVM instalada nelas.

Uma **máquina virtual** é um software que simula uma máquina física e consegue executar vários programas, gerenciar processos, memória e arquivos. Resumindo, ele constitui de uma plataforma, onde a memória, o processador e seus outros recursos, são totalmente virtuais, não dependendo de hardwares.

A execução de um código Java não está diretamente relacionada com o Sistema Operacional, ele **conversa diretamente com a JVM** (Java Virtual Machine), possibilitando assim a portabilidade de seu código. O que for escrito em um sistema operacional Windows, irá rodar em um sistema operacional Linux (salvo algumas exceções de códigos nativos).

Esse processo cria uma **independência do Sistema Operacional (SO)**, dando ao desenvolvedor uma **liberdade de desenvolver para múltiplas plataformas** sem aquela preocupação de se o código irá funcionar corretamente.

A **JVM** é desenvolvida em **código nativo**, pois ela **conversa diretamente com o sistema operacional** para que o programa Java funcione na máquina.



Passagem por Valor e por referência

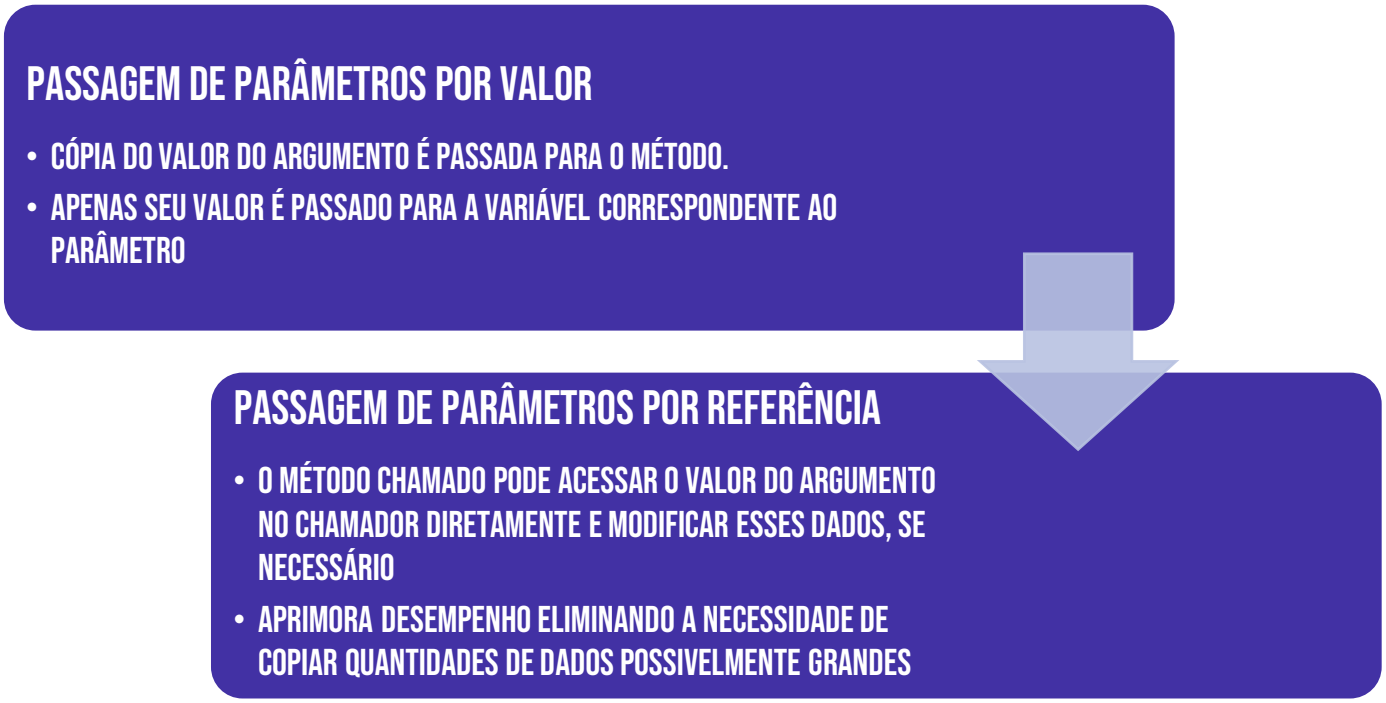
Duas maneiras de passar argumentos em chamadas de método em muitas linguagens de programação são **passagem por valor e passagem por referência** (às vezes denominadas chamada por valor e chamada por referência). Quando um argumento for passado por valor, uma cópia do valor do argumento é passada para o método chamado. O método chamado funciona exclusivamente com a cópia. As alterações na cópia do método chamado não afetam o valor da variável original no chamador.

Quando um argumento for passado por referência, o método chamado pode acessar o valor do argumento no chamador diretamente e modificar esses dados, se necessário. **Passar por referência aprimora desempenho eliminando a necessidade de copiar quantidades de dados possivelmente grandes.**

Ao contrário de algumas outras linguagens, o Java não permite escolher passagem por valor ou passagem por referência — **todos os argumentos são passados por valor**. Uma chamada de método pode passar dois tipos de valores para um método — cópias de valores primitivos (por exemplo, valores de tipo int e double) e cópias de referências para objetos. Os objetos em si não podem ser passados para os métodos. Quando um método modifica um parâmetro do tipo primitivo, as alterações no parâmetro não têm nenhum efeito no valor original do argumento no método chamador.

Se você modificar um parâmetro de tipo por referência para que ele se refira a outro objeto, apenas o parâmetro refere-se ao novo objeto — a referência armazenada na variável do chamador ainda se refere ao objeto original.

Embora uma referência do objeto seja passada por valor, um método ainda pode interagir com o objeto referenciado chamando seus métodos public que utilizam a cópia da referência do objeto. Visto que a referência armazenada no parâmetro é uma cópia da referência que foi passada como um argumento, o parâmetro no método chamado e o argumento no método chamador referenciam o mesmo objeto na memória.



(SEFAZ-CE – 2007) Na passagem de parâmetros por valor, a expressão correspondente ao parâmetro é avaliada e apenas seu valor é passado para a variável correspondente ao parâmetro dentro da função.

_____

Comentários: Quando um argumento for passado por valor, uma cópia do valor do argumento é passada para o método chamado. Apenas seu valor é passado para a variável correspondente ao parâmetro dentro da função. (Gabarito: Correto).

(AGE-PA – 2022) Em programação, existem duas maneiras de realizar a passagem de parâmetros para uma função qualquer. Sobre o tema, analise as afirmativas a seguir. A passagem de parâmetro pode ser feita por valor ou por referência.

_____

Comentários: Certo, pessoal! Como vimos, duas maneiras de passar argumentos em chamadas de método em muitas linguagens de programação são passagem por valor e passagem por referência (Gabarito: Correto).

Empacotamento

O software de aplicativos é distribuído em unidades denominadas **pacotes**. Um pacote é um conjunto de arquivos e diretórios necessários para um produto de software. Um pacote é geralmente criado e construído por um desenvolvedor de aplicativos depois de terminar de desenvolver o código do aplicativo. Um produto de software precisa ser construído em um ou mais pacotes para que possa ser facilmente transferido para um meio de distribuição. Depois, o produto de software pode ser produzido em massa e instalado por administradores. Um pacote é um conjunto de arquivos e diretórios em um formato definido.

Os componentes de um pacote se dividem em duas categorias.

- Os objetos do pacote são arquivos de aplicativo que serão instalados.
- Os arquivos de controle controlam como, onde e se o pacote está instalado.

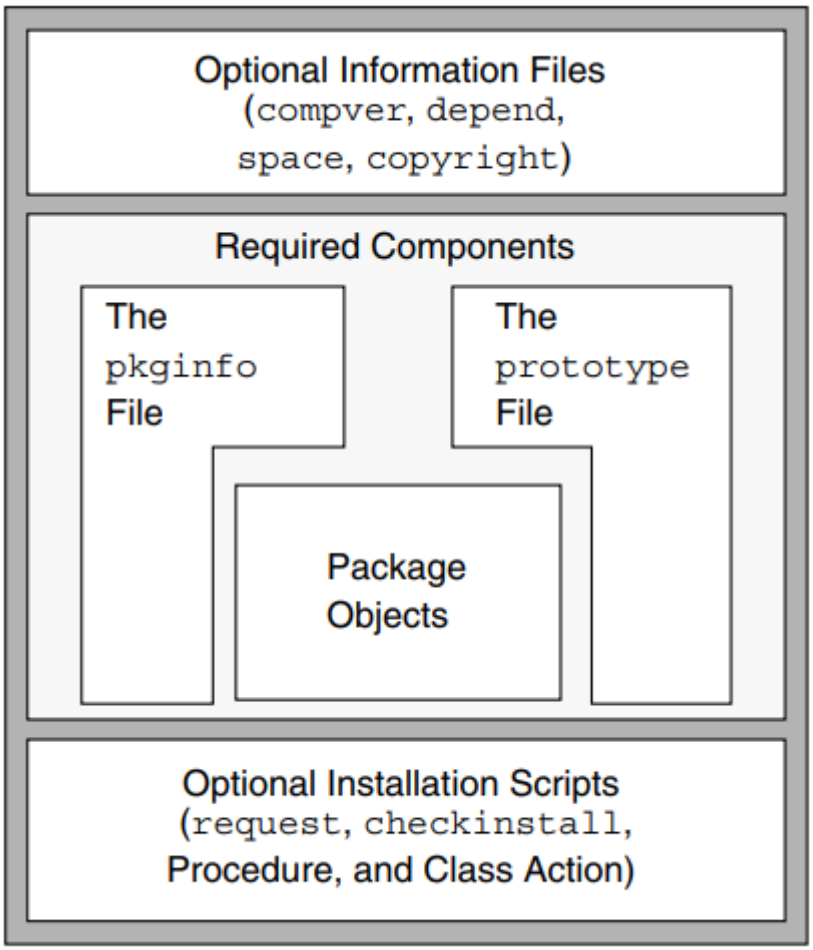
Os arquivos de controle também estão divididos em duas categorias: arquivos de informação e scripts de instalação. Alguns arquivos de controle são obrigatórios. Alguns arquivos de controle são opcionais.

Para empacotar seus aplicativos, você deve primeiro criar os componentes obrigatórios e os componentes opcionais que farão parte do seu pacote. Você pode, então, construir o pacote usando o comando pkgmk.

Para construir um pacote, você deve fornecer:

- Objetos de pacote (arquivos e diretórios do software de aplicativo)
- Dois arquivos de informação (os arquivos pkginfo e prototype)
- Arquivos de informação opcionais
- Scripts de instalação opcionais

A ilustração seguinte descreve o conteúdo de um pacote.



Você deve criar os objetos de pacote antes de construir o pacote. Os componentes abaixo fazem parte do aplicativo. Podem ser os seguintes itens:

- Arquivos (arquivos executáveis ou arquivos de dados);
- Diretórios;
- Pipes nomeados;
- Links;
- Dispositivos;
- O arquivo pkginfo;

O pkginfo é um arquivo de informação do pacote necessário que define os valores de parâmetro. Os valores de parâmetro são a **abreviação, o nome completo e a arquitetura do pacote**.

Raíz

O conceito de encapsular estrutura e comportamento em um tipo não é exclusivo da orientação a objetos; particularmente, a programação por tipos abstratos de dados segue esse mesmo conceito. O que torna a orientação a objetos única é o conceito de **herança**.

Herança é um mecanismo que permite que características comuns a diversas classes sejam fatoradas em uma classe base, ou superclasse. A partir de uma classe base, outras classes podem ser especificadas. Cada classe derivada ou subclasse apresenta as características (estrutura e métodos) da classe base e acrescenta a elas o que for definido de particularidade para ela.

Sendo uma linguagem de programação orientada a objetos, Java **oferece mecanismos para definir classes derivadas a partir de classes existentes**. É fundamental que se tenha uma boa compreensão sobre como objetos de classes derivadas são criados e manipulados, assim como das restrições de acesso que podem se aplicar a membros de classes derivadas. Também importante para uma completa compreensão da utilização desse mecanismo em Java é a compreensão de **como relacionam-se interfaces e herança**.

Herança é **sempre utilizada em Java, mesmo que não explicitamente**. Quando uma classe é criada e não há nenhuma referência à sua superclasse, implicitamente a classe criada é derivada diretamente da classe Object. É por esse motivo que to**dos os objetos podem invocar os métodos da classe Object,** tais como equals() e toString().

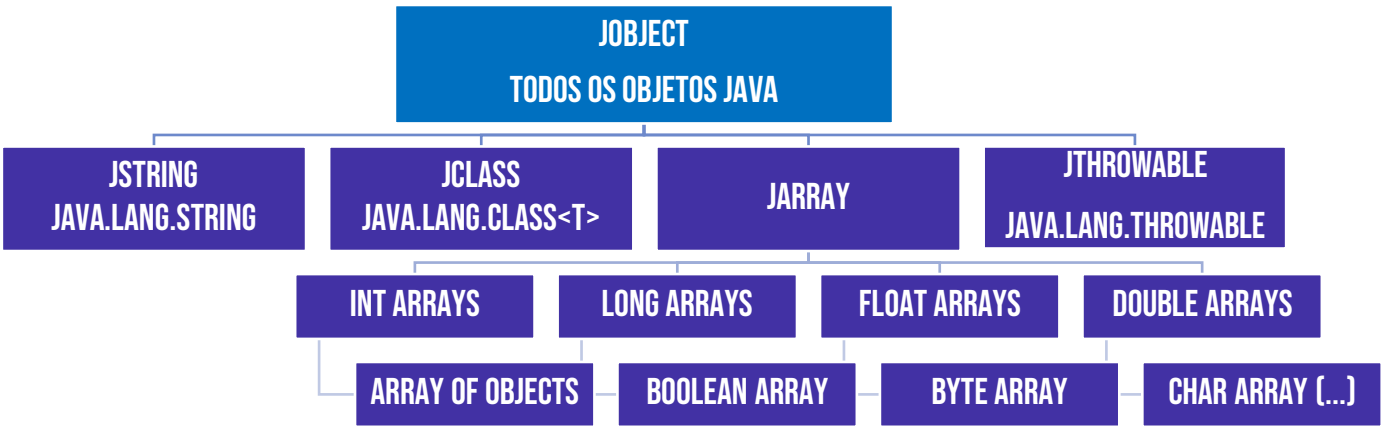
HERANÇA

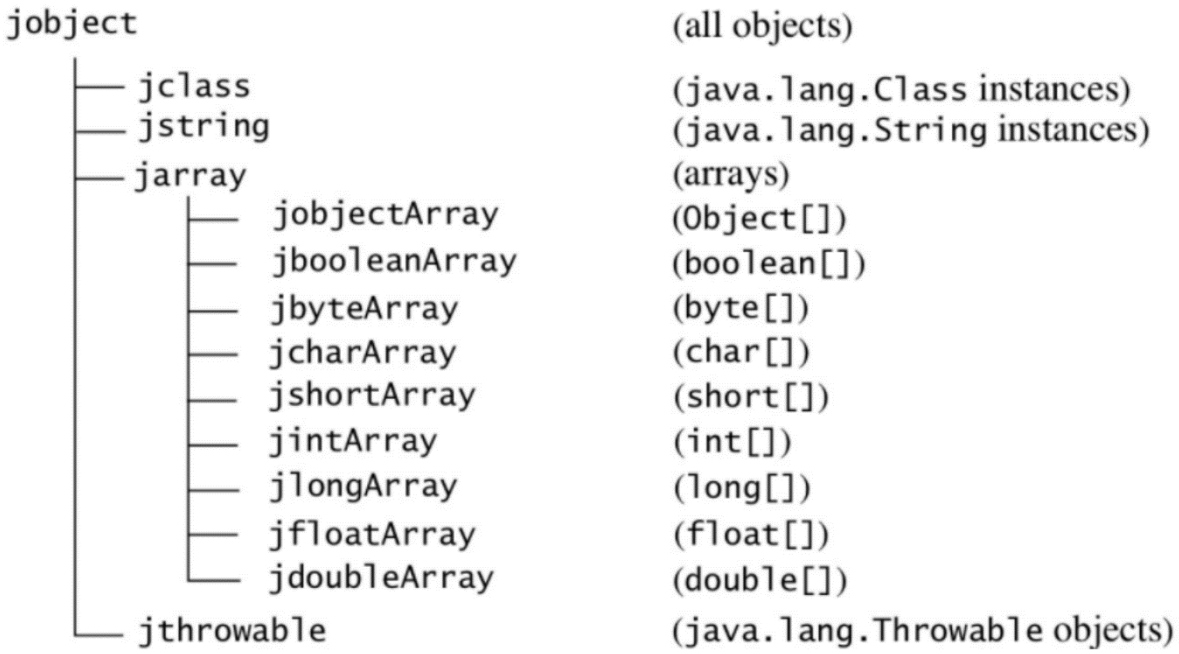
MECANISMO QUE PERMITE QUE CARACTERÍSTICAS COMUNS A DIVERSAS CLASSES SEJAM FATORADAS EM UMA CLASSE BASE, OU SUPERCLASSE

CADA CLASSE DERIVADA OU SUBCLASSE APRESENTA AS CARACTERÍSTICAS DA CLASSE BASE

SEMPRE UTILIZADA EM JAVA, MESMO QUE NÃO EXPLICITAMENTE

JAVA OFERECE MECANISMOS PARA DEFINIR CLASSES DERIVADAS A PARTIR DE CLASSES EXISTENTES





Pessoal, java é composto por uma **infinidade de pacotes e classes** que podem ser acessados no índice: https://docs.oracle.com/javase/8/docs/api/index.html

Applet

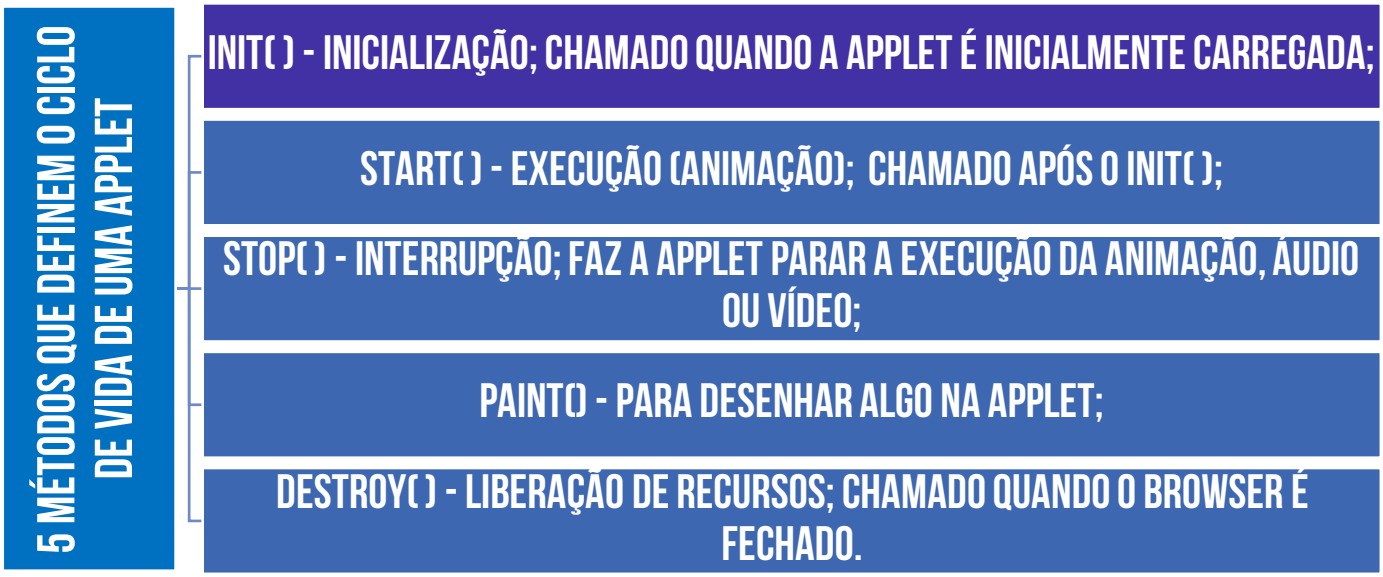
Uma **applet** é uma **pequena aplicação executada em uma janela** de uma aplicação (browser/appletviewer). Tem por finalidade estender as funcionalidades de browsers, adicionando som, animação, etc., provenientes de fontes (URLs) locais ou remotas, sendo que cada página web (arquivo .html) pode conter uma ou mais applets.

O desenvolvimento ou não de programas em Java sob a forma de applets depende do objetivo da aplicação. Se por um lado a execução necessita de um navegador, por outro pode ser disponibilizada para execução via internet. Cabe ao projetista da aplicação definir a forma mais adequada para seu sistema.

Applets sempre executam nos clientes web, nunca nos servidores. Por esta razão a carga das classes pode levar algum tempo. Para definir programas Java que rodem no servidor, veja como construir servlets.

Os 5 métodos que definem o ciclo de vida de uma applet são:

1. init( ) - inicialização; chamado quando a applet é inicialmente carregada;
2. start( ) - execução (animação); chamado após o init( );
3. stop( ) - interrupção; faz a applet parar a execução da animação, áudio ou vídeo;
4. paint() - para desenhar algo na applet;
5. destroy( ) - liberação de recursos; chamado quando o browser é fechado.



(Aeronautica / EAOAp – 2016 - Adaptado) Applets são considerados programas em Java, que são tipicamente incorporados a documentos XHTML, que também podem ser chamadas páginas web. Acerca dos applets em Java, analise as afirmativas a seguir:

I. O método start é chamado uma vez pelo contêiner de applets para inicializar um applet quando ele é carregado.

II. O método destroy é chamado pelo contêiner de applets quando o applet é removido da memória.

III. Cinco métodos do ciclo de vida de um applet são chamados pelo contêiner de applets entre o momento em que o applet é carregado no navegador e o momento em que ele é fechado pelo navegador.

Estão corretas apenas as afirmativas

a) I e II.
b) I e III.
c) II e III.
d) III, apenas.

_____
Comentários: Pessoal, Os 5 métodos que definem o ciclo de vida de uma applet são: init , start, stop e destroy.1.init( ) - inicialização; chamado quando a applet é inicialmente carregada; 2.start( ) - execução (animação); chamado após o init( ); 3.stop( ) - interrupção; faz a applet parar a execução da animação, áudio ou vídeo; 4.paint() - para desenhar algo na applet; 5. destroy( ) - liberação de recursos; chamado quando o browser é fechado. O item I está errado, o método init é chamado quando a applet é inicialmente carregada. O item II está correto, destroy é chamado pelo contêiner de applets quando o applet é removido da memória. E por fim, o item III está correto. (Gabarito: Letra C).

Toda applet é uma aplicação gráfica, não existindo, portanto, applets "modo texto". A principal diferença entre uma "Java application" e uma "applet" é o fato de que a janela base da aplicação é derivada a partir da classe Applet (ou JApplet) e não a partir da classe Frame. Além disso, a parte da aplicação que instancia a classe Applet e relaciona-a com o browser é padrão e, portanto, não precisa ser descrita. Desta forma, applets não possuem a função "main( )".

(CESPE / ANATEL – 2012) O Firefox 13, ou versão superior, disponibiliza suporte para o uso de applets e aplicações Java, desde que esta ferramenta esteja devidamente instalada no computador em uso e o plugin com suporte à Java esteja habilitado no navegador.

_____
Comentários: Pessoal, de fato, applets têm por finalidade estender as funcionalidades de browsers, adicionando som, animação, etc., provenientes de fontes (URLs) locais ou remotas. No caso do Firefox, deve estar devidamente instalada no computador em uso e o plugin com suporte à Java habilitado. (Gabarito: Correto).

(UFMT / UFMT – 2014) Applets Java são pequenos programas que podem ser embutidos em uma página Web e executados quando um navegador carrega essa página.

_____
Comentários: Pessoal, perfeita questão! (Gabarito: Correto).

(AOCP / UNIR– 2018) O Applet é um programa especial escrito em Java e adaptado para execução de uma atividade específica dentro de páginas HTML. Assim, essas páginas podem ser visualizadas em um browser.

_____

Comentários: Pessoal, applets sempre executam nos clientes web, são escritos em Java, podem ser visualizadas em um browser. (Gabarito: Correto).

Sintaxe

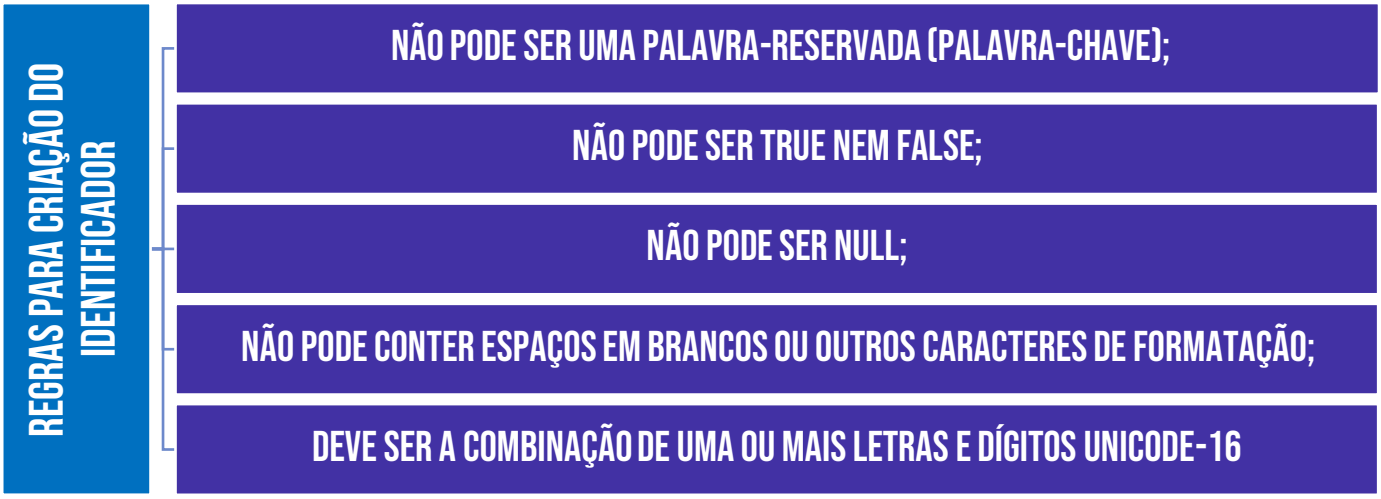
Identificadores

O Java é **case sensitive**, ou seja, Java **diferencia maiúsculas de minúsculas**: "MyClass" e "myclass" têm significados diferentes assim como Analista é diferente de analista.

Por convenção, **os nomes de classes iniciam com uma letra maiúscula** e apresentam a letra inicial de cada palavra que eles incluem em maiúscula (por exemplo, SampleClassName). O nome de uma classe é um identificador — uma série de caracteres que consiste em letras, dígitos, sublinhados (_) e sinais de cifrão ($) que não inicie com um dígito e não contenha espaços. Alguns identificadores válidos são Welcome1, $valor, _valor, m_campoDeEntrada1 e botao7. O nome 7botao não é um identificador válido porque inicia com um dígito, e o nome campo de entrada não é um identificador válido porque contém espaços. Normalmente, um identificador que não inicia com uma letra maiúscula não é um nome de classe. O Java faz distinção entre maiúsculas e minúsculas — letras maiúsculas e letras minúsculas são diferentes — assim, value e Value são identificadores diferentes (mas ambos válidos).

Utilizamos, em Java, as seguintes **regras para criação do identificador**:

- não pode ser uma palavra-reservada (palavra-chave);
- não pode ser true nem false - literais que representam os tipos lógicos (booleanos);
- não pode ser null - literal que representa o tipo nulo;
- não pode conter espaços em brancos ou outros caracteres de formatação;
- deve ser a combinação de uma ou mais letras e dígitos UNICODE-16. Por exemplo, no alfabeto latino, teríamos:
    - letras de A a Z (de \u0041 a \u005a);
    - letras de a a z (de \u0061 a \u007a);
    - sublinha _ (\u005f);
    - cifrão $ (\u0024);
    - dígitos de 0 a 9 (de \u0030 a \u0039).



Observações: caracteres compostos (acentuados) não são interpretados igualmente aos não compostos (não acentuados). Por exemplo, História e História não são o mesmo identificador. Letras maiúsculas e minúsculas diferenciam os identificadores, ou seja, **a** é um identificador diferente de **A**, **História é diferente de história** etc.

Blocos e Comandos

Blocos de programação são aglomerados de instruções e declarações que têm escopo conjunto. Ou seja, as variáveis definidas como locais dentro de um bloco somente serão presentes dentro deste bloco assim como as instruções ali presentes. Os blocos de programação são delimitados por chaves e podem ser aninhados - colocados um dentro dos outros. Por exemplo:

```
{
   // este é um bloco de programação
   int a=10;
}
```

```
{
   // este é um bloco de programação
   int a=10;
   int b=1;
   if (b==3) {
       // este é um bloco que é executado se b for igual a 3
       b=a*10;
   } else {
       // este é um bloco que é executado se b for diferente de 3
       int a=100;
       b=a*10;
   }
   System.out.println("O valor de b é " + b);
}
```

Comentários

Inserimos comentários para documentar programas e aprimorar sua legibilidade. O compilador Java ignora os comentários, portanto, eles não fazem o computador realizar qualquer ação quando o programa é executado. Por convenção, iniciamos cada programa com um comentário indicando o número da figura e o nome do arquivo.

```
// Welcome1.java
// Programa de impressão de texto.
```

O Java também tem comentários tradicionais, que podem ser distribuídos ao longo de várias linhas, como em

```
/* Esse é um comentário tradicional.
Ele pode ser dividido em várias linhas */
```

Eles começam e terminam com delimitadores, /* e */. O compilador ignora todo o texto entre os delimitadores. O Java incorporou comentários tradicionais e comentários de fim de linha das linguagens de programação C e C++, respectivamente. Preferimos usar comentários //.

O Java fornece comentários de um terceiro tipo: comentários Javadoc. Esses são delimitados por //** e */. O compilador ignora todo o texto entre os delimitadores. Os comentários no estilo Javadoc permitem-lhe incorporar a documentação do programa diretamente aos seus programas. Esses comentários são o formato de documentação Java preferido na indústria. O programa utilitário javadoc (parte do JDK) lê comentários Javadoc e os usa para preparar a documentação do programa no formato HTML.

Palavras Reservadas

Em programação, **palavras-chave**, ou **palavras reservadas**, são as palavras que **não podem ser usadas como identificadores**, ou seja, **não podem ser usadas como nome de variáveis, nome de classes, etc.** Estas palavras são assim definidas ou porque já têm uso na sintaxe da linguagem ou porque serão usadas em algum momento, seja para manter compatibilidade com versões anteriores ou mesmo com outras linguagens. No caso do Java temos as seguintes palavras-chave:

PALAVRA	DESCRIÇÃO
ABSTRACT	Um modificador sem acesso. Usado para classes e métodos: Uma classe abstrata não pode ser usada para criar objetos (para acessá-la, ela deve ser herdada de outra classe). Um método abstrato só pode ser usado em uma classe abstrata e não possui um corpo. O corpo é fornecido pela subclasse (herdado de)
BOOLEAN	Um tipo de dados que pode armazenar apenas valores verdadeiros e falsos
BREAK	Sai de um loop ou de um bloco de repetição
BYTE	Um tipo de dados que pode armazenar números inteiros de -128 e 127
CASE	Marca um bloco de código em instruções switch
CATCH	Captura exceções geradas por instruções try
CHAR	Captura exceções geradas por instruções try
CLASS	Define uma classe
CONST	Define uma constante. Não está em uso - use final.
CONTINUE	Continua para a próxima iteração de um loop
DEFAULT	Especifica o bloco de código padrão em uma instrução switch
DO	Usado junto com while para criar um loop do-while.

DOUBLE	Um tipo de dados que pode armazenar números inteiros de 1,7e−308 a 1,7e+308
ELSE	Usado em instruções condicionais
ENUM	Declara um tipo enumerado (imutável)
EXTENDS	Estende uma classe (indica que uma classe é herdada de outra classe)
FINAL	Um modificador sem acesso usado para classes, atributos e métodos, o que os torna inalteráveis (impossíveis de herdar ou substituir)
FINALLY	Usado com exceções, um bloco de código que será executado independente de haver uma exceção ou não
FLOAT	Um tipo de dados que pode armazenar números inteiros de 3.4e−038 a 3.4e+038
FOR	Criar um loop for
GOTO	Não está em uso
IF	Faz uma declaração condicional
IMPLEMENTS	Implementa uma interface
IMPORT	Usado para importar um pacote, classe ou interface
INSTANCEOF	Verifica se um objeto é uma instância de uma classe específica ou uma interface
INT	Um tipo de dados que pode armazenar números inteiros de -2147483648 a 2147483647
INTERFACE	Usado para declarar um tipo especial de classe que contém apenas métodos abstratos
LONG	Um tipo de dados que pode armazenar números inteiros de -9223372036854775808 a 9223372036854775808
MÓDULO	Declara um módulo. Novo no Java 9
NATIVE	Especifica que um método não é implementado no mesmo arquivo de origem Java (mas em outra linguagem)
NEW	Cria novos objetos
PACKAGE	Declara um pacote
PRIVATE	Um modificador de acesso usado para atributos, métodos e construtores, tornando-os acessíveis apenas dentro da classe declarada
PROTECTED	Um modificador de acesso usado para atributos, métodos e construtores, tornando-os acessíveis no mesmo pacote e subclasses
PUBLIC	Um modificador de acesso usado para classes, atributos, métodos e construtores, tornando-os acessíveis por qualquer outra classe
REQUIRES	Especifica as bibliotecas necessárias dentro de um módulo. Novo no Java 9
RETURN	Finalizou a execução de um método e pode ser usado para retornar um valor de um método
SHORT	Um tipo de dados que pode armazenar números inteiros de -32768 a 32767
STATIC	Um modificador sem acesso usado para métodos e atributos. Métodos/atributos estáticos podem ser acessados sem criar um objeto de uma classe
STRICTFP	Restringir a precisão e o arredondamento dos cálculos de ponto flutuante
SUPER	Refere-se a objetos da superclasse (pai)

SWITCH	Seleciona um dos muitos blocos de código a serem executados
SYNCHRONIZED	Um modificador sem acesso, que especifica que os métodos só podem ser acessados por um thread por vez
THIS	Refere-se ao objeto atual em um método ou construtor
THROW	Cria um erro personalizado
THROWS	Indica quais exceções podem ser lançadas por um método
TRANSIENT	Um modificador sem acesso, que especifica que um atributo não faz parte do estado persistente de um objeto
TRY	Cria uma instrução try...catch
VAR	Declara uma variável. Novo no Java 10
VOID	Especifica que um método não deve ter um valor de retorno
VOLATILE	Indica que um atributo não é armazenado em cache localmente e é sempre lido da "memória principal"
WHILE	Cria um loop while

Pessoal, estes modificadores são MUITO importantes e caem **MUITO** nas provas! Portanto repeti uma tabela com os modificadores de acesso!

MODIFICADORES	DESCRIÇÃO
DEFAULT OU PACKAGE	A classe e/ou seus membros são acessíveis somente por classes do mesmo pacote, na sua declaração não é definido nenhum tipo de modificador, sendo este identificado pelo compilador. Permite que apenas classes do mesmo pacote tenham acesso as propriedades que possuem este modificador.
PUBLIC	O código é acessível para todas as classes
PRIVATE	Membros da classe definidos como private não podem ser acessados ou usados por nenhuma outra classe. Esse modificador não se aplica às classes, somente para seus métodos e atributos. Esses atributos e métodos também não podem ser visualizados pelas classes herdadas.
PROTECTED	O modificador protected torna o membro acessível às classes do mesmo pacote ou através de herança, seus membros herdados não são acessíveis a outras classes fora do pacote em que foram declarados.

NOME	CLASSE	SUBCLASSE	PACOTE	GLOBAL
PRIVADO	Sim	Não	Não	Não
PROTEGIDO	Sim	Sim	Não	Não
PÚBLICO	Sim	Sim	Sim	Sim
PACOTE	Sim	Não	Sim	Não

Tipos Primitivos

Os tipos do Java são divididos em primitivos e por referência. Os tipos primitivos são **int, boolean, byte, char, short, long, float e double.**

Java requer que todas as variáveis tenham um tipo. Por essa razão, o Java é referido como linguagem fortemente tipada. Em C e C++, os programadores frequentemente têm de escrever versões separadas dos programas a fim de que ele suporte diferentes plataformas de computador, uma vez que não há garantia de que tipos primitivos sejam idênticos entre um computador e outro. Por exemplo, um int em uma máquina pode ser representado por 16 bits (2 bytes) de memória, em uma segunda máquina por 32 bits (4 bytes) e em outra máquina por 64 bits (8 bytes). No Java, valores int são sempre de 32 bits (4 bytes).

Uma variável de tipo primitivo pode armazenar exatamente um valor de seu tipo declarado por vez. Por exemplo, uma variável int pode armazenar um número inteiro de cada vez. Quando outro valor é atribuído a essa variável, ele substitui o anterior — que é perdido. Lembre-se de que as variáveis locais não são inicializadas por padrão. Já as variáveis de instância de tipo primitivo são inicializadas por padrão — dos tipos byte, char, short, int, long, float e double como 0, e as do tipo boolean como false.

Os programas utilizam as variáveis de tipo por referência (normalmente chamadas referências) para armazenar as localizações de objetos na memória do computador. Dizemos que essa variável referencia um objeto no programa. Objetos que são referenciados podem conter muitas variáveis de instância.

Todo tipo primitivo tem uma classe empacotadora de tipo correspondente (no pacote java.lang). Essas classes chamam-se **Boolean, Byte, Character, Double, Float, Integer, Long e Short.** Elas permitem manipular valores de tipo primitivo como objetos.

Cada uma das classes empacotadoras de tipo numéricas — **Byte, Short, Integer, Long, Float e Double** — estende a classe Number. Além disso, as classes empacotadoras de tipo são classes final, então **não é possível estendê-las**. **Os tipos primitivos não têm métodos**, então os métodos relacionados a um tipo primitivo estão localizados na classe empacotadora de tipo correspondente (por exemplo, o método parseInt, que converte uma String em um valor int, está localizado na classe Integer).

NOME	TIPO	TAMANHO	MÍNIMO	MÁXIMO	DEFAULT
LÓGICO	boolean	-	false	true	false
CARACTERE	char	16 bits	0	$2^{16}-1$	'\u0000'
INTEIRO	byte	8 bits	-2^{7}	$2^{7}-1$	0
	short	16 bits	-2^{15}	$2^{15}-1$	0
	int	32 bits	-2^{31}	$2^{31}-1$	0

DECIMAL	long	64 bits	-2^{63}	$2^{63}-1$	0
	float	32 bits	7 Casas Decimais		0.0
	double	64 bits	15 Casas Decimais		0.0

O Java fornece conversões boxing e unboxing que convertem automaticamente entre valores de tipo primitivo e objetos empacotadores de tipo. Uma conversão boxing converte um valor de um tipo primitivo em um objeto da classe empacotadora de tipo correspondente. Uma conversão unboxing converte um objeto de uma classe empacotadora de tipo em um valor do tipo primitivo correspondente. Essas conversões são executadas automaticamente — o que é chamado de autoboxing e auto-unboxing. Considere as seguintes instruções:

```
Integer[] integerArray = new Integer[5]; // cria integerArray
integerArray[0] = 10; // atribui Integer 10 a integerArray[0]
int value = integerArray[0]; // obtém valor int de Integer
```

Nesse caso, o autoboxing ocorre ao atribuir-se um valor int (10) a Array[0] integer, porque Array integer armazena referências a objetos Integer, não valores int. O auto-unboxing ocorre ao atribuir-se integerArray[0] à variável int value, porque a variável value armazena um valor value, não uma referência a um objeto Integer. As conversões boxing também ocorrem em condições que podem ser avaliadas para valores boolean primitivos ou objetos Boolean

(FCC / TRF 3ª Região – 2019) Java possui um conjunto de tipos de dados conhecidos como primitivos, dos quais NÃO faz parte o tipo

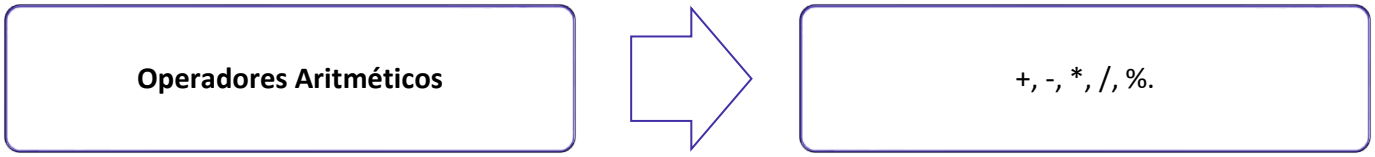
a) short.
b) long.
c) string.
d) byte.
e) float.

_____

Comentários: Pessoal, vejamos os nomes e os tipos primitivos existentes: lógico: boolean; caractere: char; inteiro: byte, short, int, long, float; decimal: double. Das opções apresentadas, a única que não consta é a string. (Gabarito: Letra C).

Operadores

Operadores são símbolos que representam atribuições, cálculos e ordem dos dados. As operações seguem uma ordem de prioridades, ou seja, alguns cálculos (ou outros) são processados antes de outros. Por exemplo, na Álgebra podemos mostrar a seguinte ordem: primeiro divisão e multiplicação e depois soma e subtração.

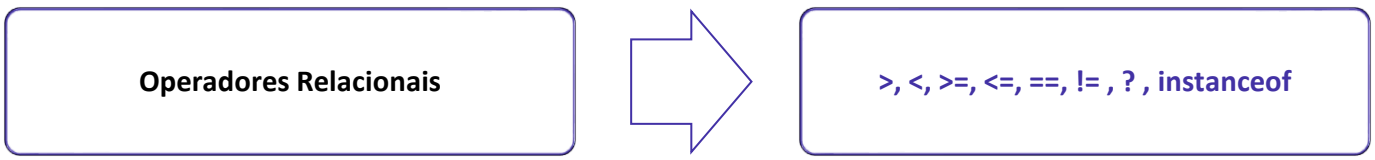


ARITMÉTICOS	ATRIBUIÇÃO	RELACIONAIS	LÓGICOS	BIT A BIT
+	=	>	!	&
-	+=	<	&&	|
*	-=	>=	||	^
/	*=	<=		<<
%	/=	!=		>>
	%=	==		>>>
	++	?		
	--	instanceof		

Assim, as operações de divisão e multiplicação, por serem de ordem 1, serão executadas antes das operações de soma e subtração (ordem 2). Também, as operações de divisão e multiplicação são de mesma ordem (1) e não importa, entre si, a ordem da operação (2 dividido por 4 vezes 9 é igual a 2 vezes 9 dividido por 4).

Os operadores são divididos em 3 tipos em relação à quantidade de operandos no qual operam: **unário, binário e ternário**.

```
int a = 5, b = 2, c = 0;
a--;                    // -- é um operador unário pois opera apenas em
a;
c = a * b;              // * é um operador binário pois opera em a e b.
c = c < 0 ? a : b;      // ?: é O operador ternário. Opera em na
expressão booleana (c < 0), e em a ou b.
```



Operadores Lógicos: !, &&, ||.

Operadores Bit a Bit (ou Bitwise): &, |, ^, <<, >>, >>>.

Precedência indica a ordem na qual um operador opera em relação à avaliação de uma expressão. A tabela seguinte elenca os operadores por precedência, do maior para o menor.

TIPO DE OPERADOR	LISTA DE OPERADORES
SUFIXAIS	expr++ expr--
PREFIXAIS	++expr --expr +expr -expr ~ !
MULTIPLICATIVOS	* / %
ADITIVOS	+ -
SHIFT BINÁRIO	<< >> >>>
COMPARATIVOS	< > <= >= instanceof
IGUALDADE	== !=
BIT-A-BIT E	&
BIT-A-BIT XOU OR	^
BIT-A-BIT OU OR	|
LÓGICO E	&&
LÓGICO OU	||
TERNÁRIO	? :
ATRIBUIÇÃO	= += -= *= /= %= &= ^= |= <<= >>= >>>=

Precedência o: atribuições

Os operadores de atribuição são os mais numerosos e os que tem uma prioridade menor de serem interpretados. Ele armazena o valor que aparecer à direita na variável presente à esquerda. Caso deseje-se que a variável da esquerda receba o valor dela mesma após passar por alguma operação com um segundo valor, basta colocar o símbolo da operação antes do sinal "=" e colocar o segundo valor à direita. Exemplos de uso:

```
 int numero = 3; //numero recebe o valor 3
 numero += 7;    //numero recebe 3+7. Ou seja, 10.
 numero -= 32;   //numero recebe o seu valor menos 32. Ou seja, -22.
 numero %= -3;   //numero recebe o resto da divisão entre seu valor e -
3. Ou seja, -1.
 numero *= 6;    //numero recebe o seu valor vezes 6. Ou seja, -6.
 numero /= 2;    //numero recebe o seu valor dividido por 2. Ou seja, -
3.
```

Quando em uma mesma linha forem encontrados vários operadores diferentes, serão executados primeiro aqueles que tiverem maior precedência. Se existirem operadores com o mesmo valor de

precedência, será realizado primeiro aquele cujo símbolo aparecer primeiro. É possível alterar a ordem natural com que são feitas as operações através do uso de parênteses. As operações entre parênteses sempre são realizadas antes.

Precedência 1: operadores ternários

O operador ternário **?** : recebe ao todo três operandos. O primeiro operando deve possuir necessariamente um valor do tipo boolean. Os outros dois operandos podem ser de qualquer tipo. Caso o valor do primeiro operando seja "verdadeiro", o operador retorna um valor igual ao do segundo operando. Caso o seu valor seja "falso", ele retorna um valor idêntico ao terceiro operando. Exemplos de uso:

```
 int numero1=245;
 int numero2=123;
 numero1=(numero1>numero2)?numero1:numero2; /* Faz com que a variavel numero 1 receba sempreo maior valor entre ela mesma e a numero2. Neste caso, ela receberá o seu prório valor por ele ser maior*/
```

Precedência 3 e 2: operadores AND e OR

Os operadores AND e OR só podem ser usados em variáveis e literais do tipo boolean. O operador AND (&&) retorna "verdadeiro" quando seus dois operandos também valem "verdadeiro" e retorna "falso" caso contrário. O operador OR (||) retorna "falso" quando seus dois operandos são falsos e retorna "verdadeiro" caso contrário. Exemplos de uso:

```
 boolean variavel;
 variavel=(2<45)&&(45<2)  //variavel passa a valer "falso"
 variavel=(2<45)||(45<2)  //variavel passa a valer "verdadeiro"
```

Precedência 6, 5 e 4: operadores Bit-a-Bit

Os Operadores Bit-a-Bit são todos aqueles que realizam suas operações sobre os bits de um número, e não sobre o seu valor. Existem ao todo três destes operadores e cada um deles tem um valor de precedência diferente. O que tem precedência mais alta é o AND bit-a-bit (&). Ele analisa dois bits e retorna 1 se ambos forem iguais à 1 e o caso contrário. Depois vem o OR exclusivo bit-a-bit (^) que retorna 1 se os bits forem diferentes e o caso contrário. Por último, vem o operador OR inclusivo (|), que retorna o caso ambos os bits valerem o e retorna 1 caso contrário. Estes operadores podem ser usados em qualquer tipo de dados, desde que possuam o mesmo tamanho em bits. Exemplos de uso:

```
int numero;
 numero=34&435;  //numero passa a valer 34
 numero=34^46;   //numero passa a valer 12
 numero=436|547; //numero passa a valer 951
```

Precedência 7: operadores de igualdade
São semelhantes aos Operadores Comparativos. Eles também recebem números como operandos e retornam um valor boolean. A diferença é que estes operadores apenas verificam se as variáveis são iguais ou não. Como exemplos de operadores assim, pode-se citar o Igual a (==) e Diferente de (!=). Estes operadores podem ser usados em qualquer tipo de variável, desde que elas sejam do mesmo tipo. Exemplos de uso:

```
boolean variavel;
 variavel=(-5==5);    //variavel recebe "falso"
 variavel=(2!=45674); //variavel recebe "verdadeiro"
Ao utilizar operadores de igualdade com objetos, a comparação é feita
entre suas referências. Dessa forma, dois objetos cognitivamente iguais,
podem ser avaliados como diferentes. Exemplo:

class Pessoa{
    String nome;

    public Pessoa(String nome){
        this.nome = nome;
    }
}
new Pessoa("miguel") == new Pessoa("miguel") // comparação avaliada como
falsa
```

Precedência 8: operadores comparativos
São operadores que comparam dois números e retornam em seguido o valor booleano "verdadeiro" ou "falso". Como exemplo, pode-se citar o Menor que(<), Maior que (>), Menor ou Igual que(<=), Maior ou Igual que (>=) e Exemplo de (instanceof). O significado dos quatro primeiros operadores é evidente. Já a operação Exemplo de, retorna "verdadeiro" se o primeiro operando for um Objeto pertencente à classe passada como segundo operando e "falso" caso contrário. Exemplos de uso:

```
boolean variavel;
 variavel=(4<4);   //variavel recebe "falso"
 variavel=(4<=4);  //variavel recebe "verdadeiro"
 variavel=(-1>-3); //variavel recebe "verdadeiro"
 variavel=(-4>=0); //variavel recebe "falso"
```

Precedência 9: operadores de shift

São operadores que deslocam os bits de um número de modo a alternar o seu valor. Exemplos de operadores deste tipo são o Shift para a Direita (>>), o Shift para a Direita Sem-Sinal(>>>) e o Shift para a Esquerda (<<). O primeiro valor a ser recebido pelo operador é o número sobre o qual será realizado um Shift e o segundo número é a quantidade de posições de bits a serem deslocados. Exemplos de uso:

```
int numero=-3;          //numero vale -3
numero=numero>>1;       //numero vale -2
numero=numero<<1;       //numero vale -4
numero=numero>>>1;      //numero vale 2147483646
numero=numero<<1;       //numero vale -4
```

Precedência 10: operadores aditivos

São operadores que realizam alguma operação igual ou equivalente à adição. Assim como os Operadores Multiplicativos, os Aditivos podem ser usados tanto em variáveis como em literais (quando fazem a concatenação de strings). Mas também não podem ser usados em variáveis char e boolean. Eles também não alteram as variáveis passadas para eles. No lugar disso, eles retornam um número que deve ser direcionado par uma variável por meio da operação de atribuição (veja abaixo). Exemplos de uso:

```
int numero=5;           //numero passa a valer 5
 numero=numero+8;        //numero passa a valer 13
 numero=numero-numero;   //numero passa a valer zero
 String x="Alo";         // x é inicializado com a string "Alo"
 String y="Mundo!";      // y é inicializado com a string "Mundo!"
 x = x + ", " + y;       // x passa a valer "Alo, Mundo!"
```

Precedência 11: operadores multiplicativos

São operadores que realizam uma operação igual ou semelhante à multiplicação. Exemplos de operações do tipo são a Multiplicação (*), a Divisão (/) e o Resto (%). O primeiro pode realizar a multiplicação entre dois valores que não sejam do tipo boolean e nem do tipo char. O segundo pode dividir o primeiro número pelo segundo. Também não pode ser usado em valores booleans ou char. O terceiro retorna o resto da divisão do primeiro pelo segundo. Exemplos de uso:

```
int numero=5;         //numero passa a valer 5
 numero=numero*4;     //numero assume o valor 20
 numero=200/numero;   //numero assume o valor 10
 numero=5%12;         //numero assume o valor 5
```

Precedência 12: operadores prefixais

São operadores unários que alteram o valor de uma variável e seus sinais são posicionados antes do identificador da variável. Como exemplo, pode-se citar o Incremento ++, Decremento --, Sinal Positivo +, Sinal Negativo -, Inversão e Incremento ~ e Negação !. O incremento e decremento, já vimos o que faz. Eles estão sendo citados aqui novamente porque seus sinais podem vir antes de variáveis também e numa operação complexa (com outros operadores binários) alteram a precedência da operação. O Sinal Positivo + retorna a variável que vem depois dele com o mesmo sinal, o Sinal Negativo - inverte o sinal de variáveis transformando números positivos em negativo e vice-versa. Ele não pode ser usado em variáveis dos tipos boolean e char. O Incremento e Inversão ~ aumenta o número em uma unidade e inverte o seu sinal. Só pode ser usado em inteiros.

Já a operação de negação ! transforma "verdadeiro" em "falso" e vice-versa, só podendo ser usado em variáveis do tipo boolean. Também só funcionam em variáveis, não em literais. Exemplos de uso:

```
 int numero=5;          //numero contém 5
 boolean ligado=false; //ligado contém "falso"
 ++numero;             //numero agora vale 6
 --numero;             //numero passa a valer 5
 numero=+numero;       //numero continua valendo 5
 numero=-numero;       //numero passa a valer -5
 numero=~numero;       //numero passa a valer 4
 ligado=!ligado;       //ligado passa a representar o valor "true"
```

Observação: uma diferença importante entre os operadores '++' e '--' prefixais e sufixais é o tempo de avaliação da expressão comparado com a alteração da variável. A saber:

```
 int x = 5;         // x contém 5
 int y, z;          // y e z não foram definidos
 y = x++;            // primeiro faz y igual ao valor (anterior) de x, e
depois modifica x
 z = ++x;           // primeiro modifica x,  e depois atribui a z o novo
valor de x
```

Neste exemplo, temos que, ao final x vale 7 (duas vezes incrementado), y vale 5 (o valor inicial de x) e z vale 7 (o valor final de x). Deve-se evitar usar mais de um operador prefixal e sufixal na mesma linha, porque isto torna o código incompreensível, por exemplo: x = (y++ + ++z - --x) + ++y.

Precedência 13: operadores sufixais

São operadores unários posicionados após o identificador da variável para incrementar ++ ou decrementar -- uma variável de tipo numérico em 1. Não podem ser utilizados em variáveis do tipo string, boolean ou de referência. Também não podem ser utilizados em valores de expressão e em literais. Diferente dos operadores de pré incremento/decremento, esses operadores sufixais retornam o valor original da variável para a expressão e depois realizam a operação sobre a variável.

```
int numero = 5;                          //A variável número é inicializada
com 5.
System.out.println(numero++);             //É exibido 5, o valor original,
e então a variável é atualizada para 6.
System.out.println(numero);              //É exibido 6, valor incrementado
na instrução anterior.
```

Pessoal, para entender o que é abordado, é necessário ter conhecimento das operações lógicas. Isso é abordado na aula de Informática Básica.[1]

(FGV / IMBEL – 2021) Com relação aos operadores bitwise do Java, considere os valores binários

a = 00111100
b = 00001101

Os valores resultantes das operações a&b e a|b são, respectivamente,

a) 00011100 e 11111101
b) 00001100 e 00111101
c) 00001111 e 00111111
d) 11001110 e 00001100
e) 01101100 e 00100101

_____

Comentários: Pessoal, vamos fazer essa operação! O ideal é fazer em uma folha/tablet – em forma escrita à mão. Vamos lá, a = 00111100 & b = 00001101 = Do último dígito para o primeiro (0&1) = 0, (0 & 0) = 0, (1 & 1) = 1, (1 & 1) = 1, (1 & 0) = 0, (1 & 0) = 0, (0 & 0) = 0, (0 & 0) = 0. O resultado de A&B é 00001100. Agora vamos ao segundo a|b, novamente do fim para o início: (0|1) = 1, (0 | 0) = 0, (1 | 1) = 1, (1 | 1) = 1, (1 | 0) = 1, (1 | 0) = 1, (0 | 0) = 0, (0 | 0) = 0. O resultado de A|B é 00111101. (Gabarito: Letra B).

[1] Sistemas Numéricos

Vetores

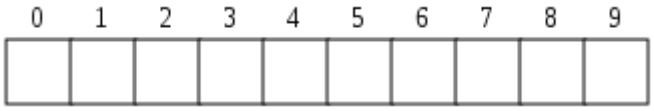
Um array ou vetor é um grupo de variáveis (chamados elementos ou componentes) que contém valores todos do mesmo tipo. É uma estrutura de dados formada por um conjunto de dados ou outros elementos de um mesmo tipo ou uma mesma estrutura. Pode ter uma dimensão ou mais. Também chamado de matriz quando de duas dimensões, funciona de modo análogo às matrizes matemáticas. O acesso aos dados é feito através de "coordenadas" (índices).

Como você logo verá, o que em geral consideramos um array é, na verdade, uma referência a um objeto array na memória. Ou seja, os arrays são objetos; portanto, são considerados tipos por referência. Os elementos de um array podem ser tipos primitivos ou tipos por referência.

Para referenciar um elemento particular em um array, especificamos o nome da referência para o array e o número de posição do elemento no array. O número de posição do elemento é chamado de índice.

A programação faz um grande uso de vetores. Cada item de um vetor é chamado de elemento. Cada um dos elementos possui uma posição dentro do vetor, à qual referenciamos através do índice do elemento. Cada um dos "domínios" (conjunto de posições, endereços de armazenamentos) do vetor, nós chamamos de dimensão. Já o tipo de dado (ou de elemento) corresponde ao "contradomínio" do vetor, ou seja, o conjunto de literais ou de outro tipo de elemento que o vetor pode armazenar.

Veja um vetor de uma dimensão e dez elementos:



Ele possui 10 elementos que são acessados (referenciados) pelos índices [0], [1], [2], [3], [4], [5], [6], [7], [8], [9]. Os índices de cada elemento são escritos entre colchetes [ ]. Para declarar um vetor, utiliza-se a seguinte sintaxe:

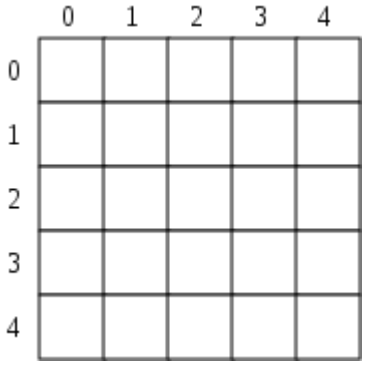
```
tipo[] identificador;
```

ou

```
tipo identificador[];
```

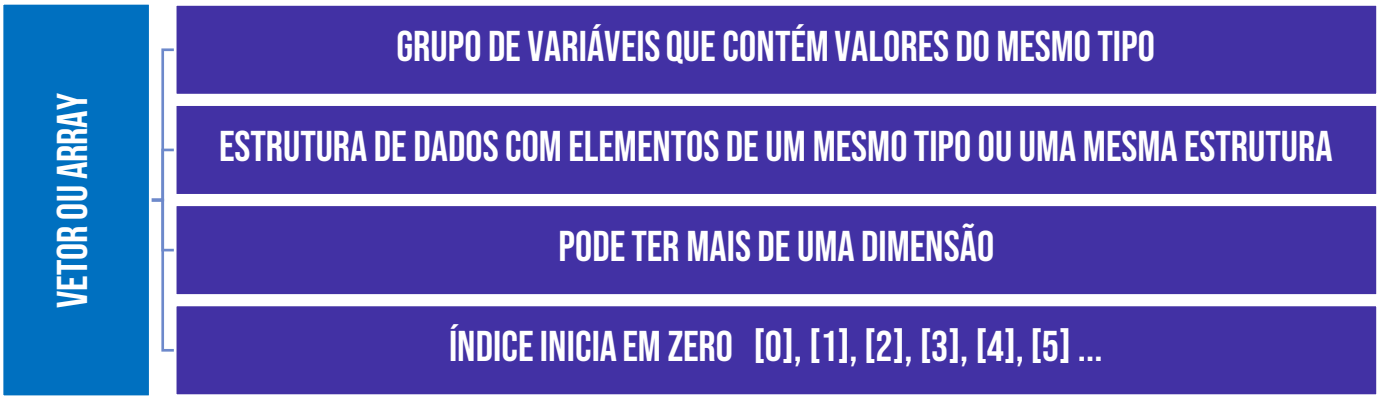
Onde:
tipo = tipo de dado ou de elemento;
identificador = nome do vetor.

Veja um vetor de duas dimensões e vinte e cinco elementos (5 x 5):



Ele possui 25 elementos que são acessados (referenciados) pelos índices:
[0][0], [1][0], [2][0], [3][0], [4][0],
[0][1], [1][1], [2][1], [3][1], [4][1],
[0][2], [1][2], [2][2], [3][2], [4][2],
[0][3], [1][3], [2][3], [3][3], [4][3],
[0][4], [1][4], [2][4], [3][4], [4][4].

Cada elemento é representado por dois índices (um para cada dimensão), entre colchetes e adjacentes [ ][ ]. É como uma matriz.



Os elementos dos vetores podem ser atribuídos da seguinte forma: separados por vírgulas e entre chaves. Por exemplo, em um vetor unidimensional poderíamos indicar os dados de seus elementos assim: {x0,x1,x2, ... ,xn}. Em que, cada um dos valores X é um valor do elemento de índice correspondente no vetor. A numeração dos índices começa a partir do zero e pode ser somente número natural, ou seja, inteiro maior ou igual a zero.

Vetores podem ser declarados e iniciados conforme o seguinte exemplo - cada dimensão é delimitada por chaves e cada elemento é separado do outro através de vírgulas:

```
int[] vetor={34, 27, 3, 2};
```

Outro modo de uso é declarando, iniciando com o tamanho (quantidade de elementos) do vetor e depois atribuindo os valores, como o equivalente a seguir:

```
int[] vetor= new int[4];
vetor={34, 27, 3, 2};
```

Ou então, atribuindo os elementos individualmente:

```
int[] vetor= new int[4];
vetor[0]=34;
vetor[1]=27;
vetor[2]=3;
vetor[3]=2;
```

Para descobrir quantos elementos um array possui, use a propriedade length():

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
System.out.println (cars.length);
```

Você pode percorrer os elementos do array com o loop for e usar a propriedade length para especificar quantas vezes o loop deve ser executado. O exemplo a seguir gera todos os elementos do array cars:

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
for (int i = 0; i < cars.length; i++) {
  System.out.println(cars[i]);
}
```

Lembre-se que o acesso ao elemento do array se dá pelo índice. Neste exemplo foi utilizado o índice i – uma variável criada dentro do for (int i = 0).

Outra possibilidade é utilizar o loop "for-each", que é usado **exclusivamente** para percorrer elementos em arrays. A sintaxe é:

```
for (type variable : arrayname) {
  ...
}
```

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
for (String i : cars) {
  System.out.println(i);
}
```

Veja que a diferença é sutil. Está apenas no **"for (String i : cars)".** O exemplo acima pode ser lido assim: para cada elemento String (chamado i – como no índice) em cars , imprima o valor de i .

Se você comparar o loop **for** com o loop **for-each** , verá que o método **for -each** é **mais fácil de escrever, não requer um contador** (usando a propriedade length) e **é mais legível**.

(FGV / IMBEL – 2021) No contexto da linguagem Java, considere um array definido como segue.

String[] nomes = new String[100];

Assinale a expressão cujo valor corresponde ao índice do último elemento do array nomes.

a) nomes.length
b) nomes.length - 1
c) nomes.size
d) nomes.size + 1
e) nomes.ubound

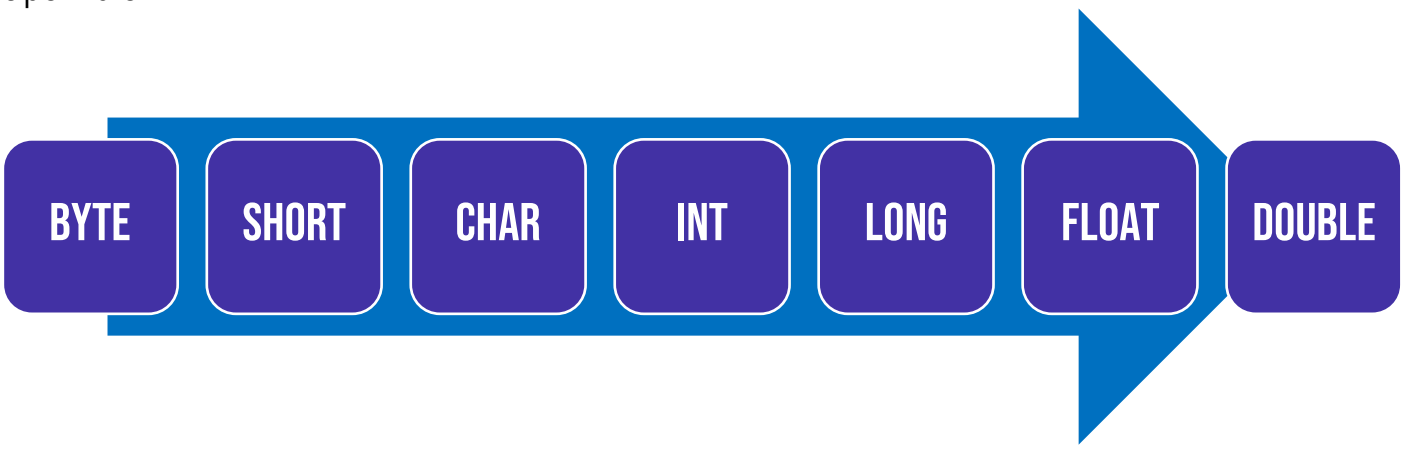
_____

Comentários: Pessoal, para retornar o comprimento de uma string especificada usamos o método length(). No caso em questão, como foi solicitado o ultimo elemento do array, o correto seria utilizar o length – 1. (Gabarito: Letra B).

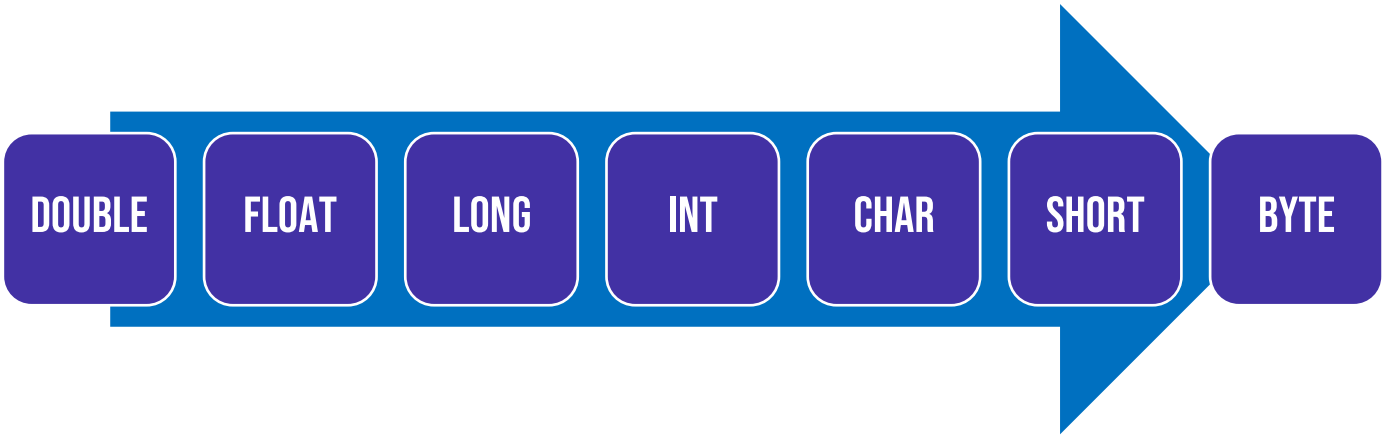
Conversão de Tipos

A conversão de tipo é quando você atribui um valor de um tipo de dados primitivo a outro tipo. Em Java, existem dois tipos de conversão:

Ampliação da transmissão (automaticamente) - convertendo um tipo menor em um tamanho de tipo maior:



Estreitando Casting (manualmente) - convertendo um tipo maior para um tipo de tamanho menor (caminho inverso)



A conversão de ampliação é feita automaticamente ao passar um tipo de tamanho menor para um tipo de tamanho maior:

```
public class Main {
  public static void main(String[] args) {
    int myInt = 9;
    double myDouble = myInt; // Automatic casting: int to double

    System.out.println(myInt);      // Outputs 9
    System.out.println(myDouble);   // Outputs 9.0
  }
```

```
}
```

A redução da conversão deve ser feita manualmente, colocando o tipo entre parênteses na frente do valor:

```
public class Main {
  public static void main(String[] args) {
    double myDouble = 9.78d;
    int myInt = (int) myDouble; // Manual casting: double to int

    System.out.println(myDouble);   // Outputs 9.78
    System.out.println(myInt);      // Outputs 9
  }
}
```

Controle de Fluxo

Como pode-se notar, o conhecimento adquirido até aqui não permite a criação de programas interativos. Eles sempre executam da mesma forma, independente do que o usuário faz ou dos valores que são recebidos. Mas agora iremos aprender a usar alguns comandos que permitem que dependendo das circunstâncias, os programas executem instruções diferentes. A instrução de seleção única if, também conhecida por if-then, possibilita a execução condicional de um bloco de instruções.

```
if (expressaoBooleana) {
    //instruções que serão executadas caso a expressaoBooleana resulte true.
}
```

A expressão Booleana pode ser uma das condições lógicas usuais da matemática:

- Menor que: a < b
- Menor ou igual a: a <= b
- Maior que: a > b
- Maior ou igual a: a >= b
- igual a a == b
- Diferente de: a != b

Você pode usar essas condições para executar ações diferentes para decisões diferentes. Depois da palavra-chave if é necessária uma expressão booleana entre parênteses. Caso a expressão booleana resulte no valor true em tempo de execução então o bloco seguinte será executado, caso resulte em false aquele será ignorado. O bloco de instruções pode conter o ou mais instruções. As chaves que delimitam o bloco são opcionais caso se tenha apenas uma instrução a ser executada.

```
int hora = 20;
boolean eManha = false;

// Exemplo 1: com bloco.
if (hora <= 12) {
    eManha = true;
    System.out.print(hora + " AM");
}

//Exemplo 2: sem bloco.
if (!eManha)
    System.out.print(hora - 12 + " PM");
System.out.println(" é o mesmo que " + hora + " horas."); //Esta linha é incondicionalmente exibida
```

A saída gera o resultado: 8 PM é o mesmo que 20 horas.

A instrução if...else, também conhecida como instrução if-then-else, a instrução de seleção dupla if...else tem função complementar à de if: executa instruções no caso da expressão booleana de if resultar em false.

```
if (expressaoBooleana) {
    //instruções que serão executadas caso a expressaoBooleana resulte
true.
} else {
    //instruções que serão executadas caso a expressaoBooleana resulte
false.
}
```

A palavra-chave **else** deve estar logo após da(s) instrução **if**. Após a palavra-chave **else** deve ser colocado o bloco de instruções a serem executadas no caso da expressão booleana de **if** resultar em **false**. Assim como if, as chaves delimitadoras de bloco são opcionais caso haja apenas uma instrução a executar.

```
int hora = 20;

if (hora <= 12)
    System.out.print(hora + " AM");
else
    System.out.print(hora - 12 + " PM");
System.out.println(" é o mesmo que " + hora + " horas."); //Esta linha é
incondicionalmente exibida
```

As instruções **if ou if...else podem ser aninhadas** dentro de outras instruções if ou if...else para casos em que antes de determinadas instruções serem executadas sejam necessárias combinações de resultados de expressões booleanas.

```
if (expressaoBooleana1) {
    if (expressaoBooleana2) {
    // Instruções a serem executadas caso as expressões booleanas 1 e 2
resultem em true.
    } else {
    // Instruções a serem executadas caso a expressão booleana 1 resulte
em true, e a 2 em false.
    }
} else {
     if (expressaoBooleana3) {
    // Instruções a serem executadas caso a expressão booleana 1 resulte
em false, e a 2 em true.
    } else {
```

```
    // Instruções a serem executadas caso as expressões booleanas 1 e 3
resultem em false.
    }
}
```

As instruções **if e if...else aninhadas não apresentam nenhum comportamento diferente do esperado caso não estivessem aninhadas**. Estão sendo abordadas apenas com o intuito de **exemplificar a possibilidade de aumento das ramificações de decisões**.

```
int hora = 20;

if (hora < 0 || hora >= 24)
    if (hora < 0)
        System.out.print("Erro: A hora deve ser maior que 0.");
    else
        System.out.print("Erro: A hora deve ser menor que 24.");
else {
    if (hora <= 12)
        System.out.print(hora + " AM");
    else
        System.out.print(hora - 12 + " PM");
    System.out.println(" é o mesmo que " + hora + " horas."); //Esta
linha é incondicionalmente exibida
}
```

É possível verificar no exemplo acima que a primeira instrução if mesmo contendo mais de uma linha de instruções consegue identificar que o **if...else forma uma única ramificação e assim executar a expressão booleana normalmente**. Isso se deve ao fato que toda else está vinculada a uma if. Já na else com o escopo mais externo, verifica-se chaves delimitadoras de bloco. Essas chaves são necessárias por conta de uma segunda instrução, nomeadamente System.out.println(), que é executada independentemente do resultado da expressão booleana da if...else.

```
int hora = 20;

if (hora < 0)
    System.out.print("Erro: A hora deve ser maior que 0.");
else if (hora >= 24)
    System.out.print("Erro: A hora deve ser menor que 24.");
else if (hora <= 12)
    System.out.print(hora + " AM é o mesmo que " + hora + " horas.");
else
    System.out.print(hora + " PM é o mesmo que " + hora + " horas.");
```

```
}
```

No exemplo acima há um recurso estilístico para indentar o código com a finalidade de aprimorar a legibilidade. As palavras-chave if foram anexadas às palavras-chave else já que as else têm somente uma instrução cada e por isso não necessitam de chaves para delimitar bloco de instruções. O código abaixo tem exatamente a mesma funcionalidade apesar das quebras de linha.

```
int hora = 20;

if (hora < 0)
    System.out.print("Erro: A hora deve ser maior que 0.");
else
    if (hora >= 24)
        System.out.print("Erro: A hora deve ser menor que 24.");
    else
        if (hora <= 12)
            System.out.print(hora + " AM é o mesmo que " + hora + "
horas.");
        else
            System.out.print(hora + " PM é o mesmo que " + hora + "
horas.");
```

A instrução switch por vezes chamada de switch...case possibilita a execução condicional de instruções de acordo com a correspondência entre a expressão avaliada e a constante em case.

```
switch (expressao) {
    case constante1:
        // Instruções
        break;
    case constante2:
        // Instruções
        break;
    case default:
        // Instruções
}
```

Dentro do parâmetro do switch pode ser utilizada expressão que resulte em: byte, short, char, int, String e enum. As chaves que delimitam o bloco são necessárias ainda que só haja uma ramificação do fluxo do código. A palavra-chave case indica as ramificações de código. Deve ser seguida de uma expressão constante que corresponda ao tipo da expressão inserida no parâmetro do switch, e essa expressão constante, por sua vez, deve ser seguida de: que é o carácter que delimita o início do bloco de instruções relativo à case. Após: podem ser inseridas o ou mais

instruções, incluindo a palavra-chave break que será abordada mais adiante. Ao iniciar outra instrução case ou inserir a chave de fechamento do bloco de switch o bloco anterior é encerrado.

```
int dia = 5;
final int segunda = 2;
final int sexta = 6;

switch (dia) {
    case segunda:
        System.out.print("Segunda ");
    case 3:
     System.out.print("Terça ");
    case 4:
     System.out.print("Quarta ");
    case 5:
     System.out.print("Quinta ");
    case sexta:
     System.out.print("Sexta ");
    case 7:
     System.out.print("Sábado ");
    case 0:
    case 1:
     System.out.print("Domingo ");
}
```

Em tempo de execução, a variável dia será comparada com as expressões constantes, definidas em tempo de compilação, de cada case. O case contendo o valor 5 tem valor igual ao da variável dia, então desse ponto em diante todas as instruções serão executadas até que chegue o término do bloco de switch.

Caso seja necessário que apenas sejam executadas instruções vinculadas a determinadas case então deve-se utilizar a instrução **break**. Após a instrução break o fluxo do programa sai do bloco de switch. A instrução break também pode ser usada para sair de um loop. Por outro lado, a instrução continue interrompe uma iteração (no loop), se ocorrer uma condição especificada, e **continua com a próxima iteração no loop**.

```
int dia = 5;
final int segunda = 2;
final int sexta = 6;

switch (dia) {
    case segunda:
        System.out.print("Segunda ");
    case 3:
```

```
     System.out.print("Terça ");
    case 4:
     System.out.print("Quarta ");
    case 5:
     System.out.print("Quinta ");
    case sexta:
     System.out.print("Sexta ");
     break;
    case 7:
     System.out.print("Sábado ");
    case 0:
    case 1:
     System.out.print("Domingo ");
}
System.out.println("\n-> Fora do bloco de instruções de switch.");
```

Com a instrução break inserida no bloco da case com valor sexta, o código será executado da case com valor 5 até essa break referida.

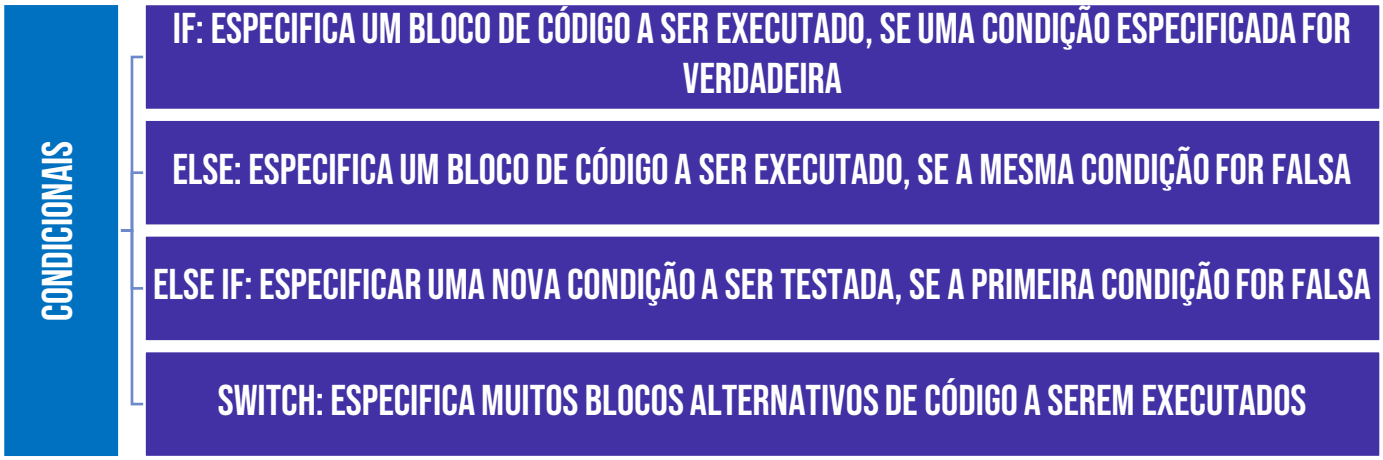
A **instrução default** pode ser utilizada para o caso de **a expressão no parâmetro de switch não corresponder a nenhum dos valores das instruções case**. **default** pode aparecer em qualquer ordem e segue o mesmo funcionamento que case no que tange a bloco de instruções e uso de break.

```
int dia = -5;
final int segunda = 2;
final int sexta = 6;

switch (dia) {
    case segunda:
        System.out.print("Segunda ");
    case 3:
     System.out.print("Terça ");
    case 4:
     System.out.print("Quarta ");
    default:
     System.out.println("erro: dia deve estar entre [0, 7]");
     break;
    case 5:
     System.out.print("Quinta ");
    case sexta:
     System.out.print("Sexta ");
     break;
    case 7:
     System.out.print("Sábado ");
```

```
    case 0:
    case 1:
     System.out.print("Domingo ");
}
```

Resumindo, Java tem as seguintes declarações condicionais:



Vejamos agora os Loops! Os loops podem executar um bloco de código desde que **uma condição especificada seja alcançada**. Os loops são úteis porque economizam tempo, reduzem erros e tornam o código mais legível. O Loop **While** percorre um bloco de código desde que uma condição especificada seja verdadeira (true):

```
while (condition) {
  // bloco de codigo a ser executado caso a condição seja verdadeira
}
```

NÃO ESQUEÇA DE AUMENTAR A VARIÁVEL USADA NA CONDIÇÃO, CASO CONTRÁRIO O LOOP NUNCA TERMINARÁ!

O loop do/while é uma variante do loop while. Este loop executará o bloco de código uma vez, antes de verificar se a condição é verdadeira, então **repetirá o loop enquanto a condição for verdadeira.**

```
do {
  // code block to be executed
}
while (condition);
```

O exemplo abaixo usa um loop do/while. O loop sempre será executado pelo menos uma vez, mesmo que a condição seja falsa, pois o bloco de código é executado antes que a condição seja testada:

```
int i = 0;
do {
  System.out.println(i);
  i++;
}
while (i < 5);
```

(CESPE / SEED – PR – 2021) Em Java, a estrutura de repetição que permite que um conjunto de instruções não seja executada nenhuma vez é representada por

a) while.
b) switch.
c) do...while.
d) case.
e) continue.

_____

Comentários: Pessoal, o comando while é pré-testado, ou seja, antes de executar qualquer comando, testa-se a condição oferecida, portanto, permite que um conjunto de instruções não seja executada nenhuma vez. (Gabarito: Letra A).

Classes e Objetos

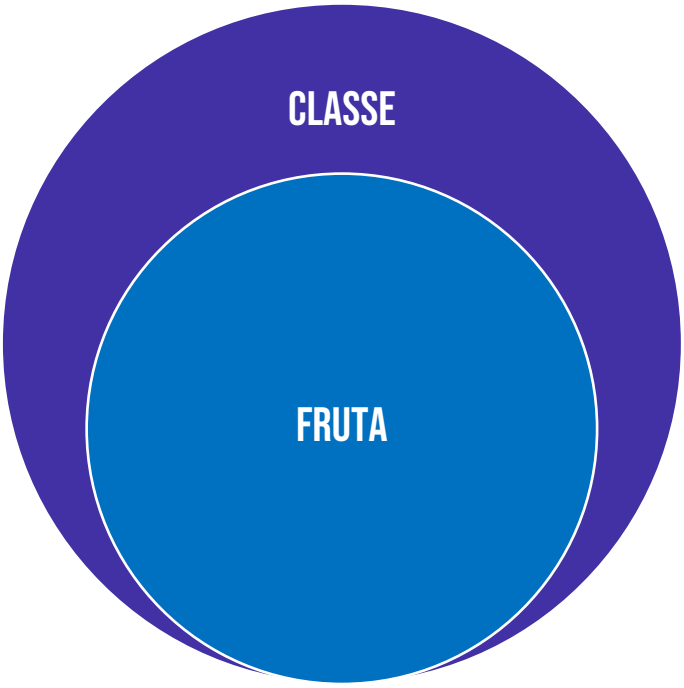
Vamos retomar os princípios da programação procedural e a programação orientada a objetos para entender classes. 😄 A programação procedural consiste em escrever procedimentos ou métodos que executam operações nos dados, enquanto a programação orientada a objetos trata da criação de objetos que contêm dados e métodos.

A programação orientada a objetos tem várias vantagens sobre a programação procedural 😉

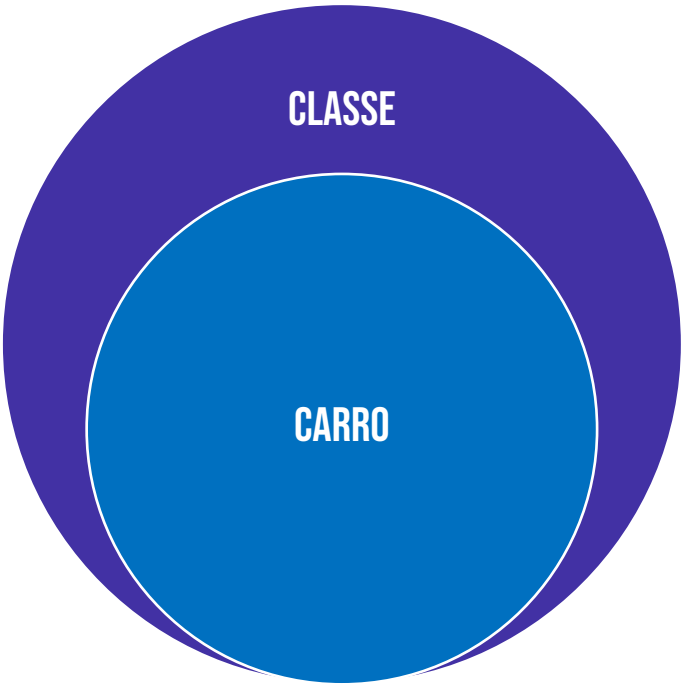
- POO é **mais rápido e fácil de executar**
- POO fornece uma **estrutura clara para os programas**
- POO ajuda a manter o código Java **DRY "Don't Repeat Yourself"** e torna o código **mais fácil de manter, modificar e depurar**
- POO possibilita a criação **de aplicativos reutilizáveis,** completos com menos código e menor tempo de desenvolvimento

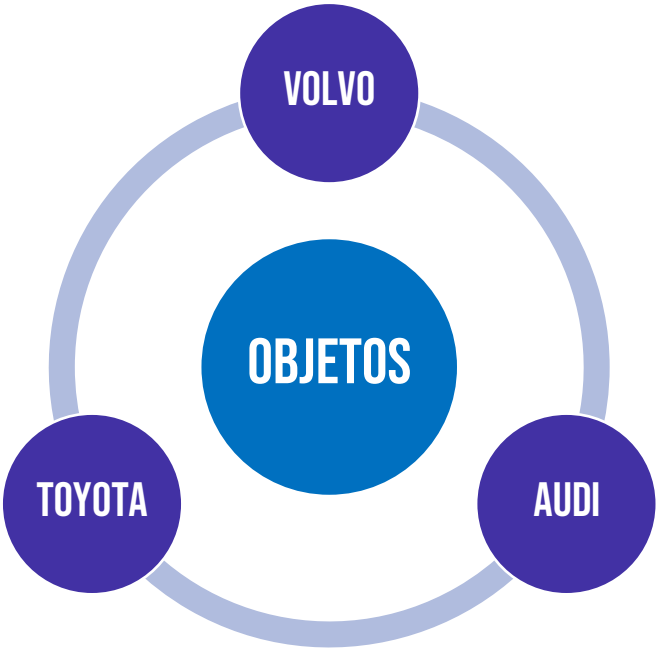
Dica: O princípio "Don't Repeat Yourself" (DRY) consiste em reduzir a repetição de código. Você deve **extrair os códigos que são comuns** para o aplicativo e **colocá-los em um único local** e **reutilizá-los em vez de repeti-los**. 🤩

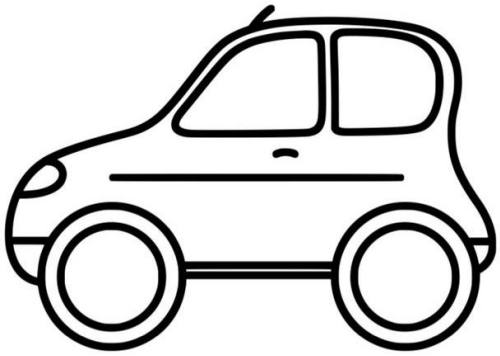
Agora, bora lá, o que são Classes e Objetos? Classes e objetos são os **dois principais aspectos da programação orientada a objetos**. Uma classe nada mais é do que **um projeto de um objeto**. Ao definirmos classes estamos **modelando uma entidade que pode ser criada várias vezes com a mesma definição**. Sendo que ao necessitarmos utilizar essa entidade temos que criar um objeto através do nosso modelo que é a classe.



Tudo em Java está associado a classes e objetos, juntamente com seus atributos e métodos. Por exemplo: na vida real, um carro é um objeto. O carro tem atributos, como peso e cor, e métodos, como tração e freio. **Uma classe é como um construtor de objetos**, ou um "projeto" para criar objetos.







Pessoal, uma classe é uma "coisa" mais generalista, por exemplo, fruta, carro... Já, o objeto consiste na "especificação" daquela "coisa". Como é o caso do carro Toyota, por exemplo. Quando você pensa em uma classe carro, o que vem a cabeça?

Quatro rodas, porta, volante... o que o carro geralmente tem, certo?

Quando pensamos no objeto "Carro Toyota" há uma visão mais elaborada, certo? Eu por exemplo já imagino um **Corolla Cross** 😍. Da mesma forma que a classe Carro possui quatro rodas, uma cor, uma marca, um modelo, o objeto especifica esses atributos. Assim, uma classe é um modelo para objetos e um objeto é uma instância de uma classe. Quando os objetos individuais são criados, eles herdam todas as variáveis e métodos da classe.

Assim como alguém tem de fabricar um carro a partir dos desenhos de engenharia antes que possa realmente dirigi-lo, você deve **construir um**

objeto de uma classe antes que um programa possa executar as tarefas que os métodos da classe definem. O processo para fazer isso é chamado **instanciação**. Um **objeto** é então referido como **uma instância da sua classe**.

Assim como os desenhos de engenharia de um carro podem ser reutilizados várias vezes para fabricar muitos carros, você pode reutilizar uma classe muitas vezes para construir vários objetos. A reutilização de classes existentes ao construir novas classes e programas economiza tempo e esforço. Também ajuda a construir sistemas mais confiáveis e eficientes, porque classes e componentes existentes costumam passar por extensos testes, depuração e ajuste de desempenho. Assim como a noção das partes intercambiáveis foi crucial para a Revolução Industrial, **classes reutilizáveis são fundamentais para a revolução de software que foi estimulada pela tecnologia de objetos**.

Ao dirigir um carro, o ato de pressionar o acelerador envia uma mensagem para o veículo realizar uma tarefa — isto é, ir mais rápido. Da mesma forma, você envia mensagens para um objeto. Cada mensagem é implementada como uma chamada de método que informa a um método do objeto a maneira de realizar sua tarefa. Por exemplo, um programa pode chamar o método depósito de um objeto conta bancária para aumentar o saldo da conta.

E na prática? Como criar uma classe? Para criar uma classe chamada " Main" com uma variável x, devemos seguir a seguinte sintaxe:

```
public class Main {
  int x = 5;
}
```

Lembre-se que uma classe deve sempre começar com uma primeira letra maiúscula e que o nome do arquivo java deve corresponder ao nome da classe. 😊

Para criar um objeto a partir de uma classe, podemos usar a classe main para criar os objetos. Para criar um objeto de Main, especifique o nome da classe, seguido pelo nome do objeto e use a palavra-chave new. No exemplo, é criado um objeto chamado " myObj" e imprime o valor de x:

```
public class Main {
  int x = 5;

  public static void main(String[] args) {
    Main myObj = new Main();
    System.out.println(myObj.x);
  }
}
```

Atributos

Relembrando nosso exemplo, um carro, além de ter a capacidade de realizar tarefas, também tem atributos, como cor, número de portas, quantidade de gasolina no tanque, velocidade atual e registro das milhas totais dirigidas (isto é, a leitura do hodômetro). Assim como suas capacidades, os **atributos do carro** são representados como **parte do seu projeto** nos diagramas de engenharia (que, por exemplo, incluem um hodômetro e um medidor de combustível).

Ao dirigir um carro real, esses atributos são incorporados a ele. Cada carro mantém seus próprios atributos. Cada carro sabe a quantidade de gasolina que há no seu tanque, mas desconhece quanto há no tanque de outros carros. Um objeto, da mesma forma, tem atributos que ele incorpora à medida que é usado em um programa. Esses atributos são especificados como parte da classe do objeto. Por exemplo, um objeto conta bancária tem um atributo saldo que representa a quantidade de dinheiro disponível. Cada objeto conta bancária sabe o saldo que ele representa, mas não os saldos de outras contas bancárias. **Os atributos são especificados pelas variáveis de instância da classe.**

Podemos acessar atributos criando um objeto da classe e usando a sintaxe de ponto (.). O exemplo a seguir criará um objeto da classe Main, com o nome myObj. Usamos o x atributo no objeto para imprimir seu valor:

```
public class Main {
  int x = 5;

  public static void main(String[] args) {
    Main myObj = new Main();
    System.out.println(myObj.x);
  }
}
```

Para modificar o atributo x, podemos realizar a seguinte operação:

```
myObj.x = 40;
```

Ou substitua os valores existentes:

```
myObj.x = 25; // x passa a ser igual a 25
```

Há também a possibilidade de impedir que os atributos sejam modificados! 😳. Para impedir a substituição de valores existentes, declare o atributo como **final**:

```
public class Main {
  final int x = 10;

  public static void main(String[] args) {
    Main myObj = new Main();
```

```
    myObj.x = 25; // will generate an error: cannot assign a value to a final
variable
    System.out.println(myObj.x);
  }
}
```

A palavra-chave **final** é útil quando você deseja que **uma variável armazene sempre o mesmo valor, como PI** (3.14159...). Ela é chamada de "**modificador**".

Vejamos como são criados métodos em classes! No exemplo, será criado um método denominado "myMethod". myMethod()imprime um texto (`Hello World!`), quando é chamado . Para chamar um método, **escreva o nome do método seguido por dois parênteses** () e um ponto e vírgula; (myMethod();)

```
public class Main {
  static void myMethod() {
    System.out.println("Hello World!");
  }
}
```

```
public class Main {
  static void myMethod() {
    System.out.println("Hello World!");
  }

  public static void main(String[] args) {
    myMethod();
  }
}
```

Frequentemente, você verá programas Java que possuem atributos e métodos static ou public. No exemplo acima, criamos um static método, o que significa que ele pode ser acessado sem criar um objeto da classe, diferente de public , que só pode ser acessado por objetos:

A palavra-chave **public** é um modificador de acesso, o que significa que é usado para definir o nível de acesso para classes, atributos, métodos e construtores. Dividimos os modificadores em dois grupos:

Modificadores de acesso: controlam o nível de acesso.
Modificadores de não acesso: não controlam o nível de acesso, mas fornecem outras funcionalidades.

Para classes, você pode usar **public default** .

MODIFICADOR	DESCRIÇÃO
PUBLIC	A classe é acessível por qualquer outra classe
DEFAULT	A classe só é acessível por classes no mesmo pacote. Isso é usado quando você não especifica um modificador.

Para atributos, métodos e construtores, você pode usar um dos seguintes:

MODIFICADOR	DESCRIÇÃO
PUBLIC	A classe, métodos ou atributos assim declarados podem ser acessadas em qualquer lugar e a qualquer momento da execução do programa – é o modificador menos restritivo.
PRIVATE	O código só é acessível dentro da classe declarada. Marca a visibilidade do método ou variável de instância para que apenas a própria classe acesse. Membros da classe definidos como private não podem ser acessados ou usados por nenhuma outra classe. Esse modificador não se aplica às classes, somente para seus métodos e atributos. Esses atributos e métodos também não podem ser visualizados pelas classes herdadas. métodos ou atributos (classes, não) assim declaradas podem ser acessadas apenas dentro da classe que os criou. Subclasses herdam-nos, mas não os acessam – é o modificador mais restritivo.
DEFAULT	O código só é acessível no mesmo pacote. Isso é usado quando você não especifica um modificador.
PROTECTED	O código é acessível no mesmo pacote e subclasses. Marca a visibilidade de um método ou variável de instância para que apenas a própria classe ou suas filhas acessem. O modificador protected torna o membro acessível às classes do mesmo pacote ou através de herança, seus membros herdados não são acessíveis a outras classes fora do pacote em que foram declarados.

Vejamos um compilado de informações: entenda, se uma classe for do tipo Privado, a própria classe pode acessar? Sim! A subclasse? Não! O pacote? Não! E o programa de forma global? Também não! Privado é o modificador mais restritivo! Já o público é o oposto! Todos (classe, sublcasse, pacote e global) conseguem acessar.

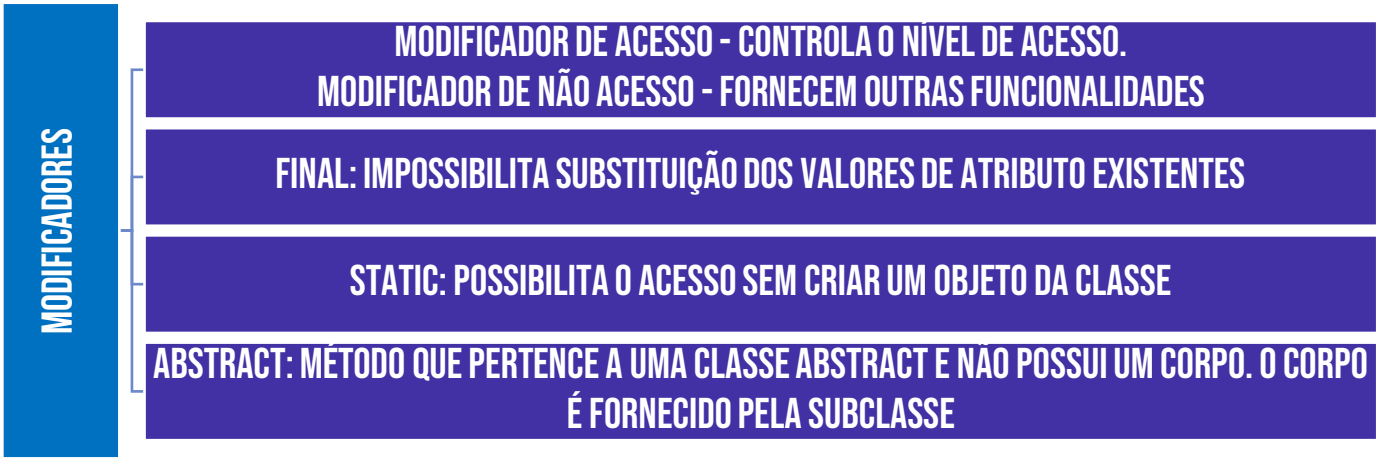
NOME	CLASSE	SUBCLASSE	PACOTE	GLOBAL
PRIVADO	Sim	Não	Não	Não
PROTEGIDO	Sim	Sim	Não	Não
PÚBLICO	Sim	Sim	Sim	Sim
PACOTE	Sim	Não	Sim	Não

Outros modificadores: Para classes, você pode usar final ou abstract:

MODIFICADOR	DESCRIÇÃO
FINAL	A classe não pode ser herdada por outras classes
ABSTRACT	A classe não pode ser usada para criar objetos. Para acessar uma classe abstrata, ela deve ser herdada de outra classe.

Para atributos e métodos, você pode usar um dos seguintes:

MODIFICADOR	DESCRIÇÃO
FINAL	Atributos e métodos finais não podem ser substituídos/modificados
STATIC	Atributos e métodos estáticos pertencem à classe, em vez de um objeto
ABSTRACT	Só pode ser usado em uma classe abstrata e só pode ser usado em métodos. O método não tem um corpo, por exemplo abstract void run();. O corpo é fornecido pela subclasse (herdado da subclasse).
TRANSIENT	Atributos e métodos são ignorados ao serializar o objeto que os contém
SYNCHRONIZED	Métodos synchronized só podem ser acessados por um thread por vez
VOLATILE	O valor de um atributo não é armazenado em cache localmente e é sempre lido na "memória principal"



(FGV / SEFAZ-AM – 2022) A serialização de objetos na linguagem de programação Java permite representar o estado de um objeto como uma sequência de bytes que podem ser armazenados em um arquivo ou banco de dados.

Para impedir que o conteúdo de determinado atributo seja convertido em bytes no processo padrão de serialização, este atributo deve ser declarado na classe com o modificador

a) final.
b) native.
c) volatile.
d) transient.
e) synchronized.

Comentários: Pessoal, olha que questão interessante! Falamos na aula que variáveis de instância não Serializable devem ser declaradas transient para indicar que elas devem ser ignoradas durante o processo de serialização. A questão solicita exatamente o que foi pedido: qual modificador usado para impedir que o conteúdo de determinado atributo seja convertido em bytes no processo padrão de serialização? É o transiente, portanto nosso gabarito é a letra D. Mas vamos ver de que trata as outras alternativas? BORA! A) Final: é usada para impossibilitar que uma classe seja estendida, que um método seja sobrescrito ou que uma variável seja reinicializada. Ou seja, no caso da questão, mantém o atributo constante. B) native é utilizado para dizer que sua implementação é feita em uma outra linguagem específica (por exemplo, C), para que se possa acessar recursos específicos do sistema operacional. Já volatile, indica que uma determinada variável de instância pode ser modificada em duas threads distintas ao mesmo tempo. Por fim, a letra E) synchronized, indica que método com essa marcação será controlado para que não se possa ter duas threads acessando o mesmo objeto. (Gabarito: Letra D).

Métodos

Vamos usar nosso exemplo do carro para introduzir alguns conceitos fundamentais da programação orientada a objetos. Para realizar uma tarefa em um programa é necessário um método.

O método armazena as declarações do programa que, na verdade, executam as tarefas; além disso, ele oculta essas declarações do usuário, assim como o pedal do acelerador de um carro oculta do motorista os mecanismos para fazer o veículo ir mais rápido.

No Java, criamos uma unidade de programa chamada classe para armazenar o conjunto de métodos que executam as tarefas dela. Por exemplo, uma classe que representa uma conta bancária poderia conter um método para fazer depósitos de dinheiro, outro para fazer saques e um terceiro para perguntar qual é o saldo atual.

Uma classe é similar em termos do conceito aos desenhos de engenharia de um carro, que armazenam o projeto de um pedal de acelerador, volante etc.

Construtores

Um construtor em Java é um método especial usado para inicializar objetos. O construtor é chamado quando um objeto de uma classe é criado. Ele pode ser usado para definir valores iniciais para atributos de objeto:

Para criar um construtor:

```
// Create a Main class
public class Main {
  int x;  // Create a class attribute

  // Create a class constructor for the Main class
  public Main() {
    x = 5;  // Set the initial value for the class attribute x
  }

  public static void main(String[] args) {
    Main myObj = new Main(); // Create an object of class Main (This will call the constructor)
    System.out.println(myObj.x); // Print the value of x
  }
}

// Outputs 5
```

Observe que o nome do construtor deve corresponder ao nome da classe e não pode ter um tipo de retorno (como void). Observe também que o construtor é chamado quando o objeto é criado. **Todas as classes têm construtores por padrão**: se você não criar um construtor de classe, o Java cria um para você. No entanto, **você não poderá definir valores iniciais para atributos de objeto**.

Os construtores também podem receber parâmetros, que são usados para inicializar atributos. O exemplo a seguir adiciona um int yparâmetro ao construtor. Dentro do construtor definimos x para y (x=y). Quando chamamos o construtor, passamos um parâmetro para o construtor (5), que definirá o valor de x para 5:

EM JAVA APENAS AS INTERFACES NÃO POSSUEM CONSTRUTORES.

Toda classe tem pelo menos um construtor sempre definido. Se nenhum construtor for explicitamente definido pelo programador da classe, um construtor default, que não recebe argumentos, é criado pelo compilador Java. No entanto, se o programador da classe criar pelo menos um método construtor, o construtor default não será criado automaticamente -- se ele o desejar, deverá criar um construtor sem argumentos explicitamente.

No momento em que um construtor é invocado, a seguinte sequência de ações é executada para a criação de um objeto:

- O espaço para o objeto é alocado e seu conteúdo é inicializado (bitwise) com zeros.
- O construtor da classe base é invocado.
- Os membros da classe são inicializados para o objeto, seguindo a ordem em que foram declarados na classe.
- O restante do corpo do construtor é executado.

Seguir essa sequência é uma necessidade de forma a garantir que, quando o corpo de um construtor esteja sendo executado, o objeto já terá à disposição as funcionalidades mínimas necessárias, quais sejam aquelas definidas por seus ancestrais. O primeiro passo garante que nenhum campo do objeto terá um valor arbitrário, que possa tornar erros de não inicialização difíceis de detectar.

(FGV – DPE-RJ – 2019) Considere as seguintes afirmativas sobre class constructor na linguagem Java.

I. Deve receber o mesmo nome da classe a ele associada.

II. Não deve ser especificado um tipo de retorno na sua declaração.

III. É útil para a definição de valores iniciais para os atributos da classe.

IV. É sempre declarado como public.

É correto somente o que se afirma em:
a) I e II;
b) II e III;
c) III e IV;
d) I, II e III;
e) I, III e IV.

_____
Comentários: Vamos lá, um construtor é um método especial usado para inicializar objetos. O construtor é chamado quando um objeto de uma classe é criado. Ele pode ser usado para definir valores iniciais para atributos de objeto. o construtor é apenas invocado no momento da criação do objeto através do operador new. A assinatura de um construtor diferencia-se das assinaturas dos outros métodos por não ter nenhum tipo de retorno (nem mesmo void). Além disto, o nome do construtor deve ser o próprio nome da classe. O construtor pode receber argumentos, como qualquer método. Usando o mecanismo de sobrecarga, mais de um construtor pode ser definido para uma classe. Assim, temos que os itens I, II e III estão corretos. Já, a assertiva IV está errada, pois pode ser public, protected ou private. (Gabarito: Letra D).

(FGV / MPE-AL – 2018) Sobre as variáveis e os métodos declarados como private, em Java, analise as afirmativas a seguir.

I. Ficam acessíveis somente aos membros da própria classe.

II. Ficam acessíveis somente às classes definidas no mesmo package.

III. Ficam acessíveis somente para suas classes derivadas.

Está correto o que se afirma em

a) I, apenas.
b) II, apenas.
c) III, apenas.
d) I e II, apenas.
e) II e III, apenas.

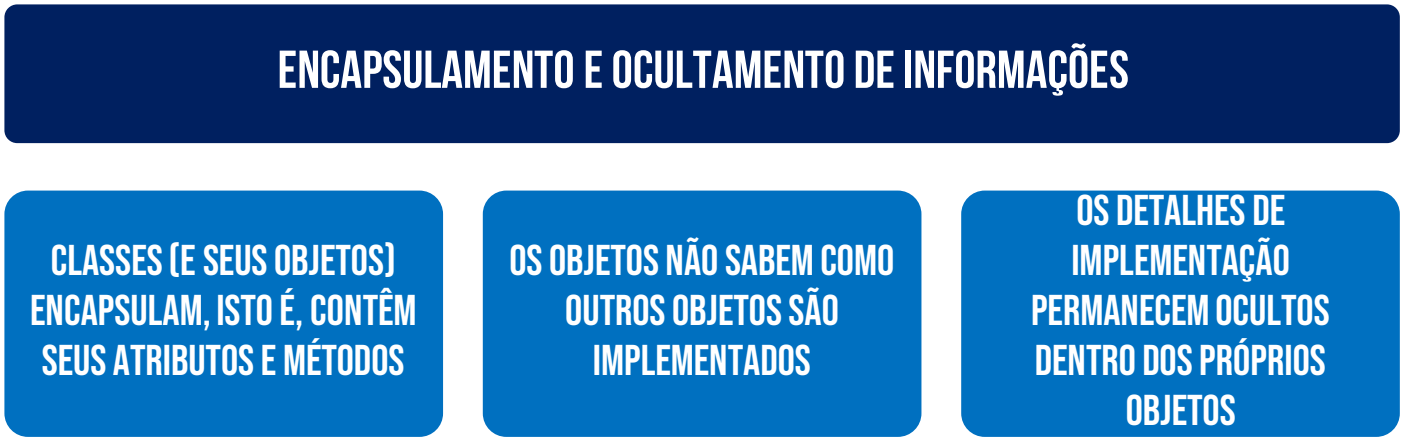
_____

Comentários: Pessoal, vamos relembrar a tabela? As variáveis e os métodos private ficam acessíveis apenas aos membros da própria classe! Não ficam disponíveis no pacote e também não ficam acessíveis ás subclasses! Portanto nosso gabarito é apenas o item I. (Gabarito: Letra A).

Herança e Encapsulamento

Encapsulamento e ocultamento de informações

Classes (e seus objetos) encapsulam, isto é, contêm seus atributos e métodos. Os atributos e métodos de uma classe (e de seu objeto) estão **intimamente relacionados**. Os objetos podem se comunicar entre si, mas eles em geral **não sabem como outros objetos são implementados** — **os detalhes de implementação permanecem ocultos dentro dos próprios objetos.** Esse ocultamento de informações, como veremos, é crucial à boa engenharia de software.



Herança

Uma nova classe de objetos pode ser criada convenientemente por meio de herança — ela (chamada subclasse) começa com as características de uma classe existente (chamada superclasse), possivelmente personalizando-as e adicionando aspectos próprios.

Na nossa analogia do carro, um objeto da classe "conversível" decerto é um objeto da classe mais geral "automóvel", mas, especificamente, o teto pode ser levantado ou baixado.

Em java a herança é realizada de maneira simples ao utilizar a palavra-chave **extends**:

```
package academico;

public abstract class Pessoa {
    public String strNome;
    public String strTelefone;
// Métodos
    public void getStrNome(String Nome) {
        this.StrNome = Nome;
    }
    public String setStrNome() {
        return StrNome;
```

```
    }
}

public class Aluno extends Pessoa {
    // strNome e strTelefone, bem como seus metodos são herdados nesta calasse por
meio da palavra "extends"

}
```

Quando uma classe é criada como sub-classe de outra classe, a palavra-chave super é usada para que a sub-classe possa acessar métodos public ou protected (mas não private) da superclasse. super também é usado para invocar o constructor da superclasse, durante o constructor da subclasse.

```
public class SuperClass
{
   SuperClass(String title)
   {
      System.out.println( "Super: " + title );
   }

   public void printHello()
   {
      System.out.println( "Hello from SuperClass" );
     return;
   }
}

public class SubClass extends SuperClass
{
   SubClass(String title)// constructor
   {
       super(title);  // chama o constructor de Frame
   }
   public void printHello()
   {
      super.printHello();
      System.out.println( "Hello from SubClass" );
     return;
   }

   public static main( String[] args )
   {
      SubClass obj = new SubClass();
      obj.printHello();
   }
}
```

(FGV / SEFAZ-AM – 2022) Na programação orientada a objetos, a herança é uma técnica de abstração que permite categorizar as classes de objetos sob certos critérios, especificando-se as características dessas classes. As classes que são vinculadas por meio de relacionamentos de herança formam uma hierarquia de herança. Na linguagem de programação Java, o relacionamento de herança é definido pela palavra-chave

a) static.
b) extends.
c) public.
d) new.
e) this.

_____

Comentários: Pessoal, a palavra-chave da herança é extends! A palavra-chave extends faz com que uma subclasse herde (receba) todos os atributos e métodos declarados na classe-pai. (Gabarito: Letra B).

ESPECIFICADOR	CLASSE	SUBCLASSE	PACOTE	GLOBAL
PRIVADO	Sim	Não	Não	Não
PROTEGIDO	Sim	Sim	Não	Não
PÚBLICO	Sim	Sim	Sim	Sim
PACOTE	Sim	Não	Sim	Não

(FGV – TJDFT – 2022) Observe as classes abaixo descritas na linguagem de programação Java.

public class DocumentoProcesso extends Object

```
{
   private String docNumero;
   private int classificacao;


   public DocumentoProcesso (String docNumero, int

classificacao){
    if (classificacao < 1)
      throw new IllegalArgumentException (
        "A classificação do documento deve ser no mínimo 1.");

     this.docNumero = docNumero;
     this.classificacao = classificacao;
    }
    public String getDocNumero()
    { return docNumero; }
    public int getClassificacao()
    { return classificacao; }
    public void setDocNumero(String docNumero)
    { this.docNumero = docNumero; }
    public void setClassificacao(int classificacao)
    { this.classificacao = classificacao; }
    public int promove()
    { return classificacao + 1; }
}

public class Oficio extends DocumentoProcesso
{

   private int precedencia;
   public Oficio (String docNumero, int classificacao, int precedencia)
   {super(docNumero,classificacao);
   this.precedencia = precedencia; }
public int getPrecedencia()
{ return precedencia; }
public void setPrecedencia(int precedencia)
{ this.precedencia = precedencia; }
@Override
public int promove()
{ return precedencia + 1; }
}
```

Com base nos conceitos de orientação a objetos, é correto afirmar que:
a) os atributos private de DocumentoProcesso são acessíveis por Oficio;
b) a anotação @Override indica que o método promove() é abstrato e polimórfico;
c) a classe Oficio representa uma herança múltipla das classes DocumentoProcesso e Object;
d) a classe Oficio é composta pela classe DocumentoProcesso, permitindo que uma instância da classe Oficio faça referências a instâncias da classe DocumentoProcesso;
e) os métodos getDocNumero() e setDocNumero() da classe DocumentoProcesso encapsulam o atributo docNumero e asseguram que os objetos da classe mantenham estados consistentes.

Comentários: temos o nosso gabarito na letra E. Pessoal, os métodos getDocNumero() e setDocNumero() de fato, são utilizados para encapsular o atributo docNumero. Ademais, asseguram que os objetos da classe mantenham estados consistentes. Getters e setters são usados para proteger seus dados, especialmente na criação de classes. Para cada instância de variável, um método getter retorna seu valor, enquanto um método setter o define ou atualiza seu valor. Nas questões comentada comento todas as demais assertivas! (Gabarito: Letra E).

Interfaces

O Java também **suporta interfaces** — **coleções de métodos relacionados que normalmente permitem informar aos objetos o que fazer**, **mas não como fazer**. Na analogia do carro, uma interface das capacidades "básicas de dirigir" consistindo em um volante, um pedal de acelerador e um pedal de freio permitiria que um motorista informasse ao carro o que fazer. Depois que você sabe como usar essa interface para virar, acelerar e frear, você pode dirigir muitos tipos de carro, embora os fabricantes possam implementar esses sistemas de forma diferente.

Uma classe implementa zero ou mais interfaces — cada uma das quais pode ter um ou mais métodos —, assim como um carro implementa interfaces separadas para as funções básicas de dirigir, controlar o rádio, controlar os sistemas de aquecimento, ar-condicionado e afins. Da mesma forma que os fabricantes de automóveis implementam os recursos de forma distinta, classes podem implementar métodos de uma interface de maneira diferente. Por exemplo, um sistema de software pode incluir uma interface de "backup" que ofereça os métodos save e restore. As classes podem implementar esses métodos de modo diferente, dependendo dos tipos de formato em que é feito o backup, como programas, textos, áudios, vídeos etc., além dos tipos de dispositivo em que esses itens serão armazenados.

INTERFACES

TODOS OS MÉTODOS DA INTERFACE SÃO IMPLICITAMENTE MÉTODOS PUBLIC ABSTRACT

TODOS OS CAMPOS SÃO IMPLICITAMENTE PUBLIC, STATIC E FINAL.

(FCC / SABESP – 2018) As interfaces são usadas nas aplicações Java quando se deseja permitir que diversas classes implementem determinados métodos, mesmo que de formas diferentes. Em uma interface Java

a) os métodos não podem ter os modificadores protected ou private.
b) não pode haver assinaturas de métodos cujo tipo de retorno seja void.
c) pode haver múltiplos construtores, desde que recebam parâmetros diferentes.
d) não pode haver dois ou mais métodos com o mesmo nome, mesmo que recebam parâmetros diferentes.
e) todo método deverá ser implementado por uma das subclasses da aplicação pelo menos uma vez.

_____

Comentários: A alternativa 'a' diz que os métodos não podem ter os modificadores protected ou private, o que está correto! Vimos em aula que em geral, as interfaces são compostas basicamente de um conjunto de assinaturas de métodos públicos e abstratos. Ou seja, os métodos das interfaces são públicos! B: Não há nenhuma restrição relacionado a assinatura de métodos com tipo de retorno void! Errada letra B! A restrição da letra C também está incorreta! Não há nenhuma restrição relacionada a quantidade de métodos, sequer métodos com o mesmo nome! Se houver, estaremos diante de um caso de sobrecarrega, ou polimorfismo estático. Por fim, a letra E também está errada! Pode sim haver método sem a implementação na subclasse.Não existe obrigação de implementação de uma interface. (Gabarito: Letra A).

Polimorfismo

O polimorfismo permite escrever programas que processam objetos que **compartilham a mesma superclasse**, direta ou indiretamente, como se todos fossem objetos da superclasse; isso pode simplificar a programação.

Considere o exemplo de polimorfismo a seguir. Suponha que criamos um programa que simula o movimento de vários tipos de animais para um estudo biológico. As classes Peixe, Anfíbio e Pássaro representam os três tipos de animais sob investigação.

Imagine que cada classe estende a superclasse Animal, que contém um método **mover** e mantém a localização atual de um animal como coordenadas x-y. **Cada subclasse implementa o método mover.** Nosso programa mantém um array Animal que contém referências a objetos das várias subclasses Animal. Para simular os movimentos dos animais, o programa envia a mesma mensagem a cada objeto uma vez por segundo — a saber, mover.

Cada tipo específico de Animal responde a uma mensagem mover de uma maneira única — um Peixe poderia nadar um metro, um Anfíbio poderia pular um metro e meio e um Pássaro poderia voar três metros. Cada objeto sabe como modificar suas coordenadas x-y de forma adequada para seu tipo específico de movimento. Contar com o fato de que cada objeto sabe "fazer a coisa certa" (isto é, faz o que é apropriado a esse tipo de objeto) em resposta à mesma chamada de método é o conceito-chave do polimorfismo.

A mesma mensagem (nesse caso, mover) enviada a uma variedade de objetos tem muitas formas de resultados — daí o termo polimorfismo.

Com o polimorfismo, podemos **projetar e implementar** sistemas que são **facilmente extensíveis** — novas classes podem ser adicionadas com pouca ou nenhuma modificação a partes gerais do programa, contanto que as novas classes façam parte da hierarquia de herança que o programa processa genericamente.

As novas classes simplesmente se "encaixam". As únicas partes de um programa que devem ser alteradas são aquelas que exigem conhecimento direto das novas classes que adicionamos à hierarquia. Por exemplo, se estendermos a classe Animal para criar a classe Tartaruga (que poderia responder a uma mensagem mover deslizando uma polegada), precisaremos escrever somente a classe Tartaruga e a parte da simulação que instancia um objeto Tartaruga. As partes da simulação que dizem para que cada Animal se mova genericamente podem permanecer as mesmas.

Vejamos um exemplo clássico de polimorfismo para fixação do conhecimento. Se a classe Retângulo é derivada da classe Quadrilátero, então **um objeto Retângulo é uma versão mais específica de um objeto Quadrilátero**. Qualquer operação (por exemplo, calcular o perímetro ou a área) que pode ser realizada em um objeto Quadrilátero também pode ser realizada em um objeto Retângulo. Essas operações podem ser realizadas em outros Quadriláteros, como Quadrados, Paralelogramos e Trapezoides. O polimorfismo ocorre quando um programa invoca um método por meio de uma variável de superclasse Quadrilátero — em tempo de execução, a versão correta da subclasse do método é chamada com base no tipo de referência armazenado na variável da superclasse.

POLIMORFISMO

A MESMA MENSAGEM ENVIADA A UMA VARIEDADE DE OBJETOS TEM MUITAS FORMAS DE RESULTADOS.

(CESPE / SERPRO – 2021) O polimorfismo ocorre quando a mesma operação é construída em uma mesma classe ou quando o método da subclasse sobrepõe-se ao método da superclasse.

_____

Comentários: Pessoal, o polimorfismo ocorre quando um programa invoca um método por meio de uma variável de superclasse em tempo de execução, a versão correta da subclasse do método é chamada com base no tipo de referência armazenado na variável da superclasse. Portanto, trata-se da capacidade de um objeto poder se comportar de diversas formas dependendo da mensagem recebida. (Gabarito: Correto).

Dois conceitos muito importantes em Java são: sobrecarga e sobrescrita. Com a sobrecarga (overload) de métodos, **vários métodos podem ter o mesmo nome com parâmetros diferentes**. Em vez de definir dois métodos que devem fazer a mesma coisa, é melhor sobrecarregar um. Sobrecarga ou Polimorfismo Estático ocorre quando **uma classe possui métodos com mesmo nome, entretanto assinaturas diferentes**. Ocorre em Tempo de Compilação.

A **sobrescrita** (ou override) está diretamente relacionada à orientação a objetos, mais especificamente com a herança. Com a sobrescrita, conseguimos **especializar os métodos herdados** das superclasses, alterando o seu comportamento nas subclasses por um mais específico. A sobrescrita de métodos consiste basicamente em e mesmo tipo de retorno do método sobrescrito.

Os métodos com o mesmo nome podem ser declarados na mesma classe, contanto que tenham diferentes conjuntos de parâmetros (determinados pelo número, tipos e ordem dos parâmetros) — isso é chamado de **sobrecarga de métodos**. Quando um método sobrecarregado é chamado, o compilador Java seleciona o método adequado examinando o número, os tipos e a ordem dos argumentos na chamada. A **sobrecarga** de métodos é comumente utilizada para **criar vários**

métodos com o mesmo nome que realizam as mesmas tarefas, ou tarefas semelhantes, mas sobre **tipos diferentes ou números diferentes de argumentos**.

O compilador **distingue os métodos sobrecarregados pelas suas assinaturas** — uma combinação do nome e o número do método, tipos e ordens de seus parâmetros, mas não de seu tipo de retorno. Se o compilador examinasse somente os nomes do método durante a compilação, o código do programas seria ambíguo. Internamente, o compilador utiliza nomes de método mais longos, que incluem o nome original do método, os tipos de cada parâmetro e a ordem exata dos parâmetros para determinar se os métodos em uma classe são únicos nela.

Em resumo, o Java permite métodos sobrecarregados em uma classe, desde que os métodos tenham diferentes conjuntos de parâmetros (determinados pelo número, ordem e tipo de parâmetros). Métodos sobrecarregados são **distinguidos por suas assinaturas** — combinações dos nomes e número, tipos e ordem dos parâmetros dos métodos, mas não pelos tipos de retorno.

Tratamento de Exceções

Ao executar o código Java, diferentes erros podem ocorrer: erros de codificação feitos pelo programador, erros devido a entrada errada ou outros imprevistos. Quando ocorre um erro, o Java normalmente para e gera uma mensagem de erro. O termo técnico para isso é: Java lançará uma exceção (jogará um erro).

Uma **exceção** é uma indicação de um problema que ocorre durante a execução de um programa. O **tratamento de exceção** permite criar aplicativos que podem **resolver (ou tratar) exceções**. Em muitos casos, o tratamento de uma exceção permite que um programa continue executando como se nenhum problema tivesse sido encontrado.

Somente as classes que estendem **Throwable** (pacote java.lang) direta ou indiretamente podem ser utilizadas com **o tratamento de exceção**. Exceções encadeadas podem ser usadas ao chamar um método que indica uma exceção, você pode lançar outra exceção e encadear o original com o novo. Isso permite adicionar informações específicas do aplicativo à exceção original.

Ademais, há pré-condições e pós-condições, que devem ser verdadeiras quando seus métodos são chamados e quando eles retornam, respectivamente.

A instrução **try** permite que você **defina um bloco de código para ser testado** quanto a erros enquanto está sendo executado. A instrução **catch** permite definir um **bloco de código a ser executado, caso ocorra um erro no bloco try**. As palavras-chave try e catch vêm em pares:

```
try {
  //  bloco de código para ser testado
}
catch(Exception e) {
  //  bloco de código a ser executado, caso ocorra um erro no bloco try
}
```

Sincronismo e Multithreading

Seria interessante se pudéssemos concentrar nossa atenção na realização de uma única tarefa de cada vez e fazer isso bem, o que geralmente é difícil de alcançar em um mundo complexo em que muitas coisas acontecem ao mesmo tempo. Principalmente atualmente, com a tecnologia avançando rapidamente, são inúmeros processos executando de forma concorrente.

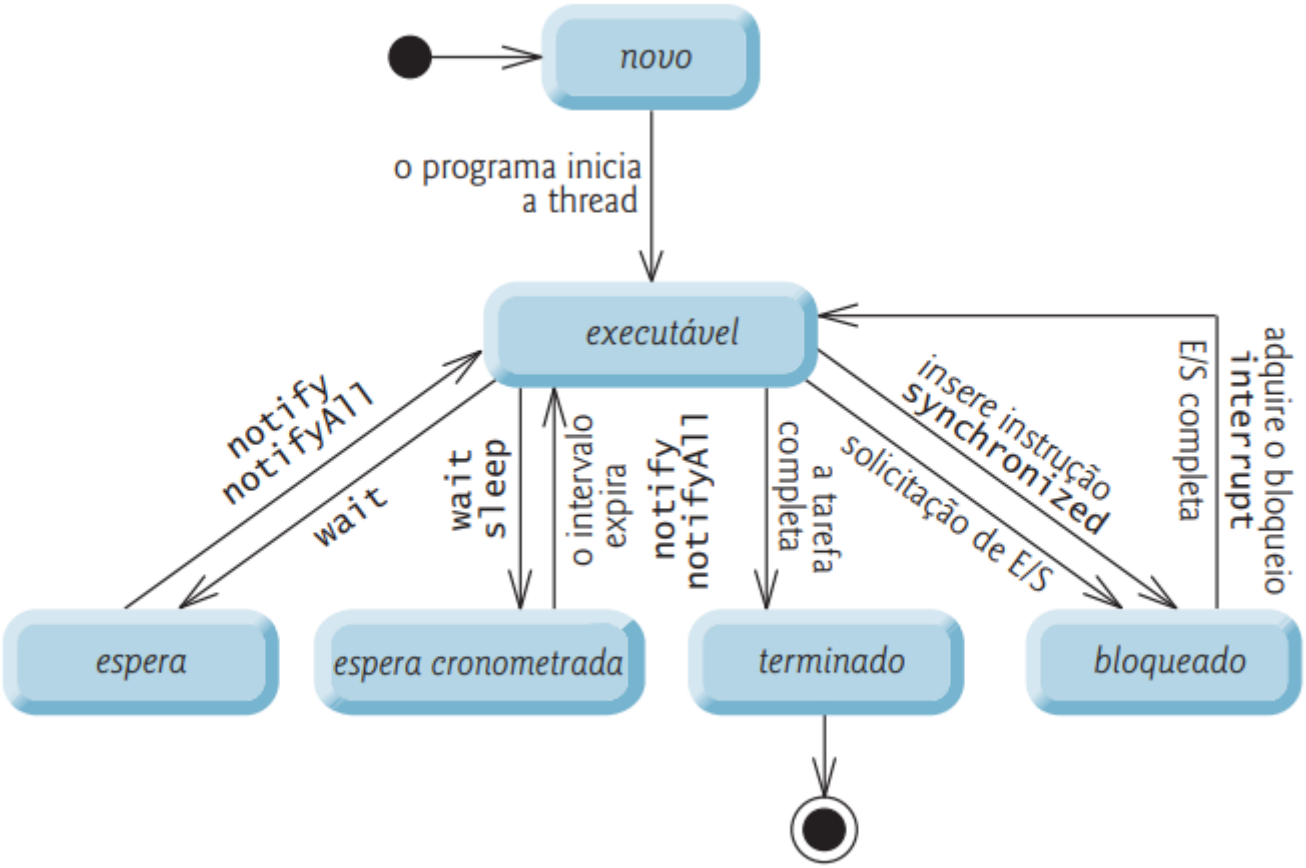
Quando dizemos que duas tarefas operam concorrentemente, queremos dizer que ambas **progridem ao mesmo tempo**. Até recentemente, a maioria dos computadores tinha apenas **um único processador**. Sistemas operacionais nesses computadores executam tarefas de forma concorrente alternando rapidamente entre elas, fazendo uma pequena parte de cada uma antes de passar para a próxima, de modo que todas as tarefas continuem progredindo. Por exemplo, é

comum que computadores pessoais compilem um programa, enviem um arquivo para uma impressora, recebam mensagens de correio eletrônico por uma rede e muito mais, concorrentemente. Desde sua criação, o Java suporta a concorrência.

Quando dizemos que duas tarefas operam em paralelo, queremos dizer que elas são executadas simultaneamente. Nesse sentido, o paralelismo é um subconjunto da concorrência. O corpo humano realiza uma grande variedade de operações paralelas.

O Java disponibiliza a concorrência por meio da **linguagem e APIs**. Programas Java podem ter **várias threads** de execução, em **que cada thread tem sua própria pilha de chamadas de método** e seu próprio contador de programa, permitindo que seja **executada concorrentemente com outras threads** enquanto compartilha os recursos de nível de aplicativo como memória e handles de arquivo. Essa capacidade é chamada **multithreading**

A qualquer momento, diz-se que uma thread está em um de vários estados de thread

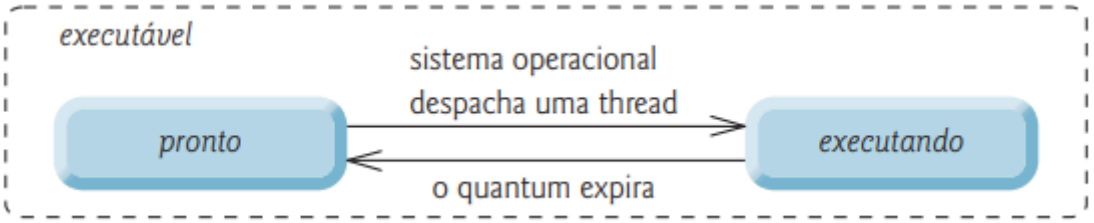


ESTADO	DESCRIÇÃO
EXECUTÁVEL	Uma nova thread inicia seu ciclo de vida no estado novo. Ela permanece nesse estado até que o programa inicie a thread, o que a coloca no estado executável. Considera-se que uma thread no estado executável está executando sua tarefa.

ESPERA	Às vezes a thread executável transita para o estado de espera enquanto aguarda outra thread realizar uma tarefa. Uma thread em espera volta ao estado executável somente quando outra thread a notifica para continuar a execução.
ESPERA SINCRONIZADA	Uma thread executável pode entrar no estado de espera sincronizada por um intervalo especificado de tempo. Ela faz a transição de volta ao estado executável quando esse intervalo de tempo expira ou o evento que ela espera ocorre. Threads de espera sincronizada não podem utilizar um processador, mesmo se houver um disponível. Uma thread executável pode transitar para o estado de espera sincronizada se fornecer um intervalo de espera opcional quando ela estiver esperando outra thread realizar uma tarefa. Essa thread retornará ao estado executável quando ela for notificada por outra thread ou quando o intervalo sincronizado expirar — o que ocorrer primeiro. Outra maneira de posicionar uma thread no estado em espera sincronizada é colocar uma thread executável para dormir — uma thread adormecida permanece no estado de espera sincronizada por um determinado período de tempo (chamado intervalo de adormecimento), depois do qual ela retorna ao estado executável. As threads dormem quando, por um breve período, não têm de realizar nenhuma tarefa. Por exemplo, um processador de texto pode conter uma thread que grave periodicamente backups (isto é, grava uma cópia) do documento atual no disco para fins de recuperação. Se a thread não dormisse entre os sucessivos backups, seria necessário um loop em que testaria continuamente se deve ou não gravar uma cópia do documento em disco. Esse loop consumiria tempo de processador sem realizar trabalho produtivo, reduzindo assim o desempenho do sistema. Nesse caso, é mais eficiente para a thread especificar um intervalo de adormecimento (igual ao período entre backups sucessivos) e entrar no estado de espera sincronizada. Essa thread é retornada ao estado executável quando seu intervalo de adormecimento expira, ponto em que ela grava uma cópia do documento no disco e entra novamente no estado de espera sincronizada.
BLOQUEADO	Uma thread executável passa para o estado bloqueado quando tenta realizar uma tarefa que não pode ser concluída imediatamente e deve esperar temporariamente até que a tarefa seja concluída. Por exemplo, quando uma thread emite uma solicitação de entrada/saída, o sistema operacional só permite que ela seja executada depois que a solicitação de E/S estiver concluída — nesse ponto, a thread bloqueada faz uma transição para o estado executável, assim pode continuar a execução. Uma thread bloqueada não pode utilizar um processador, mesmo se algum estiver disponível.
TERMINADO	Uma thread executável entra no estado terminado (às vezes chamado estado morto) quando ela conclui com sucesso suas tarefas ou é de outro modo terminada (talvez em razão de um erro).

No nível do sistema operacional, o estado executável do Java geralmente inclui dois estados separados. O sistema operacional oculta esses estados da Java Virtual Machine (JVM), que vê apenas o estado executável. Quando uma thread entra pela primeira vez no estado executável a

partir do estado novo, ela está no estado pronto. Uma thread pronta entra no estado de execução (isto é, começa a executar) quando o sistema operacional a atribui a um processador — também conhecido como despachar a thread. Na maioria dos sistemas operacionais, cada thread recebe uma pequena quantidade de tempo de processador — chamada de quantum ou fração de tempo — com a qual realiza sua tarefa. Decidir qual deve ser o tamanho máximo do quantum é um tema-chave nos cursos sobre sistemas operacionais. Quando o quantum expira, a thread retorna ao estado pronto, e o sistema operacional atribui outra thread ao processador. As transições entre os estados pronto e em execução são tratadas exclusivamente pelo sistema operacional. A JVM não "vê" as transições — ela simplesmente vê a thread como executável e deixa para o sistema operacional fazer a transição entre a thread pronta e em execução. O processo que um sistema operacional usa para determinar qual thread despachar é chamado agendamento de thread e depende das prioridades de thread.



Coleções

A Java API fornece várias estruturas de dados predefinidas, chamadas coleções, usadas para **armazenar grupos de objetos relacionados na memória**. Essas classes fornecem métodos eficientes que **organizam, armazenam e recuperam seus dados** sem a necessidade de conhecer como os dados são armazenados.

Isso reduz o tempo de desenvolvimento de aplicativos. Você já usou arrays para armazenar sequências de objetos. Arrays não mudam automaticamente o tamanho em tempo de execução para acomodar elementos adicionais. A classe de coleção ArrayList<T> (pacote java.util) fornece uma solução conveniente para esse problema — ela pode alterar dinamicamente seu tamanho para acomodar mais elementos. O T (por convenção) é um espaço reservado — ao declarar um novo ArrayList, substitua-o pelo tipo dos elementos que você deseja que o ArrayList armazene. Por exemplo,

```
ArrayList<String> list;
```

declara **list** como uma **coleção ArrayList** que só pode armazenar **Strings**. Classes com esse tipo de espaço reservado que podem ser usadas com qualquer tipo são chamadas classes genéricas. Somente tipos não primitivos podem ser usados para declarar variáveis e criar objetos das classes genéricas. Mas o Java fornece um mecanismo conhecido como boxing, que permite que valores primitivos sejam empacotados como objetos para uso com classes genéricas. Assim, por exemplo,

```
ArrayList<Integer> integers;
```

declara **integers** como um **ArrayList** que só pode armazenar **Integers**. Ao inserir um valor int em um ArrayList<Integer>, o valor int é empacotado como um objeto Integer, e quando você obtém um objeto Integer de um ArrayList<Integer>, e então atribui o objeto a uma variável int, o valor int dentro do objeto é desempacotado.

A classe array é um ArrayList **redimensionável**, que pode ser encontrado no pacote.java.util. A diferença entre um array embutido e um ArrayList em Java, é que **o tamanho de um array não pode ser modificado** (se você quiser adicionar ou remover elementos de/para um array, você tem que criar um novo). **Enquanto os elementos podem ser adicionados e removidos de um ArrayList sempre que você quiser.** Vejamos alguns métodos e propriedades da classe ArrayList<T>.

MÉTODO	DESCRIÇÃO
ADD	Adiciona um elemento ao final do ArrayList.
CLEAR	Remove todos os elementos do ArrayList.
CONTAINS	Retorna true se o ArrayList contém o elemento especificado; caso contrário, retorna false.
GET	Retorna o elemento no índice especificado.
INDEXOF	Retorna o índice da primeira ocorrência do elemento especificado no ArrayList.
REMOVE	Sobrecarregado. Remove a primeira ocorrência do valor especificado ou o elemento no índice especificado.
SIZE	Retorna o número de elementos armazenados em ArrayList.
TRIMTOSIZE	Corta a capacidade do ArrayList para o número atual de elementos.

A classe **LinkedList** é quase idêntica à **ArrayList**. A classe **LinkedList** é uma coleção que pode conter muitos objetos do mesmo tipo, assim como o **ArrayList**. A classe **LinkedList** tem todos os mesmos métodos que a classe **ArrayList** porque ambos implementam a interface **List**. Isso significa que você pode **adicionar itens, alterar itens, remover itens e limpar a lista da mesma maneira**.

No entanto, embora a classe **ArrayList** e a classe **LinkedList** possam ser usadas da mesma maneira, elas são construídas de maneira muito diferente.

A classe **ArrayList** tem um array regular dentro dela. Quando um elemento é adicionado, ele é colocado no array. Se a matriz não for grande o suficiente, uma nova matriz maior é criada para substituir a antiga e a antiga é removida.

A **LinkedList** armazena seus itens em "contêineres". A lista tem um link para o primeiro contêiner e cada contêiner tem um link para o próximo contêiner na lista. Para adicionar um elemento à lista,

o elemento é colocado em um novo contêiner e esse contêiner é vinculado a um dos outros contêineres da lista.

MÉTODOS LINKEDLIST	DESCRIÇÃO
ADDFIRST	Adiciona um item ao início da lista.
ADDLAST	Adiciona um item ao final da lista.
REMOVEFIRST	Remove um item do início da lista.
REMOVELAST	Remove um item do final da lista.
GETFIRST	Obtém o item no início da lista.
GETLAST	Obtém o item no final da lista.

(FGV/ IBGE – 2017) Um programador Java precisa utilizar um array unidimensional dinâmico armazenando números inteiros e decide importar a classe java.util.ArrayList. A declaração da referência ao array que ele deverá utilizar é:

a) ArrayList<int> arr;
b) ArrayList<int>[] arr;
c) ArrayList<int> arr[];
d) ArrayList<Integer> arr[];
e) ArrayList<Integer> arr.

_____

Comentários: Pessoal, quando criamos o ArrayList carro – no exemplo – utilizamos a seguinte sintaxe: ArrayList<String> carro. Da mesma forma, para criar um ArrayList do tipo inteiro, teríamos que usar: ArrayList<Integer> arr. Portanto, nosso gabarito é a Letra E (Gabarito: Letra E).

Um Set é uma **Collection** não ordenada de elementos únicos (isto, sem duplicatas). A estrutura de coleções contém diversas implementações de Set, incluindo **HashSet** e **TreeSet**. **HashSet** armazena seus elementos em uma tabela de hash; e **TreeSet** armazena seus elementos em uma árvore.

A estrutura de coleções também inclui a interface SortedSet (que estende Set) para conjuntos que mantêm seus elementos em ordem classificada — a ordem natural dos elementos (por exemplo, números estão em ordem crescente) ou uma ordem especificada por um Comparator. A classe TreeSet implementa **SortedSet**

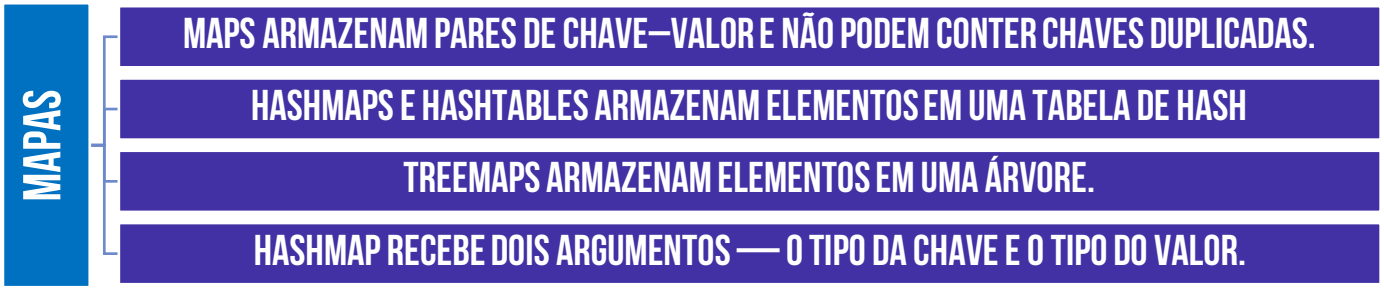
Vamos conhecer os mapas! Mapa ou **Map** associam chaves a valores. As chaves em um Map devem ser únicas, mas os valores associados não precisam ser. Se um Map contém chaves únicas e valores únicos, diz-se que implementa um mapeamento de um para um. Se somente as chaves são únicas, diz-se que o Map implementa um mapeamento de muitos para um — muitas chaves podem mapear para um valor.

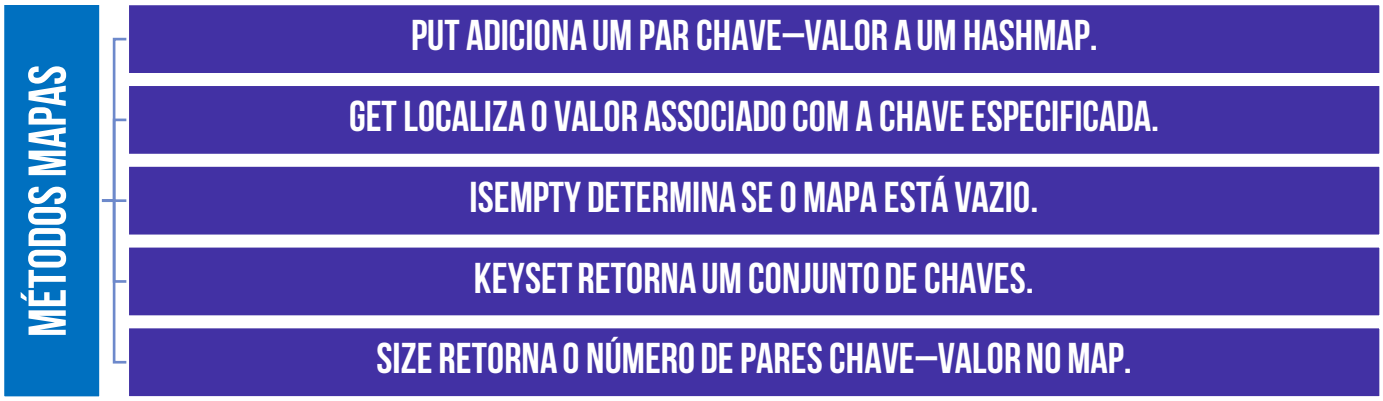
Maps diferem de Sets pelo fato de que Maps contêm chaves e valores, enquanto Sets contêm somente valores. Três das várias classes que implementam a interface Map são Hashtable, HashMap e TreeMap. Hashtables e HashMaps armazenam elementos em tabelas de hash e TreeMaps armazenam elementos em árvores. Esta seção discute as tabelas de hash e fornece um exemplo que utiliza um HashMap para armazenar pares chave–valor. A interface SortedMap estende Map e mantém suas chaves em ordem classificada — na ordem natural dos elementos ou em uma ordem especificada por uma implementação Comparator. A classe TreeMap implementa SortedMap.

MÉTODOS	DESCRIÇÃO
CLEAR()	Remove todos os pares chave/valor do mapa;
CONTAINSKEY(K)	Retorna true se o mapa invocador contiver o objeto k como chave;
CONTAINSVALUE(V)	Retorna true se o mapa contiver o objeto v como chave;
ENTRYSET()	Retorna um conjunto que contenha as entradas no mapa;
EQUALS()	Retorna true se mapas contiverem as mesmas entradas;
GET()	Retorna o valor associado com a chave k;
REMOVE(K)	Remove a entrada que tiver chave igual a k;

CONJUNTOS

- UM SET É UMA COLLECTION NÃO ORDENADA QUE NÃO CONTÉM ELEMENTOS DUPLICADOS.
- HASHSET ARMAZENA SEUS ELEMENTOS EM UMA TABELA DE HASH.
- TREESET ARMAZENA SEUS ELEMENTOS EM UMA ÁRVORE.
- A INTERFACE SORTEDSET ESTENDE SET E REPRESENTA UM CONJUNTO QUE MANTÉM SEUS ELEMENTOS NA ORDEM DE CLASSIFICAÇÃO.
- A CLASSE TREESET IMPLEMENTA SORTEDSET.





Streams e Serialização

O JAVA 8 foi usado para incluir o método **stream**() na API de Collections e possibilitar que todas as coleções sejam **fontes de dados para streams**. Além deste, outros métodos padrão também foram incorporados à API de coleções, como o **removeIf**() na interface Collection, e em Comparator, o método **reversed**(), que retorna um novo comparador que realiza a ordenação ao contrário.

O Java 8 apresenta diversas funcionalidades adicionadas à linguagem , uma das principais funções é a Streams API, recurso que oferece ao desenvolvedor a possibilidade de trabalhar com conjuntos de elementos de forma mais simples e com um número menor de linhas de código. Isso se tornou possível graças à incorporação do paradigma funcional, combinado com as expressões lambda, o que facilita a manutenção do código e aumenta a eficiência no processamento devido ao uso de paralelismo.

A proposta em torno da Streams API é **reduzir a preocupação do desenvolvedor com a forma de implementar controle de fluxo ao lidar com coleções**, deixando isso a cargo da API. A ideia é iterar sobre essas coleções de objetos e, a cada elemento, realizar alguma ação, seja ela de filtragem, mapeamento, transformação, etc. Caberá ao desenvolvedor apenas definir qual ação será realizada sobre o objeto.

Stream map(Function mapper) **retorna um stream que consiste nos resultados da aplicação da função dada aos elementos desse stream**. Stream map (Mapeador de funções) é uma operação intermediária. As operações intermediárias são invocadas em uma instância do Stream e, depois que terminam seu processamento, fornecem uma instância do Stream como saída.

A Streams API traz uma nova opção para a manipulação de coleções em Java seguindo os princípios da programação funcional. Combinada com as expressões lambda, ela proporciona uma forma diferente de lidar com conjuntos de elementos, oferecendo ao desenvolvedor uma maneira simples e concisa de escrever código que resulta em facilidade de manutenção e paralelização sem efeitos indesejados em tempo de execução.

A proposta em torno da Streams API é fazer com que **o desenvolvedor não se preocupe mais com a forma de se programar o comportamento**, deixando a parte relacionada ao **controle de fluxo e loop a cargo da API**. É algo muito parecido com o que é feito com Threads, onde os aspectos mais complexos ficam encapsulados em APIs e as regras de negócio passam a ser a única responsabilidade do desenvolvedor.

Portanto, sabemos que o Java 8 introduz o conceito de streams, que são semelhantes aos iteradores. Fluxos são objetos das classes que implementam a interface Stream (do pacote java.util.stream) ou uma das interfaces de fluxo especializadas para processar coleções de valores int, long ou double, Juntamente com lambdas, fluxos **permitem realizar tarefas sobre coleções de elementos, muitas vezes a partir de um objeto array ou coleção**.

Fluxos movem elementos por meio de uma sequência de passos de processamento conhecidos como **pipeline de fluxo** — que começa com uma origem de dados (como um array ou coleção), realiza **várias operações intermediárias** sobre os elementos da origem de dados e termina com uma operação terminal. Um pipeline de fluxo é formado encadeando chamadas de método. Ao contrário de coleções, **fluxos não têm um armazenamento próprio** — depois que o fluxo é processado, ele **não pode ser reutilizado**, porque **não mantém uma cópia da origem de dados original.**

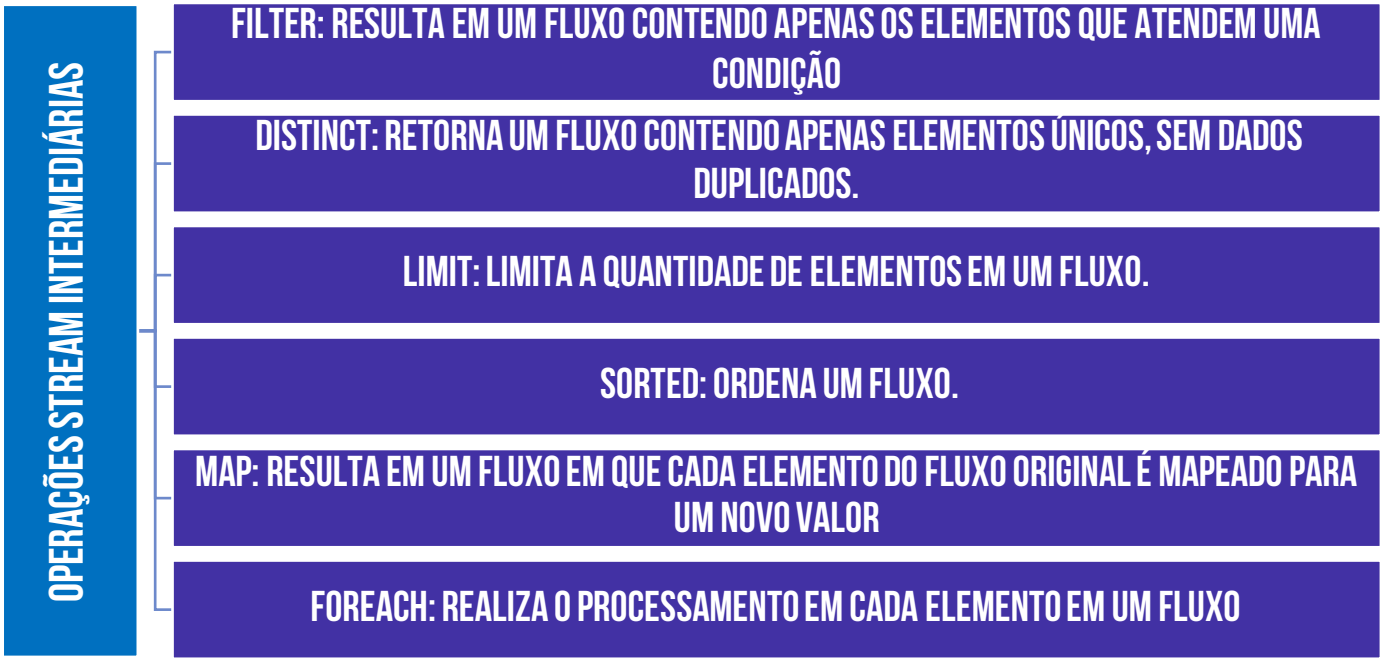
Uma **operação intermediária** especifica as tarefas a realizar sobre os elementos do fluxo e sempre resulta em um novo fluxo. Operações intermediárias são "preguiçosas" — elas só são executadas depois que uma operação terminal é invocada. Isso permite que desenvolvedores de biblioteca otimizem o desempenho do processamento de fluxo. Por exemplo, se você tem uma coleção de 1.000.000 objetos Person e está procurando o primeiro com o sobrenome "Jones", o processamento de fluxo pode terminar assim que o primeiro desses objetos Person for encontrado.

A **operação terminal** inicia o processamento das operações intermédias de um pipeline de fluxo e **produz um resultado**. Operações terminais são "gulosas" — elas realizam a operação solicitada quando são chamadas.

CLASSE	DESCRIÇÃO
FILTER	Resulta em um fluxo contendo apenas os elementos que atendem uma condição.
DISTINCT	Resulta em um fluxo que contém somente os elementos únicos.

LIMIT	Resulta em um fluxo com o número especificado de elementos a partir do início do fluxo original.
MAP	Resulta em um fluxo em que cada elemento do fluxo original é mapeado para um novo valor (possivelmente de um tipo diferente) — por exemplo, mapear valores numéricos para as raízes quadradas dos valores numéricos. O novo fluxo tem o mesmo número de elementos que o fluxo original.
SORTED	Resulta em um fluxo em que os elementos estão em ordem classificada. O novo fluxo tem o mesmo número de elementos que o fluxo original.
FOREACH	Realiza o processamento em cada elemento em um fluxo (por exemplo, exibir cada elemento).

Método filter: filtra os elementos de um fluxo de acordo com alguma condição fornecida, ao final ele nos retorna um fluxo apenas com os elementos que se enquadaram na condição. Dessa forma, o método filter() cria um novo array com **todos os elementos que passaram no teste implementado pela função fornecida**.



Code Conventions

Code Conventions Java é uma regra a ser seguida ao decidir **como nomear seus identificadores**, como **classe, pacote, variável, constante, método** etc. Mas, não é forçado a seguir. Então, é conhecido como **convenção e não regra**. Essas convenções são sugeridas por várias comunidades Java, como Sun Microsystems e Netscape.

Todas as classes, interfaces, pacotes, métodos e campos da linguagem de programação Java são fornecidos de acordo com a convenção de nomenclatura Java. **Se você não seguir** essas convenções, **poderá gerar confusão ou código errôneo**.

Vantagem de convenções de nomenclatura em Java: Ao usar convenções de nomenclatura Java padrão, você **torna seu código mais fácil de ler** para você e outros programadores. A legibilidade do programa Java é muito importante. Indica que menos tempo é gasto para descobrir o que o código faz.

TIPO	DESCRIÇÃO
NOMES E IDENTIFICADORES DE CLASSE	Os nomes de classes iniciam com uma letra maiúscula e apresentam a letra inicial de cada palavra que eles incluem em maiúscula (por exemplo, SampleClassName). Essa convenção de nomenclatura é conhecida como notação camelo, porque as letras maiúsculas destacam-se como corcovas desse animal.
COMENTÁRIOS INICIAIS	Todo arquivo de código deve iniciar com comentários de início, que devem conter informações a respeito da classe/interface, como por exemplo, nome da classe, informações de versão, data e autor
DECLARAÇÕES DE PACOTES E IMPORTAÇÕES	A primeira linha de código após os comentários iniciais deve ser a declaração do pacote (se necessário), e em sequência as declarações de importações
DECLARAÇÃO DA CLASSE/INTERFACE	A ordem em que elas devem aparecer é: Comentário de documentação, Declaração da classe/interface, Comentários de implementação da classe, Atributos estáticos, Demais atributos, Construtores, Demais métodos.
INDENTAÇÃO OU RECUO DE CÓDIGO	Os recuos são utilizados para alinhar visualmente comandos pertencentes a blocos de código. Um bloco é o código envolvido pelos delimitadores { e }, como por exemplo o if. A regra geral é abrir o bloco na linha do comando e fechar alinhado a ele.
QUEBRAS DE LINHA	Quando uma expressão não couber numa única linha, quebrá-la de acordo com as seguintes regras básicas: Após uma vírgula, Antes de um operador, Alinhar a nova linha com o início da expressão da linha anterior, Se as regras acima gerarem código confuso, ou se a linha de baixo ficar colada na margem, use uma tabulação de 8 espaços.
COMENTÁRIOS DE IMPLEMENTAÇÃO	Há quatro possíveis formatos de comentário de implementação: bloco, linha, à direita e no fim da linha. Os comentários de bloco são utilizados para descrever arquivos de código, métodos ou algum algoritmo dentro de um método. Procure sempre inserir uma linha em branco antes do bloco de comentários. Os comentários de linha são para comentários curtos, e devem ser precedidos com uma linha em branco, assim como o comentário de bloco. Comentários muito curtos podem aparecer na mesma linha do código que descrevem, mas devem ser deslocados o suficiente para separá-los das declarações. O delimitador de comentário // pode servir para comentar toda uma linha ou apenas parte dela.
DECLARAÇÕES	Ao declarar variáveis, observe as seguintes regras: Faça apenas uma declaração por linha, de modo a incentivar o uso de comentários, Inicialize apenas uma variável por linha, Use letras minúsculas e evite caracteres especiais.
PACOTES	Letras minúsculas sem caracteres especiais
CLASSES E INTERFACES	Os nomes de classe devem ser substantivos, com a primeira letra de cada palavra interna em maiúscula. Tente manter seus nomes de classe simples e descritivos. Evite siglas e abreviações (a menos que a sigla seja muito mais usada do que a forma longa, como a URL ou HTML) .

MÉTODOS	Métodos devem ser verbos, em maiúsculas e minúsculas com a primeira letra minúscula, com a primeira letra de cada palavra interna em maiúscula.
VARIÁVEIS	Devem iniciar com minúscula. Palavras internas começam com letras maiúsculas. Não deve começar com underline ou $, mesmo que ambos sejam permitidos. Use nomes curtos, mas significativos. O nome deve indicar a intenção da utilização da variável. Os nomes comuns para variáveis temporárias são i, j, k, m, n e para inteiros, c, d, e e para caracteres.
CONSTANTES	Constantes devem ter todas as letras maiúsculas separadas por underline

As **boas práticas de programação** chamam a atenção a técnicas que irão ajudá-lo a criar programas que são mais claros, mais compreensíveis e mais fáceis de manter. Vejamos algumas.

- Algumas organizações exigem que todo programa comece com um comentário que informa o objetivo e o autor dele, a data e a hora em que foi modificado pela última vez.
- Utilize linhas e espaços em branco para aprimorar a legibilidade do programa.
- Recue o corpo inteiro de cada declaração de classe por um "nível" entre a chave esquerda e a chave direita que delimitam o corpo da classe. Esse formato enfatiza a estrutura da declaração de classe e torna mais fácil sua leitura. Usamos três espaços para formar um nível de recuo — muitos programadores preferem dois ou quatro espaços. Qualquer que seja o estilo que você escolher, utilize-o de modo consistente.
- Recue o corpo inteiro de cada declaração de método um "nível" entre as chaves que definem o corpo do método. Isso faz com que a estrutura do método se destaque, tornando a declaração do método mais fácil de ler.
- Coloque um espaço depois de cada vírgula (,) em uma lista de argumentos para tornar os programas mais legíveis.
- Declarar cada variável em uma declaração própria. Esse formato permite que um comentário descritivo seja inserido ao lado de cada ser variável que é declarado.
- Escolher nomes de variáveis significativos ajuda um programa a ser autodocumentado (isto é, pode-se entender o programa simplesmente lendo-o em vez de ler os documentos associados ou visualizar um número excessivo de comentários).
- Coloque espaços de ambos os lados de um operador binário para legibilidade.
- Insira uma única declaração por linha em um programa para facilitar a leitura
- Ao escrever expressões que contêm muitos operadores, consulte a tabela do operador de precedência. Confirme se as operações na expressão são realizadas na ordem que você

espera. Se, em uma expressão complexa, você não estiver seguro quanto à ordem da avaliação, utilize parênteses para forçar essa ordem, exatamente como você faria em expressões algébricas.

- Recue as duas instruções do corpo de uma instrução if...else. Muitos IDEs fazem isso por você. Se existem vários níveis de recuo, cada nível deve ser recuado pela mesma quantidade adicional de espaço.

- Sempre inclua chaves em uma instrução do...while. Isso ajuda a eliminar ambiguidade entre a instrução while e uma instrução do...while que contém apenas uma instrução.

Java 8

O Java 8 é a release mais recente do Java que contém novas funcionalidades, aprimoramentos e correções de bug para aumentar a eficiência do desenvolvimento e execução de programas Java. A nova release do Java primeiro é disponibilizada para desenvolvedores, a fim de permitir um tempo adequado de teste e certificação, e só então fica disponível no site java.com para que usuários finais façam download. As novas funcionalidades do Java 8 são:

Métodos de Expressão Lambda e Extensão Virtual: O destaque do Java SE 8 é a implementação de expressões Lambda e funcionalidades de suporte para a linguagem de programação e plataforma Java.

API de Data e Hora: Essa nova API permitirá que os desenvolvedores tratem data e hora de maneira mais natural, clara e fácil de entender.

Nashhorn JavaScript Engine: Uma nova implementação leve de alto desempenho do motor JavaScript foi integrada ao JDk e está disponível para aplicações Java por meio das APIs existentes.

Maior Segurança: A lista manual existente de métodos sensíveis do chamador foi substituída por um mecanismo que identifica com precisão esses métodos e permite que seus chamadores sejam descobertos com confiança.

Java 8 trouxe também o recurso denominado Default Methods, que foi introduzido para possibilitar a evolução de interfaces ao permitir que interfaces já existentes ofereçam métodos novos sem que os códigos que as implementem também tenham que fornecer uma implementação para esses métodos.

Diante disso, note que um dos focos dessa nova versão continua sendo manter a compatibilidade com códigos legados e ser o menos intrusivo possível, ou seja, afetar o menos possível as antigas APIs.

Java 17

A Oracle anunciou em setembro de 2021 a disponibilização do Java 17, a versão mais recente da plataforma de desenvolvimento e linguagem de programação que é **número um do mundo**. O Java 17 oferece **milhares de atualizações de desempenho, estabilidade e segurança**, bem como **14 JEPs (JDK Enhancement Proposals)** que melhoram ainda mais a linguagem e a plataforma Java para ajudar os desenvolvedores a serem mais produtivos.

Java 17 é o mais recente lançamento de **suporte de longo prazo (LTS)** sob a cadência de lançamento de seis meses do Java e é o resultado de uma ampla colaboração entre engenheiros da Oracle e outros membros da comunidade mundial de desenvolvedores Java, por meio da OpenJDK Community e do Java Community Process (JCP). Desde o lançamento do JDK 11 LTS anterior há três anos, mais de 70 JEPs foram implementados.

Os JEPs são divididos em: Aprimoramento da linguagem Java (JEP 409), Atualizações e melhorias para bibliotecas(JEPs 306, 356, e 382), Suporte para nova plataforma (JEP 391), Removals and Deprecations (JEPs 398, 407, 410 e 411), Programas Java de Prova Futura (JEP 403), Visualizações e incubadoras para versões posteriores do JDK (JEPs 406, 412, 414). Vejamos os nomes e as descrições dos 17 **JEPs!**

JEP	DESCRIÇÃO
JEP 409: CLASSES SELADAS	Classes e interfaces seladas restringem outras classes ou interfaces que podem estendê-las ou implementá-las. Esse aprimoramento é mais um aprimoramento do Projeto Amber, que visa aumentar a produtividade do desenvolvedor por meio da evolução da linguagem Java.
JEP 306: RESTAURE A SEMÂNTICA DE PONTO FLUTUANTE ALWAYS-STRICT	A linguagem de programação Java e a máquina virtual Java, originalmente tinham apenas semântica de ponto flutuante estrita. A partir do Java 1.2, pequenas variações nessas semânticas estritas foram permitidas por padrão para acomodar as limitações das arquiteturas de hardware atuais. Essas variações não são mais úteis ou necessárias, por isso foram removidas pelo JEP 306.
JEP 356: GERADOR DE NÚMERO PSEUDO-ALEATÓRIO APRIMORADO	Fornece novos tipos de interface e implementações para geradores de número pseudo-aleatório (PRNGs). Essa mudança melhora a interoperabilidade de diferentes PRNGs e torna mais fácil solicitar um algoritmo com base em requisitos, em vez de codificar uma implementação específica.
JEP 382: NOVO PIPELINE DE RENDERIZAÇÃO DO MACOS	Implementa um pipeline Java 2D para macOS usando a API Apple Metal. O novo pipeline reduzirá a dependência do JDK na API Apple OpenGL obsoleta.
JEP 391: MACOS AARCH64 PORT	Portas do JDK para a plataforma macOS / AArch64. Esta porta permitirá que os aplicativos Java sejam executados nativamente nos novos computadores Apple Silicon baseados no Arm 64.
JEP 398: DESCONTINUAR A API DO MINIAPLICATIVO PARA REMOÇÃO	Todos os fornecedores de navegadores da web removeram o suporte para plug-ins de navegador Java ou anunciaram planos para iniciarem a solução. A API Applet foi descontinuada, mas não para remoção, no Java 9 em setembro de 2017.

JEP 407: Remove RMI Activation	Remove o mecanismo de ativação de Remote Method Invocation (RMI), preservando o resto do RMI.
JEP 410: Remova o AOT experimental e o compilador JIT	O compilador experimental baseado em Java (AOT) e just-in-time (JIT) foram recursos experimentais que não tiveram muita adoção. Por serem opcionais, eles já foram removidos do JDK 16. Este JEP remove esses componentes do código-fonte do JDK.
JEP 411: Obsoleta o Security Manager para remoção	O Security Manager remonta ao Java 1.0. Não foi o principal meio de proteger o código Java do lado do cliente por muitos anos e, raramente, foi usado para proteger o código do lado do servidor. Removê-lo em uma versão futura ajudará a eliminar uma carga de manutenção significativa e permitirá que a plataforma Java avance.
JEP 403: Encapsular fortemente JDK	Não será mais possível relaxar o forte encapsulamento de elementos internos por meio de uma única opção de linha de comando, como era possível no JDK 9 ao JDK 16. Ainda será possível acessar os APIs internos existentes, mas agora exigirá enumerar, como parâmetros de linha de comando ou atributos de manifesto de arquivo JAR, cada pacote para o qual o encapsulamento deve ser relaxado. Essa mudança levará a aplicativos mais seguros e menos dependências de detalhes de implementação JDK internos que não são padrão.
JEP 406: Correspondência de padrões para switch (visualização)	Permite que uma expressão seja testada em vários padrões, cada um com uma ação específica, de forma que consultas complexas orientadas a dados, possam ser expressas de forma concisa e segura.
JEP 412: Função externa e API de memória (incubadora)	Melhora as APIs de incubação introduzidas no JDK 14 e JDK 15 que permitem que, programas Java interoperem com código e dados fora do tempo de execução Java. Invocando com eficiência funções externas (ou seja, código fora da JVM) e acessando com segurança a memória externa, essas APIs permitem que os programas Java chamem bibliotecas nativas e processem dados nativos sem a fragilidade e complexidade da Java Native Interface (JNI). Essas APIs estão sendo desenvolvidas no Projeto Panamá, que visa melhorar a interação entre código Java e não Java.
JEP 414: API Vector (segunda incubadora)	Permite expressar cálculos vetoriais que compilam de forma confiável em tempo de execução para instruções vetoriais otimizadas em arquiteturas de CPU suportadas, alcançando, assim, um desempenho superior aos cálculos escalares equivalentes.

O Java 17 LTS é a versão de suporte de longo prazo **mais recente para a plataforma Java SE**. Os binários JDK 18 e JDK 17 são gratuitos para uso em produção e para redistribuição, sem custo.

No site da Oracle temos a informação que o JDK 18 receberá atualizações **até setembro de 2022**, quando será substituído pelo **JDK 19.** O JDK 17 receberá atualizações até pelo menos **setembro de 2024**.

REFERÊNCIAS

https://www.oracle.com/br/java/

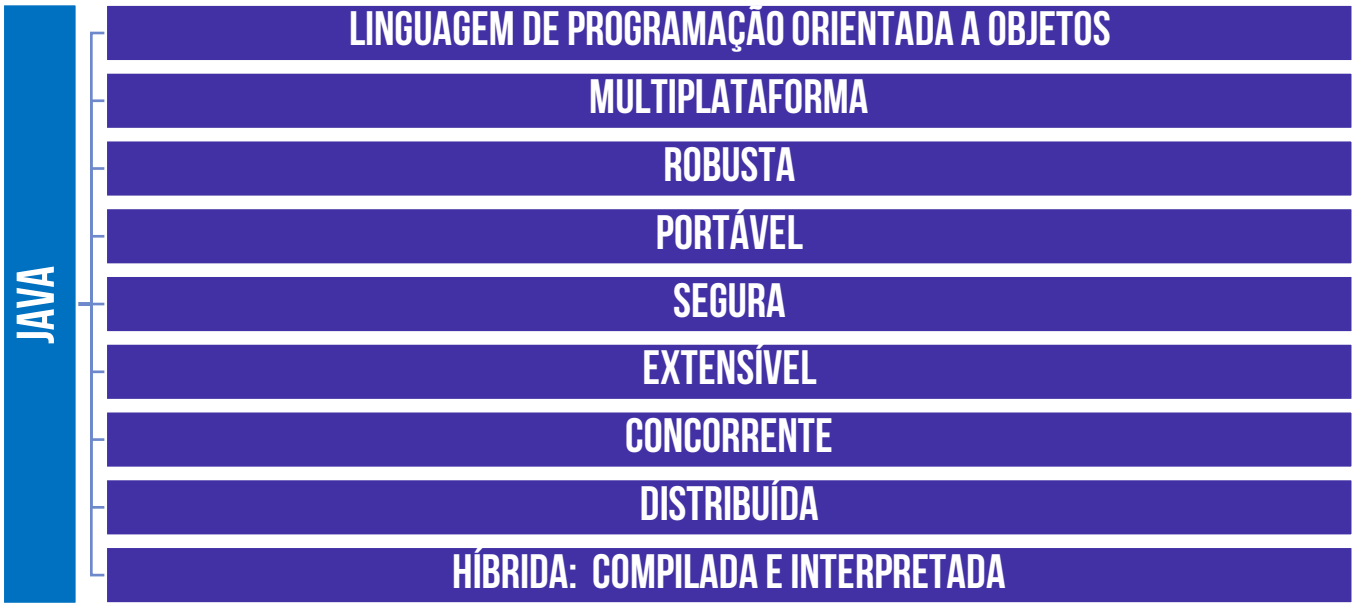
https://www.w3schools.com/java/java_ref_keywords.asp

Deitel, Paul. Java: como programar / Paul Deitel, Harvey Deitel; tradução Edson Furmankiewicz; revisão técnica Fabio Lucchini. -- São Paulo: Pearson Education do Brasil, 2017

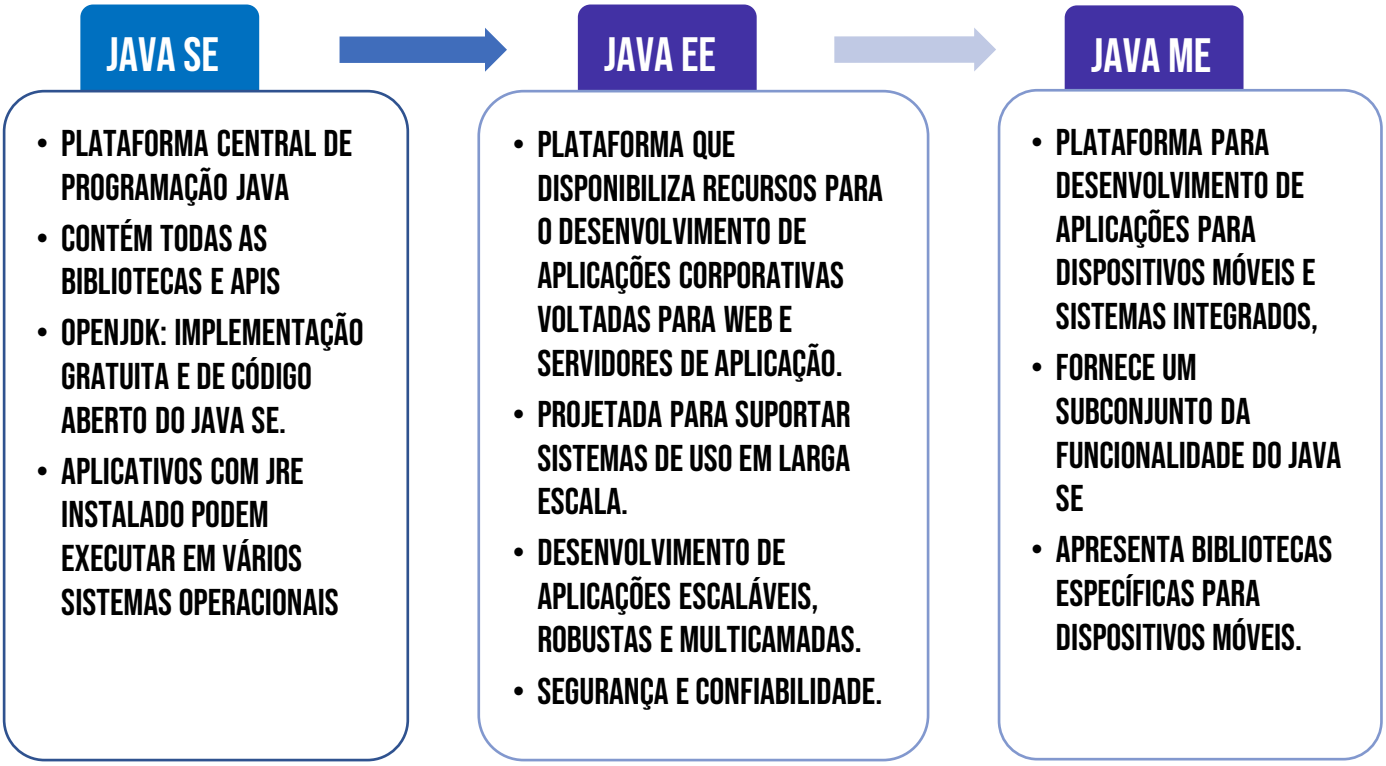
RESUMO

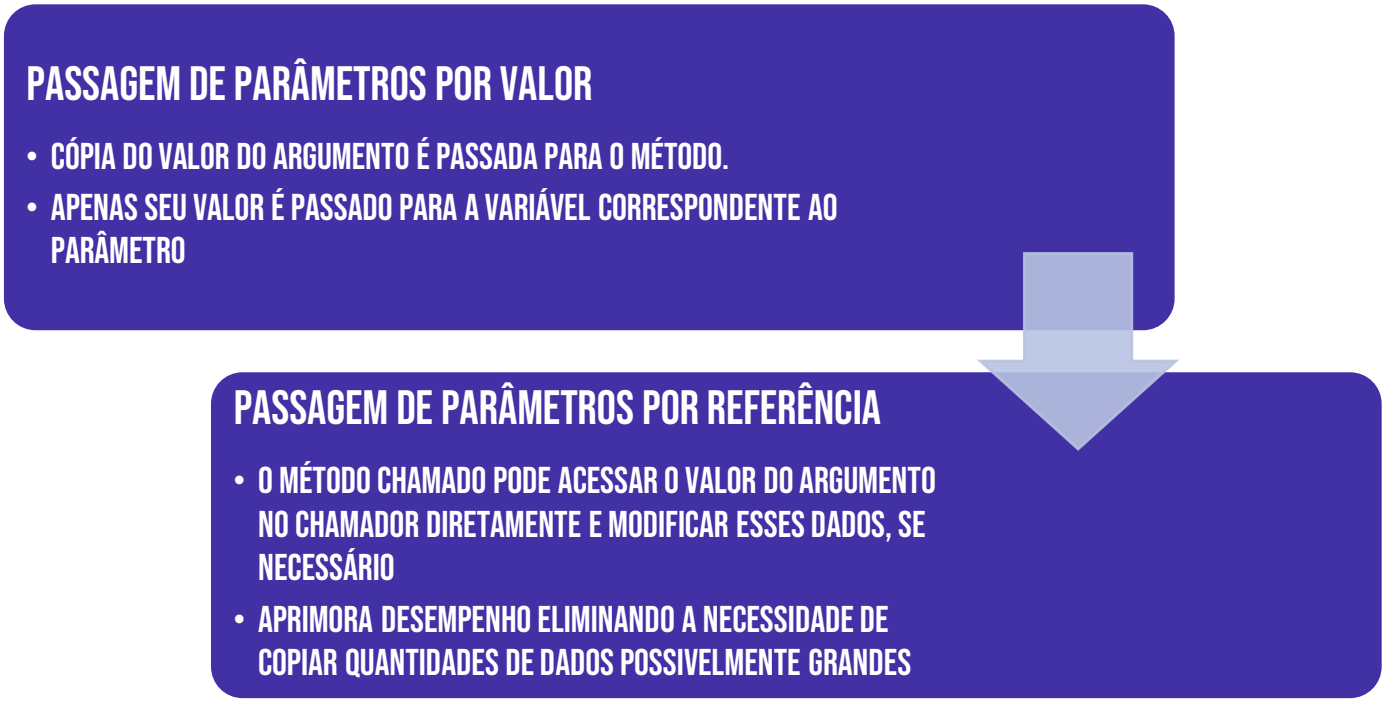
O Java é a principal linguagem de programação e plataforma de desenvolvimento. Reduz custos e prazos de desenvolvimento, impulsiona a inovação e aprimora os serviços de aplicativos. Com milhões de desenvolvedores executando **mais de 51 bilhões de Java Virtual Machines em todo o mundo.**

CARACTERÍSTICA	DESCRIÇÃO
CONCISA E SIMPLES	Não contém redundâncias e é fácil de entender, implementar e usar. Parecida com C++ para facilitar compreensão por grande parte de programadores. É uma evolução de C++: não suporta aritmética de ponteiros, registros, coercions, etc.
ORIENTADA A OBJETOS	Suporta os principais conceitos de orientação a objetos. Favorece reusabilidade.
ROBUSTA	Altamente tipada. Programas são confiáveis. Reduz imprevistos em tempo de execução: variáveis são automaticamente inicializadas, uso disciplinado de ponteiros, rotinas devem ser chamadas corretamente, etc.
PORTÁVEL	Completamente especificada. Não contém aspectos dependentes da implementação: o tamanho dos tipos é fixo para qualquer implementação, etc.
SEGURA	Restrições de acesso a arquivos, manipulação de ponteiros, etc. Implica que não é útil para desenvolver certas aplicaçoes como `device drivers'.
CONCORRENTE	Suporta aplicações concorrentes: multithreads, monitores, execução atômica.
INDEPENDENTE DE PLATAFORMA	Geração de bytecode que pode ser interpretado para qualquer arquitetura e sistema operacional tendo o sistema Java. Facilita distribuição de software.
INTERPRETADA	Facilita desenvolvimento exploratório.
COMPILADA	Utilizando compiladores, bytecodes podem ser traduzidos em tempo de execução para código de máquina.



Java é uma linguagem **WORA**, que significa Write once, run anywhere ou "**Escreva uma vez, execute em qualquer lugar**"





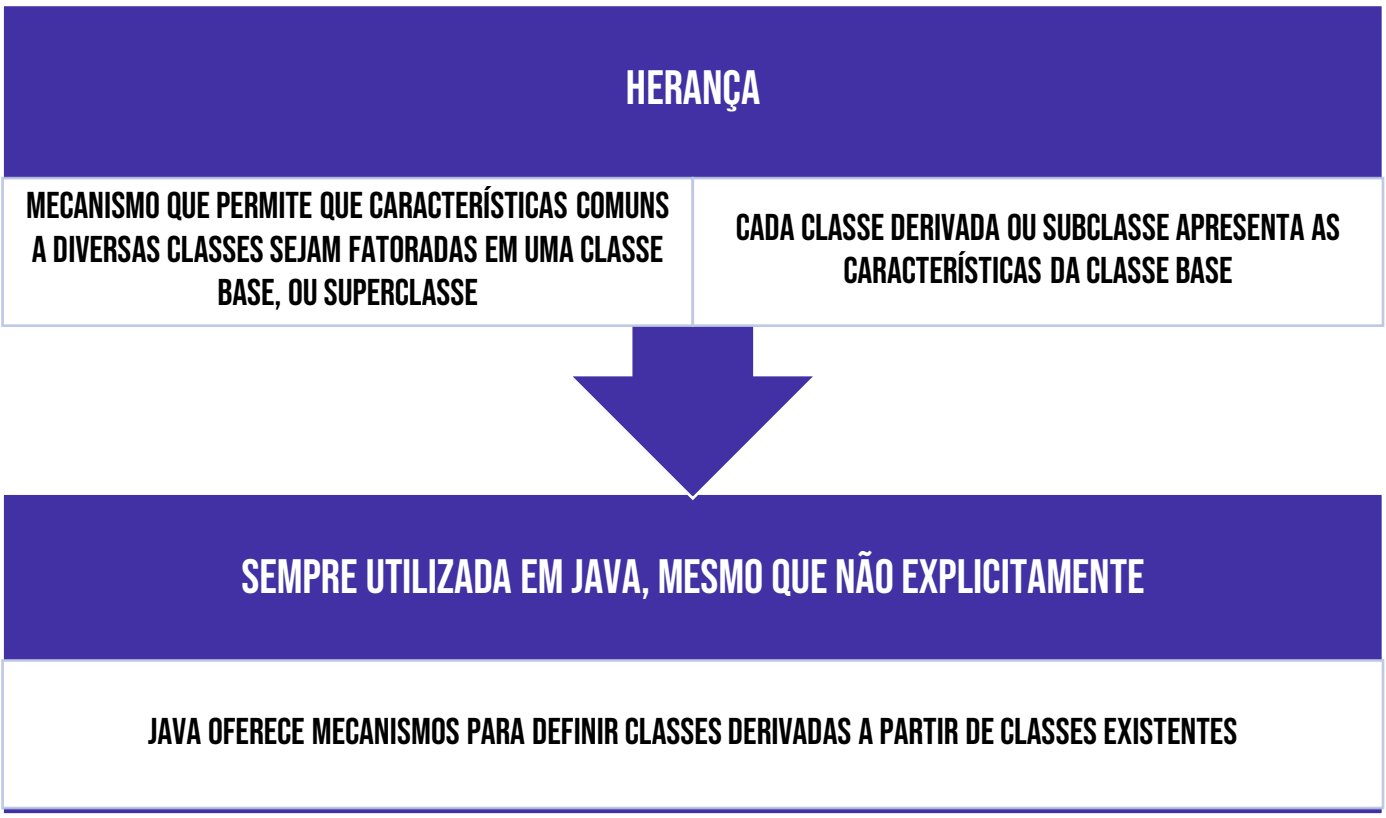
Pacotes

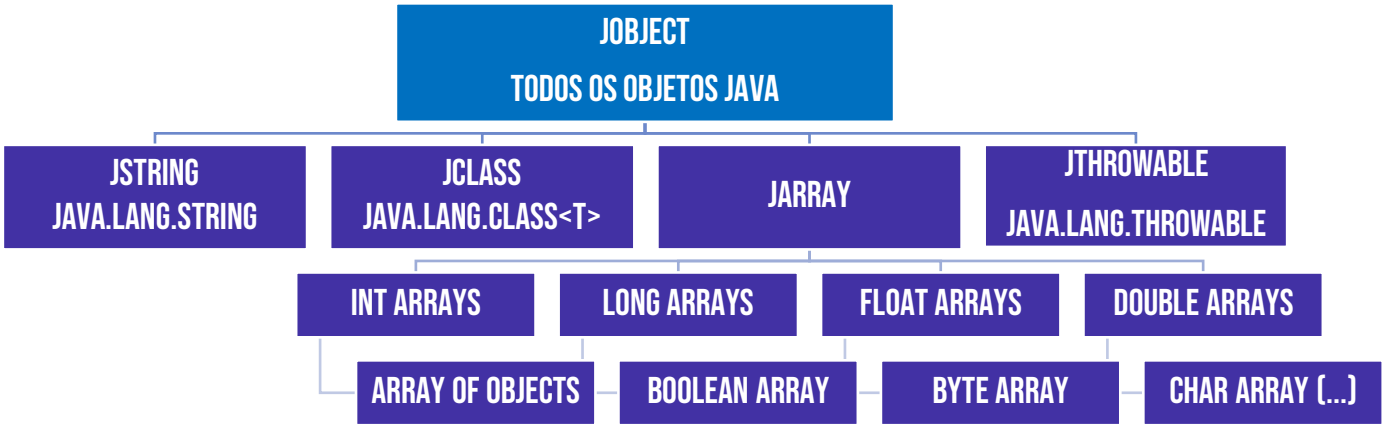
O software de aplicativos é distribuído em unidades denominadas **pacotes**. Um pacote é um conjunto de arquivos e diretórios necessários para um produto de software. Um pacote é geralmente criado e construído por um desenvolvedor de aplicativos depois de terminar de

desenvolver o código do aplicativo. Um produto de software precisa ser construído em um ou mais pacotes para que possa ser facilmente transferido para um meio de distribuição. Depois, o produto de software pode ser produzido em massa e instalado por administradores. Um pacote é um **conjunto de arquivos e diretórios em um formato definido**.

Para construir um pacote, você deve fornecer:

- Objetos de pacote (arquivos e diretórios do software de aplicativo)
- Dois arquivos de informação (os arquivos pkginfo e prototype)
- Arquivos de informação opcionais
- Scripts de instalação opcionais





```
jobject                          (all objects)
    ├── jclass                   (java.lang.Class instances)
    ├── jstring                  (java.lang.String instances)
    ├── jarray                   (arrays)
    │       ├── jobjectArray     (Object[])
    │       ├── jbooleanArray    (boolean[])
    │       ├── jbyteArray       (byte[])
    │       ├── jcharArray       (char[])
    │       ├── jshortArray      (short[])
    │       ├── jintArray        (int[])
    │       ├── jlongArray       (long[])
    │       ├── jfloatArray      (float[])
    │       └── jdoubleArray     (double[])
    └── jthrowable               (java.lang.Throwable objects)
```

APPLET

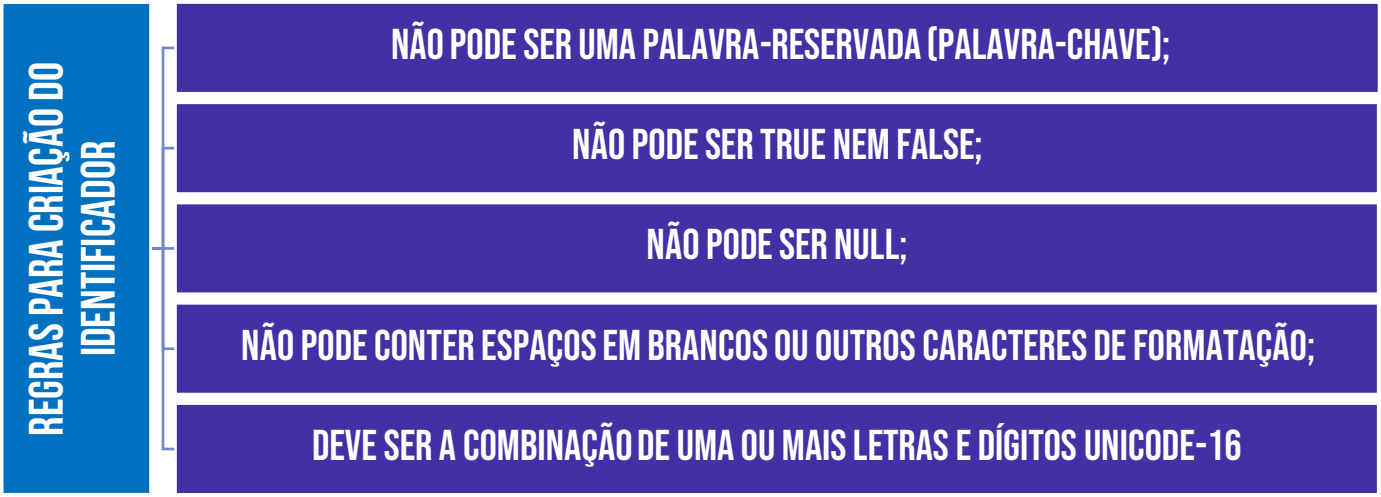
- PEQUENA APLICAÇÃO EXECUTADA EM UMA JANELA
- ESTENDE AS FUNCIONALIDADES DE BROWSERS, ADICIONANDO SOM, ANIMAÇÃO
- NECESSITA DE UM NAVEGADOR, POR OUTRO
- PODE SER DISPONIBILIZADA PARA EXECUÇÃO VIA INTERNET

5 MÉTODOS QUE DEFINEM O CICLO DE VIDA DE UMA APPLET

- INIT( ) - INICIALIZAÇÃO; CHAMADO QUANDO A APPLET É INICIALMENTE CARREGADA;
- START( ) - EXECUÇÃO (ANIMAÇÃO); CHAMADO APÓS O INIT( );
- STOP( ) - INTERRUPÇÃO; FAZ A APPLET PARAR A EXECUÇÃO DA ANIMAÇÃO, ÁUDIO OU VÍDEO;
- PAINT() - PARA DESENHAR ALGO NA APPLET;
- DESTROY( ) - LIBERAÇÃO DE RECURSOS; CHAMADO QUANDO O BROWSER É FECHADO.

Identificadores

O Java é **case sensitive**, ou seja, Java **diferencia maiúsculas de minúsculas**: "MyClass" e "myclass" têm significados diferentes assim como Analista é diferente de analista.



Palavras Reservadas

Palavras-chave, ou **palavras reservadas não podem ser usadas como identificadores**, ou seja, **não podem ser usadas como nome de variáveis, nome de classes, etc.**

PALAVRA	DESCRIÇÃO
ABSTRACT	Um modificador sem acesso. Usado para classes e métodos: Uma classe abstrata não pode ser usada para criar objetos (para acessá-la, ela deve ser herdada de outra classe). Um método abstrato só pode ser usado em uma classe abstrata e não possui um corpo. O corpo é fornecido pela subclasse (herdado de)
BOOLEAN	Um tipo de dados que pode armazenar apenas valores verdadeiros e falsos
BREAK	Sai de um loop ou de um bloco de repetição
BYTE	Um tipo de dados que pode armazenar números inteiros de -128 e 127
CASE	Marca um bloco de código em instruções switch
CATCH	Captura exceções geradas por instruções try
CHAR	Captura exceções geradas por instruções try
CLASS	Define uma classe
CONST	Define uma constante. Não está em uso - use final.
CONTINUE	Continua para a próxima iteração de um loop
DEFAULT	Especifica o bloco de código padrão em uma instrução switch
DO	Usado junto com while para criar um loop do-while.
DOUBLE	Um tipo de dados que pode armazenar números inteiros de 1,7e−308 a 1,7e+308
ELSE	Usado em instruções condicionais
ENUM	Declara um tipo enumerado (imutável)

EXTENDS	Estende uma classe (indica que uma classe é herdada de outra classe)
FINAL	Um modificador sem acesso usado para classes, atributos e métodos, o que os torna inalteráveis (impossíveis de herdar ou substituir)
FINALLY	Usado com exceções, um bloco de código que será executado independente de haver uma exceção ou não
FLOAT	Um tipo de dados que pode armazenar números inteiros de 3.4e−038 a 3.4e+038
FOR	Criar um loop for
GOTO	Não está em uso
IF	Faz uma declaração condicional
IMPLEMENTS	Implementa uma interface
IMPORT	Usado para importar um pacote, classe ou interface
INSTANCEOF	Verifica se um objeto é uma instância de uma classe específica ou uma interface
INT	Um tipo de dados que pode armazenar números inteiros de -2147483648 a 2147483647
INTERFACE	Usado para declarar um tipo especial de classe que contém apenas métodos abstratos
LONG	Um tipo de dados que pode armazenar números inteiros de -9223372036854775808 a 9223372036854775808
MÓDULO	Declara um módulo. Novo no Java 9
NATIVE	Especifica que um método não é implementado no mesmo arquivo de origem Java (mas em outra linguagem)
NEW	Cria novos objetos
PACKAGE	Declara um pacote
PRIVATE	Um modificador de acesso usado para atributos, métodos e construtores, tornando-os acessíveis apenas dentro da classe declarada
PROTECTED	Um modificador de acesso usado para atributos, métodos e construtores, tornando-os acessíveis no mesmo pacote e subclasses
PUBLIC	Um modificador de acesso usado para classes, atributos, métodos e construtores, tornando-os acessíveis por qualquer outra classe
REQUIRES	Especifica as bibliotecas necessárias dentro de um módulo. Novo no Java 9
RETURN	Finalizou a execução de um método e pode ser usado para retornar um valor de um método
SHORT	Um tipo de dados que pode armazenar números inteiros de -32768 a 32767
STATIC	Um modificador sem acesso usado para métodos e atributos. Métodos/atributos estáticos podem ser acessados sem criar um objeto de uma classe
STRICTFP	Restringir a precisão e o arredondamento dos cálculos de ponto flutuante
SUPER	Refere-se a objetos da superclasse (pai)
SWITCH	Seleciona um dos muitos blocos de código a serem executados
SYNCHRONIZED	Um modificador sem acesso, que especifica que os métodos só podem ser acessados por um thread por vez

THIS	Refere-se ao objeto atual em um método ou construtor
THROW	Cria um erro personalizado
THROWS	Indica quais exceções podem ser lançadas por um método
TRANSIENT	Um modificador sem acesso, que especifica que um atributo não faz parte do estado persistente de um objeto
TRY	Cria uma instrução try...catch
VAR	Declara uma variável. Novo no Java 10
VOID	Especifica que um método não deve ter um valor de retorno
VOLATILE	Indica que um atributo não é armazenado em cache localmente e é sempre lido da "memória principal"
WHILE	Cria um loop while

Pessoal, estes modificadores são MUITO importantes e caem **MUITO** nas provas! Portanto repeti uma tabela com os modificadores de acesso!

MODIFICADORES	DESCRIÇÃO
DEFAULT OU PACKAGE	A classe e/ou seus membros são acessíveis somente por classes do mesmo pacote, na sua declaração não é definido nenhum tipo de modificador, sendo este identificado pelo compilador. Permite que apenas classes do mesmo pacote tenham acesso as propriedades que possuem este modificador.
PUBLIC	O código é acessível para todas as classes
PRIVATE	Membros da classe definidos como private não podem ser acessados ou usados por nenhuma outra classe. Esse modificador não se aplica às classes, somente para seus métodos e atributos. Esses atributos e métodos também não podem ser visualizados pelas classes herdadas.
PROTECTED	O modificador protected torna o membro acessível às classes do mesmo pacote ou através de herança, seus membros herdados não são acessíveis a outras classes fora do pacote em que foram declarados.

NOME	CLASSE	SUBCLASSE	PACOTE	GLOBAL
PRIVADO	Sim	Não	Não	Não
PROTEGIDO	Sim	Sim	Não	Não
PÚBLICO	Sim	Sim	Sim	Sim
PACOTE	Sim	Não	Sim	Não

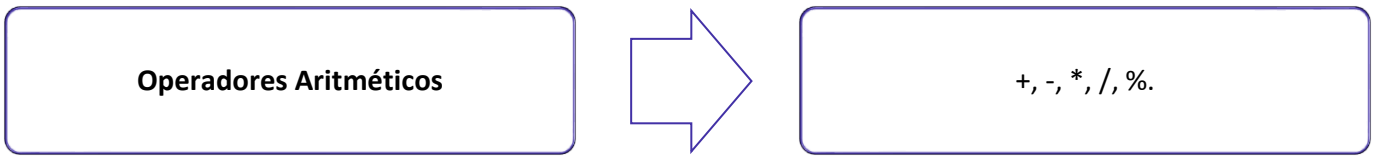
Tipos Primitivos

NOME	TIPO	TAMANHO	MÍNIMO	MÁXIMO	DEFAULT

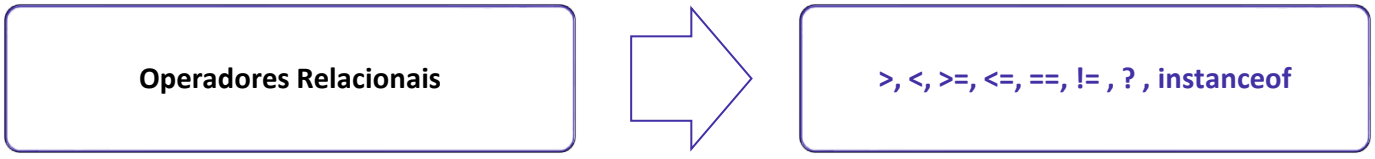
LÓGICO	boolean	-	false	true	false
CARACTERE	char	16 bits	0	$2^{16}-1$	'\uoooo'
INTEIRO	byte	8 bits	-2^{7}	$2^{7}-1$	0
	short	16 bits	-2^{15}	$2^{15}-1$	0
	int	32 bits	-2^{31}	$2^{31}-1$	0
	long	64 bits	-2^{63}	$2^{63}-1$	0
	float	32 bits	7 Casas Decimais		0.0
DECIMAL	double	64 bits	15 Casas Decimais		0.0

Operadores

Operadores são símbolos que representam atribuições, cálculos e ordem dos dados. As operações seguem uma ordem de prioridades, ou seja, alguns cálculos (ou outros) são processados antes de outros. Por exemplo, na Álgebra podemos mostrar a seguinte ordem: primeiro divisão e multiplicação e depois soma e subtração.



ARITMÉTICOS	ATRIBUIÇÃO	RELACIONAIS	LÓGICOS	BIT A BIT
+	=	>	!	&
-	+=	<	&&	|
*	-=	>=	||	^
/	*=	<=		<<
%	/=	!=		>>
	%=	==		>>>
	++	?		
	--	instanceof		



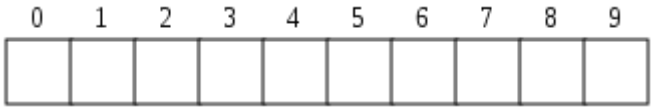
Precedência indica a ordem na qual um operador opera em relação à avaliação de uma expressão. A tabela seguinte elenca os operadores por precedência, do maior para o menor.

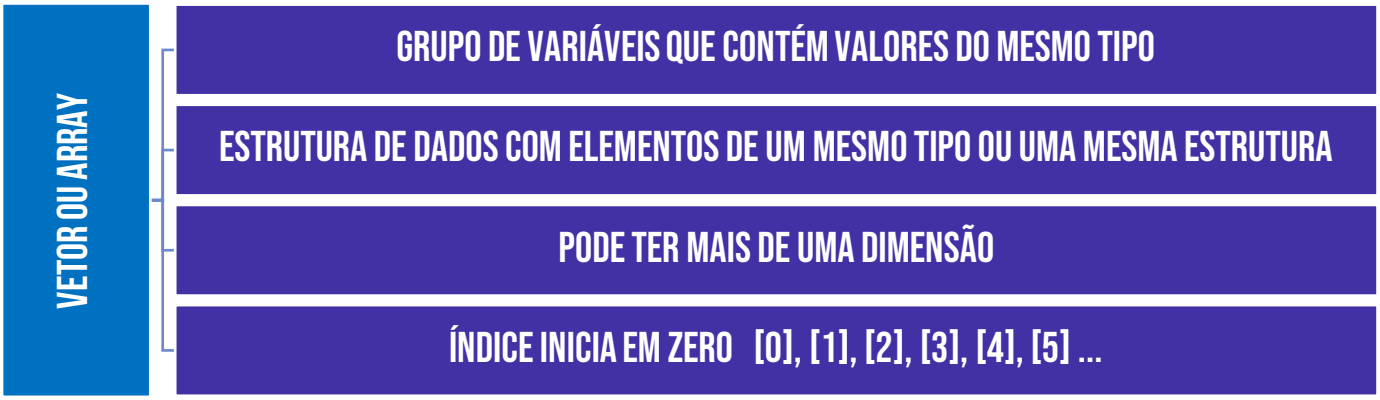
TIPO DE OPERADOR	LISTA DE OPERADORES
SUFIXAIS	expr++ expr--

PREFIXAIS	++expr --expr +expr -expr ~ !
MULTIPLICATIVOS	* / %
ADITIVOS	+ -
SHIFT BINÁRIO	<< >> >>>
COMPARATIVOS	< > <= >= instanceof
IGUALDADE	== !=
BIT-A-BIT E	&
BIT-A-BIT XOU OR	^
BIT-A-BIT OU OR	|
LÓGICO E	&&
LÓGICO OU	||
TERNÁRIO	? :
ATRIBUIÇÃO	= += -= *= /= %= &= ^= |= <<= >>= >>>=

Vetores

Um array ou vetor é um grupo de variáveis (chamados elementos ou componentes) que contém valores todos do mesmo tipo. Veja um vetor de uma dimensão e dez elementos:

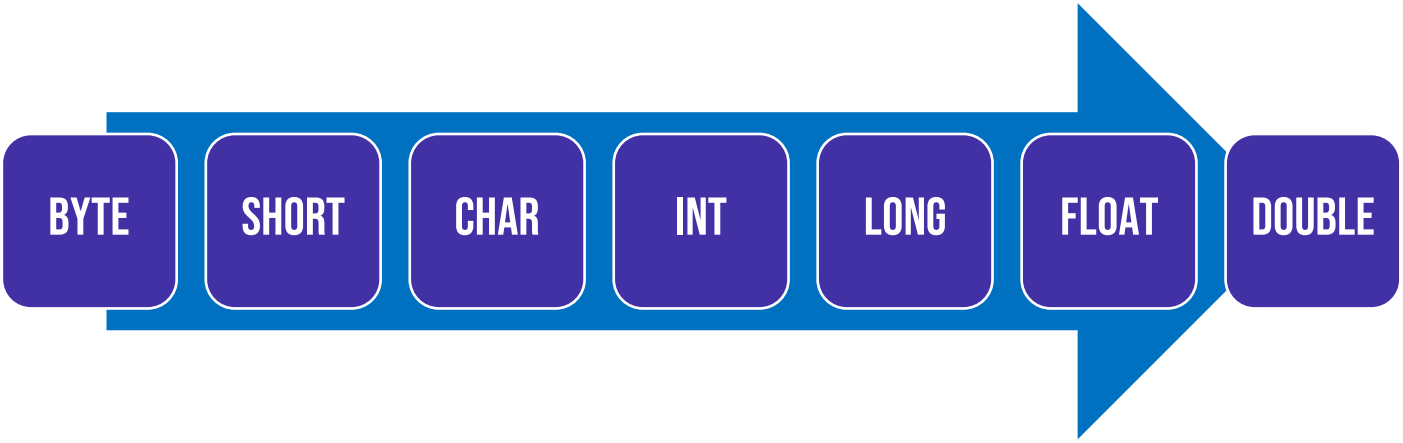




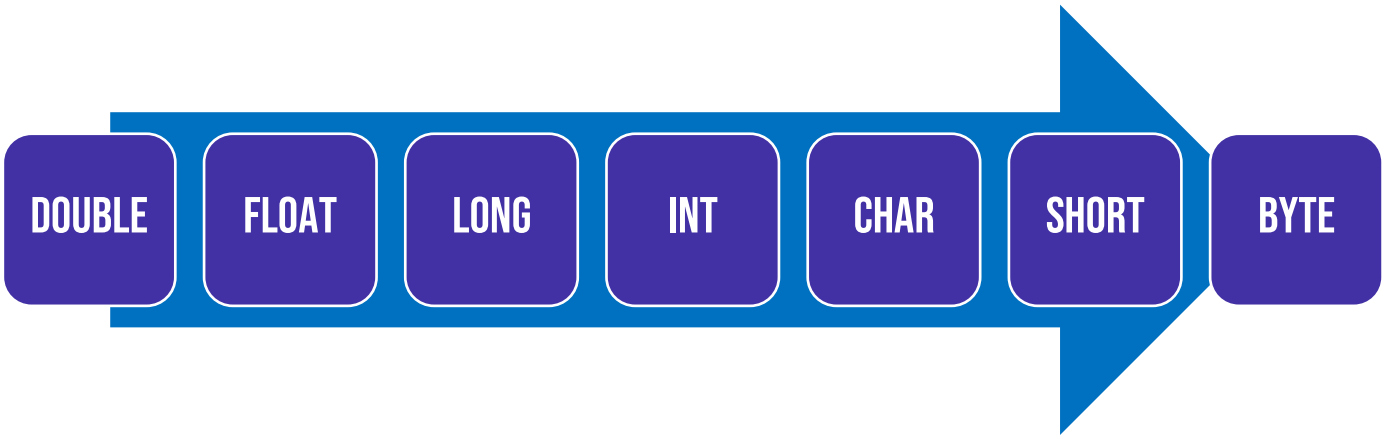
Conversão de Tipos

A conversão de tipo é quando você atribui um valor de um tipo de dados primitivo a outro tipo. Em Java, existem dois tipos de conversão:

Ampliação da transmissão (automaticamente) - convertendo um tipo menor em um tamanho de tipo maior:



Estreitando Casting (manualmente) - convertendo um tipo maior para um tipo de tamanho menor (caminho inverso)



Controle de Fluxo

CONDICIONAIS

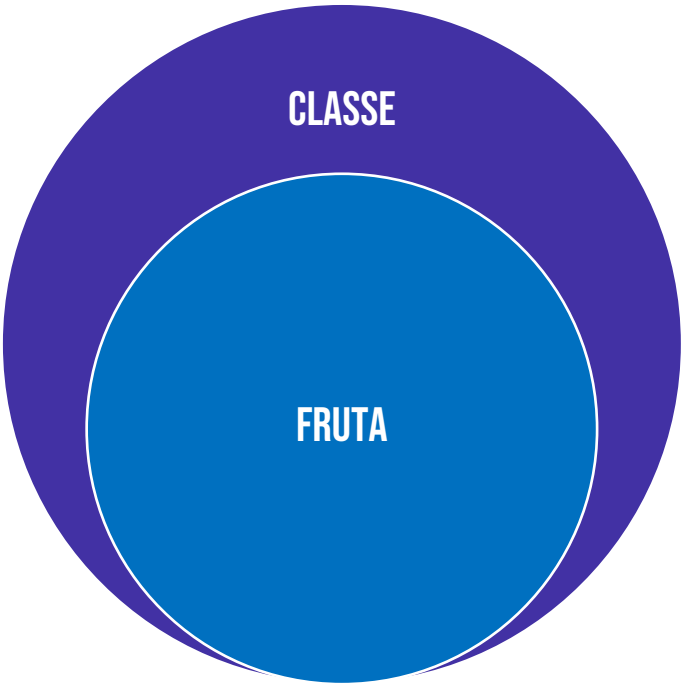
- IF: ESPECIFICA UM BLOCO DE CÓDIGO A SER EXECUTADO, SE UMA CONDIÇÃO ESPECIFICADA FOR VERDADEIRA
- ELSE: ESPECIFICA UM BLOCO DE CÓDIGO A SER EXECUTADO, SE A MESMA CONDIÇÃO FOR FALSA
- ELSE IF: ESPECIFICAR UMA NOVA CONDIÇÃO A SER TESTADA, SE A PRIMEIRA CONDIÇÃO FOR FALSA
- SWITCH: ESPECIFICA MUITOS BLOCOS ALTERNATIVOS DE CÓDIGO A SEREM EXECUTADOS

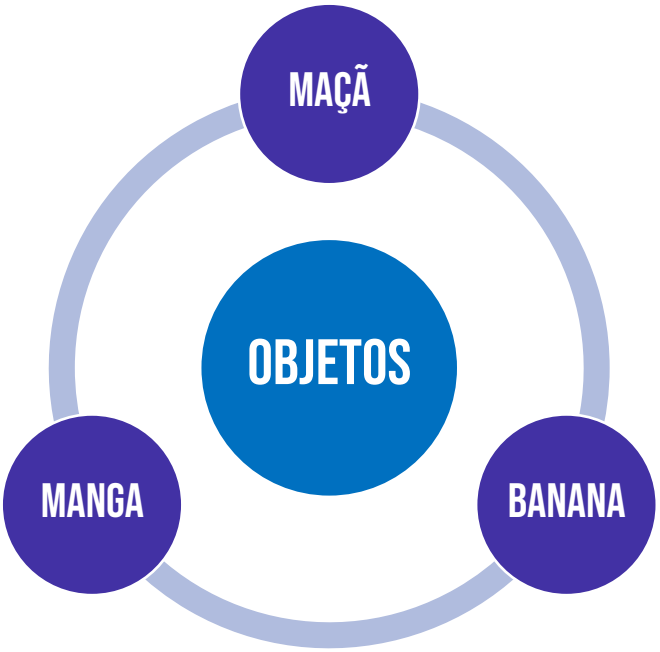
NÃO ESQUEÇA DE AUMENTAR A VARIÁVEL USADA NA CONDIÇÃO, CASO CONTRÁRIO O LOOP NUNCA TERMINARÁ!

Classes e Objetos

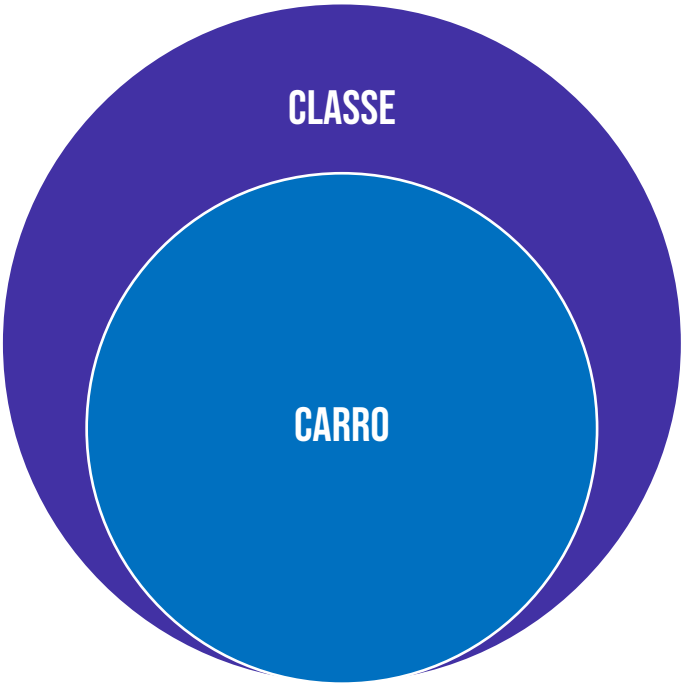
A programação orientada a objetos tem várias vantagens sobre a programação procedural 😉

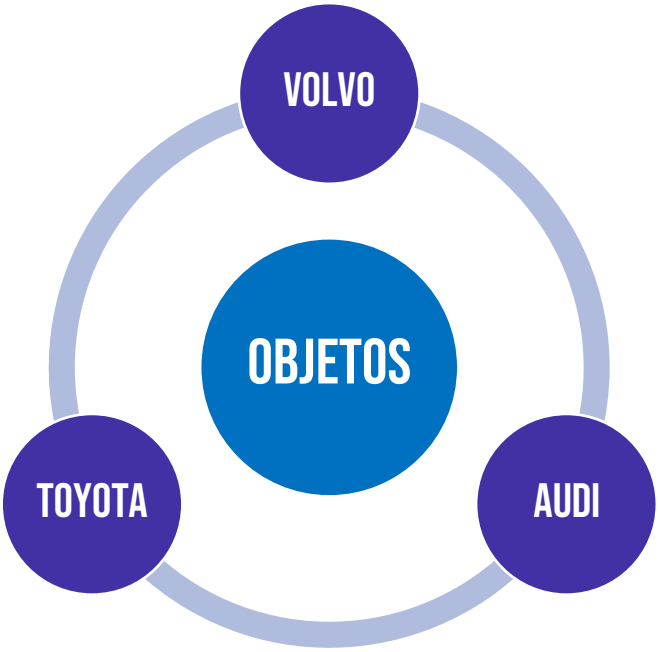
- POO é **mais rápido e fácil de executar**
- POO fornece uma **estrutura clara para os programas**
- POO ajuda a manter o código Java **DRY "Don't Repeat Yourself"** e torna o código **mais fácil de manter, modificar e depurar**
- POO possibilita a criação **de aplicativos reutilizáveis,** completos com menos código e menor tempo de desenvolvimento

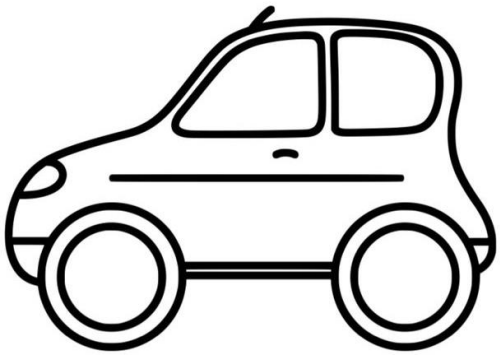




Tudo em Java está associado a classes e objetos, juntamente com seus atributos e métodos. Por exemplo: na vida real, um carro é um objeto. O carro tem atributos, como peso e cor, e métodos, como tração e freio. **Uma classe é como um construtor de objetos**, ou um "projeto" para criar objetos.



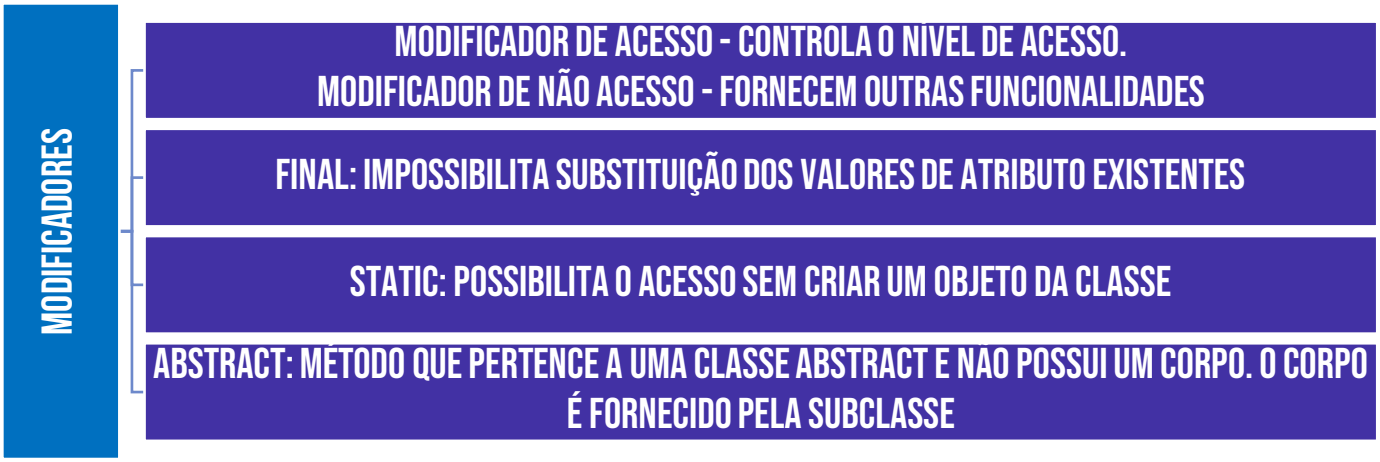




Pessoal, uma classe é uma "coisa" mais generalista, por exemplo, fruta, carro... Já, o objeto consiste na "especificação" daquela "coisa". Como é o caso do carro Toyota, por exemplo. Quando você pensa em uma classe carro, o que vem a cabeça?

Quatro rodas, porta, volante... o que o carro geralmente tem, certo?

Quando pensamos no objeto "Carro Toyota" há uma visão mais elaborada, certo? Eu por exemplo já imagino um **Corolla Cross** 😍. Da mesma forma que a classe Carro possui quatro rodas, uma cor, uma marca, um modelo, o objeto especifica esses atributos. Assim, uma classe é um modelo para objetos e um objeto é uma instância de uma classe. Quando os objetos individuais são criados, eles herdam todas as variáveis e métodos da classe.



NOME	CLASSE	SUBCLASSE	PACOTE	GLOBAL
PRIVADO	Sim	Não	Não	Não
PROTEGIDO	Sim	Sim	Não	Não
PÚBLICO	Sim	Sim	Sim	Sim
PACOTE	Sim	Não	Sim	Não

Outros modificadores: Para classes, você pode usar final ou abstract:

MODIFICADOR	DESCRIÇÃO
FINAL	A classe não pode ser herdada por outras classes
ABSTRACT	A classe não pode ser usada para criar objetos. Para acessar uma classe abstrata, ela deve ser herdada de outra classe.

Para atributos e métodos, você pode usar um dos seguintes:

MODIFICADOR	DESCRIÇÃO
FINAL	Atributos e métodos finais não podem ser substituídos/modificados
STATIC	Atributos e métodos estáticos pertencem à classe, em vez de um objeto
ABSTRACT	Só pode ser usado em uma classe abstrata e só pode ser usado em métodos. O método não tem um corpo, por exemplo abstract void run();. O corpo é fornecido pela subclasse (herdado da subclasse).
TRANSIENT	Atributos e métodos são ignorados ao serializar o objeto que os contém
SYNCHRONIZED	Métodos synchronized só podem ser acessados por um thread por vez
VOLATILE	O valor de um atributo não é armazenado em cache localmente e é sempre lido na "memória principal"

ENCAPSULAMENTO E OCULTAMENTO DE INFORMAÇÕES

CLASSES (E SEUS OBJETOS) ENCAPSULAM, ISTO É, CONTÊM SEUS ATRIBUTOS E MÉTODOS

OS OBJETOS NÃO SABEM COMO OUTROS OBJETOS SÃO IMPLEMENTADOS

OS DETALHES DE IMPLEMENTAÇÃO PERMANECEM OCULTOS DENTRO DOS PRÓPRIOS OBJETOS

INTERFACES

TODOS OS MÉTODOS DA INTERFACE SÃO IMPLICITAMENTE MÉTODOS PUBLIC ABSTRACT

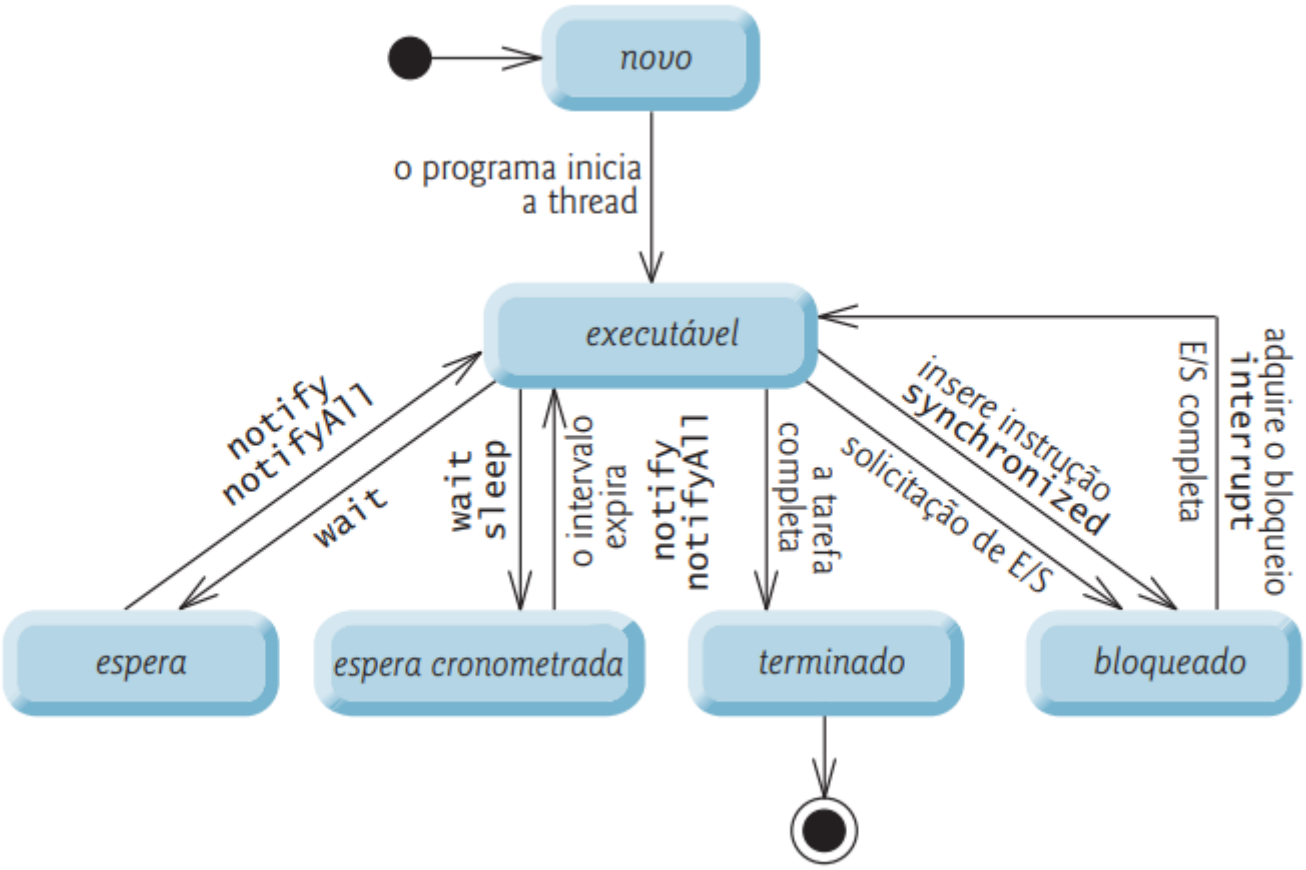
TODOS OS CAMPOS SÃO IMPLICITAMENTE PUBLIC, STATIC E FINAL.

POLIMORFISMO

A MESMA MENSAGEM ENVIADA A UMA VARIEDADE DE OBJETOS TEM MUITAS FORMAS DE RESULTADOS.

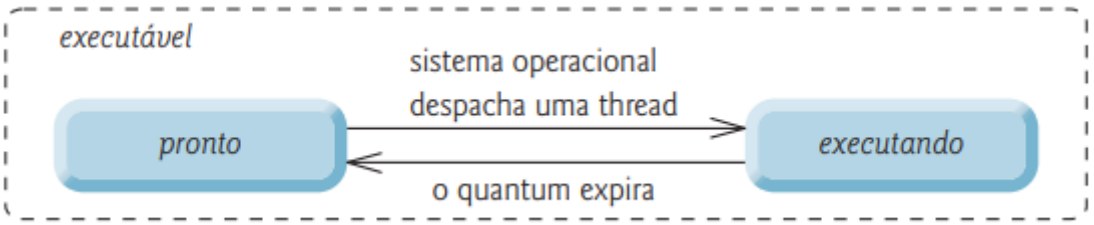
Sincronismo e Multithreading

A qualquer momento, diz-se que uma thread está em um de vários estados de thread



ESTADO	DESCRIÇÃO
EXECUTÁVEL	Uma nova thread inicia seu ciclo de vida no estado novo. Ela permanece nesse estado até que o programa inicie a thread, o que a coloca no estado executável. Considera-se que uma thread no estado executável está executando sua tarefa.
ESPERA	Às vezes a thread executável transita para o estado de espera enquanto aguarda outra thread realizar uma tarefa. Uma thread em espera volta ao estado executável somente quando outra thread a notifica para continuar a execução.
ESPERA SINCRONIZADA	Uma thread executável pode entrar no estado de espera sincronizada por um intervalo especificado de tempo. Ela faz a transição de volta ao estado executável quando esse intervalo de tempo expira ou o evento que ela espera ocorre. Threads de espera sincronizada não podem utilizar um processador, mesmo se houver um disponível. Uma thread executável pode transitar para o estado de espera sincronizada se fornecer um intervalo de espera opcional quando ela estiver esperando outra thread realizar uma tarefa. Essa thread retornará ao estado executável quando ela for notificada por outra thread ou quando o intervalo sincronizado expirar — o que ocorrer primeiro. Outra maneira de posicionar uma thread no estado em espera sincronizada é colocar uma thread executável para dormir — uma thread adormecida permanece no estado de espera sincronizada por um determinado período de tempo (chamado intervalo de adormecimento), depois do qual ela retorna ao estado executável. As threads dormem quando, por um breve período, não têm de realizar nenhuma tarefa. Por exemplo, um processador de texto

	pode conter uma thread que grave periodicamente backups (isto é, grava uma cópia) do documento atual no disco para fins de recuperação. Se a thread não dormisse entre os sucessivos backups, seria necessário um loop em que testaria continuamente se deve ou não gravar uma cópia do documento em disco. Esse loop consumiria tempo de processador sem realizar trabalho produtivo, reduzindo assim o desempenho do sistema. Nesse caso, é mais eficiente para a thread especificar um intervalo de adormecimento (igual ao período entre backups sucessivos) e entrar no estado de espera sincronizada. Essa thread é retornada ao estado executável quando seu intervalo de adormecimento expira, ponto em que ela grava uma cópia do documento no disco e entra novamente no estado de espera sincronizada.
BLOQUEADO	Uma thread executável passa para o estado bloqueado quando tenta realizar uma tarefa que não pode ser concluída imediatamente e deve esperar temporariamente até que a tarefa seja concluída. Por exemplo, quando uma thread emite uma solicitação de entrada/saída, o sistema operacional só permite que ela seja executada depois que a solicitação de E/S estiver concluída — nesse ponto, a thread bloqueada faz uma transição para o estado executável, assim pode continuar a execução. Uma thread bloqueada não pode utilizar um processador, mesmo se algum estiver disponível.
TERMINADO	Uma thread executável entra no estado terminado (às vezes chamado estado morto) quando ela conclui com sucesso suas tarefas ou é de outro modo terminada (talvez em razão de um erro).



Coleções

A Java API fornece várias estruturas de dados predefinidas, chamadas coleções, usadas para **armazenar grupos de objetos relacionados na memória**. Essas classes fornecem métodos eficientes que **organizam, armazenam e recuperam seus dados** sem a necessidade de conhecer como os dados são armazenados.

Arrays não mudam automaticamente o tamanho em tempo de execução para acomodar elementos adicionais. A classe de coleção ArrayList<T> (pacote java.util) fornece uma solução conveniente para esse problema — ela pode alterar dinamicamente seu tamanho para acomodar mais elementos.

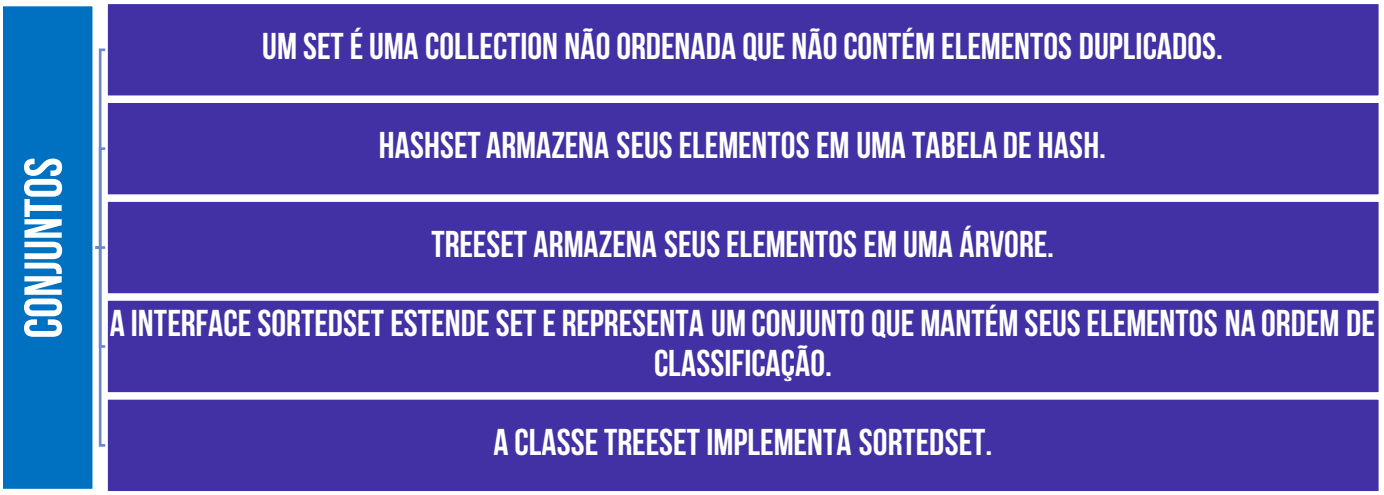
MÉTODOS ARRAYLIST	DESCRIÇÃO
ADD	Adiciona um elemento ao final do ArrayList.
CLEAR	Remove todos os elementos do ArrayList.
CONTAINS	Retorna true se o ArrayList contém o elemento especificado; caso contrário, retorna false.
GET	Retorna o elemento no índice especificado.
INDEXOF	Retorna o índice da primeira ocorrência do elemento especificado no ArrayList.
REMOVE	Sobrecarregado. Remove a primeira ocorrência do valor especificado ou o elemento no índice especificado.
SIZE	Retorna o número de elementos armazenados em ArrayList.
TRIMTOSIZE	Corta a capacidade do ArrayList para o número atual de elementos.

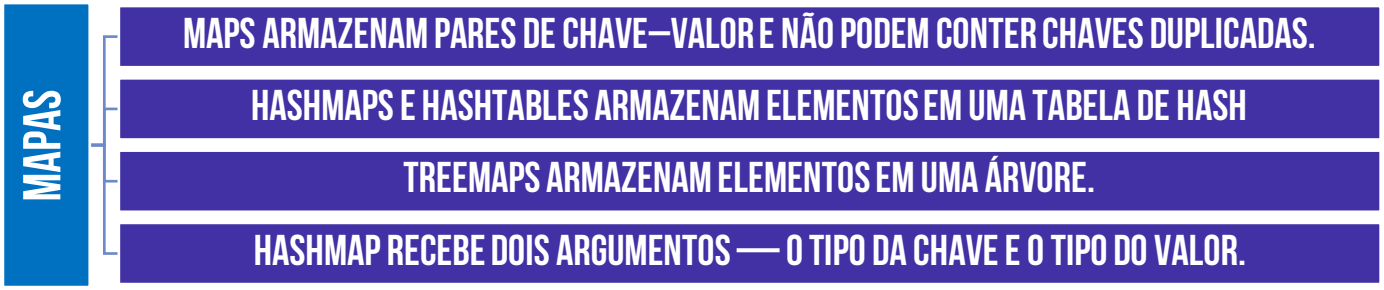
A classe **LinkedList** é quase idêntica à **ArrayList**. A classe **LinkedList** é uma coleção que pode conter muitos objetos do mesmo tipo, assim como o **ArrayList**. A classe **LinkedList** tem todos os mesmos métodos que a classe **ArrayList** porque ambos implementam a interface **List**. Isso significa que você pode **adicionar itens, alterar itens, remover itens e limpar a lista da mesma maneira**.

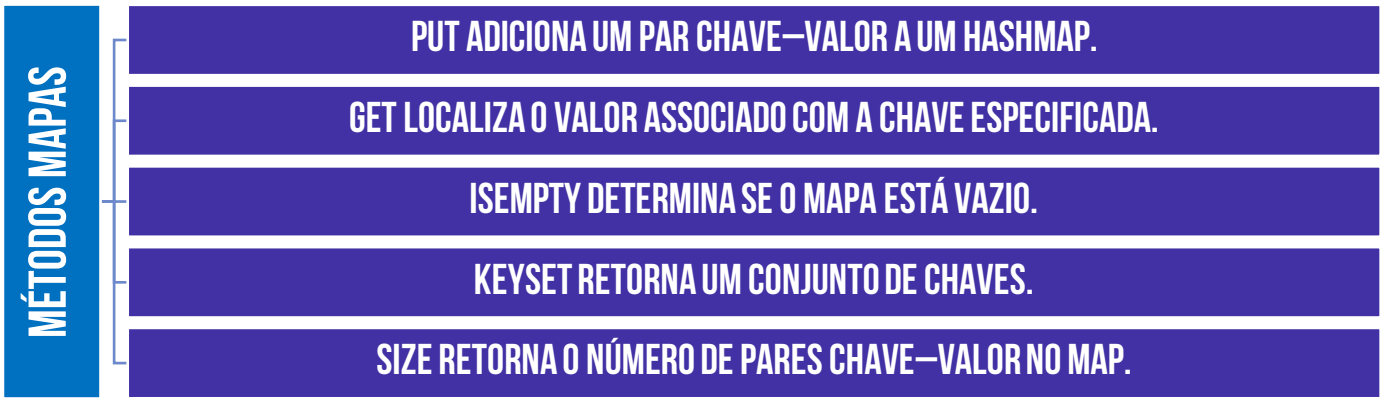
A **LinkedList** armazena seus itens em "contêineres". A lista tem um link para o primeiro contêiner e cada contêiner tem um link para o próximo contêiner na lista. Para adicionar um elemento à lista, o elemento é colocado em um novo contêiner e esse contêiner é vinculado a um dos outros contêineres da lista.

MÉTODOS LINKEDLIST	DESCRIÇÃO
ADDFIRST	Adiciona um item ao início da lista.
ADDLAST	Adiciona um item ao final da lista.
REMOVEFIRST	Remove um item do início da lista.
REMOVELAST	Remove um item do final da lista.
GETFIRST	Obtém o item no início da lista.
GETLAST	Obtém o item no final da lista.

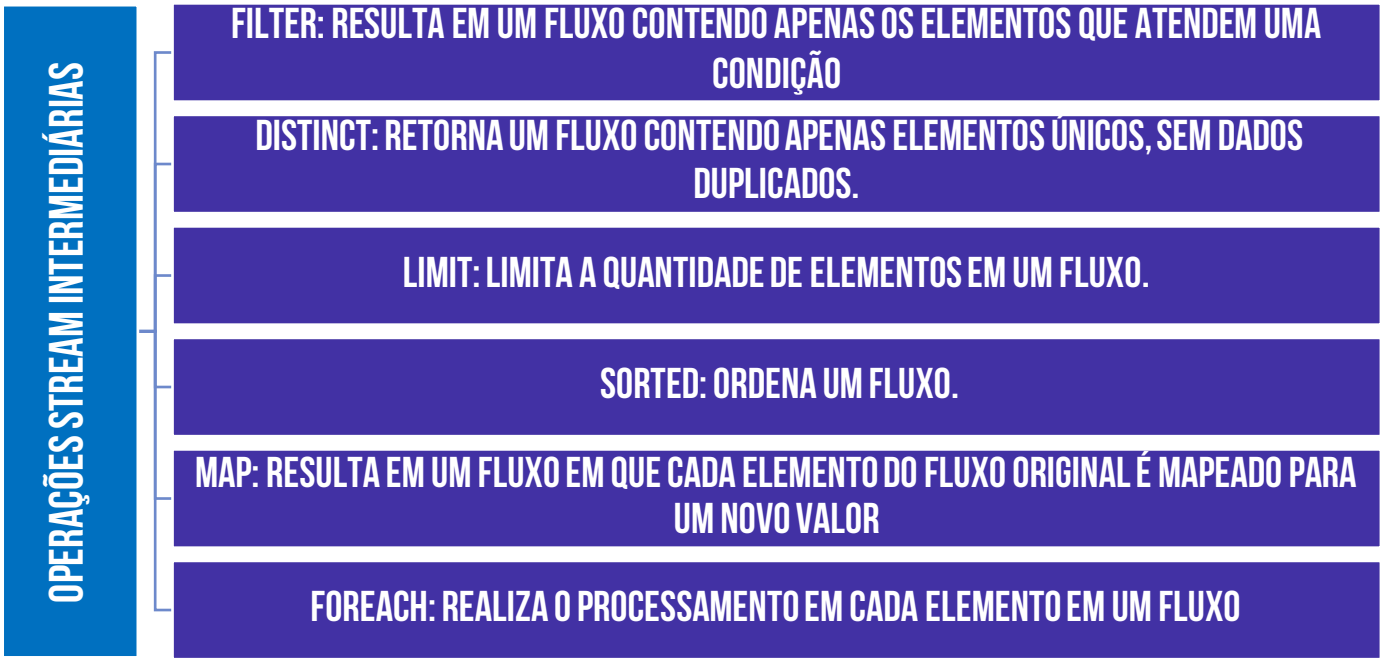
MÉTODOS	DESCRIÇÃO
CLEAR()	Remove todos os pares chave/valor do mapa;
CONTAINSKEY(K)	Retorna true se o mapa invocador contiver o objeto k como chave;
CONTAINSVALUE(V)	Retorna true se o mapa contiver o objeto v como chave;
ENTRYSET()	Retorna um conjunto que contenha as entradas no mapa;
EQUALS()	Retorna true se mapas contiverem as mesmas entradas;
GET()	Retorna o valor associado com a chave k;
REMOVE(K)	Remove a entrada que tiver chave igual a k;







Streams e Serialização



Code Conventions

TIPO	DESCRIÇÃO
NOMES E IDENTIFICADORES DE CLASSE	Os nomes de classes iniciam com uma letra maiúscula e apresentam a letra inicial de cada palavra que eles incluem em maiúscula (por exemplo, SampleClassName). Essa convenção de nomenclatura é conhecida como notação camelo, porque as letras maiúsculas destacam-se como corcovas desse animal.
COMENTÁRIOS INICIAIS	Todo arquivo de código deve iniciar com comentários de início, que devem conter informações a respeito da classe/interface, como por exemplo, nome da classe, informações de versão, data e autor
DECLARAÇÕES DE PACOTES E IMPORTAÇÕES	A primeira linha de código após os comentários iniciais deve ser a declaração do pacote (se necessário), e em sequência as declarações de importações
DECLARAÇÃO DA CLASSE/INTERFACE	A ordem em que elas devem aparecer é: Comentário de documentação, Declaração da classe/interface, Comentários de implementação da classe, Atributos estáticos, Demais atributos, Construtores, Demais métodos.
INDENTAÇÃO OU RECUO DE CÓDIGO	Os recuos são utilizados para alinhar visualmente comandos pertencentes a blocos de código. Um bloco é o código envolvido pelos delimitadores { e }, como por exemplo o if. A regra geral é abrir o bloco na linha do comando e fechar alinhado a ele.
QUEBRAS DE LINHA	Quando uma expressão não couber numa única linha, quebrá-la de acordo com as seguintes regras básicas: Após uma vírgula, Antes de um operador, Alinhar a nova linha com o início da expressão da linha anterior, Se as regras acima gerarem código confuso, ou se a linha de baixo ficar colada na margem, use uma tabulação de 8 espaços.
COMENTÁRIOS DE IMPLEMENTAÇÃO	Há quatro possíveis formatos de comentário de implementação: bloco, linha, à direita e no fim da linha. Os comentários de bloco são utilizados para descrever arquivos de código, métodos ou algum algoritmo dentro de um método. Procure

	sempre inserir uma linha em branco antes do bloco de comentários. Os comentários de linha são para comentários curtos, e devem ser precedidos com uma linha em branco, assim como o comentário de bloco. Comentários muito curtos podem aparecer na mesma linha do código que descrevem, mas devem ser deslocados o suficiente para separá-los das declarações. O delimitador de comentário // pode servir para comentar toda uma linha ou apenas parte dela.
DECLARAÇÕES	Ao declarar variáveis, observe as seguintes regras: Faça apenas uma declaração por linha, de modo a incentivar o uso de comentários, Inicialize apenas uma variável por linha, Use letras minúsculas e evite caracteres especiais.
PACOTES	Letras minúsculas sem caracteres especiais
CLASSES E INTERFACES	Os nomes de classe devem ser substantivos, com a primeira letra de cada palavra interna em maiúscula. Tente manter seus nomes de classe simples e descritivos. Evite siglas e abreviações (a menos que a sigla seja muito mais usada do que a forma longa, como a URL ou HTML) .
MÉTODOS	Métodos devem ser verbos, em maiúsculas e minúsculas com a primeira letra minúscula, com a primeira letra de cada palavra interna em maiúscula.
VARIÁVEIS	Devem iniciar com minúscula. Palavras internas começam com letras maiúsculas. Não deve começar com underline ou $, mesmo que ambos sejam permitidos. Use nomes curtos, mas significativos. O nome deve indicar a intenção da utilização da variável. Os nomes comuns para variáveis temporárias são i, j, k, m, n e para inteiros, c, d, e e para caracteres.
CONSTANTES	Constantes devem ter todas as letras maiúsculas separadas por underline

Java 17

O Java 17 oferece **milhares de atualizações de desempenho, estabilidade e segurança**, bem como **14 JEPs (JDK Enhancement Proposals)** que melhoram ainda mais a linguagem e a plataforma Java para ajudar os desenvolvedores a serem mais produtivos.

JEP	DESCRIÇÃO
JEP 409: CLASSES SELADAS	Classes e interfaces seladas restringem outras classes ou interfaces que podem estendê-las ou implementá-las. Esse aprimoramento é mais um aprimoramento do Projeto Amber, que visa aumentar a produtividade do desenvolvedor por meio da evolução da linguagem Java.
JEP 306: RESTAURE A SEMÂNTICA DE PONTO FLUTUANTE ALWAYS-STRICT	A linguagem de programação Java e a máquina virtual Java, originalmente tinham apenas semântica de ponto flutuante estrita. A partir do Java 1.2, pequenas variações nessas semânticas estritas foram permitidas por padrão para acomodar as limitações das arquiteturas de hardware atuais. Essas variações não são mais úteis ou necessárias, por isso foram removidas pelo JEP 306.
JEP 356: GERADOR DE NÚMERO PSEUDO-ALEATÓRIO APRIMORADO	Fornece novos tipos de interface e implementações para geradores de número pseudo-aleatório (PRNGs). Essa mudança melhora a interoperabilidade de diferentes PRNGs e torna mais fácil solicitar um algoritmo com base em requisitos, em vez de codificar uma implementação específica.

JEP 382: NOVO PIPELINE DE RENDERIZAÇÃO DO MACOS	Implementa um pipeline Java 2D para macOS usando a API Apple Metal. O novo pipeline reduzirá a dependência do JDK na API Apple OpenGL obsoleta.
JEP 391: MACOS AARCH64 PORT	Portas do JDK para a plataforma macOS / AArch64. Esta porta permitirá que os aplicativos Java sejam executados nativamente nos novos computadores Apple Silicon baseados no Arm 64.
JEP 398: DESCONTINUAR A API DO MINIAPLICATIVO PARA REMOÇÃO	Todos os fornecedores de navegadores da web removeram o suporte para plug-ins de navegador Java ou anunciaram planos para iniciarem a solução. A API Applet foi descontinuada, mas não para remoção, no Java 9 em setembro de 2017.
JEP 407: REMOVE RMI ACTIVATION	Remove o mecanismo de ativação de Remote Method Invocation (RMI), preservando o resto do RMI.
JEP 410: REMOVA O AOT EXPERIMENTAL E O COMPILADOR JIT	O compilador experimental baseado em Java (AOT) e just-in-time (JIT) foram recursos experimentais que não tiveram muita adoção. Por serem opcionais, eles já foram removidos do JDK 16. Este JEP remove esses componentes do código-fonte do JDK.
JEP 411: OBSOLETA O SECURITY MANAGER PARA REMOÇÃO	O Security Manager remonta ao Java 1.0. Não foi o principal meio de proteger o código Java do lado do cliente por muitos anos e, raramente, foi usado para proteger o código do lado do servidor. Removê-lo em uma versão futura ajudará a eliminar uma carga de manutenção significativa e permitirá que a plataforma Java avance.
JEP 403: ENCAPSULAR FORTEMENTE JDK	Não será mais possível relaxar o forte encapsulamento de elementos internos por meio de uma única opção de linha de comando, como era possível no JDK 9 ao JDK 16. Ainda será possível acessar os APIs internos existentes, mas agora exigirá enumerar, como parâmetros de linha de comando ou atributos de manifesto de arquivo JAR, cada pacote para o qual o encapsulamento deve ser relaxado. Essa mudança levará a aplicativos mais seguros e menos dependências de detalhes de implementação JDK internos que não são padrão.
JEP 406: CORRESPONDÊNCIA DE PADRÕES PARA SWITCH (VISUALIZAÇÃO)	Permite que uma expressão seja testada em vários padrões, cada um com uma ação específica, de forma que consultas complexas orientadas a dados, possam ser expressas de forma concisa e segura.
JEP 412: FUNÇÃO EXTERNA E API DE MEMÓRIA (INCUBADORA)	Melhora as APIs de incubação introduzidas no JDK 14 e JDK 15 que permitem que, programas Java interoperem com código e dados fora do tempo de execução Java. Invocando com eficiência funções externas (ou seja, código fora da JVM) e acessando com segurança a memória externa, essas APIs permitem que os programas Java chamem bibliotecas nativas e processem dados nativos sem a fragilidade e complexidade da Java Native Interface (JNI). Essas APIs estão sendo desenvolvidas no Projeto Panamá, que visa melhorar a interação entre código Java e não Java.
JEP 414: API VECTOR (SEGUNDA INCUBADORA)	Permite expressar cálculos vetoriais que compilam de forma confiável em tempo de execução para instruções vetoriais otimizadas em arquiteturas de CPU suportadas, alcançando, assim, um desempenho superior aos cálculos escalares equivalentes.

QUESTÕES COMENTADAS – FGV

1. **(FGV – CGU – 2022)** Observe o trecho de código a seguir.

```
1 import java.io.*;
2
3 interface Segunda { public void segunda();}
4 interface Terca { public void terca();}
5 interface Quarta extends Segunda, Terca { public void quarta();}
6
7 class Quinta implements Segunda {
8 public void segunda(){System.out.println("Segunda");}
9 }
10
11 class Sexta extends Quinta implements Quarta {
12 public void segunda(){System.out.println("Segunda!");}
13 public void terca(){System.out.println("Segunda!!");}
14 public void quarta(){System.out.println("Segunda!!!");}
15 }
16
17 public class teste
18 {
19 public static void main( String[] args )
20 {
21 Quinta dia = (Quinta) new Sexta();
22 dia.segunda();
23 }
24 }
```

A saída produzida pela execução do código é:

a) Erro de compilação na linha 5
b) Segunda
c) Segunda!
d) Segunda!!
e) Segunda!!!

Comentários:

Primeiramente, temos que a assertiva a está incorreta, já que a linha 5 não gera erro, pois entre interfaces pode haver "herança múltipla", ou seja, herança de duas ou mais classes. Agora vamos

analisar o código. No código há três interfaces, Segunda, Terca e Quarta além das classes Sexta e Quinta. Sendo que a classe Quinta é classe pai, ou superclasse, e a classe Sexta é classe filha, ou subclasse. O foco da questão está na chamada da linha 21 Quinta dia = (Quinta) new Sexta(); nessa linha, temos que o objeto dia chama o método construtor Sexta, que está na classe Sexta, que é filha de Quinta. Quando esse objeto for criado, ele vai à classe Sexta e procura pelo método segunda, se o encontrar ali, executa o que estiver no corpo do método, senão, ele vai à classe Quinta a procura de um método segunda. Daí, quando a linha dia.segunda(); for executada, o objeto dia procura um método segunda. Ele vai à classe Sexta, pois esse é o método construtor que foi passado na criação do objeto, após isso, caso não encontre o método segunda na classe Sexta, ele vai à classe Quinta, que deve ter um método segunda presente em seu corpo. Assim, o resultado que será apresentado na tela é Segunda! E temos nosso gabarito na alternativa C.

Gabarito: Letra C

---

2. (FGV – TJDFT – 2022) Observe as classes abaixo descritas na linguagem de programação Java.

```
public class DocumentoProcesso extends Object
{
   private String docNumero;
   private int classificacao;


   public DocumentoProcesso (String docNumero, int

classificacao){
    if (classificacao < 1)
      throw new IllegalArgumentException (
        "A classificação do documento deve ser no mínimo 1.");

     this.docNumero = docNumero;
     this.classificacao = classificacao;
    }
    public String getDocNumero()
    { return docNumero; }
    public int getClassificacao()
    { return classificacao; }
    public void setDocNumero(String docNumero)
    { this.docNumero = docNumero; }
    public void setClassificacao(int classificacao)
    { this.classificacao = classificacao; }
    public int promove()
    { return classificacao + 1; }
}
```

```
public class Oficio extends DocumentoProcesso
{

   private int precedencia;
   public Oficio (String docNumero, int classificacao, int precedencia)
   {super(docNumero,classificacao);
   this.precedencia = precedencia; }
public int getPrecedencia()
{ return precedencia; }
public void setPrecedencia(int precedencia)
{ this.precedencia = precedencia; }
@Override
public int promove()
{ return precedencia + 1; }
}
```

Com base nos conceitos de orientação a objetos, é correto afirmar que:
a) os atributos private de DocumentoProcesso são acessíveis por Oficio;
b) a anotação @Override indica que o método promove() é abstrato e polimórfico;
c) a classe Oficio representa uma herança múltipla das classes DocumentoProcesso e Object;
d) a classe Oficio é composta pela classe DocumentoProcesso, permitindo que uma instância da classe Oficio faça referências a instâncias da classe DocumentoProcesso;
e) os métodos getDocNumero() e setDocNumero() da classe DocumentoProcesso encapsulam o atributo docNumero e asseguram que os objetos da classe mantenham estados consistentes.

Comentários:

Vamos analisar as assertivas: a letra a diz que os atributos private de DocumentoProcesso são acessíveis por Oficio. Vejamos a classe Oficio: a declaração dela é a seguinte: public class Oficio extends DocumentoProcesso. Portanto, ofício é filha da classe DocumentoProcesso. Já que os atributos possuem o modificador private, eles não podem ser compartilhados entre as classes. Lembrem-se da definição: Membros da classe definidos como private não podem ser acessados ou usados por nenhuma outra classe. Assim, temos uma assertiva errada. Letra B: a anotação @Override indica que o método promove() é abstrato e polimórfico. Errada assertiva B porque @Override indica que um método da classe-pai será sobrescrito por um método da classe-filha, ou seja, a classe filha modifica o método que foi declarado na classe pai. Para indica que o método promove() é abstrato deveria ser utilizado abstract. A letra C também está errada, pois na verdade, é a classe DocumentoProcesso que faz referência à Oficio. Por fim, temos o nosso gabarito na letra E. Pessoal, os métodos getDocNumero() e setDocNumero() de fato, são utilizados para encapsular o atributo docNumero. Ademais, asseguram que os objetos da classe mantenham estados consistentes. Getters e setters são usados para proteger seus dados, especialmente na criação de

classes. Para cada instância de variável, um método getter retorna seu valor, enquanto um método setter o define ou atualiza seu valor.

Gabarito: Letra E

---

3. **(FGV – PGE-AM – 2022)** Considere o trecho de programa Java abaixo.

```
public class Calcular {
    public double reajuste(double salario, double percentual) {
        return salario + salario * percentual / 100;
    }
    public double reajuste(double salario) {
        return salario * 1.30;
    }
    public static double reajuste(float salario) {
        return salario * 0.20;
    }
}
```

É correto afirmar que

a) o código mostra um exemplo de sobrescrita de métodos, pois há vários métodos com o mesmo nome, porém, com parâmetros diferentes.
b) ocorrerá um erro no último método reajuste, pois este não pode ser estático.
c) uma chamada correta ao método reajuste, em condições ideais, é double sal = 1000.5; double c = Calcular.reajuste(sal);
d) ocorrerá um erro na classe Calcular, pois não é permitido criar vários métodos com o mesmo nome, como ocorre com o método reajuste.
e) uma chamada correta ao método reajuste, em condições ideais, é float sal = (float) 1000.5; double c = Calcular.reajuste(sal);

Comentários:

Sobrecarga ou Polimorfismo Estático ocorre quando uma classe possui métodos com mesmo nome, entretanto assinaturas diferentes. Ocorre em Tempo de Compilação. Veja que o método reajuste possui diferentes implementações. Com essa explicação inicial podemos verificar que a letra A está incorreta, pois não ocorre sobrescrita, mas sim, sobrecarga. O erro da letra B consiste em dizer que método reajuste não pode ser estático. O erro da letra C consiste na falta do "double" indicado double sal = (double) 1000.5; double c = Calcular.reajuste(sal). O erro da alternativa d é dizer que não é permitido criar vários métodos com o mesmo nome. Na verdade, não vai ocorrer um erro. Ocorre uma sobrescrita que é possível em Java! Por fim, temos nosso gabarito Letra E!

Gabarito: Letra E

---

4. **(FGV – DPE-RJ – 2019)** Considere as seguintes afirmativas sobre class constructor na linguagem Java.

I. Deve receber o mesmo nome da classe a ele associada.

II. Não deve ser especificado um tipo de retorno na sua declaração.

III. É útil para a definição de valores iniciais para os atributos da classe.

IV. É sempre declarado como public.

É correto somente o que se afirma em:
a) I e II;
b) II e III;
c) III e IV;
d) I, II e III;
e) I, III e IV.

Comentários:

Vamos lá, um construtor é um método especial usado para inicializar objetos. O construtor é chamado quando um objeto de uma classe é criado. Ele pode ser usado para definir valores iniciais para atributos de objeto. o construtor é apenas invocado no momento da criação do objeto através do operador new. A assinatura de um construtor diferencia-se das assinaturas dos outros métodos por não ter nenhum tipo de retorno (nem mesmo void). Além disto, o nome do construtor deve ser o próprio nome da classe. O construtor pode receber argumentos, como qualquer método. Usando o mecanismo de sobrecarga, mais de um construtor pode ser definido para uma classe. Assim, temos que os itens I, II e III estão corretos. Já, a assertiva IV está errada, pois pode ser public, protected ou private.

Gabarito: Letra D

---

5. **(FGV – MPE-AL – 2018)** Analise o código Java a seguir.

```
public class Exemplo {
 public static void main(String[] args) {
  int  a = 4, b = 16;
  String out = (a^b)==o ? "Sim": (a & b)!=o
                  ? "Talvez" : "Não";
  System.out.println(out);
 }
}
```

A execução do código acima produzirá

a) um erro de compilação.
b) um erro de execução.
c) a linha "Sim".
d) a linha "Não".

e) a linha "Talvez".

Comentários:

A questão está totalmente focada no operador ternário String out = (a^b)==0 ? "Sim": (a & b)!=0. Vamos destrinchá-lo. Inicialmente temos: (a^b)==0. O operador ^é um XOR que consiste em um ou exclusivo que retorna 1 se os bits forem diferentes: Portanto, temos a = 4 e b = 16. a é igual 4 que é igual a 00100 em binário e b – 16 – é igual a 10000 em binário. Fazendo a operação XOR entre os valores temos como resultado 10100, que é igual a 20 em decimal. Podemos perceber que o valor não é sim, já que 20 não é igual a 0. Portanto vamos ao próximo argumento que é (a & b)!=0., ou seja, a e b é diferente de zero. Vejamos como fica o resultado de a & b = 00000 porque o operador "E" (&) só retorna 1 se ambos bits forem 1: a - 00100 (4) b - 10000 (16) = 00000 (0). Veja que o teste vai ficar da seguinte forma: (a & b) != 0. Sabemos que a & b é 0 (de acordo com o cálculo acima. Assim fica o != 0 ? ou seja, zero é diferente de zero? Não né! Então é retornado o valor "Não". Já que o operador ternário (TESTE)? X : Y retorna X, se TESTE for verdadeiro, e Y se teste for falso.

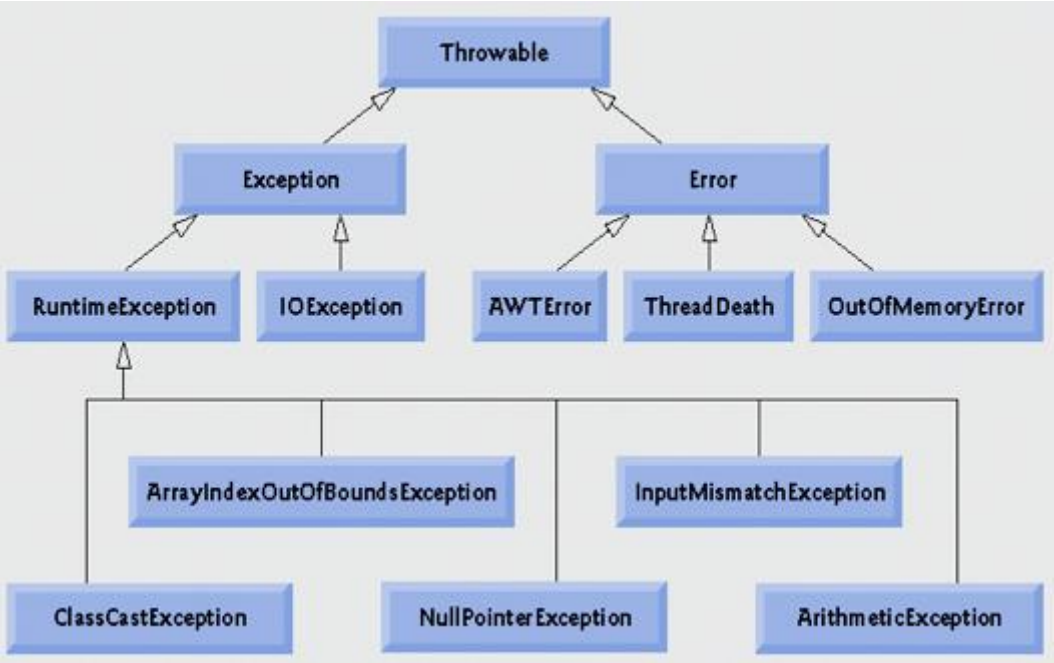
Gabarito: Letra D

---

6. **(FGV – MPE – AL – 2018)** No Java, a classe Error e Exception derivam da classe
   a) ClassNotFoundException.
   b) IOException.
   c) MainException.
   d) RuntimeException.
   e) Throwable.

Comentários:

A classe Throwable é a superclasse de todos os erros e exceções na linguagem Java. Somente objetos que são instâncias desta classe (ou uma de suas subclasses) são lançados pela Java Virtual Machine ou podem ser lançados pela instrução throw Java. Da mesma forma, apenas esta classe ou uma de suas subclasses pode ser o tipo de argumento em uma cláusula catch. Para fins de verificação de exceções em tempo de compilação, Throwable e qualquer subclasse Throwable também não é uma subclasse de nenhuma RuntimeExceptionou Error é considerada como exceções verificadas. Instâncias de duas subclasses, Errore Exception, são convencionalmente usadas para indicar que ocorreram situações excepcionais. Normalmente, essas instâncias são recém-criadas no contexto da situação excepcional para incluir informações relevantes (como dados de rastreamento de pilha).Error e Exception são, portanto, derivadas da superclasse Throwable. Uma imagem para lembrar da hierarquia de exceções.



Gabarito: Letra E

---

7. **(FGV – MPE-AL – 2018)** Sobre as variáveis e os métodos declarados como private, em Java, analise as afirmativas a seguir.

I. Ficam acessíveis somente aos membros da própria classe.

II. Ficam acessíveis somente às classes definidas no mesmo package.

III. Ficam acessíveis somente para suas classes derivadas.

Está correto o que se afirma em

a) I, apenas.
b) II, apenas.
c) III, apenas.
d) I e II, apenas.
e) II e III, apenas.

Comentários:

Pessoal, vamos relembrar a tabela? As variáveis e os métodos private ficam acessíveis apenas aos membros da própria classe! Não ficam disponíveis no pacote e também não ficam acessíveis ás subclasses! Portanto nosso gabarito é apenas o item I.

Gabarito: Letra A

---

8. **(FGV – BANESTES – 2018)** Considere a compilação de um ou mais programas por meio da linha de comando, num ambiente Java. Nesse caso, o comando que está corretamente formado para esse fim é:

a) compile teste.java –type java
b) java teste.java
c) javac *.java
d) jvm Teste1.java teste2.java
e) parse java teste.java

Comentários:

A Linguagem Java tem dois processos de execução de código-fonte: Compilação e Interpretação! Vamos lá... o programador escreve um código em Java em um editor de texto, por exemplo. Ele salva com a extensão .java e passa por um compilador (JavaC)! O javac é utilizado para compilar as classes em Java. É necessário usá-lo antes de executar uma classe para que seja gerado um arquivo .class da classe em questão. Assim, devemos usar a seguinte sintaxe: javac *.java

Gabarito: Letra C

---

9. **(FGV – ALE-RO – 2018)** Analise a definição de um array em Java.

int[ ][ ] multi = new int[5][10];

Assinale o código que exibe corretamente um elemento de cada linha desse array.

a) for (int i = 0; i < 5; i++) { System.out.println(multi[i][0]); };
b) for (int i = 0; i < 10; i++) { System.out.println(multi[i,1]); };
c) for (int i = 0; i <= 5; i++) { System.out.println(multi[i,0]); };
d) for (int i = 0; i <= multi.length; i++) { System.out.println(multi[1,i]); };
e) for (int i = 1; i < multi[0].length; i++) { System.out.println(multi[i,1]); };

Comentários:

Bom, o array é multidimentional, possui duas dimensões, costuma-se dizer "i" e "j". A questão solicita exibir a primeira linha do array. A primeira linha é composta de 6 elementos (0,1,2,3,4,5). É necessário que o for seja de 0 até 5. Ademais, para exibir o conteúdo de cada linha usa-se o nome do array (multi) com a respectiva referência: [i][0]. Assim, temos como gabarito a letra A.

Gabarito: Letra A.

---

10. **(FGV – ALE-RO – 2018)** No contexto da linguagem Java, assinale o modificador (modifier) que se refere ao nível de acesso.

a) abstract
b) final
c) protected
d) static
e) volatile

Comentários:

É nesse momento que você pensa: podia tanto cair uma questão dessas na minha prova 😍.os modificadores são 4: Privado, Protegido, Público e Pacote. Portanto, nosso gabarito é a letra C. Vejamos as definições das demais alternativas: abstract: Aplicado a um método ou classe indica que a implementação completa deste método ou classe é efetuada posteriormente, por uma subclasse. Caso seja uma classe, significa que ela não pode ser instanciada. Final: é usada para impossibilitar que uma classe seja estendida, que um método seja sobrescrito ou que uma variável seja reinicializada. Ou seja, no caso da questão, mantém o atributo constante. Static: é usada para marcar um método ou variável para que se tenha apenas uma cópia da memória desse membro. Volatile: para que se possa acessar recursos específicos do sistema operacional, indica que uma determinada variável de instância pode ser modificada em duas threads distintas ao mesmo tempo.

Gabarito: Letra C

---

11. **(FGV – ALE-RO – 2018)** Sobre construtores (constructors), no contexto da linguagem Java, analise as afirmativas a seguir.

I. Os construtores devem ser declarados como private.

II. Uma interface não pode ter um construtor.

III. Uma classe abstrata pode ter um construtor.

Está correto o que se afirma em
a) I, somente.
b) II, somente.
c) III, somente.
d) I e II, somente.
e) II e III, somente.

Comentários:

Pessoal, o item II está perfeito! A interface só declara métodos e constantes. Ela não implementa! Uma classe abstrata pode ter um ou mais construtores. O item II também está correto: Embora não seja possível criar objetos de uma classe abstrata, em uma hierarquia de classes o construtor

de uma subclasse chama (explícita ou implicitamente) o construtor da superclasse. Como uma classe abstrata deve, necessariamente, ter subclasses, a classe abstrata precisa ter um construtor. Já, o item I está incorreto: Construtores podem ter qualquer modificador de acesso: private, <default>, protected ou public.

Gabarito: Letra E

---

12. (FGV – MPE AL – 2018) O package "java.util.regex" do Java permite a manipulação de expressões regulares.

As três classes mais importantes desse pacote são denominadas:
a) RegExp, Parser e Lexicon.
b) RegExPattern, RegExVerifier e RegExCompiler.
c) Pattern, Matcher e PatternSyntaxException.
d) Collector, Retriever e SyntaxParser.
e) Parser, Inspector e Evaluator.

Comentários:

A questão diz que Java permite a manipulação de expressões regulares, e solicita as três classes mais importantes do pacote. Vejamos o que diz a definição do W3Schools, primeiro sobre expressões regulares: Uma expressão regular é uma sequência de caracteres que forma um padrão de pesquisa. Uma expressão regular pode ser um único caractere ou um padrão mais complicado. As expressões regulares podem ser usadas para executar todos os tipos de pesquisa de texto e operações de substituição de texto . Java não tem uma classe de Expressão Regular embutida, mas podemos importar o java.util.regex pacote para trabalhar com expressões regulares. O pacote inclui as seguintes classes: PatternClasse - Define um padrão (para ser usado em uma pesquisa). MatcherClasse - Usada para procurar o padrão. PatternSyntaxExceptionClasse - Indica erro de sintaxe em um padrão de expressão regular. Portanto as três classes mais importantes são: Pattern, Matcher e PatternSyntaxException e o nosso gabarito é a letra C.

Gabarito: Letra C

---

13. (FGV – MPE – AL – 2018) Servidores de aplicação baseados em Java Platform Enterprise Edition possuem vários tipos de API.

Assinale a opção que indica a API utilizada para fornecer acesso ao servidor de nomes e diretórios.
a) JNDI.
b) RMI.
c) JMS.
d) JTS.
e) JDBC.

Comentários:

Vamos traduzir essa sopa de letrinhas? JNDI - Java Naming and Directory Interface (Nomeação Java e Interface de Diretório); RMI - Remote Method Invocation (Invocação do Método Remoto); JMS - Java Message Service (Serviço de Mensagens Java); JTS - Java Topology Suite (Suíte de Topologia Java); JDBC - Java Database Connectivity (Conexão de Banco de Dados Java). Após desvendar as siglas, podemos marcar nosso gabarito a alternativa A: acesso ao servidor de nomes e diretórios: JNDI.

Gabarito: Letra A

---

14. (FGV – IBGE – 2017) Os servidores de aplicação fornecem a infraestrutura básica necessária para o desenvolvimento e a distribuição de aplicações. Em relação aos servidores de aplicação da plataforma Java EE, analise as afirmativas a seguir:

I. GlassFish e Apache Geronimo são servidores de aplicação open source compatível com a especificação J2EE.

II. O Apache Tomcat não dispõe de um container EJB.

III. JBoss é programado em Java, portanto, pode ser usado em qualquer sistema operacional que suporte essa linguagem de programação.

Está correto o que se afirma em:

a) somente I;
b) somente II;
c) somente III;
d) somente I e II;
e) I, II e III.

Comentários:

Vamos analisar cada alternativa: I. GlassFish e Apache Geronimo são servidores de aplicação open source compatível com a especificação J2EE. O GlassFish é um servidor de aplicação open source liderado pela Sun Microsystems para a plataforma Java EE. Sua versão proprietária é chamada Sun GlassFish Enterprise Server. Já o Apache Geronimo é um conjunto de projetos de software livre que se concentra em fornecer bibliotecas JavaEE/JakartaEE e implementações de Microperfil. No entanto, estamos entregando ativamente componentes Java EE reutilizáveis. Eles são amplamente utilizados e ainda mantidos ativamente. Portanto, ambos são open source e compatível com a especificação J2EE. O item I está correto! Vamos para o item II. O Apache Tomcat não dispõe de um container EJB. De fato, o Apache Tomcat não dispõe de um container EJB. EJB é um dos principais componentes da plataforma Java EE. Eles são executados em um container EJB no lado do servidor. A principal característica do EJB é a rapidez em desenvolver

soluções, pois, ele fornece um conjunto de componentes prontos, o que torna o desenvolvimento eficiente. Vocês viram a importância do EJB, certo? Em função disso, o TomEE incorpora as tecnologias EJB, CDI e as outras features do Java EE dentro do Tomcat, ou seja, atualmente a assertiva estaria errada. Porem foi considerada correta. Item II certo! Por fim, vejamos o item III. JBoss é programado em Java, portanto, pode ser usado em qualquer sistema operacional que suporte essa linguagem de programação. Item correto, galera! Vejamos: JBoss é um servidor de aplicação de código fonte aberto baseado na plataforma JEE e implementado completamente na linguagem de programação Java. Em virtude disso, ele pode ser usado em qualquer Sistema Operacional que suporte a referida linguagem

Gabarito: Letra E

---

15. **(FGV – IBGE – 2017)** Em Java, certo método da classe A deve poder ser invocado independentemente da criação prévia de uma instância de A.

O modificador obrigatório na declaração desse método é:

a) abstract;
b) static;
c) public;
d) protected;
e) final.

Comentários:

Como já dissemos em questões anteriores, os métodos static tem um relacionamento com uma classe como um todo, enquanto os métodos que não são static são associados a uma instância de classe específica (objeto) e podem manipular as variáveis de instância do objeto. Portanto, para classe A poder ser invocado independentemente da criação prévia de uma instância de A o método deve receber o modificador static.

Gabarito: Letra B

QUESTÕES COMENTADAS – FCC

16. (FCC – PGE-AM - 2022) Considere o trecho de programa Java abaixo.

```
public class Exemplo1 {

    public static void main(String[] args) {
        int[] vet = {78, 89, 90, 34, 32, 56};
        int[] dados = verificar(vet);
    }

      I
    ......{

        int a, b, x, t, max;
        max = vet.length;
        for (a = 0; a <= max - 1; a++) {
            for (b = 0; b <= max - a; b++) {
                if (b < max - 1) {
                    if (vet[b] > vet[b + 1]) {
                        x = vet[b];
                        vet[b] = vet[b + 1];
                        vet[b + 1] = x;
                    }
                }
            }
        }
        return vet;
    }
}
```

A assinatura correta do método que deve ser colocada na lacuna I é

a) public int[] verificar(int vet[])
b) private int[] verificar(int vet[])
c) public static int verificar(int vet[])
d) public int verificar(int vet)
e) public static int[] verificar(int vet[])

Comentários: A forma correta para a assinatura do método é conforme a sintaxe: [Palavras-Chave] TipoRetorno NomeMetodo ([Lista de Parâmetros]). Assim, temos um método public, daí eliminamos a letra B. É definido como static porque os métodos static tem um relacionamento com uma classe como um todo, enquanto os métodos que não são static são associados a uma instância de classe específica (objeto) e podem manipular as variáveis de instância do objeto. Portanto eliminamos as alternativas a, b e d. O retorno é um vetor de inteiros, portanto int[], daí eliminamos as alternativas c e d. Ficamos então com a alternativa E que é o nosso gabarito: public static int[] verificar(int vet[])

Gabarito: Letra E

17. (FCC – PGE-AM – 2022) Considere st um objeto do tipo PreparedStatement e conn um objeto do tipo Connection, ambos em condições ideais para a execução do método salvar, de uma classe Java.

```
public boolean salvar(Cliente cliente){
        try {
            st = conn.prepareStatement("..... I ....");
            st.setString(1, cliente.getCpf());
            st.setString(2, cliente.getNome());
            st.setString(3, cliente.getEndereco());
            st.setString(4, cliente.getBairro());
            st.setString(5, cliente.getCidade());
            st.setString(6, cliente.getTelefone());
            st.setDouble(7, cliente.getRenda());
            st.executeUpdate();
            return true;
        } catch (SQLException ex) {
            return false;
        }
}
```

Nas condições apresentadas, para que o método salvar grave as informações do cliente no banco de dados aberto, a lacuna I deve ser corretamente preenchida com a instrução
a) INSERT INTO cliente VALUES(cpf,nome,endereco,bairro,cidade,telefone,renda)
b) INSERT INTO cliente VALUES(?,?,?,?,?,?,?)
c) INSERT INTO cliente (nome,cpf,endereco,bairro,cidade,telefone,renda) VALUES(?,?,?,?,?,?,?)
d) INSERT INTO cliente VALUES([cpf],[nome],[endereço],[bairro],[cidade],[telefone],[renda])
e) INSERT TO cliente (cpf,nome,endereco,bairro,cidade,telefone,renda) VALUES(?,?,?,?,?,?,?)

Comentários:

Pessoal, essa questão é um pouco mais avançada pois trata do do tipo PreparedStatement e conn. Um PreparedStatement é um objeto que representa uma instrução SQL pré-compilada. Uma instrução SQL é pré-compilada e armazenada em um objeto PreparedStatement. Esse objeto pode ser usado para executar com eficiência essa instrução várias vezes. Às vezes, é mais conveniente usar um objeto PreparedStatement para enviar instruções SQL ao banco de dados. Esse tipo especial de instrução é derivado da classe mais geral, Statement, que você já conhece. Se você deseja executar um objeto Statement muitas vezes, geralmente reduz o tempo de execução para usar um objeto PreparedStatement. A principal característica de um objeto PreparedStatement é que, diferentemente de um objeto Statement, ele recebe uma instrução

SQL quando é criado. A vantagem disso é que, na maioria dos casos, essa instrução SQL é enviada imediatamente ao DBMS, onde é compilada. Como resultado, o objeto PreparedStatement contém não apenas uma instrução SQL, mas uma instrução SQL que foi pré-compilada. Isso significa que quando o PreparedStatement é executado, o DBMS pode apenas executar a instrução SQL PreparedStatement sem ter que compilá-lo primeiro. A ideia é preparar as querys do SQL para armazenamento no banco de dados. O tutorial da Oracle diz que nós devemos usar a interrogação para inserir na tabela cliente (?,?,?,?,?,?,?,?). Portanto o gabarito é a letra B.

Gabarito: Letra B

18.(FCC – PGE-AM – 2022) No método doPost de uma servlet Java, deseja-se receber um parâmetro de um campo de nome renda contido em um formulário HTML, converter o valor recebido deste campo para número real e armazená-lo em uma variável chamada renda. A instrução correta para realizar esta tarefa é

a) long renda = Long.parseLong(request.getParameter("renda"));
b) double renda = parseDouble(request.getParameter("renda"));
c) double renda = request.getParameter("renda").toDouble();
d) double renda = Double.parseDouble(request.getParameter("renda"));
e) number renda = parseNumber(request.getParameter("renda"));

Comentários:

O método parseDouble() da classe Java Double é um método embutido em Java que retorna um novo double inicializado com o valor representado pela String especificada, como feito pelo método valueOf da classe Double. A sintaxe é: public static double parseDouble(String s). Parâmetros: Aceita um único parâmetro obrigatório que especifica a string a ser analisada. Tipo de retorno: Retorna um valor double representado pelo argumento string. A questão, entre parênteses, usaremos request.getParameter("renda"), de forma a pegar o valor recebido no método post. A resposta da questão é a letra D: double renda = Double.parseDouble(request.getParameter("renda"))

Gabarito: Letra D

19.(FCC – AFAP – 2019) Considere o método Java abaixo.

```
public int conectar() {
    try {
        Class.forName("com.mysql.jdbc.Driver");
        con = DriverManager.getConnection("jdbc:mysql://localhost:3306/trf3", "root", "ad3jA");
        st = con.createStatement();
        return 1;
    } catch (.....I ex) {
        return 0;
    } catch (.....II ex) {
        return 2;
    }
}
```

Caso o driver JDBC não esteja disponível na aplicação e não exista o banco de dados trf3, as exceções que deverão ser tratadas nas lacunas I e II são, correta e respectivamente,

a) NullPointerExceptionException e DatabaseException.
b) DriverNotFoundException e MySQLException.
c) JDBCDriverException e DatabaseException.
d) ClassNotFoundException e SQLException.
e) JDBCDriverException e DatabaseSQLException.

Comentários:

Pessoal, caso o driver JDBC não esteja disponível na aplicação, irá acusar um erro chamado ClassNotFoundException. Os comandos não irão conseguir encontrar a classe informada, pois não há o driver disponível. Por fim, como não existe o BD, o JDBC informa que é um erro chamado SQLException, ele acusa algum erro no banco de dados, como a questão nos informou que ele não existe não precisamos ficar procurando esse erro na aplicação.

Gabarito: Letra D

---

20. (FCC – AFAP – 2019) Para criar array um Analista de Informática digitou os comandos abaixo no método main de uma classe Java em condições ideais.

I. int[] a = {1,3};
II. int b[] = new int[2]; b[0]=1; b[1]=3;
III. int[] c = new int[2]; c.add(1); c.add(2);
IV. int[][] d = {{1,3},{4,5}};
V. int e[][] = {{1,3},{4,5}};

Ao tentar compilar e executar a classe, foi exibida a mensagem "Exception in thread "main" java.lang.RuntimeException: Uncompilable source code". Essa mensagem foi exibida porque o item

a) IV está incorreto, pois não especifica o tamanho da array.
b) V está incorreto, pois deve haver apenas um par de colchetes após o nome da array.
c) I está incorreto, pois não é permitido declarar um array e já incluir os valores nele.
d) II está incorreto, pois os colchetes devem ficar antes do nome da variável b.
e) III está incorreto, pois não existe o método add neste contexto.

Comentários:

Pessoal, a resposta correta é a letra E. Não existe o método add para array no Java, mas sim no ArrayList. Ademais, a letra a erra porque não é necessário especificar o tamanho. O erro da letra B está em dizer que deve haver apenas um par de colchetes. E a letra C está errada pois é possível sim declarar e incluir valores no array. Por fim, os colchetes podem ficar antes ou depois da variável.

Gabarito: Letra E

---

21. (FCC – AFAP – 2019) Considere a classe Java abaixo, que usa um método de ordenação por inserção para rearranjar os valores em um vetor em ordem crescente.

```
public class Organiza{
   public static void main(String[] args) {
      int k[] = {88, 44, 42, 7, 72, 5};
      int n = k.length;
      ordena(n, k);
   }
   static void ordena(int n, int v[]) {
      for (int j = 1; j < n; ++j) {
         int x = v[j];
         int i;
         for (..I..){
            v[i + 1] = v[i];
         }
         ..II.. ;
      }
   }
}
```

Para que a ordenação por inserção ocorra corretamente, as lacunas I e II devem ser corretamente preenchidas com

a) i=j-1; i >=0 && v[i]>x; – –i e v[i-1]=x
b) i=j-1; i >=0; i++ e v[i+1]=x
c) i=j-1; i >=0 && v[i]>x; – –i e v[i+1]=x
d) i=j-1; i >=0 && v[i]>x; ++i e v[i]=x
e) i=j-1; i >=0 || v[i]>x; – –i e v[i+1]=x

Comentários:

Pessoal, essa é uma questão de lógica. A resposta correta é a letra c)i=j-1; i >=0 && v[i]>x; – –i e v[i+1]=x. No caso, existe uma comparação de valores um a um dado a condição "i >=0 && v[i]>x" e o decremento de "i".Durante o loop, o programa percorre o vetor comparando o valor contido em x (v[j]) com o valor da próxima posição do vetor, realizando as trocas entre os elementos do vetor.

Gabarito: Letra C

---

22. (FCC - TRF 3ª Região /2019) No interior do método main da classe principal de uma aplicação Java SE um método foi chamado usando as instruções abaixo.

```
double[] dados;
double [] d={1, 20, 7, 8};
dados = Aluno.cadastraDados(d);
```

O método cadastraDados na classe Aluno deve ter a declaração

a) public static double[] cadastraDados(double[] d)
b) public static double[] cadastraDados(double d)
c) public double[] cadastraDados(double[] d)
d) public final double[] cadastraDados(double d[])
e) public static double cadastraDados(double[] d)

Comentários:

Para responder a questão devemos ter em mente que é necessário fazer o método cadastraDados passar algo por parâmetro. Para isso, devemos utilizar a alternativa A: public static double[] cadastraDados(double[] d)

Gabarito: Letra A

---

23. (FCC – SANASA – 2019) Na orientação a objetos uma classe abstrata é construída para ser um modelo para classes derivadas e na sua construção há algumas restrições. Assim, considere a classe abstrata abaixo, criada na Linguagem Java.

```
public abstract class Calcula {
    private static final double VALOR=10;
    public abstract double soma(double n1, double n2);
    public abstract void exibeResultado( );
    protected abstract double soma(double n1, double n2, double n3);
    private abstract int multiplica(double n1, int n2);
    private double multiplica(double n1, double n2){return n1*n2;}
```

```
        public Calcula( ) { }
    }
```

A instrução que NÃO é permitida nessa classe é

a) private abstract int multiplica(double n1, int n2);
b) public abstract void exibeResultado( );
c) public Calcula( ) { }
d) private static final double VALOR=10;
e) private double multiplica(double n1, double n2){return n1*n2;}

Comentários:

Pessoal, analisando as alternativas, podemos observar na letra A a criação de um método "private abstract". Porém, em Java, não é permitido declarar um método como "private abstract"! daí já sabemos que a alternativa errada (gabarito da questão) é a letra A. Já, a letra B está correta, veja é a declaração de um método abstrato sem implementação. As demais alternativas também estão plenamente corretas.

Gabarito: Letra A.

---

24. (FCC – TJ TRF3 – 2019) Em uma aplicação Java, um vetor n foi criado por meio da instrução double n=new double[3]; e alimentado com 3 valores reais. Para exibir o conteúdo da segunda posição (índice) deste vetor utiliza-se a instrução

a) JOptionPane.showMessageDialog(null, n[2]);
b) System.out.println(n[1]);
c) JOptionPane.ShowMessageDialog(n[2]);
d) System.Out.Println(null, n[1]);
e) JOptionPane.showWindows(0, n[2]);

Comentários:

o comando para imprimir a segunda posição de um vetor de três posições deverá pegar a posição n[1] já que, em Java, o vetor inicia na posição 0. Portanto ficamos apenas com as alternativas b e d. A letra d apresenta um erro porque System.out.Println não possui o parâmetro null como é apresentado na questão. Portanto nosso gabarito é a letra B.

Gabarito: Letra B

---

25. (FCC – AFAP – 2019) Considere a classe Java a seguir em condições ideais.

```
import java.sql.*;
public class ClienteDados {
    public int conectar(){
```

```
    Class.forName("com.mysql.jdbc.Driver");
    Connection con = DriverManager.getConnection
    ("jdbc:mysql://localhost:3306/banco", "root", "");
    Statement st = con.createStatement();
    return 1;
  }
}
```

No método conectar podem ser lançadas duas exceções que o Analista de Informática deseja que sejam tratadas não no interior do método, mas sim por quem o chamar. Para que isso seja permitido, deve-se inserir o comando

No método conectar podem ser lançadas duas exceções que o Analista de Informática deseja que sejam tratadas não no interior do método, mas sim por quem o chamar. Para que isso seja permitido, deve-se inserir o comando

a) try na linha abaixo da declaração do método e catch (DataBaseException, SQLQueryException) abaixo do comando return 1.
b) throws ClassNotFoundException, SQLException na linha de declaração do método.
c) Exception ClassNotFoundException, SQLException na linha de declaração do método.
d) try na linha abaixo da declaração do método e catch (ClassNotFoundException, SQLException) abaixo do comando return 1.
e) throws DataBaseException, SQLQueryException na linha de declaração do método.

Comentários:

Para realizar um tratamento de exceção da forma que é solicitado, ou seja, não no interior do método, mas sim por quem o chamar, podemos usar o throws na assinatura do método com a exceção. O correto é utilizar a sintaxe: tipo_retorno nome_metodo() throws tipo_exceção_1, ..., tipo_exceção_n. Assim, é possível utilizar a alternativa B: throws ClassNotFoundException, SQLException. Por que devemos usar o ClassNotFoundException? Porque há na classe um Class.forName que é um dos métodos básicos em que essa exceção pode ocorrer. As outras são: findSystemClass da classe ClassLoader e loadClass também da classe ClassLoader. Ademais, devemos usar o SQLException porque há uma conexão ao banco de dados na classe.

Gabarito: Letra B

---

26. (FCC / TRF3 – 2019) Uma classe Colaborador de uma aplicação Java tem o método abaixo.

```
public void setNome(String nome) {
this.nome = nome;
}
```

Para instanciar um objeto dessa classe e armazenar o nome "Pedro" no atributo nome utiliza-se

a) Colaborador.setNome("Pedro");
b) Colaborador c -> new Colaborador(); c->Nome="Pedro";

c) Colaborador c = new Colaborador(c.setNome("Pedro"));
d) Colaborador c = new Colaborador(); c.setNome("Pedro");
e) Colaborador c = new Colaborador().setNome("Pedro");

Comentários:

Pessoal, a palavra-chave da herança é extends! A palavra-chave extends faz com que uma subclasse herde (receba) todos os atributos e métodos declarados na classe-pai.

Gabarito: Letra B

---

27. (FCC / SABESP – 2018) As interfaces são usadas nas aplicações Java quando se deseja permitir que diversas classes implementem determinados métodos, mesmo que de formas diferentes. Em uma interface Java

a) os métodos não podem ter os modificadores protected ou private.
b) não pode haver assinaturas de métodos cujo tipo de retorno seja void.
c) pode haver múltiplos construtores, desde que recebam parâmetros diferentes.
d) não pode haver dois ou mais métodos com o mesmo nome, mesmo que recebam parâmetros diferentes.
e) todo método deverá ser implementado por uma das subclasses da aplicação pelo menos uma vez

Comentários:

Pessoal, aqui vamos analisar cada alternativa! A alternativa 'a' diz que os métodos não podem ter os modificadores protected ou private, o que está completamente correto! Vimos em aula que em geral, as interfaces são compostas basicamente de um conjunto de assinaturas de métodos públicos e abstratos. Ou seja, os métodos das interfaces são públicos! B: Não há nenhuma restrição relacionado a assinatura de métodos com tipo de retorno void! Errada letra B! A restrição da letra C também está incorreta! Não há nenhuma restrição relacionada a quantidade de métodos, sequer métodos com o mesmo nome! Se houver, estaremos diante de um caso de sobrecarrega, ou polimorfismo estático. Por fim, a letra E também está errada! Pode sim haver método sem a implementação na subclasse.Não existe obrigação de implementação de uma interface

Gabarito: Letra A.

---

28. (FCC – TRF 3ª Região – 2014) Herança e interfaces são conceitos da orientação a objetos que possibilitam a obtenção de polimorfismo. Em Java, uma interface pode

a) herdar de mais de uma interface.
b) ser instanciada diretamente por meio da instrução new.
c) possuir apenas assinaturas de métodos públicos e protegidos.

d) possuir métodos abstratos e estáticos.
e) conter declarações de constantes e de variáveis não inicializadas.

Comentários:

Em Java, uma interface pode estender múltiplas interfaces, assim como uma classe pode implementar múltiplas interfaces. Além disso, uma classe pode estender apenas uma classe. Sabendo disso, nosso gabarito é a Letra a) herdar de mais de uma interface.

Gabarito: Letra A

---

29. (FCC – TRE-RN – 2011) Em relação ao Java Standard Edition, é INCORRETO afirmar:

a) Possui gerenciamento de memória embutido, por meio do coletor de lixo.
b) Ambiente indicado para o desenvolvimento de aplicativos para dispositivos móveis ou portáteis.
c) Permite o desenvolvimento de aplicações desktop de linha de comando e interfaces gráficas Swing.
d) Portabilidade dos programas compilados para diversos sistemas operacionais, sem necessidade de recompilação.
e) Usa conceitos tais como orientação a objetos e multithreading.

Comentários:

O acesso a arrays e strings, e a conversão de tipos são checados em tempo de execução para assegurar a sua validade. O Garbage Collector faz a desalocação automática de memória evitando, erros de referência e desperdício de memória. Finalmente, o recurso de Exception Handling permite o tratamento de erros em tempo de execução, por um mecanismo robusto, análogo ao do C++. (a) Conforme vimos em aula, ele contém um Garbage Collector para gerenciamento de memória; Java Micro Edition (Java ME): trata-se do padrão aplicado a dispositivos compactos ou móveis, como smartphones, tablets, controles remotos, etc. Permite o desenvolvimento de softwares embarcados, i.e., aplicações que rodam em um dispositivo de propósito específico, desempenhando alguma tarefa útil. Em geral, possuem limitações de recursos como memória ou processamento. (b) Conforme vimos em aula, esse é o Java ME (Java Micro Edition); A Plataforma Java oferece recursos para construção de interfaces gráficas de usuário (GUI), entre eles: AWT (java.awt) e Swing (javax.swing)! O primeiro é um conjunto básico de classes e interfaces que definem os componentes de uma janela desktop. Já o Swing é um conjunto sofisticado de classes e interfaces que definem os componentes visuais necessários para construir uma interface gráfica de usuário. (c) Conforme vimos em aula, ele permite desenvolvimento de aplicações desktop e de interfaces gráficos (Swing e AWT); Vocês sabiam que Java é uma Linguagem WORA? Pois é, esse acrônimo significa Write Once, Run Anywhere ou Escreva uma vez, execute em qualquer lugar. Trata-se de um slogan para exemplificar os benefícios multiplataforma da linguagem Java! Idealmente, isso significa que um programa em Java (uma vez compilado em um bytecode) pode rodar em qualquer equipamento que possua uma JVM! (d) Conforme vimos em aula, é uma linguagem WORA (Write Once, Run Anywhere); Dessa

forma, aplicações funcionam da mesma maneira em qualquer ambiente. Podemos dizer que Java é uma linguagem concorrente ou multithreaded, i.e., pode realizar diversas tarefas assincronamente com o uso de threads, que são suportadas de modo nativo. Java torna a manipulação de threads tão simples quanto trabalhar com qualquer variável. (e) Conforme vimos em aula, é uma linguagem orientada a objetos e possui suporte nativo a threads.

Gabarito: Letra B

---

30. (FCC – TRT 22ª Região – 2010) A plataforma Java disponibiliza um interpretador que traduz, em tempo de execução, o bytecode para instruções nativas do processador, permitindo, dessa forma, que uma mesma aplicação seja executada em qualquer plataforma computacional que possua essa implementação. Trata-se de:

a) Java Virtual Machine.
b) Java API.
c) JavaBeans.
d) J2SE.
e) JavaFX.

Comentários:

O bytecode é um código intermediário, que é posteriormente interpretado e executado por uma Java Virtual Machine (JVM). O que é isso, professora? É um programa que carrega e executa os aplicativos Java, convertendo bytecodes em código executável. Lembram que eu falei que Java é uma Linguagem WORA? Pois é, isso ocorre em grande parte por conta do bytecode e da Máquina Virtual Java. Conforme vimos em aula, trata-se da Java Virtual Machine (JVM)!

Gabarito: Letra A

---

31. (FCC – Sergipe Gás– 2010) É tida como uma das principais linguagens de programação orientada a objeto; tem como característica a compilação para um bytecode e execução por uma máquina virtual. Trata-se da linguagem:

a) Algol.
b) Delphi.
c) C++.
d) Java.
e) PHP.

Comentários:

Java é também uma linguagem portável e multiplataforma! O Compilador é capaz de gerar um código intermediário (bytecode), que permite que o mesmo programa possa ser executado em qualquer máquina ou sistema operacional que possua uma JVM. Ademais, busca que todos os

aspectos da linguagem sejam independentes de plataforma (Ex: ela especifica o tamanho e comportamento de cada tipo de dado). Conforme vimos em aula, trata-se da linguagem Java.

Gabarito: Letra D

---

32. (FCC – TCE-SP – 2010) Os aplicativos Java "rodam" em diferentes ambientes. A tradução dos códigos Java (bytecode), para instruções específicas de cada sistema e dispositivo, é uma função do programa:

a) Java Community Process (JCP).
b) Java Virtual Module (JVM).
c) Java Virtual Machine (JVM).
d) Java Comunication Process (JCP).
e) Java Enterprise Machine (JEM).

Comentários:

O bytecode é um código intermediário, que é posteriormente interpretado e executado por uma Java Virtual Machine (JVM). O que é isso, professora? É um programa que carrega e executa os aplicativos Java, convertendo bytecodes em código executável. Lembram que eu falei que Java é uma Linguagem WORA? Pois é, isso ocorre em grande parte por conta do bytecode e da Máquina Virtual Java. Conforme vimos em aula, trata-se da Java Virtual Machine (JVM)!

Gabarito: Letra C

QUESTÕES COMENTADAS – CESPE

33. (CESPE – Petrobrás – 2022)

```
package cadastroUsuario;


import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.*;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;


public class CadastroServlet extends HttpServlet
{
   protected void doPost(HttpServletRequest
     request,HttpServletResponse response)
   throws ServletException, IOException {
   String nome = request.getParameter("nome");
String endereco =
request.getParameter("endereco");
  String telefone =
   request.getParameter("telefone");
  response.setContentType("text/html");
  PrintWriter out = response.getWriter( );
  out.println("<HTML>");
  out.println("<HEAD>");
  out.println("<STYLE>");
  out.println("background-color: green;");
  out.println("</STYLE>");
  out.println("</HEAD>");
  out.println("<BODY>");
  out.println("<P>Prezado(a) ", " + nome + "
</P>");
  out.println("<P>Obrigado por se
cadastrar.</P>");
  out.println("<P>Entraremos em contato através
do telefone", " + telefone+ " em breve.</P>");
  out.println("</BODY>");
  out.println("</HTML>");
  out.close( );
```

```
  }
}
```

Tendo como referência o código precedente, julgue o item que se segue.

A linha
String nome = request.getParameter("nome");

pode ser alterada para
String nome = request.getAttribute("nome");

sem perda de funcionalidade no código.

Comentários:

Pessoal, conforme descrito no tópico "Métodos", há diferenças entre esses métodos, um deles é o retorno: getParameter retorna uma String representando o valor único do parâmetro. Já, o getAttribute retorna um Object contendo o valor do atributo, ou null se o atributo não existir. Portanto, há perda de funcionalidade no código tornando errada a assertiva.

Gabarito: Errada

---

34. (CESPE - DP DF - 2022) No Java 8, o uso do método map() permite aplicar uma função a todos os elementos de um stream.

Comentários:

Stream map(Function mapper) retorna um stream que consiste nos resultados da aplicação da função dada aos elementos desse stream. Stream map (Mapeador de funções) é uma operação intermediária. As operações intermediárias são invocadas em uma instância do Stream e, depois que terminam seu processamento, fornecem uma instância do Stream como saída.

Gabarito: Correto

---

35. (CESPE - DP DF– 2022) Quando a mensageria é utilizada com linguagem Java, as requisições são processadas exclusivamente de forma síncrona.

Comentários:

Os sistemas de mensageria permitem que você acople livremente sistemas heterogêneos, enquanto fornece também confiabilidade, transações e muitos outros recursos. Ao contrário dos

sistemas baseados em um padrão de Chamada de Procedimento Remoto (RPC), o sistema de mensageria usa principalmente um padrão de passagem de mensagem assíncrona sem relação entre pedidos e respostas. Ou seja, o serviço é realizado de forma assíncrona, o que torna a questão incorreta.

Gabarito: Errado

---

36. (CESPE – SEED PR –2021) Java é uma linguagem construída a partir de um legado das linguagens C e C++. No entanto, ela apresenta características únicas que a diferem das demais, como:

I. o applet, que é um tipo especial de programa Java projetado para ser transmitido pela Internet e executado por um navegador web compatível com Java.

II. a saída de seu compilador não gera um código executável e, sim, um bytecode.

III. o fato de um programa Java ser executado somente pela Java virtual machine (JVM).

Assinale a opção correta.

a) Apenas o item I está certo.
b) Apenas o item II está certo.
c) Apenas os itens I e III estão certos.
d) Apenas os itens II e III estão certos.
e) Todos os itens estão certos.

Comentários:

Pessoal, perfeita a assertiva I: applet é um tipo especial de programa Java projetado para ser transmitido pela Internet e executado por um navegador web compatível com Java. A assertiva II também está de acordo com o que vimos! O compilador transforma o arquivo .java em código de máquina e em um arquivo .class, também chamado bytecode. Por fim, a assertiva III também está correta, exatamente como dissemos anteriormente: O bytecode é um código intermediário, que é posteriormente interpretado e executado por uma Java Virtual Machine (JVM). O código fonte é entendido pelo sistema operacional, mas varia em cada um deles. Quando você usa o bytecode, através do .class você consegue rodar facilmente em qualquer sistema operacional, sem ter que usar o código-fonte.

Gabarito: Letra E

---

37. (CESPE – PGDDF – 2021) Julgue o item subsecutivo, a respeito de lógica e de estrutura de programação.

```
função avaliar( a, b )
início
    ma <-a;
    se (ma < b) então ma <- b;
    me <-a;
    se (me > b) então me <- b;
  resultado <- ( ma % me );
  se (resultado = 0)
      então retorne me
      senão avaliar(me, ma)
fim
escreva avaliar (120,30);
```

O resultado do pseudocódigo precedente será 120.

Comentários:

Pessoal, o código consiste em comparar dois valores, 120 e 130 (avaliar (120,30)). Vocês devem observar que os valores de a e b são respectivamente 120 e 30. Daí, no início da função, avaliar ma recebe o valor de a (120), e passa pelo primeiro se (se (ma < b) então ma <- b) e não entra, já que ma (120) é maior que b (30). Já, o segundo "se – então" é executado, veja: se (me > b) então me <- b; me é maior que b porque me = 120 e b = 30. Com o resultado verdadeiro do se, me recebe o valor 30 – que era o valor de b (no código me=b). Após isso, a variável resultado recebe o resto da divisão de ma por me que é igual a 0. Como resultado é igual a zero, entra no próximo se (se (resultado = 0)), daí retorna o valor de me que é 30. Portanto, o resultado do pseudocódigo será 30, e não 120 conforme diz a questão.

Gabarito: Errado

---

38. (CESPE – TJ-RJ – 2021) Considere o trecho de código a seguir, de um programa escrito na linguagem Java.

```
1   public class Carro
2   {
3     private String modelo;
4     private int cilindradas;
5     public Carro(String modelo, int cilindradas)
6     {
7       this.modelo = modelo;
8       this.cilindradas = cilindradas;
9     }
...
15  }
```

Quanto a esse contexto, considere, ainda, as instruções a seguir, da classe `CarroEsportivo`.

```
1  public class CarroEsportivo extends Carro
2  {
3          private String marca;
4          public CarroEsportivo(String modelo,  int cilindradas, String marca)
5          {
6                  super(modelo, cilindradas);
7                  this.marca = marca;
8          }
9  }
```

Com relação aos trechos de código precedentes, é correto afirmar que

a) o parâmetro this, na linha 7 da classe CarroEsportivo, é uma palavra reservada usada para mostrar que está sendo feita referência ao atributo privado marca da classe Carro.
b) o comando this, na linha 7 da classe CarroEsportivo, é uma palavra reservada que faz referência ao atributo público marca da classe Carro.
c) a instrução super, na linha 6 da classe CarroEsportivo, é usada para importar o pacote atributos privados da classe Carro, a fim de se tornarem públicos para a classe CarroEsportivo.
d) a instrução super, na linha 6 da classe CarroEsportivo, permite à classe CarroEsportivo acessar os atributos privados da classe Carro.
e) a instrução super, na linha 6 da classe CarroEsportivo, pode permitir à classe CarroEsportivo acessar atributos públicos da classe Carro.

Comentários:

Vejamos cada uma das alternativas a partir da análise do código apresentado. A letra a está errada, pois diz que o parâmetro this, na linha 7 da classe CarroEsportivo, é uma palavra reservada usada para mostrar que está sendo feita referência ao atributo privado marca da classe Carro, sendo que o parâmetro this faz referência ao atributo privado marca da classe CarroEsportivo. A letra B também está errada porque o comando this, na linha 7, faz referência a marca da classe CarroEsportivo. Já a alternativa c está errada pois não é possível importar o pacote atributos privados da classe Carro, a fim de se tornarem públicos dado que eles estão com o modificador private. A alternativa d também está errada, assim como a anterior, pois não é possível acessar atributos privados, mesmo que seja uma classe filha. Por fim, a alternativa e é o nosso gabarito! De fato, a instrução super chama membros da classe-pai. Ou seja, é uma variável de referência usada para referenciar o objeto da classe pai.

Gabarito: Letra E

39. (CESPE – PGDF – 2021) Com relação a servidores de aplicação e seus serviços e a teoria de becapes, julgue o item a seguir.

Apenas uma única instância do Tomcat pode existir em uma única JVM (Java virtual machine); no entanto, é possível ter múltiplas instâncias em uma mesma máquina virtual, desde que em processos Java separados, em execução em portas separadas.

Comentários:

Pessoal, na verdade, é possível uma mesma máquina física, porém com processos Java separados e sendo executados em portas separadas. Veja: uma instância do Tomcat é o componente de mais alto nível na hierarquia do contêiner Tomcat. Apenas uma única instância do Tomcat pode existir em uma única JVM. Essa abordagem faz que todas as outras aplicações Java que estejam sendo executadas em uma mesma máquina física em um servidor Tomcat, seguras, caso o Tomcat ou a JVM travem. Podem-se ter múltiplas instâncias em uma mesma máquina física, porém com processos Java separados e sendo executados em portas separadas.

Gabarito: Errado

---

40. (CESPE -– Ministério da Economia – 2020) Em Java 8, o método de limite de fluxo tem como objetivo eliminar elementos com base em um critério.

Comentários:

Na verdade, o método limit é utilizado para limitar a quantidade de elementos em um fluxo.

Gabarito: Errado

---

41. (CESPE – Ministério da Economia – 2020) Uma expressão lambda é usada principalmente para definir a implementação procedural de uma interface associativa.

Comentários:

Uma expressão lambda é usada principalmente para definir a implementação procedural de uma interface funcional.

Gabarito: Errado

---

42. (CESPE – Pref. Barra dos Coqueiros – 2020) A máquina virtual do Java (JVM) utiliza uma tecnologia especificamente para aumentar o desempenho da aplicação, identificando códigos que serão muito executados, normalmente dentro de loops. Essa tecnologia é chamada de

a) hotspot.
b) bytecode.
c) compilação estática.

d) JRE (Java Runtime Environment).
e) JDK (Java Development Kit).

Comentários:

A questão solicita uma tecnologia especificamente para aumentar o desempenho da aplicação, identificando códigos que serão muito executados, normalmente dentro de loops. De acordo com Deitel, em geral, as JVMs atuais executam bytecodes utilizando uma combinação de interpretação e a chamada compilação Just-In-Time (JIT). Nesse processo, a JVM analisa os bytecodes à medida que eles são interpretados, procurando hotspots (pontos ativos) — partes dos bytecodes que executam com frequência. Para essas partes, um compilador Just-In-Time (JIT) — conhecido como compilador Java HotSpot — traduz os bytecodes para a linguagem de máquina de um computador subjacente. Da mesma forma, temos a definição de HotSpot que é uma máquina virtual Java para desktops e servidores, mantido e distribuído pela Oracle Corporation. Ele apresenta técnicas como just-in-time de compilação e otimização adaptativa projetado para melhorar o desempenho. Portanto, temos a certeza que o gabarito é a letra A. A assertiva b está errada porque o bytecode é um código intermediário, que é posteriormente interpretado e executado por uma Java Virtual Machine (JVM). A letra c também está errada, foi apenas para confundir o candidato. A letra d trata da JRE, camada de software que é executado sobre um software de sistema operacional de um computador e fornece as bibliotecas de classe e outros recursos que um programa específico do Java precisa executar. Por fim, JDK é um conjunto de utilitários cuja a finalidade é a permissão para criação de jogos e programas para a plataforma Java. Este pacote é disponibilizado pela Oracle, e nele vem todo o ambiente necessário para a criação e execução dos aplicativos java.

Gabarito: Letra A

---

43. (CESPE – MPE-CE – 2020) Com base na linguagem de programação Java, julgue o item a seguir.

O código

```
public class mpce {
  public static void main(String[] args) {
    int[][] num = { {3, 5}, {9, 7} };
    for (int i = 0; i < num.length; ++i) {
     for(int j = 0; j < num[i].length; ++j) {

System.out.println(num[i][j]);
    }
   }
  }
}
```

apresentará, quando executado, o resultado a seguir.

3
5
7
9

Comentários:

Na verdade, a sequência que será apresentada é: 3, 5, 9, 7. Portanto a questão está errada, pois a matriz não fará a ordenação dos dados, mas apenas imprimir, primeiro preenchendo as linhas e em seguida preenchendo as colunas.

Gabarito: Errado

44.(CESPE – MPE-CE – 2020) Com base na linguagem de programação Java, julgue o item a seguir.

O código

```
public class mpce {

public static int Funcao(int num, int resultado){
  if(num<1)
    return resultado;
  resultado+=(num%10);
  return Funcao(num/10, resultado);
}

public static void main(String[] args) {

  int res = Funcao(17, 0);
  System.out.println("O resultado é: " +
res);
 }
}
```

apresentará a seguinte saída ao ser executado.

O resultado é: 8

Comentários:

Vejamos o funcionamento da função: ela executa até o valor de num ser menor que 1, ou seja, até que o valor de num seja 0. Para o início da execução temos os valores iniciais de 17 para num e o para resultado. O primeiro passo a se fazer é entrar no if, da linha 3, e fazer a comparação: 17 < 1,

dezessete é menor que um? A resposta é não, assim ele ignora a instrução de retorno que está na linha 4 e parta para a operação matemática da linha 5. resultado = resultado + (num % 10), a variável resultado está recebendo a soma dela mesma com o resto da divisão de num por 10, em números temos: 0 = 0 + (17 % 10) -> o resto da divisão de 17 por 10 é 7, assim temos: resultado = 0 + 7. Neste momento o valor da variável resultado é 7. E entramos na instrução de retorno da linha 6, em que retorna a própria função com o valor da variável num dividido por 10 e o valor da variável resultado. Assim temos: 17 / 10 = 1,7, mas como a função retorna apenas a parte inteira, então aqui esta divisão assume valor 1. Então a instrução de retorno da linha 6 tem como valores num = 1 e resultado = 7. E o processo se repete novamente e voltamos ao if da linha 3. Ele vai repetir esses passos até que chegue a comparação 0,1 < 1, zero vírgula um é menor que 1? A resposta é sim, então agora ele entra na instrução de retorno da linha 4, neste caso ele imprime a variável resultado que possui valor 8. Então, quando chama a função na linha 9 ela imprime a variável resultado como 8. Dessa forma, o nosso gabarito está correto.

Gabarito: Correto

---

45. (CESPE – TJPA – 2020)

```
class GeraNumeros {
  public static void main (String args []) {
    int num;
    num = 36;
    for(int i=0; i < num; i++) {
      if(i*i >= num) break;
      System.out.print(i + " ");
    }
   }
}
```

Assinale a opção que apresenta corretamente a saída gerada pelo código Java precedente.

a) 36
b) 1 2 3 4 5 6
c) 1 2 3 4 5
d) 0 1 2 3 4 5
e) 0 1 2 3 4 5 6

Comentários:

Pessoal, a função compara i a num. E irá parar quando i for menor que 36. Aparentemente, i é incrementado em 1, porém dentro do if, i = i*i. Podemos visualizar o foco da questão nessa condição. Assim, i vale respectivamente 0,1,4,9,16,25,36 a cada iteração. Portanto sendo executado 7 vezes. E imprimindo respectivamente os valores 0 1 2 3 4 5.

Gabarito: Letra D

46.(CESPE – STM – 2018) Os membros de uma classe definidos como PUBLIC não podem ser acessados ou usados por nenhuma outra classe.

Comentários:

Vamos relembrar todos os modificadores? Default (modificador): A classe e/ou seus membros são acessíveis somente por classes do mesmo pacote, na sua declaração não é definido nenhum tipo de modificador, sendo este identificado pelo compilador. Public: Marca a visibilidade de uma classe, método ou variável de instância para que todas as classes em todos os pacotes tenham acesso. Package: Informa em que estrutura de diretórios a classe está localizada. Private: Marca a visibilidade de um método ou variável de instância para que apenas a própria classe acesse. Membros da classe definidos como private não podem ser acessados ou usados por nenhuma outra classe. Esse modificador não se aplica às classes, somente para seus métodos e atributos. Esses atributos e métodos também não podem ser visualizados pelas classes herdadas. Protected: Marca a visibilidade de um método ou variável de instância para que apenas a própria classe ou suas filhas acessem. O modificador protected torna o membro acessível às classes do mesmo pacote ou através de herança, seus membros herdados não são acessíveis a outras classes fora do pacote em que foram declarados. Portanto, Public: Marca a visibilidade de uma classe, método ou variável de instância para que todas as classes em todos os pacotes tenham acesso e a questão está incorreta!

Gabarito: Errado

47.(CESPE – CGM-Joao Pessoa– 2018) A JME oferece um ambiente robusto e flexível para aplicativos executados em dispositivos móveis e integrados cuja capacidade de memória, de vídeo e de processamento são limitados, tais como set-top boxes e reprodutores de discos blu-ray.

Comentários:

A plataforma Java, Micro Edition (Java ME) fornece um ambiente robusto e flexível para aplicativos executados em dispositivos embarcados e móveis na Internet das Coisas: microcontroladores, sensores, gateways, telefones celulares, assistentes digitais pessoais (PDAs), aparelhos de TV top boxes, impressoras e muito mais. O Java ME inclui interfaces de usuário flexíveis, segurança robusta, protocolos de rede integrados e suporte para aplicativos em rede e offline que podem ser baixados dinamicamente. Os aplicativos baseados em Java ME são portáteis em muitos dispositivos, mas aproveitam os recursos nativos de cada dispositivo. Essa é a definição exata do site da Oracle! Perfeita questão.

Gabarito: Correto

48.(CESPE – TER-TO– 2017) Na orientação a objetos, a alteração do comportamento dos métodos herdados das superclasses para um comportamento mais específico nas subclasses, de forma a se criar um novo método na classe filha que contém a mesma assinatura e o mesmo tipo de retorno, relaciona-se a

a) sobrecarga.
b) overloading.
c) portabilidade.
d) abstração.
e) sobrescrita.

Comentários:

Pessoal, isso é um caso de sobrescrita. Utilizando a sobrescrita conseguimos especializar os métodos herdados das superclasses, alterando o seu comportamento nas subclasses por um mais específico. A sobrescrita (ou override) está diretamente relacionada à orientação a objetos, mais especificamente com a herança. Com a sobrescrita, conseguimos especializar os métodos herdados das superclasses, alterando o seu comportamento nas subclasses por um mais específico. A sobrescrita de métodos consiste basicamente em mesmo tipo de retorno do método sobrescrito.

Gabarito: Letra E

49.(CESPE – TCE-PA – 2016) O fato de as expressões lambda terem sido excluídas do Java 8 possibilitou que a linguagem Java ficasse mais flexível e que a implementação de seu código se tornasse mais precisa.

Comentários:

Pessoal, erradíssimo! Lambda foi incluída no Java 8! Um lambda permite que um trecho de código seja passado como parâmetro para uma função ou ser armazenado em uma variável para ser invocado posteriormente

Gabarito: Errado

50.(CESPE – FUNPRESP-JUD – 2016) Para lidar com um conjunto de objetos em JSon, é necessário utilizar um array que permita realizar, em uma única operação, a carga de todos os objetos.

Comentários:

Não é necessário utilizar um array.

Gabarito: Errado

---

51. (CESPE – TRE-PI – 2016) A linguagem Java foi, originalmente, desenvolvida para

a) permitir a comunicação entre aparelhos domésticos e computadores.
b) traduzir fórmulas matemáticas utilizando-se cartões perfurados.
c) processar valores inteiros, em um ambiente negócios, em computadores de grande porte.
d) trabalhar com inteligência artificial por meio de uma abordagem puramente lógica.
e) demonstrar a viabilidade da implementação da álgebra relacional de dados.

Comentários:

A história de Java começou a ser escrita quando James Gosling, Patrick Naughton e Mike Sheridan se uniram a dois fundadores da Sun, Andy Bechtolsheim e Bill Joy, para pensar sobre a nova onda do mundo digital. Eles não demoraram muito para concluir que seria a convergência de computadores aos dispositivos e eletrodomésticos utilizados no dia a dia, tudo interconectado e remotamente controlado. Uma questão sobre "curiosidades", temos como gabarito a letra A, já que, Java foi, originalmente, desenvolvida para permitir a comunicação entre computadores aos dispositivos e eletrodomésticos

Gabarito: Letra A

---

52. (CESPE – TER-GO – 2015) Em um grupo do tipo Array, podem-se armazenar dados de diferentes tipos.

Comentários:

Errado pessoal! Um Array é um objeto contêiner que contém um número fixo de valores de um único tipo. O comprimento de um array é estabelecido quando o array é criado. Após a criação, seu comprimento é fixo.

Gabarito: Errado

---

53. (CESPE – TJDFT – 2015) Na tecnologia JSP (Java Server Pages), cada documento é convertido pelo contêiner JSP em um servlet, o que ajuda a separar a apresentação do conteúdo.

Comentários:

Perfeito, pessoal! O JSP é uma ferramenta baseada em Java para criação de páginas dinâmicas para uso na web. Com isso, cada parte do documento é convertida, através de um Contêiner, como um TomCat por exemplo, em um Servlet.

Gabarito: Correto

---

54. (CESPE – SERPRO – 2013) A tecnologia JSE (Java Small Editon) utilizada pela plataforma iOS permite o desenvolvimento de aplicações para todos os seus dispositivos, como estações gráficas, iPad, iPod, entre outros.

Comentários:

Java Micro Edition (Java ME): trata-se do padrão aplicado a dispositivos compactos ou móveis, como smartphones, tablets, controles remotos, etc. Permite o desenvolvimento de softwares embarcados, i.e., aplicações que rodam em um dispositivo de propósito específico, desempenhando alguma tarefa útil. Em geral, possuem limitações de recursos como memória ou processamento. Conforme vimos em aula, a questão está cheia de erros! Primeiro, JSE é a sigla de Java Standard Edition. Segundo, a tecnologia proposta no item é o JME!

Gabarito: Errado

---

55. (CESPE – MPOG – 2013) O JME foi criado para trabalhar com ambientes de programação multiprocessados em ambientes distribuídos.

Comentários:

Java Micro Edition (Java ME): trata-se do padrão aplicado a dispositivos compactos ou móveis, como smartphones, tablets, controles remotos, etc. Permite o desenvolvimento de softwares embarcados, i.e., aplicações que rodam em um dispositivo de propósito específico, desempenhando alguma tarefa útil. Em geral, possuem limitações de recursos como memória ou processamento. Java Enterprise Edition (Java EE): trata-se do padrão para desenvolvimento de sistemas corporativos, voltada para aplicações multicamadas, baseadas em componentes executados em servidores de aplicações – ele inclui o Java SE. Contém bibliotecas para acesso a base de dados, RPC, CORBA, entre outras. As aplicações podem ou não estar na internet. Conforme vimos em aula, Java ME possui diversas limitações de recursos. Na verdade, é o Java EE que é ideal para ambientes distribuídos.

Gabarito: Errado

---

56. (CESPE – ANTT – 2013) JSE, JME, JEE e JCE são edições da tecnologia Java voltadas para o desenvolvimento de aplicações para desktop/servidores, dispositivos móveis, ambientes corporativos e ambientes em nuvem, respectivamente.

Comentários:

Java Micro Edition (Java ME): trata-se do padrão aplicado a dispositivos compactos ou móveis, como smartphones, tablets, controles remotos, etc. Permite o desenvolvimento de softwares embarcados, i.e., aplicações que rodam em um dispositivo de propósito específico, desempenhando alguma tarefa útil. Em geral, possuem limitações de recursos como memória ou

processamento. Conforme vimos em aula, está quase tudo certo! No entanto, JCE é uma API de Criptografia (Java Cryptography Extension). Não se trata de uma plataforma ou ambiente de desenvolvimento em nuvem!

Gabarito: Errado

---

57. (CESPE – TRT 10ª Região – 2013) O uso de System.out.println() possibilita a exibição de textos; para a exibição de valores de variáveis, é necessário utilizar showAttributes().

Comentários:

Errado! O System.out.println() pode perfeitamente ser utilizada para exibir variáveis.

Gabarito: Errado

---

58. (CESPE – TRT 10ª Região – 2013) No código abaixo, caso a variável salário assuma o valor igual a 5000, o valor b da variável avaliação será atribuído corretamente.

```
if (salario > 10000) {
avaliacao=”a”;
else
avaliacao=”b”;
}
```

Comentários:

Como o valor apresentado (5.000) é menor que 10.000, a assertiva está correta!

Gabarito: Correto

---

59. (CESPE – TRT 10ª Região – 2013) A execução do código abaixo informará a classe do objeto que foi atribuído à variável de nome var4.

```
String nome = var4.getname();
```

Comentários:

Errado, na verdade, getname retorna o nome do objeto. O código apresentado pelo item declara e inicializa uma variável nome, do tipo String. O valor que essa variável receberá é o que for retornado pelo método getname, invocado sobre o objeto var4. Apenas com as informações trazidas, é impossível saber qual a classe à qual o objeto var4 pertence.

Gabarito: Errado

60.(CESPE – MPU – 2013) O tratamento de exceção em Java consiste no bloco try-catch-finally. O bloco finally sempre executa ao término do bloco try.

Comentários:

Questão, ao generalizar, erra porque finally não está relacionada com o sucesso do bloco try-catch-finally. A instrução finally permite que você execute código, depois do try...catch, independentemente do resultado.

Gabarito: Errado

61.(CESPE – MPU – 2013) Se a thread executando um código try ou catch for interrompida ou abortada, o bloco finally poderá não ser executado, apesar de a aplicação como um todo continuar.

Comentários:

A instrução try permite que você defina um bloco de código para ser testado quanto a erros enquanto está sendo executado. A instrução catch permite definir um bloco de código a ser executado, caso ocorra um erro no bloco try. Por fim, a instrução finally permite que você execute código, depois do try...catch, independentemente do resultado. Mas há casos em que o finally não é executado. Questão correta!

Gabarito: Correto

62.(CESPE – TRE-MA – 2009) Para definição e manipulação de uma exceção em Java, devem constar no programa, obrigatoriamente, os termos:

a) try e catch.
b) try e finally.
c) finally e catch.
d) finally e retry.
e) try e retry.

Comentários:

Pessoal, o correto é Try e catch.

Gabarito: Letra A

63.(CESPE – SERPRO – 2008) A linguagem Java, orientada a objetos, tem como característica ser compilada em um código executado em máquina virtual.

Comentários:

Java é também uma linguagem portável e multiplataforma! O Compilador é capaz de gerar um código intermediário (bytecode), que permite que o mesmo programa possa ser executado em qualquer máquina ou sistema operacional que possua uma JVM. Ademais, busca que todos os aspectos da linguagem sejam independentes de plataforma (Ex: ela especifica o tamanho e comportamento de cada tipo de dado). Conforme vimos em aula, está perfeito! Qual o nome desse código? Bytecode!

Gabarito: Correto

64.(CESPE – TRT - 5ª Região – 2008) A instrução import Java.awt.* indica que o programa irá utilizar componentes gráficos.

Comentários:

Essa questão é polêmica! Alguns afirmar que é possível inserir a instrução, mas não utilizar componentes gráficos. Não faria sentido importar um pacote para não utilizar suas funcionalidades, mas é possível – apesar de má prática! Eu acredito que a questão cabe recurso, sim!

Gabarito: Correto

65.(CESPE – TRT - 5ª Região – 2008) Em Java, os métodos public de uma classe são utilizados pelos clientes da classe para manipular dados armazenados em objetos dessa classe.

Comentários:

Pessoal, esses Modificadores de Acesso determinam quão acessíveis são esses elementos. Vamos vê-los agora em mais detalhes:<public>: essa instrução indica que a classe, método ou atributo assim declaradas podem ser acessadas em qualquer lugar e a qualquer momento da execução do programa – é o modificador menos restritivo. Perfeito, são públicos para toda e qualquer classe.

Gabarito: Correto

QUESTÕES COMENTADAS – OUTRAS BANCAS

66. (UFMA – UFMA – 2019) Duas características importantes e relacionadas entre si, presentes em Java por ser uma linguagem orientada a objetos, são a herança e o polimorfismo. Considere as afirmativas I e II a seguir e depois marque a alternativa correta.

I. Herança múltipla é um recurso existente em Java para permitir que uma classe possa herdar atributos e métodos de mais de uma classe.

II. Polimorfismo em Java é a capacidade de duas ou mais classes derivadas de uma mesma superclasse possuírem a mesma assinatura de um método, porém com comportamento diferente.

a) Apenas a afirmativa II está correta.
b) Ambas as afirmativas I e II estão corretas.
c) Ambas as afirmativas I e II estão erradas.
d) Apenas a afirmativa I está correta.
e) A correção ou não das afirmativas I e II depende de qual versão de Java se está levando em consideração.

Comentários:

Pessoal, é necessário saber os conceitos de herança e polimorfismo. Primeiramente, devemos ter em mente que não existe Herança múltipla em Java. Daí a assertiva I está incorreta. Vejamos o que diz a assertiva II: Polimorfismo em Java é a capacidade de duas ou mais classes derivadas de uma mesma superclasse possuírem a mesma assinatura de um método, porém com comportamento diferente. Está perfeita a definição.

Gabarito: Letra A

---

67. (FUNDEP – Prefeitura de Lagoa Santa– 2019) Qual é a forma correta de se criar uma classe que não poderá ser instanciada, mas ainda poderá ser reutilizada?

a) public abstract class NomeDaClasse
b public abstract NomeDaClasse
c) public class NomeDaClasse
d) public class private NomeDaClasse

Comentários:

Pessoal, para criar uma classe que não poderá ser instanciada, mas ainda poderá ser reutilizada devemos utilizar public abstract class NomeDaClasse. Porque a classe abstrata é sempre uma superclasse que não possui instâncias.

Gabarito: Letra A

---

68. (UFRN – UFRN – 2019) Na programação orientada a objetos, o polimorfismo é a habilidade de objetos de classes diferentes responderem à mesma mensagem de maneiras diferentes. Um tipo de polimorfismo é

a) polimorfismo de sobrecarga.
b) polimorfismo de instanciação.
c) polimorfismo de abstração.
d) polimorfismo de classificação.

Comentários:

Questão que todo mundo que estudou a aula gostaria que caísse em sua prova: resposta letra a) polimorfismo de sobrecarga. Todas as demais são invenções.

Gabarito: Letra A

---

69. (IADES – Hemocentro – DF– 2017) Considerando que Luta e Jogo são classes, e que Esporte, Individual e Coletivo são interfaces, com relação aos conceitos de classes e interfaces da linguagem Java, assinale a alternativa correta.

a) interface Futebol implements Esporte, Jogo{}.
b) interface Futebol extends Esporte,Coletivo{}
c) interface Futebol implements Esporte{}
d) class Judo extends Esporte, Individual{}
e) interface Karate extends Luta{}

Comentários:

Pessoal, a questão cobra conceitos sobre classes e interfaces. Em Java, uma interface pode estender múltiplas interfaces, assim como uma classe pode implementar múltiplas interfaces. Além disso, uma classe pode estender apenas uma classe. Temos que nos atentar ao fato que Luta e Jogo são classes e Esporte Individual e Coletivo são interfaces. Sabendo que classes implementam interfaces e classes podem estender classes. Dessa forma, podemos entender que o gabarito é a letra b, porque a interface Futebol pode estender de esporte e de Coletivo (uma interface pode estender múltiplas interfaces) que são interfaces.

Gabarito: Letra B

---

70. (ESAF – Receita Federal – 2012) Em programação Java, o comando while:

a) executa um bloco exclusivamente de comandos de atribuição.
b) executa um bloco de comandos enquanto sua condição for verdadeira.
c) executa um bloco de comandos até que sua condição seja verdadeira.
d) equivale ao comando what-if.
e) é idêntico ao comando do while.

Comentários:

(a) Na verdade, podem ter outros comandos – não é só atribuição; (b) Perfeito, enquanto for verdadeira, continua a iteração; (c) Não, enquanto ela for verdadeira; (d) Não, esse comando não existe; (e) Não, esse comando entra no bloco e só depois avalia a condição.

Gabarito: Letra B

LISTA DE QUESTÕES – FGV

1. **(FGV – CGU – 2022)** Observe o trecho de código a seguir.

```
1 import java.io.*;
2
3 interface Segunda { public void segunda();}
4 interface Terca { public void terca();}
5 interface Quarta extends Segunda, Terca { public void quarta();}
6
7 class Quinta implements Segunda {
8 public void segunda(){System.out.println("Segunda");}
9 }
10
11 class Sexta extends Quinta implements Quarta {
12 public void segunda(){System.out.println("Segunda!");}
13 public void terca(){System.out.println("Segunda!!");}
14 public void quarta(){System.out.println("Segunda!!!");}
15 }
16
17 public class teste
18 {
19 public static void main( String[] args )
20 {
21 Quinta dia = (Quinta) new Sexta();
22 dia.segunda();
23 }
24 }
```

A saída produzida pela execução do código é:

a) Erro de compilação na linha 5
b) Segunda
c) Segunda!
d) Segunda!!
e) Segunda!!!

2. **(FGV – TJDFT – 2022)** Observe as classes abaixo descritas na linguagem de programação Java.

public class DocumentoProcesso extends Object

```
{
   private String docNumero;
   private int classificacao;


   public DocumentoProcesso (String docNumero, int

classificacao){
    if (classificacao < 1)
      throw new IllegalArgumentException (
        "A classificação do documento deve ser no mínimo 1.");

     this.docNumero = docNumero;
     this.classificacao = classificacao;
    }
    public String getDocNumero()
    { return docNumero; }
    public int getClassificacao()
    { return classificacao; }
    public void setDocNumero(String docNumero)
    { this.docNumero = docNumero; }
    public void setClassificacao(int classificacao)
    { this.classificacao = classificacao; }
    public int promove()
    { return classificacao + 1; }
}

public class Oficio extends DocumentoProcesso
{

   private int precedencia;
   public Oficio (String docNumero, int classificacao, int precedencia)
   {super(docNumero,classificacao);
   this.precedencia = precedencia; }
public int getPrecedencia()
{ return precedencia; }
public void setPrecedencia(int precedencia)
{ this.precedencia = precedencia; }
@Override
public int promove()
{ return precedencia + 1; }
}
```

Com base nos conceitos de orientação a objetos, é correto afirmar que:
a) os atributos private de DocumentoProcesso são acessíveis por Oficio;
b) a anotação @Override indica que o método promove() é abstrato e polimórfico;
c) a classe Oficio representa uma herança múltipla das classes DocumentoProcesso e Object;
d) a classe Oficio é composta pela classe DocumentoProcesso, permitindo que uma instância da classe Oficio faça referências a instâncias da classe DocumentoProcesso;
e) os métodos getDocNumero() e setDocNumero() da classe DocumentoProcesso encapsulam o atributo docNumero e asseguram que os objetos da classe mantenham estados consistentes.

3. **(FGV – PGE-AM – 2022)** Considere o trecho de programa Java abaixo.

```
public class Calcular {
    public double reajuste(double salario, double percentual) {
        return salario + salario * percentual / 100;
    }
    public double reajuste(double salario) {
        return salario * 1.30;
    }
    public static double reajuste(float salario) {
        return salario * 0.20;
    }
}
```

É correto afirmar que

a) o código mostra um exemplo de sobrescrita de métodos, pois há vários métodos com o mesmo nome, porém, com parâmetros diferentes.
b) ocorrerá um erro no último método reajuste, pois este não pode ser estático.
c) uma chamada correta ao método reajuste, em condições ideais, é double sal = 1000.5; double c = Calcular.reajuste(sal);
d) ocorrerá um erro na classe Calcular, pois não é permitido criar vários métodos com o mesmo nome, como ocorre com o método reajuste.
e) uma chamada correta ao método reajuste, em condições ideais, é float sal = (float) 1000.5; double c = Calcular.reajuste(sal);

4. **(FGV – DPE-RJ – 2019)** Considere as seguintes afirmativas sobre class constructor na linguagem Java.

I. Deve receber o mesmo nome da classe a ele associada.

II. Não deve ser especificado um tipo de retorno na sua declaração.

III. É útil para a definição de valores iniciais para os atributos da classe.

IV. É sempre declarado como public.

É correto somente o que se afirma em:

a) I e II;
b) II e III;
c) III e IV;
d) I, II e III;
e) I, III e IV.

5. **(FGV – MPE-AL – 2018)** Analise o código Java a seguir.

```
public class Exemplo {
 public static void main(String[] args) {
  int  a = 4, b = 16;
  String out = (a^b)==0 ? "Sim": (a & b)!=0
                  ? "Talvez" : "Não";
  System.out.println(out);
 }
}
```

A execução do código acima produzirá

a) um erro de compilação.
b) um erro de execução.
c) a linha "Sim".
d) a linha "Não".
e) a linha "Talvez".

6. **(FGV – MPE – AL – 2018)** No Java, a classe Error e Exception derivam da classe
a) ClassNotFoundException.
b) IOException.
c) MainException.
d) RuntimeException.
e) Throwable.

7. **(FGV – MPE-AL – 2018)** Sobre as variáveis e os métodos declarados como private, em Java, analise as afirmativas a seguir.

I. Ficam acessíveis somente aos membros da própria classe.

II. Ficam acessíveis somente às classes definidas no mesmo package.

III. Ficam acessíveis somente para suas classes derivadas.

Está correto o que se afirma em

a) I, apenas.

b) II, apenas.
c) III, apenas.
d) I e II, apenas.
e) II e III, apenas.

8. **(FGV – BANESTES – 2018)** Considere a compilação de um ou mais programas por meio da linha de comando, num ambiente Java. Nesse caso, o comando que está corretamente formado para esse fim é:

a) compile teste.java –type java
b) java teste.java
c) javac *.java
d) jvm Teste1.java teste2.java
e) parse java teste.java

9. **(FGV – ALE-RO – 2018)** Analise a definição de um array em Java.

int[ ][ ] multi = new int[5][10];

Assinale o código que exibe corretamente um elemento de cada linha desse array.

a) for (int i = 0; i < 5; i++) { System.out.println(multi[i][0]); };
b) for (int i = 0; i < 10; i++) { System.out.println(multi[i,1]); };
c) for (int i = 0; i <= 5; i++) { System.out.println(multi[i,0]); };
d) for (int i = 0; i <= multi.length; i++) { System.out.println(multi[1,i]); };
e) for (int i = 1; i < multi[0].length; i++) { System.out.println(multi[i,1]); };

10. **(FGV – ALE-RO – 2018)** No contexto da linguagem Java, assinale o modificador (modifier) que se refere ao nível de acesso.

a) abstract
b) final
c) protected
d) static
e) volatile

11. **(FGV – ALE-RO – 2018)** Sobre construtores (constructors), no contexto da linguagem Java, analise as afirmativas a seguir.

I. Os construtores devem ser declarados como private.

II. Uma interface não pode ter um construtor.

III. Uma classe abstrata pode ter um construtor.

Está correto o que se afirma em
a) I, somente.
b) II, somente.
c) III, somente.
d) I e II, somente.
e) II e III, somente.

12. **(FGV – MPE AL – 2018)** O package "java.util.regex" do Java permite a manipulação de expressões regulares.

As três classes mais importantes desse pacote são denominadas:
a) RegExp, Parser e Lexicon.
b) RegExPattern, RegExVerifier e RegExCompiler.
c) Pattern, Matcher e PatternSyntaxException.
d) Collector, Retriever e SyntaxParser.
e) Parser, Inspector e Evaluator.

13. **(FGV – MPE – AL – 2018)** Servidores de aplicação baseados em Java Platform Enterprise Edition possuem vários tipos de API.

Assinale a opção que indica a API utilizada para fornecer acesso ao servidor de nomes e diretórios.
a) JNDI.
b) RMI.
c) JMS.
d) JTS.
e) JDBC.

14. **(FGV – IBGE – 2017)** Os servidores de aplicação fornecem a infraestrutura básica necessária para o desenvolvimento e a distribuição de aplicações. Em relação aos servidores de aplicação da plataforma Java EE, analise as afirmativas a seguir:
I. GlassFish e Apache Geronimo são servidores de aplicação open source compatível com a especificação J2EE.

II. O Apache Tomcat não dispõe de um container EJB.

III. JBoss é programado em Java, portanto, pode ser usado em qualquer sistema operacional que suporte essa linguagem de programação.

Está correto o que se afirma em:

a) somente I;
b) somente II;

c) somente III;
d) somente I e II;
e) I, II e III.

15. **(FGV – IBGE – 2017)** Em Java, certo método da classe A deve poder ser invocado independentemente da criação prévia de uma instância de A.

O modificador obrigatório na declaração desse método é:

a) abstract;
b) static;
c) public;
d) protected;
e) final.

Comentários:

Como já dissemos em questões anteriores, os métodos static tem um relacionamento com uma classe como um todo, enquanto os métodos que não são static são associados a uma instância de classe específica (objeto) e podem manipular as variáveis de instância do objeto. Portanto, para classe A poder ser invocado independentemente da criação prévia de uma instância de A o método deve receber o modificador static.

Gabarito: Letra B

LISTA DE QUESTÕES – FCC

16. (FCC – PGE-AM - 2022) Considere o trecho de programa Java abaixo.

```
public class Exemplo1 {

    public static void main(String[] args) {
        int[] vet = {78, 89, 90, 34, 32, 56};
        int[] dados = verificar(vet);
    }

      I
    ......{

        int a, b, x, t, max;
        max = vet.length;
        for (a = 0; a <= max - 1; a++) {
            for (b = 0; b <= max - a; b++) {
                if (b < max - 1) {
                    if (vet[b] > vet[b + 1]) {
                        x = vet[b];
                        vet[b] = vet[b + 1];
                        vet[b + 1] = x;
                    }
                }
            }
        }
        return vet;
    }
}
```

A assinatura correta do método que deve ser colocada na lacuna I é

a) public int[] verificar(int vet[])
b) private int[] verificar(int vet[])
c) public static int verificar(int vet[])
d) public int verificar(int vet)
e) public static int[] verificar(int vet[])

17. (FCC – PGE-AM – 2022) Considere st um objeto do tipo PreparedStatement e conn um objeto do tipo Connection, ambos em condições ideais para a execução do método salvar, de uma classe Java.

```
public boolean salvar(Cliente cliente){
        try {
            st = conn.prepareStatement("I.....");
            st.setString(1, cliente.getCpf());
            st.setString(2, cliente.getNome());
            st.setString(3, cliente.getEndereco());
            st.setString(4, cliente.getBairro());
            st.setString(5, cliente.getCidade());
            st.setString(6, cliente.getTelefone());
            st.setDouble(7, cliente.getRenda());
            st.executeUpdate();
            return true;
        } catch (SQLException ex) {
            return false;
        }
}
```

Nas condições apresentadas, para que o método salvar grave as informações do cliente no banco de dados aberto, a lacuna I deve ser corretamente preenchida com a instrução
a) INSERT INTO cliente VALUES(cpf,nome,endereco,bairro,cidade,telefone,renda)
b) INSERT INTO cliente VALUES(?,?,?,?,?,?,?)
c) INSERT INTO cliente (nome,cpf,endereco,bairro,cidade,telefone,renda) VALUES(?,?,?,?,?,?,?)
d) INSERT INTO cliente VALUES([cpf],[nome],[endereço],[bairro],[cidade],[telefone],[renda])
e) INSERT TO cliente (cpf,nome,endereco,bairro,cidade,telefone,renda) VALUES(?,?,?,?,?,?,?)

18.(FCC – PGE-AM – 2022) No método doPost de uma servlet Java, deseja-se receber um parâmetro de um campo de nome renda contido em um formulário HTML, converter o valor recebido deste campo para número real e armazená-lo em uma variável chamada renda. A instrução correta para realizar esta tarefa é

a) long renda = Long.parseLong(request.getParameter("renda"));
b) double renda = parseDouble(request.getParameter("renda"));
c) double renda = request.getParameter("renda").toDouble();
d) double renda = Double.parseDouble(request.getParameter("renda"));
e) number renda = parseNumber(request.getParameter("renda"));

19.(FCC – AFAP – 2019) Considere o método Java abaixo.

```
public int conectar() {
    try {
        Class.forName("com.mysql.jdbc.Driver");
        con = DriverManager.getConnection("jdbc:mysql://localhost:3306/trf3", "root", "ad3jA");
        st = con.createStatement();
        return 1;
    } catch (.....I..... ex) {
        return 0;
    } catch (.....II..... ex) {
        return 2;
    }
}
```

Caso o driver JDBC não esteja disponível na aplicação e não exista o banco de dados trf3, as exceções que deverão ser tratadas nas lacunas I e II são, correta e respectivamente,

a) NullPointerExceptionException e DatabaseException.
b) DriverNotFoundException e MySQLException.
c) JDBCDriverException e DatabaseException.
d) ClassNotFoundException e SQLException.
e) JDBCDriverException e DatabaseSQLException.

20. (FCC – AFAP – 2019) Para criar array um Analista de Informática digitou os comandos abaixo no método main de uma classe Java em condições ideais.

I. int[] a = {1,3};
II. int b[] = new int[2]; b[0]=1; b[1]=3;
III. int[] c = new int[2]; c.add(1); c.add(2);
IV. int[][] d = {{1,3},{4,5}};
V. int e[][] = {{1,3},{4,5}};

Ao tentar compilar e executar a classe, foi exibida a mensagem "Exception in thread "main" java.lang.RuntimeException: Uncompilable source code". Essa mensagem foi exibida porque o item

a) IV está incorreto, pois não especifica o tamanho da array.
b) V está incorreto, pois deve haver apenas um par de colchetes após o nome da array.
c) I está incorreto, pois não é permitido declarar um array e já incluir os valores nele.
d) II está incorreto, pois os colchetes devem ficar antes do nome da variável b.
e) III está incorreto, pois não existe o método add neste contexto.

21. (FCC – AFAP – 2019) Considere a classe Java abaixo, que usa um método de ordenação por inserção para rearranjar os valores em um vetor em ordem crescente.

```
public class Organiza{
   public static void main(String[] args) {
      int k[] = {88, 44, 42, 7, 72, 5};
      int n = k.length;
      ordena(n, k);
   }
   static void ordena(int n, int v[]) {
      for (int j = 1; j < n; ++j) {
         int x = v[j];
         int i;
         for (..I..){
            v[i + 1] = v[i];
         }
         ..II.. ;
      }
   }
}
```

Para que a ordenação por inserção ocorra corretamente, as lacunas I e II devem ser corretamente preenchidas com

a) i=j-1; i >=0 && v[i]>x; – –i e v[i-1]=x
b) i=j-1; i >=0; i++ e v[i+1]=x
c) i=j-1; i >=0 && v[i]>x; – –i e v[i+1]=x
d) i=j-1; i >=0 && v[i]>x; ++i e v[i]=x
e) i=j-1; i >=0 || v[i]>x; – –i e v[i+1]=x

22. (FCC - TRF 3ª Região /2019) No interior do método main da classe principal de uma aplicação Java SE um método foi chamado usando as instruções abaixo.

```
double[] dados;
double [] d={1, 20, 7, 8};
dados = Aluno.cadastraDados(d);
```

O método cadastraDados na classe Aluno deve ter a declaração

a) public static double[] cadastraDados(double[] d)
b) public static double[] cadastraDados(double d)
c) public double[] cadastraDados(double[] d)
d) public final double[] cadastraDados(double d[])
e) public static double cadastraDados(double[] d)

23. (FCC – SANASA – 2019) Na orientação a objetos uma classe abstrata é construída para ser um modelo para classes derivadas e na sua construção há algumas restrições. Assim, considere a classe abstrata abaixo, criada na Linguagem Java.

```
public abstract class Calcula {
    private static final double VALOR=10;
    public abstract double soma(double n1, double n2);
    public abstract void exibeResultado( );
    protected abstract double soma(double n1, double n2, double n3);
    private abstract int multiplica(double n1, int n2);
    private double multiplica(double n1, double n2){return n1*n2;}
    public Calcula( ) { }
}
```

A instrução que NÃO é permitida nessa classe é

a) private abstract int multiplica(double n1, int n2);
b) public abstract void exibeResultado( );
c) public Calcula( ) { }
d) private static final double VALOR=10;
e) private double multiplica(double n1, double n2){return n1*n2;}

24. (FCC – TJ TRF3 – 2019) Em uma aplicação Java, um vetor n foi criado por meio da instrução double n=new double[3]; e alimentado com 3 valores reais. Para exibir o conteúdo da segunda posição (índice) deste vetor utiliza-se a instrução

a) JOptionPane.showMessageDialog(null, n[2]);
b) System.out.println(n[1]);
c) JOptionPane.ShowMessageDialog(n[2]);
d) System.Out.Println(null, n[1]);
e) JOptionPane.showWindows(0, n[2]);

25. (FCC – AFAP – 2019) Considere a classe Java a seguir em condições ideais.

```
import java.sql.*;
public class ClienteDados {
    public int conectar(){
    Class.forName("com.mysql.jdbc.Driver");
    Connection con = DriverManager.getConnection
    ("jdbc:mysql://localhost:3306/banco", "root", "");
    Statement st = con.createStatement();
    return 1;
  }
}
```

No método conectar podem ser lançadas duas exceções que o Analista de Informática deseja que sejam tratadas não no interior do método, mas sim por quem o chamar. Para que isso seja permitido, deve-se inserir o comando

No método conectar podem ser lançadas duas exceções que o Analista de Informática deseja que sejam tratadas não no interior do método, mas sim por quem o chamar. Para que isso seja permitido, deve-se inserir o comando

a) try na linha abaixo da declaração do método e catch (DataBaseException, SQLQueryException) abaixo do comando return 1.
b) throws ClassNotFoundException, SQLException na linha de declaração do método.
c) Exception ClassNotFoundException, SQLException na linha de declaração do método.
d) try na linha abaixo da declaração do método e catch (ClassNotFoundException, SQLException) abaixo do comando return 1.
e) throws DataBaseException, SQLQueryException na linha de declaração do método.

26. (FCC / TRF3 – 2019) Uma classe Colaborador de uma aplicação Java tem o método abaixo.

```
public void setNome(String nome) {
this.nome = nome;
}
```

Para instanciar um objeto dessa classe e armazenar o nome "Pedro" no atributo nome utiliza-se

a) Colaborador.setNome("Pedro");
b) Colaborador c -> new Colaborador(); c->Nome="Pedro";
c) Colaborador c = new Colaborador(c.setNome("Pedro"));
d) Colaborador c = new Colaborador(); c.setNome("Pedro");
e) Colaborador c = new Colaborador().setNome("Pedro");

27. (FCC / SABESP – 2018) As interfaces são usadas nas aplicações Java quando se deseja permitir que diversas classes implementem determinados métodos, mesmo que de formas diferentes. Em uma interface Java

a) os métodos não podem ter os modificadores protected ou private.
b) não pode haver assinaturas de métodos cujo tipo de retorno seja void.
c) pode haver múltiplos construtores, desde que recebam parâmetros diferentes.
d) não pode haver dois ou mais métodos com o mesmo nome, mesmo que recebam parâmetros diferentes.
e) todo método deverá ser implementado por uma das subclasses da aplicação pelo menos uma vez

28. (FCC – TRF 3ª Região – 2014) Herança e interfaces são conceitos da orientação a objetos que possibilitam a obtenção de polimorfismo. Em Java, uma interface pode

a) herdar de mais de uma interface.
b) ser instanciada diretamente por meio da instrução new.

c) possuir apenas assinaturas de métodos públicos e protegidos.
d) possuir métodos abstratos e estáticos.
e) conter declarações de constantes e de variáveis não inicializadas.

29. **(FCC – TRE-RN – 2011)** Em relação ao Java Standard Edition, é INCORRETO afirmar:

a) Possui gerenciamento de memória embutido, por meio do coletor de lixo.
b) Ambiente indicado para o desenvolvimento de aplicativos para dispositivos móveis ou portáteis.
c) Permite o desenvolvimento de aplicações desktop de linha de comando e interfaces gráficas Swing.
d) Portabilidade dos programas compilados para diversos sistemas operacionais, sem necessidade de recompilação.
e) Usa conceitos tais como orientação a objetos e multithreading.

30. **(FCC – TRT 22ª Região – 2010)** A plataforma Java disponibiliza um interpretador que traduz, em tempo de execução, o bytecode para instruções nativas do processador, permitindo, dessa forma, que uma mesma aplicação seja executada em qualquer plataforma computacional que possua essa implementação. Trata-se de:

a) Java Virtual Machine.
b) Java API.
c) JavaBeans.
d) J2SE.
e) JavaFX.

31. **(FCC – Sergipe Gás– 2010)** É tida como uma das principais linguagens de programação orientada a objeto; tem como característica a compilação para um bytecode e execução por uma máquina virtual. Trata-se da linguagem:

a) Algol.
b) Delphi.
c) C++.
d) Java.
e) PHP.

32. **(FCC – TCE-SP – 2010)** Os aplicativos Java "rodam" em diferentes ambientes. A tradução dos códigos Java (bytecode), para instruções específicas de cada sistema e dispositivo, é uma função do programa:

a) Java Community Process (JCP).
b) Java Virtual Module (JVM).
c) Java Virtual Machine (JVM).
d) Java Comunication Process (JCP).

e) Java Enterprise Machine (JEM).

LISTA DE QUESTÕES – CESPE

33. (CESPE – Petrobrás – 2022)

```
package cadastroUsuario;


import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.*;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;


public class CadastroServlet extends HttpServlet
{
   protected void doPost(HttpServletRequest
     request,HttpServletResponse response)
   throws ServletException, IOException {
   String nome = request.getParameter("nome");
String endereco =
request.getParameter("endereco");
  String telefone =
   request.getParameter("telefone");
  response.setContentType("text/html");
  PrintWriter out = response.getWriter( );
  out.println("<HTML>");
  out.println("<HEAD>");
  out.println("<STYLE>");
  out.println("background-color: green;");
  out.println("</STYLE>");
  out.println("</HEAD>");
  out.println("<BODY>");
  out.println("<P>Prezado(a) ", " + nome + "
</P>");
  out.println("<P>Obrigado por se
cadastrar.</P>");
  out.println("<P>Entraremos em contato através
do telefone", " + telefone+ " em breve.</P>");
  out.println("</BODY>");
  out.println("</HTML>");
  out.close( );
```

```
  }
}
```

Tendo como referência o código precedente, julgue o item que se segue.

A linha
String nome = request.getParameter("nome");

pode ser alterada para
String nome = request.getAttribute("nome");

sem perda de funcionalidade no código.

34. **(CESPE - DP DF - 2022)** No Java 8, o uso do método map() permite aplicar uma função a todos os elementos de um stream.

35. **(CESPE - DP DF– 2022)** Quando a mensageria é utilizada com linguagem Java, as requisições são processadas exclusivamente de forma síncrona.

36. **(CESPE – SEED PR –2021)** Java é uma linguagem construída a partir de um legado das linguagens C e C++. No entanto, ela apresenta características únicas que a diferem das demais, como:

I. o applet, que é um tipo especial de programa Java projetado para ser transmitido pela Internet e executado por um navegador web compatível com Java.

II. a saída de seu compilador não gera um código executável e, sim, um bytecode.

III. o fato de um programa Java ser executado somente pela Java virtual machine (JVM).

Assinale a opção correta.

a) Apenas o item I está certo.
b) Apenas o item II está certo.
c) Apenas os itens I e III estão certos.
d) Apenas os itens II e III estão certos.
e) Todos os itens estão certos.

37. **(CESPE – PGDDF – 2021)** Julgue o item subsecutivo, a respeito de lógica e de estrutura de programação.

função avaliar(a, b)

```
início
    ma <-a;
    se (ma < b) então ma <- b;
    me <-a;
    se (me > b) então me <- b;
  resultado <- ( ma % me );
  se (resultado = 0)
      então retorne me
      senão avaliar(me, ma)
fim
escreva avaliar (120,30);
```

O resultado do pseudocódigo precedente será 120.

38. **(CESPE – TJ-RJ – 2021)** Considere o trecho de código a seguir, de um programa escrito na linguagem Java.

```
1   public class Carro
2   {
3     private String modelo;
4     private int cilindradas;
5     public Carro(String modelo, int cilindradas)
6     {
7       this.modelo = modelo;
8       this.cilindradas = cilindradas;
9     }
...
15  }
```

Quanto a esse contexto, considere, ainda, as instruções a seguir, da classe `CarroEsportivo`.

```
1  public class CarroEsportivo extends Carro
2  {
3          private String marca;
4          public CarroEsportivo(String modelo,  int cilindradas, String marca)
5          {
6                  super(modelo, cilindradas);
7                  this.marca = marca;
8          }
9  }
```

Com relação aos trechos de código precedentes, é correto afirmar que

a) o parâmetro this, na linha 7 da classe CarroEsportivo, é uma palavra reservada usada para mostrar que está sendo feita referência ao atributo privado marca da classe Carro.
b) o comando this, na linha 7 da classe CarroEsportivo, é uma palavra reservada que faz referência ao atributo público marca da classe Carro.
c) a instrução super, na linha 6 da classe CarroEsportivo, é usada para importar o pacote atributos privados da classe Carro, a fim de se tornarem públicos para a classe CarroEsportivo.
d) a instrução super, na linha 6 da classe CarroEsportivo, permite à classe CarroEsportivo acessar os atributos privados da classe Carro.
e) a instrução super, na linha 6 da classe CarroEsportivo, pode permitir à classe CarroEsportivo acessar atributos públicos da classe Carro.

39. **(CESPE – PGDF – 2021)** Com relação a servidores de aplicação e seus serviços e a teoria de becapes, julgue o item a seguir.

Apenas uma única instância do Tomcat pode existir em uma única JVM (Java virtual machine); no entanto, é possível ter múltiplas instâncias em uma mesma máquina virtual, desde que em processos Java separados, em execução em portas separadas.

40. **(CESPE -– Ministério da Economia – 2020)** Em Java 8, o método de limite de fluxo tem como objetivo eliminar elementos com base em um critério.

41. **(CESPE – Ministério da Economia – 2020)** Uma expressão lambda é usada principalmente para definir a implementação procedural de uma interface associativa.

Comentários:

42. **(CESPE – Pref. Barra dos Coqueiros – 2020)** A máquina virtual do Java (JVM) utiliza uma tecnologia especificamente para aumentar o desempenho da aplicação, identificando códigos que serão muito executados, normalmente dentro de loops. Essa tecnologia é chamada de

a) hotspot.
b) bytecode.
c) compilação estática.
d) JRE (Java Runtime Environment).
e) JDK (Java Development Kit).

43. **(CESPE – MPE-CE – 2020)** Com base na linguagem de programação Java, julgue o item a seguir.

O código

```
public class mpce {
 public static void main(String[] args) {
  int[][] num = { {3, 5}, {9, 7} };
  for (int i = 0; i < num.length; ++i) {
   for(int j = 0; j < num[i].length; ++j) {

System.out.println(num[i][j]);
   }
  }
 }
}
```

apresentará, quando executado, o resultado a seguir.

3
5
7
9

44. (CESPE – MPE-CE – 2020) Com base na linguagem de programação Java, julgue o item a seguir.

O código

```
public class mpce {

public static int Funcao(int num, int resultado){
  if(num<1)
    return resultado;
  resultado+=(num%10);
  return Funcao(num/10, resultado);
}

public static void main(String[] args) {

  int res = Funcao(17, 0);
  System.out.println("O resultado é: " +
res);
 }
}
```

apresentará a seguinte saída ao ser executado.

O resultado é: 8

45. (CESPE – TJPA – 2020)

```
class GeraNumeros {
  public static void main (String args []) {
    int num;
    num = 36;
    for(int i=0; i < num; i++) {
      if(i*i >= num) break;
      System.out.print(i + " ");
    }
   }
}
```

Assinale a opção que apresenta corretamente a saída gerada pelo código Java precedente.

a) 36
b) 1 2 3 4 5 6
c) 1 2 3 4 5
d) 0 1 2 3 4 5
e) 0 1 2 3 4 5 6

46. **(CESPE – STM – 2018)** Os membros de uma classe definidos como PUBLIC não podem ser acessados ou usados por nenhuma outra classe.

47. **(CESPE – CGM-Joao Pessoa– 2018)** A JME oferece um ambiente robusto e flexível para aplicativos executados em dispositivos móveis e integrados cuja capacidade de memória, de vídeo e de processamento são limitados, tais como set-top boxes e reprodutores de discos blu-ray.

48. **(CESPE – TER-TO– 2017)** Na orientação a objetos, a alteração do comportamento dos métodos herdados das superclasses para um comportamento mais específico nas subclasses, de forma a se criar um novo método na classe filha que contém a mesma assinatura e o mesmo tipo de retorno, relaciona-se a

a) sobrecarga.
b) overloading.
c) portabilidade.
d) abstração.
e) sobrescrita.

49. **(CESPE – TCE-PA – 2016)** O fato de as expressões lambda terem sido excluídas do Java 8 possibilitou que a linguagem Java ficasse mais flexível e que a implementação de seu código se tornasse mais precisa.

50. **(CESPE – FUNPRESP-JUD – 2016)** Para lidar com um conjunto de objetos em JSon, é necessário utilizar um array que permita realizar, em uma única operação, a carga de todos os objetos.

51. **(CESPE – TRE-PI – 2016)** A linguagem Java foi, originalmente, desenvolvida para

a) permitir a comunicação entre aparelhos domésticos e computadores.
b) traduzir fórmulas matemáticas utilizando-se cartões perfurados.
c) processar valores inteiros, em um ambiente negócios, em computadores de grande porte.
d) trabalhar com inteligência artificial por meio de uma abordagem puramente lógica.
e) demonstrar a viabilidade da implementação da álgebra relacional de dados.

52. **(CESPE – TER-GO – 2015)** Em um grupo do tipo Array, podem-se armazenar dados de diferentes tipos.

53. **(CESPE – TJDFT – 2015)** Na tecnologia JSP (Java Server Pages), cada documento é convertido pelo contêiner JSP em um servlet, o que ajuda a separar a apresentação do conteúdo.

54. **(CESPE – SERPRO – 2013)** A tecnologia JSE (Java Small Editon) utilizada pela plataforma iOS permite o desenvolvimento de aplicações para todos os seus dispositivos, como estações gráficas, iPad, iPod, entre outros.

55. **(CESPE – MPOG – 2013)** O JME foi criado para trabalhar com ambientes de programação multiprocessados em ambientes distribuídos.

56. **(CESPE – ANTT – 2013)** JSE, JME, JEE e JCE são edições da tecnologia Java voltadas para o desenvolvimento de aplicações para desktop/servidores, dispositivos móveis, ambientes corporativos e ambientes em nuvem, respectivamente.

57. **(CESPE – TRT 10ª Região – 2013)** O uso de System.out.println() possibilita a exibição de textos; para a exibição de valores de variáveis, é necessário utilizar showAttributes().

Comentários:

58. **(CESPE – TRT 10ª Região – 2013)** No código abaixo, caso a variável salário assuma o valor igual a 5000, o valor b da variável avaliação será atribuído corretamente.

```
if (salario > 10000) {
avaliacao=”a”;
else
avaliacao=”b”;
}
```

59. **(CESPE – TRT 10ª Região – 2013)** A execução do código abaixo informará a classe do objeto que foi atribuído à variável de nome var4.

```
String nome = var4.getname();
```

60. **(CESPE – MPU – 2013)** O tratamento de exceção em Java consiste no bloco try-catch-finally. O bloco finally sempre executa ao término do bloco try.

61. **(CESPE – MPU – 2013)** Se a thread executando um código try ou catch for interrompida ou abortada, o bloco finally poderá não ser executado, apesar de a aplicação como um todo continuar.

62. **(CESPE – TRE-MA – 2009)** Para definição e manipulação de uma exceção em Java, devem constar no programa, obrigatoriamente, os termos:

a) try e catch.

b) try e finally.
c) finally e catch.
d) finally e retry.
e) try e retry.

63. **(CESPE – SERPRO – 2008)** A linguagem Java, orientada a objetos, tem como característica ser compilada em um código executado em máquina virtual.

64. **(CESPE – TRT - 5ª Região – 2008)** A instrução import Java.awt.* indica que o programa irá utilizar componentes gráficos.

65. **(CESPE – TRT - 5ª Região – 2008)** Em Java, os métodos public de uma classe são utilizados pelos clientes da classe para manipular dados armazenados em objetos dessa classe.

LISTA DE QUESTÕES – OUTRAS BANCAS

66. (UFMA – UFMA – 2019) Duas características importantes e relacionadas entre si, presentes em Java por ser uma linguagem orientada a objetos, são a herança e o polimorfismo. Considere as afirmativas I e II a seguir e depois marque a alternativa correta.

I. Herança múltipla é um recurso existente em Java para permitir que uma classe possa herdar atributos e métodos de mais de uma classe.

II. Polimorfismo em Java é a capacidade de duas ou mais classes derivadas de uma mesma superclasse possuírem a mesma assinatura de um método, porém com comportamento diferente.

a) Apenas a afirmativa II está correta.
b) Ambas as afirmativas I e II estão corretas.
c) Ambas as afirmativas I e II estão erradas.
d) Apenas a afirmativa I está correta.
e) A correção ou não das afirmativas I e II depende de qual versão de Java se está levando em consideração.

67. (FUNDEP – Prefeitura de Lagoa Santa– 2019) Qual é a forma correta de se criar uma classe que não poderá ser instanciada, mas ainda poderá ser reutilizada?

a) public abstract class NomeDaClasse
b public abstract NomeDaClasse
c) public class NomeDaClasse
d) public class private NomeDaClasse

68. (UFRN – UFRN – 2019) Na programação orientada a objetos, o polimorfismo é a habilidade de objetos de classes diferentes responderem à mesma mensagem de maneiras diferentes. Um tipo de polimorfismo é

a) polimorfismo de sobrecarga.
b) polimorfismo de instanciação.
c) polimorfismo de abstração.
d) polimorfismo de classificação.

69. (IADES – Hemocentro – DF– 2017) Considerando que Luta e Jogo são classes, e que Esporte, Individual e Coletivo são interfaces, com relação aos conceitos de classes e interfaces da linguagem Java, assinale a alternativa correta.

a) interface Futebol implements Esporte, Jogo{}.

b) interface Futebol extends Esporte,Coletivo{}
c) interface Futebol implements Esporte{}
d) class Judo extends Esporte, Individual{}
e) interface Karate extends Luta{}

70. (ESAF – Receita Federal – 2012) Em programação Java, o comando while:

a) executa um bloco exclusivamente de comandos de atribuição.
b) executa um bloco de comandos enquanto sua condição for verdadeira.
c) executa um bloco de comandos até que sua condição seja verdadeira.
d) equivale ao comando what-if.
e) é idêntico ao comando do while.

GABARITO

1. Letra C
2. Letra E
3. Letra E
4. Letra D
5. Letra D
6. Letra D
7. Letra A
8. Letra C
9. Letra A
10. Letra C
11. Letra E
12. Letra C
13. Letra A
14. Letra E
15. Letra B
16. Letra E
17. Letra B
18. Letra D
19. Letra D
20. Letra E
21. Letra C
22. Letra A
23. Letra A
24. Letra B
25. Letra B
26. Letra B
27. Letra A
28. Letra A
29. Letra B
30. Letra A
31. Letra D
32. Letra C
33. Errada
34. Correto
35. Errado
36. Letra E
37. Errado
38. Letra E
39. Errado
40. Errado
41. Errado
42. Letra A
43. Errado
44. Correto
45. Correto
46. Errado
47. Correto
48. Letra E
49. Errado
50. Errado
51. Letra A
52. Errado
53. Correto
54. Errado
55. Errado
56. Errado
57. Errado
58. Correto
59. Errado
60. Errado
61. Correto
62. Letra A
63. Correto
64. Correto
65. Correto
66. Letra A
67. Letra A
68. Letra A
69. Letra B
70. Letra B

QUESTÕES COMENTADAS – JAVA – CONCEITOS BÁSICOS - CEBRASPE

1. **(CESPE - 2013 - SERPRO - Analista - Desenvolvimento de Sistemas) Garbage Collector é a tecnologia que gerencia a memória alocada para o programa, a fim de liberar objetos que não estão sendo utilizados.**

Comentários:

O acesso a arrays e strings, e a conversão de tipos são checados em tempo de execução para assegurar a sua validade. O Garbage Collector faz a desalocação automática de memória evitando, erros de referência e desperdício de memória. Finalmente, o recurso de Exception Handling permite o tratamento de erros em tempo de execução, por um mecanismo robusto, análogo ao do C++.

Conforme vimos em aula, essa é a função precípua do Garbage Collector! **Gabarito: C**

2. **(CESPE - 2012 - TRE-RJ - Técnico Judiciário - Programação de Sistemas) A linguagem de programação Java é muito utilizada por ter como característica gerar um código independente de plataforma que pode ser executado em qualquer arquitetura e sistema operacional que tenha o sistema Java.**

Comentários:

Vocês sabiam que Java é uma Linguagem WORA? Pois é, esse acrônimo significa Write Once, Run Anywhere ou Escreva uma vez, execute em qualquer lugar. Trata-se de um slogan para exemplificar os benefícios multiplataforma da linguagem Java! Idealmente, isso significa que um programa em Java (uma vez compilado em um bytecode) pode rodar em qualquer equipamento que possua uma JVM!

Conforme vimos em aula, seria mais correto dizer Java Virtual Machine (JVM) e, não, Sistema Java. No entanto, a questão está correta. **Gabarito: C**

3. **(CESPE - 2012 - Banco da Amazônia - Técnico Científico - Análise de Sistemas) No Java, a JRE possui tudo que é necessário para desenvolver programas em Java.**

Comentários:

Portanto, é o seguinte: se você deseja somente executar alguma aplicação Java no seu computador ou navegador, basta instalar um JRE! No entanto, se você planeja programar em Java, você precisará de um JDK (que contém a JRE)! Entenderam? É bastante simples! JRE é o mínimo que você precisa para rodar uma aplicação e o JDK é o que você precisa para desenvolver uma aplicação!

Conforme vimos em aula, ela possui tudo que é necessário para executar programas em Java. Para desenvolver programas, seria necessária uma JDK! **Gabarito: E**

4. (CESPE - 2012 - Banco da Amazônia - Técnico Científico - Análise de Sistemas) Em Java, para toda classe, método e variável de instância que se declara há um controle de acesso, independentemente de o controle ser explicitamente indicado.

Comentários:

Especificador	Própria Classe	Subclasse	Pacote	Global
Private (-)	Sim	Não	Não	Não
<vazio> (~)	Sim	Não*	Sim	Não
Protegido (#)	Sim	Sim	Sim	Não
Público (+)	Sim	Sim	Sim	Sim

Conforme vimos em aula, está perfeito! Por que ele diz "independentemente de o controle ser explicitamente indicado"? Porque quando não se indica o modificador de acesso, assume-se que é Pacote ou Default. **Gabarito: C**

5. (CESPE - 2012 - Banco da Amazônia - Técnico Científico - Análise de Sistemas) O operador instanceof só pode ser usado para testar valores null.

Comentários:

Palavras	Descrição
Instanceof	Testa se um objeto é uma instância de uma classe específica ou se é null.

Conforme vimos em aula, ele também pode ser utilizado para testar se um objeto é instância de uma classe específica. **Gabarito: E**

6. (CESPE - 2012 - MPE-PI - Analista Ministerial - Informática - Cargo 6) Em Java, qualquer método de uma classe pode ser sobrescrito por métodos de outra classe.

Comentários:

```
//Indica que esse método não possui corpo
abstract int soma (int a, int b) {//...//}

//Indica que esse método não pode ser sobrescrito
final int soma (int a, int b) {//...//}

//Indica que esse método só pode acessar atributos de classe e não pode ser sobrescrito
static int soma (int a, int b) {//...//}

//Indica que esse método foi escrito outra linguagem
native int soma (int a, int b) {//...//}

//Indica que esse método só é executável por uma thread por vez
synchronized soma (int a, int b) {//...//}
```

Não, métodos finais, estáticos e privados não podem ser sobrescritos por métodos de outra classe. **Gabarito: E**

7. (CESPE - 2011 - TJ-ES - Técnico de Informática - Específicos) O JVM (Java Virtual Machine) é um interpretador que atribui portabilidade à linguagem Java, possibilitando, consequentemente, a sua execução em qualquer sistema operacional.

Comentários:

Java é também uma linguagem portável e multiplataforma! O Compilador é capaz de gerar um código intermediário (bytecode), que permite que o mesmo programa possa ser executado em qualquer máquina ou sistema operacional que possua uma JVM. Ademais, busca que todos os aspectos da linguagem sejam independentes de plataforma (Ex: ela especifica o tamanho e comportamento de cada tipo de dado).

Perfeito, desde que haja uma Java Virtual Machine (JVM) específica para esse Sistema Operacional. Portanto, a questão está incompleta, mas não é bom brigar com a banca. **Gabarito: C**

8. (CESPE - 2011 - TRE-ES - Técnico - Programação de Sistemas – Específicos) O encapsulamento em Java somente pode ser realizado por meio do modificador de acesso protegido.

Comentários:

Especificador	Própria Classe	Subclasse	Pacote	Global
Private (-)	Sim	Não	Não	Não
<vazio> (~)	Sim	Não*	Sim	Não
Protegido (#)	Sim	Sim	Sim	Não
Público (+)	Sim	Sim	Sim	Sim

Não, pode ser Público, Privado, Protegido ou Padrão. **Gabarito: E**

9. (CESPE - 2010 - TRT - 21ª Região (RN) - Técnico Judiciário - Tecnologia da Informação) A linguagem de programação Java, em razão de sua portabilidade — uma vez que o compilador Java converte o código fonte em bytecodes, executados por uma máquina virtual — é bastante utilizada para oferecer conteúdos dinâmicos na Web.

Comentários:

O bytecode é um código intermediário, que é posteriormente interpretado e executado por uma Java Virtual Machine (JVM). O que é isso, professor? É um programa que carrega e executa os aplicativos Java, convertendo bytecodes em código executável. Lembram que eu falei que Java é uma Linguagem WORA? Pois é, isso ocorre em grande parte por conta do bytecode e da Máquina Virtual Java.

Conforme vimos em aula, está perfeito! Para tal, utiliza-se o Java EE. **Gabarito: C**

10. (CESPE - 2010 - TRE-BA - Técnico Judiciário - Programação de Sistemas) Em programação orientada a objetos, o pacote tem como função agrupar classes dentro de um grupo. Em Java, o pacote Swing (javax.swing) é composto de várias classes para a implementação de interfaces gráficas em desktop.

Comentários:

A Plataforma Java oferece recursos para construção de interfaces gráficas de usuário (GUI), entre eles: AWT (java.awt) e Swing (javax.swing)! O primeiro é um conjunto básico de classes e interfaces que definem os componentes de uma janela desktop. Já o Swing é um conjunto sofisticado de classes e interfaces que definem os componentes visuais necessários para construir uma interface gráfica de usuário.

Conforme vimos em aula, a questão está correta! **Gabarito: C**

11. (CESPE - 2009 - ANAC - Analista Administrativo - Tecnologia da Informação) Pelo uso de polimorfismo, uma chamada de método pode fazer que diferentes ações ocorram, dependendo do tipo do objeto que recebe a chamada.

Comentários:

A palavra Polimorfismo vem do grego: muitas formas. Trata-se da capacidade de um objeto poder se comportar de diversas formas dependendo da mensagem recebida. Observem que isso não quer dizer que o objeto fica transformando seu tipo a todo momento. Na verdade, um objeto nasce com um tipo e morre com esse mesmo tipo. O que muda, então? É a forma como nós nos referimos a esse objeto!

Conforme vimos em aula, dependendo do tipo de objeto (se é da Classe-Pai ou da Classe-Filho), ações diferentes podem ocorrer. **Gabarito: C**

12. (CESPE - 2010 - EMBASA - Analista de Saneamento - Analista de Tecnologia da Informação - Desenvolvimento) O trecho de código a seguir está incorreto porque uma variável booleana em Java usa a sintaxe == e não =.

```
Public void disconnect() {
    Connected = false;
}
```

Comentários:

ARITMÉTICOS	ATRIBUIÇÃO	RELACIONAIS	LÓGICOS	BIT A BIT
+	=	>	!	&
-	+=	<	&&	|
*	-=	>=	||	^
/	*=	<=		<<
%	/=	!=		>>
	%=	==		>>>
	++	?		
	--	instanceof		

Conforme vimos em aula, o operador de atribuição (=) é diferente do operador relacional (==). **Gabarito: E**

13. (CESPE - 2009 - TRE-MA - Técnico Judiciário - Programação de Sistemas) Para definição e manipulação de uma exceção em Java, devem constar no programa, obrigatoriamente, os termos:

a) try e catch.

b) try e finally.

c) finally e catch.

d) finally e retry.

e) try e retry.

Comentários:

É útil para liberar recursos do sistema quando utilizamos, por exemplo, conexões de banco de dados e abertura de buffer para leitura ou escrita de arquivos. finally virá após os blocos de catch. Portanto, o try indica que um bloco de código pode ocorrer erro; o catch tem o objetivo de capturar, manipula e trata erros; e o finally busca realizar ações mesmo após a captura de erros. Vejamos a estrutura básica:

Conforme vimos em aula, utilizam-se Try e Catch! **Gabarito: A**

14. (CESPE - 2009 - TRT - 17ª Região (ES) - Técnico Judiciário - Tecnologia da Informação) Ao contrário dos tipos primitivos que não são objetos, os tipos de objetos são determinados pela classe de origem.

Comentários:

Perfeito, perfeito, perfeito! **Gabarito: C**

15. (CESPE - 2009 - TRT - 17ª Região (ES) - Técnico Judiciário - Tecnologia da Informação) Uma classe final indica uma classe que não pode ser estendida. Um método final não pode ser redefinido em classes derivadas.

Comentários:

```
//Indica que esse método não pode ser sobrescrito
final int soma (int a, int b) {//...//}
```

```
//Essa classe não pode ser estendida
final class Carro { //...// }
```

Conforme vimos em aula, a classe final não pode ter filhos e Método Final não pode ser sobrescrita. **Gabarito: C**

16. (CESPE - 2008 - SERPRO - Analista - Desenvolvimento de Sistemas) A linguagem Java, orientada a objetos, tem como característica ser compilada em um código executado em máquina virtual.

Comentários:

Java é também uma linguagem portável e multiplataforma! O Compilador é capaz de gerar um código intermediário (bytecode), que permite que o mesmo programa possa ser executado em qualquer máquina ou sistema operacional que possua uma JVM. Ademais, busca que todos os aspectos da linguagem sejam independentes de plataforma (Ex: ela especifica o tamanho e comportamento de cada tipo de dado).

Conforme vimos em aula, está perfeito! Qual o nome desse código? Bytecode**! Gabarito: C**

17. (CESPE - 2008 - TRT - 5ª Região (BA) - Técnico Judiciário - Tecnologia da Informação) A instrução import Java.awt.* indica que o programa irá utilizar componentes gráficos.

Comentários:

Essa questão é polêmica! Alguns afirmar que é possível inserir a instrução, mas não utilizar componentes gráficos. Não faria sentido importar um pacote para não utilizar suas funcionalidades, mas é possível – apesar de má prática! Eu acredito que a questão cabe recurso, sim! **Gabarito: C**

18. (CESPE - 2008 - TRT - 5ª Região (BA) - Técnico Judiciário - Tecnologia da Informação) Em Java, os métodos public de uma classe são utilizados pelos clientes da classe para manipular dados armazenados em objetos dessa classe.

Comentários:

Pessoal, esses Modificadores de Acesso determinam quão acessíveis são esses elementos. Vamos vê-los agora em mais detalhes:

<public>: essa instrução indica que a classe, método ou atributo assim declaradas podem ser acessadas em qualquer lugar e a qualquer momento da execução do programa – é o modificador menos restritivo.

Perfeito, são públicos para toda e qualquer classe. **Gabarito: C**

19. (CESPE - 2013 - SERPRO - Analista - Desenvolvimento de Sistemas) A tecnologia JSE (Java Small Editon) utilizada pela plataforma iOS permite o desenvolvimento de aplicações para todos os seus dispositivos, como estações gráficas, iPad, iPod, entre outros.

Comentários:

Java Micro Edition (Java ME): trata-se do padrão aplicado a dispositivos compactos ou móveis, como smartphones, tablets, controles remotos, etc. Permite o desenvolvimento de softwares embarcados, i.e., aplicações que rodam em um dispositivo de propósito

específico, desempenhando alguma tarefa útil. Em geral, possuem limitações de recursos como memória ou processamento.

Conforme vimos em aula, a questão está cheia de erros! Primeiro, JSE é a sigla de Java Standard Edition. Segundo, a tecnologia proposta no item é o JME! **Gabarito: E**

20. (CESPE - 2013 - MPOG - Tecnologia da Informação) O JME foi criado para trabalhar com ambientes de programação multiprocessados em ambientes distribuídos.

Comentários:

Java Micro Edition (Java ME): trata-se do padrão aplicado a dispositivos compactos ou móveis, como smartphones, tablets, controles remotos, etc. Permite o desenvolvimento de softwares embarcados, i.e., aplicações que rodam em um dispositivo de propósito específico, desempenhando alguma tarefa útil. Em geral, possuem limitações de recursos como memória ou processamento.

Java Enterprise Edition (Java EE): trata-se do padrão para desenvolvimento de sistemas corporativos, voltada para aplicações multicamadas, baseadas em componentes executados em servidores de aplicações – ele inclui o Java SE. Contém bibliotecas para acesso a base de dados, RPC, CORBA, entre outras. As aplicações podem ou não estar na internet.

Conforme vimos em aula, Java ME possui diversas limitações de recursos. Na verdade, é o Java EE que é ideal para ambientes distribuídos. **Gabarito: E**

21. (CESPE - 2013 - ANTT - Analista Administrativo - Desenvolvimento de Sistemas da Informação) JSE, JME, JEE e JCE são edições da tecnologia Java voltadas para o desenvolvimento de aplicações para desktop/servidores, dispositivos móveis, ambientes corporativos e ambientes em nuvem, respectivamente.

Comentários:

Java Micro Edition (Java ME): trata-se do padrão aplicado a dispositivos compactos ou móveis, como smartphones, tablets, controles remotos, etc. Permite o desenvolvimento de softwares embarcados, i.e., aplicações que rodam em um dispositivo de propósito específico, desempenhando alguma tarefa útil. Em geral, possuem limitações de recursos como memória ou processamento.

Conforme vimos em aula, está quase tudo certo! No entanto, JCE é uma API de Criptografia (Java Cryptography Extension). Não se trata de uma plataforma ou ambiente de desenvolvimento em nuvem! **Gabarito: E**

22. (CESPE - 2008 - HEMOBRÁS - Técnico de Informática) O Java dá suporte a programação concorrente (multithreading).

Comentários:

Dessa forma, aplicações funcionam da mesma maneira em qualquer ambiente. Podemos dizer que Java é uma linguagem concorrente ou multithreaded, i.e., pode realizar diversas tarefas assincronamente com o uso de threads, que são suportadas de modo nativo. Java torna a manipulação de threads tão simples quanto trabalhar com qualquer variável.

Conforme vimos em aula, Java dá suporte à programação concorrente. Observem que a própria questão escreveu "a" sem crase. As bancas não ajudam :-(**Gabarito: C**

23. (CESPE - 2009 - ANAC - Técnico Administrativo - Informática) A linguagem de programação Java permite operações de bit, como, por exemplo, AND (&) e OR (|).

Comentários:

ARITMÉTICOS	ATRIBUIÇÃO	RELACIONAIS	LÓGICOS	BIT A BIT
+	=	>	!	&
-	+=	<	&&	|
*	-=	>=	||	^
/	*=	<=		<<
%	/=	!=		>>
	%=	==		>>>
	++	?		
	--	instanceof		

Conforme vimos em aula, esses são de fato operadores de bit! **Gabarito: C**

24. (CESPE - 2010 - MPU - Técnico de Informática) Na linguagem Java, um objeto do tipo Integer pode receber valor nulo, porém uma variável primitiva int não pode.

Comentários:

NOME	TIPO	TAMANHO	MÍNIMO	MÁXIMO	DEFAULT
LÓGICO	boolean	-	false	true	false
CARACTERE	char	16 bits	0	$2^{16} - 1$	'\u0000'

INTEIRO	byte	8 bits	-2^7	$2^7 - 1$	0
	short	16 bits	-2^{15}	$2^{15} - 1$	0
	int	32 bits	-2^{31}	$2^{31} - 1$	0
	long	64 bits	-2^{63}	$2^{63} - 1$	0
DECIMAL	float	32 bits	7 Casas Decimais		0.0
	double	64 bits	15 Casas Decimais		0.0

Conforme vimos em aula, int é uma variável primitiva que recebe valores inteiros, i.e., não pode receber valor nulo. Já o tipo Integer é um objeto, logo pode receber valores nulos. **Gabarito: C**

25. (CESPE - 2013 - TRT - 10ª REGIÃO (DF e TO) - Técnico Judiciário - Tecnologia da Informação) É possível indicar que parte de um código em um método pode gerar uma exceção, por meio da utilização da palavra-chave finally.

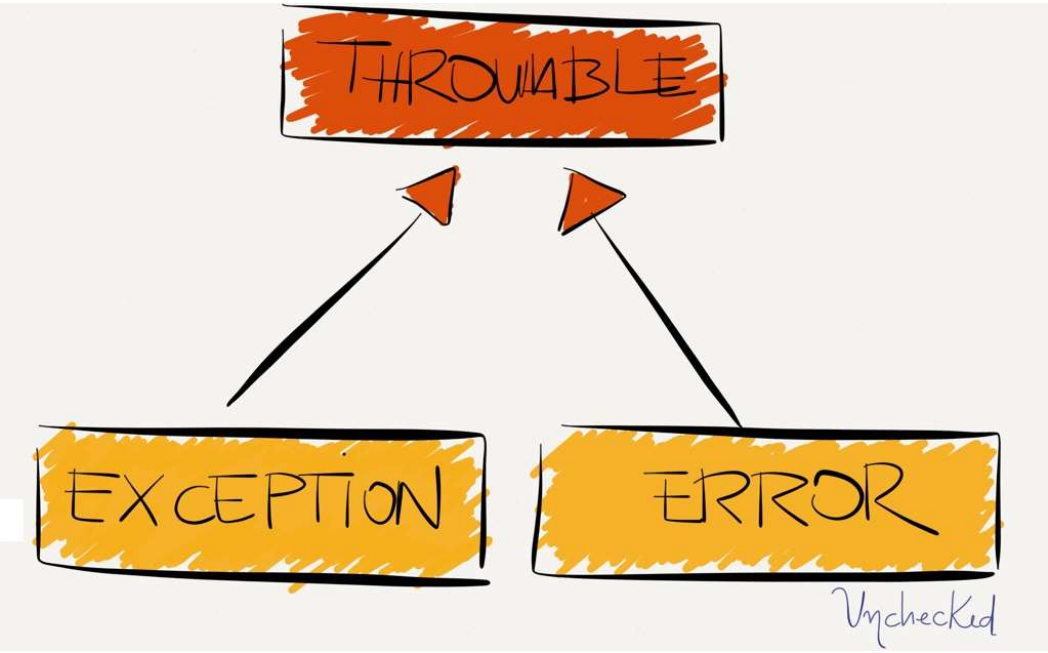
Comentários:

É útil para liberar recursos do sistema quando utilizamos, por exemplo, conexões de banco de dados e abertura de buffer para leitura ou escrita de arquivos. finally virá após os blocos de catch. Portanto, o try indica que um bloco de código pode ocorrer erro; o catch tem o objetivo de capturar, manipula e trata erros; e o finally busca realizar ações mesmo após a captura de erros. Vejamos a estrutura básica:

Conforme vimos em aula, a questão trata do try e, não, do finally. **Gabarito: E**

26. (CESPE - 2014 - ANATEL - Analista Administrativo - Tecnologia da Informação e Comunicação) A linguagem Java possui checked exceptions que estendem java.lang.Exception, em que o compilador força o programador a capturar tais exceções. Essas exceções devem ser tratadas com um bloco try-catch ou com um throws.

Comentários:



A segunda define erros para os quais as aplicações normalmente têm condições de realizar um tratamento, logo Exception e Error são subtipos de Throwable. As exceções ainda se dividem em verificadas (Checked), quando obrigatoriamente devem ser tratadas e não-verificadas (Unchecked), quando não há essa obrigação – programador decide! E como se detectam, manipulam e tratam as exceções?

Conforme vimos em aula, podemos ter vários blocos catch de um único try. Além disso, de fato elas devem tratadas (try-catch) ou propagadas (throws). **Gabarito: C**

27. (CESPE – 2004 – STJ - Analista Judiciário - Análise de Sistemas) O Java collections framework da API Java J2SE possui um conjunto de interfaces e implementações que define estruturas usadas para manipular coleções de objetos. As interfaces fundamentais do framework estão associadas à identificação de funcionalidades típicas de estruturas de dados clássicas. Assim, a interface java.util.List está ligada a estruturas de listas, a interface java.util.Set está associada a estruturas do tipo conjuntos e a interface java.util.Map refere-se a estruturas do tipo mapas. Set, List e Map possuem a interface abstrata java.util.Collection como superinterface.

Comentários:

Por fim, vamos falar sobre os Mapas (java.util.Map). Aqui já começamos diferente, porque mapas não são coleções. Pensem comigo: muitas vezes queremos buscar rapidamente um objeto, dada alguma informação sobre ele (Ex: dada a placa do carro, obter todos os dados). Poderíamos utilizar uma lista e percorrer todos os elementos, mas isso é péssimo para a performance – aqui entra o mapa!

Conforme vimos em aula, mapas não são coleções – não são filhos de java.util.Collection. **Gabarito: E**

28. (CESPE – 2004 – STJ - Analista Judiciário - Análise de Sistemas) As classes java.util.LinkedList e java.util.ArrayList são implementações para listas encadeadas e coleções do tipo arranjos com tamanho modificável, respectivamente, para a interface java.util.List.

Comentários:

Galera, não sei se algum de vocês já programou em C! Se sim, vocês se lembram de como era complicado utilizar ponteiros, ponteiros para ponteiros, alocação dinâmica de memória? Pois é, Java não exige que você faça nada disso! As coisas aqui são muito mais simples e já existe bastante funcionalidade implementada e pronta para ser utilizada. As principais classes implementadoras são: ArrayList, LinkedList, Vector.

Conforme vimos em aula, essas são – de fato – classes implementadoras de java.util.List.
Gabarito: C

29. (CESPE – 2006 – ANCINE – Analista de Sistemas) Os tipos de dados HashSet, ArrayList e HashMap são classes que implementam os tipos de dados Set, List e Map, respectivamente.

Comentários:

Galera, não sei se algum de vocês já programou em C! Se sim, vocês se lembram de como era complicado utilizar ponteiros, ponteiros para ponteiros, alocação dinâmica de memória? Pois é, Java não exige que você faça nada disso! As coisas aqui são muito mais simples e já existe bastante funcionalidade implementada e pronta para ser utilizada. As principais classes implementadoras são: ArrayList, LinkedList, Vector.

Como os conjuntos não possuem ordem, as operações baseadas em índice que existem nas listas não aparecem nos conjuntos. Set é a interface Java que define os métodos que um conjunto deve implementar. As principais classes implementadoras são: HashSet, TreeSet e LinkedHashSet. Cada implementação possui suas características sendo apropriadas para contextos diferentes.

Em um dicionário eu associo um vocábulo a uma definição. A chave é um objeto utilizado para recuperar um valor. O mapa costuma aparecer junto com outras coleções, para poder realizar essas buscas. As principais classes implementadoras da interface Map são: HashMap, TreeMap e LinkedHashMap. Cada implementação possui suas características sendo apropriadas para contextos diferentes.

Conforme vimos em aula, a questão está perfeita! **Gabarito: C**

30. (CESPE – 2006 – SGA/AC – Analista de Sistemas – C) ArrayList implementa a interface List e Collection.

Comentários:

Galera, não sei se algum de vocês já programou em C! Se sim, vocês se lembram de como era complicado utilizar ponteiros, ponteiros para ponteiros, alocação dinâmica de memória? Pois é, Java não exige que você faça nada disso! As coisas aqui são muito mais simples e já existe bastante funcionalidade implementada e pronta para ser utilizada. As principais classes implementadoras são: ArrayList, LinkedList, Vector.

Conforme vimos em aula, ArrayList é uma classe implementadora da Interface List, que por sua vez estende a interface java.util.Collection. ArrayList não implementa Collection diretamente, no entanto a questão não disse se era diretamente ou não – na minha opinião, essa questão deveria ser anulada por ambiguidade. **Gabarito: E**

31. (CESPE – 2008 – MPE/RR - Analista de Sistemas) O pacote java.awt contém diversas classes para criar interfaces gráficas de usuário em aplicações Java.

Comentários:

A Plataforma Java oferece recursos para construção de interfaces gráficas de usuário (GUI), entre eles: AWT (java.awt) e Swing (javax.swing)! O primeiro é um conjunto básico de classes e interfaces que definem os componentes de uma janela desktop. Já o Swing é um conjunto sofisticado de classes e interfaces que definem os componentes visuais necessários para construir uma interface gráfica de usuário.

Conforme vimos em aula, trata-se de um pacote de classes para criar interfaces gráficas de usuário em Java. **Gabarito: C**

32. (CESPE – 2008 – MPE/RR - Analista de Sistemas) No AWT, um componente é qualquer classe que possa ser representada em uma tela sem interação com usuário.

Comentários:

A Plataforma Java oferece recursos para construção de interfaces gráficas de usuário (GUI), entre eles: AWT (java.awt) e Swing (javax.swing)! O primeiro é um conjunto básico de classes e interfaces que definem os componentes de uma janela desktop. Já o Swing é um conjunto sofisticado de classes e interfaces que definem os componentes visuais necessários para construir uma interface gráfica de usuário.

Conforme vimos em aula, trata-se de uma interface com o usuário, logo interage com o usuário. **Gabarito: E**

33. (CESPE – 2008 – MPE/RR - Analista de Sistemas) O Swing suporta o desenvolvimento de interfaces gráficas para usuários (GUI) com o uso da IDE NetBeans

Comentários:

A Plataforma Java oferece recursos para construção de interfaces gráficas de usuário (GUI), entre eles: AWT (java.awt) e Swing (javax.swing)! O primeiro é um conjunto básico de classes e interfaces que definem os componentes de uma janela desktop. Já o Swing é um conjunto sofisticado de classes e interfaces que definem os componentes visuais necessários para construir uma interface gráfica de usuário.

Conforme vimos em aula, a questão está correta! Qual a IDE utilizada pouco importa, o swing suporta o desenvolvimento de GUI e ponto final! **Gabarito: C**

34. (CESPE – 2011 – MEC - Analista de Sistemas) AWT é um conjunto de componentes GUI, integrante da arquitetura JEE6, constante no pacote java.awt, desenvolvidas para substituir as GUIs do SWING.

Comentários:

O AWT (javax.awt) veio primeiro, é mais pesado, é gerado pelo sistema operacional, logo é dependente de plataforma. O Swing (javax.swing) é mais leve, é gerado por uma Máquina Virtual Java (JVM), logo é independente de plataforma. Galera, nem tudo é diferente! Vejam: ambos são fáceis de programar, porque a orientação a objetos proporciona alterar partes do programa, sem alterar toda a estrutura.

Conforme vimos em aula, AWT veio primeiro que o Swing! Esse último veio dar mais dinamismo e leveza aos componentes – hoje em dia, pouco se usa de AWT! **Gabarito: E**

35. (CESPE – 2015 – TCU - Analista de Sistemas) No contexto de um código na linguagem Java, o comando a seguir é utilizado com a finalidade específica de passar para o campo de visibilidade pública saldo o valor da variável deposito com o tipo double. public void saldo (double deposito).

Comentários:

A redação ficou um pouco confusa, mas vejam que a questão fala em passar para campo. Que campo? O saldo é um método e, não, um campo. **Gabarito: E**

36. (CESPE – 2014 – ANATEL - Analista de Sistemas) No JSE (Java Standard Edition) versão 8, é possível utilizar recursos inerentes à programação funcional por meio de uma nova característica da linguagem chamada expressões Lambda, que permitem o tratamento de funções como argumentos de métodos.

Comentários:

Pessoal, vocês já devem saber que nós temos uma nova versão! A cobrança ainda é raríssima (só encontrei uma questão), mas é bom saber um pouco sobre as principais novidades. A grande novidade foram as Lambda Expressions! Pois é, Java agora tem

algumas características de programação funcional. Professor, o que é exatamente programação funcional?

É um paradigma de programação que trata a computação como uma avaliação de funções matemáticas e que evita estados ou dados mutáveis. Ela enfatiza a aplicação de funções, em contraste com a programação imperativa, que enfatiza mudanças no estado do programa. As Lambda Expressions permitem passar comportamentos, ou funções como argumentos em uma chamada de método.

Conforme vimos em aula, a questão está perfeita! **Gabarito: C**

37. (CESPE – 2017 – TRE/BA - Analista de Sistemas)

```
1 class abc {
2   String nome;
3   String cpf;
4   double salario;
5   int senha;
6 }
7 class xyz extends abc {
8   int numerodeImoveis;
9 }
```

No trecho de código antecedente, é correto apenas o que se afirma na(s):

a) linhas 7 e 8, pois há a especificação de sobrecarga.

b) linhas 7 e 8, pois há a especificação de sobrescrita.

c) linha 8, pois há especificação de uma variável privada que somente pode ser acessada pela classe xyz.

d) linha 7, pois há uma especificação de herança entre xyz e abc.

e) linha 8, porque há um erro material, pois não pode haver novos atributos nesse tipo de especificação.

Comentários:

(a) Errado. Essa classe sequer tem métodos, logo não há como haver sobrecarga; (b) Errado, pelo mesmo motivo do comentário anterior; (c) Errado. Essa variável não é privada – ela tem visibilidade padrão; (d) Correto. De fato, há uma especificação de herança entre essas classes – basta ver a palavra-chave extends; (e) Errado, pode haver novos atributos, sim. **Gabarito: D**

38. (CESPE - 2004 – ABIN - Analista de Sistemas) A máquina virtual Java (Java Virtual Machine — JVM) é especificada para interpretar instruções expressas em bytecodes e compiladas a partir de uma linguagem de programação do tipo Java, mas não necessariamente Java.

Comentários:

Uma observação importante: Código Java é sempre compilado em um bytecode. No entanto, nem todo bytecode é proveniente de Código Java. Como assim, professor? É isso mesmo! Por exemplo: eu posso compilar o Código Ada em um bytecode e rodá-lo em uma JVM! E quais outras linguagens? Temos também Eiffel, Pascal, Python, C.

Conforme vimos em aula, não é necessariamente de Java! **Gabarito: C**

39. (CESPE - 2010 – TRE/MT - Analista de Sistemas – A) JVM é um tipo de linguagem de máquina, resultado da compilação do código-fonte Java, que é interpretado e executado pela máquina virtual Java.

Comentários:

O bytecode é um código intermediário, que é posteriormente interpretado e executado por uma Java Virtual Machine (JVM). O que é isso, professor? É um programa que carrega e executa os aplicativos Java, convertendo bytecodes em código executável. Lembram que eu falei que Java é uma Linguagem WORA? Pois é, isso ocorre em grande parte por conta do bytecode e da Máquina Virtual Java.

Conforme vimos em aula, não se trata de um tipo de linguagem, mas um programa! **Gabarito: E**

40. (CESPE - 2011 – TJ/ES - Analista de Sistemas) O JVM (Java Virtual Machine) é um interpretador que atribui portabilidade à linguagem Java, possibilitando, consequentemente, a sua execução em qualquer sistema operacional.

Comentários:

Por conta deles, programas escritos em Java podem funcionar em qualquer plataforma de hardware e software que possua uma JVM, tornando assim essas aplicações independentes da plataforma, como apresenta a imagem acima (Win32, UNIX e MacOS)! Galera, qualquer plataforma... desde um computador a uma geladeira. A imagem abaixo é similar à anterior, apenas para solidificar!

Conforme vimos em aula, está perfeito! Observem que a questão afirma que "possibilita a sua execução em qualquer sistema operacional". De fato, o Sistema Operacional deverá ter uma JVM implementada para ele. No entanto, a questão afirma apenas que possibilita (desde que haja essa JVM). De fato, possibilita! **Gabarito: C**

41. (CESPE - 2008 - TJ-DF - Analista Judiciário - Tecnologia da Informação) Na linguagem Java, durante a interpretação do código, a JVM (Java Virtual Machine) verifica se o applet faz tentativas de forjar ponteiros, de violar restrições de acesso em membros de classes privadas e de gerar falhas na pilha.

Comentários:

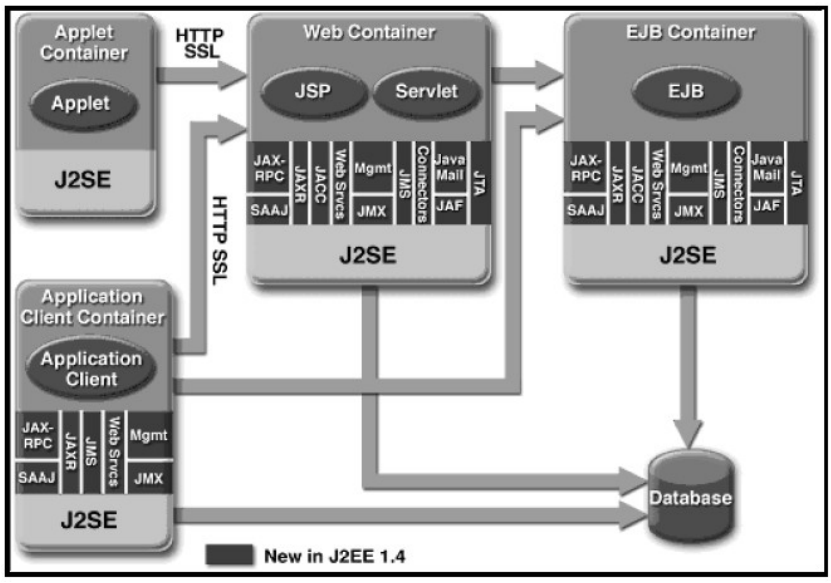
Perfeito! A JVM realiza algumas verificações de segurança importantes, principalmente, para o desenvolvimento de aplicações web. **Gabarito: C**

42. (CESPE - 2004 - SEAD - Analista de Sistemas) O programa abaixo escreverá, na tela do computador no qual for executado como aplicação ou Applet, o trecho Ola mundo.

```
import java.applet.Applet;
import java.awt.Graphics;
public class Ola extends Applet {
    public static void main (String[] args) {
System.out.println("Ola mundo");
}
    public void paint(Graphics g) {
        g.drawString("Ola mundo", 50, 25);
    }
}
```

Comentários:

Perfeito! Observem que ele escreveu "Ola Mundo" no Método Main e no Método Paint. Então, não importa aonde seja executado (na aplicação ou na applet), essa mensagem será exibida. **Gabarito: C**



43. (CESPE - 2006 – ANCINE - Analista de Sistemas) Para que um sistema aplicativo funcione nessa arquitetura, é necessário que os contêineres Web e EJB utilizem máquinas com o mesmo sistema operacional, seja ele Windows ou Linux, embora os contêineres Applet e de aplicação cliente possam ser heterogêneos com relação ao sistema operacional.

Comentários:

Não, o Contêineres Web e Contêiner EJB põem ser heterogêneos quanto ao sistema operacional também. **Gabarito: E**

44. (CESPE - 2006 – DATAPREV - Analista de Sistemas) Os aplicativos Java que executam no Applet Container são instalados originalmente no Web Container.

Comentários:

Em 1995, a tecnologia Java e seu modelo de código móvel conhecido como applet foram criados pela Sun Microsystems. Applets são pequenos programas Java que são instalados em Contêineres Web e referenciados através de Páginas HTML em um navegador. Quando um browser acessa esta página HTML que referencia o applet, ele automaticamente também transfere o código do applet e o executa.

Perfeito, conforme visto em aula. **Gabarito: C**

45. (CESPE - 2006 – DATAPREV - Analista de Sistemas) Um applet pode ser armazenado em um servidor e depois transferido para as máquinas dos usuários. Uma classe que modela um applet deriva da classe Applet e contém um método init executado na carga do applet.

Comentários:

Em 1995, a tecnologia Java e seu modelo de código móvel conhecido como applet foram criados pela Sun Microsystems. Applets são pequenos programas Java que são instalados em Contêineres Web e referenciados através de Páginas HTML em um navegador. Quando um browser acessa esta página HTML que referencia o applet, ele automaticamente também transfere o código do applet e o executa.

O primeiro código que uma applet executa é o código definido para sua inicialização, e seu construtor, através do método init que executa a inicialização básica da applet – geralmente usado para iniciar as variáveis globais, obter recursos de rede e configurar a interface. O método init é chamado apenas uma vez, quando o código da applet é carregado pela primeira vez, ao iniciar.

Perfeito, conforme visto em aula! **Gabarito: C**

46. (CESPE - 2008 – TJDFT - Analista de Sistemas) Na linguagem Java, durante a interpretação do código, a JVM (Java Virtual Machine) verifica se o applet faz tentativas de forjar ponteiros, de violar restrições de acesso em membros de classes privadas e de gerar falhas na pilha.

Comentários:

Perfeito! A JVM possui medidas de segurança para proteger o código de ataques (não usa ponteiros, não permite que applets tenham acesso ao computador dos clientes). Ademais, ela possui um verificador de bytecodes confirma se código não contém erros ou se não violam restrições de segurança de Java. **Gabarito: C**

47. (CESPE - 2009 – UNIPAMPA - Analista de Sistemas) O trecho de código a seguir é um exemplo de como inserir applet em um documento HTML.

```
<applet
 code=MinhaClasse.class
 width=120
 height=120>
</applet>
```

Comentários:

Perfeito. O HTML possui uma tag <applet> que permite inserir uma applet em um documento HTML. No entanto, ele não é mais aceito no HTML 5 e não é suportado pelos navegadores Chrome e Opera. **Gabarito: C**

48. (CESPE - 2008 - TRT - 5ª Região (BA) - Técnico Judiciário - Tecnologia da Informação) Applets são componentes que podem ser executados tanto do lado servidor quanto do lado cliente em qualquer aplicação Java.

Comentários:

Applets, ao contrário das Servlets, rodam no Cliente e são hoje as principais responsáveis pela grande disseminação do conceito e das tecnologias de código móvel em geral. O uso de código móvel na web, e em especial o modelo de applets Java, permite que, adicionalmente ao uso de páginas HTML e imagens, os recursos transferidos entre servidores e clientes web sejam programas de computador. Portanto, applets podem ser executadas apenas do lado do cliente e, não, do servidor. **Gabarito: E**

QUESTÕES COMENTADAS- JAVA – CONCEITOS BÁSICOS - FCC

1. **(FCC - 2011 - TRE-RN - Técnico Judiciário - Programação de Sistemas) Em relação ao Java Standard Edition, é INCORRETO afirmar:**

   a) Possui gerenciamento de memória embutido, por meio do coletor de lixo.

   b) Ambiente indicado para o desenvolvimento de aplicativos para dispositivos móveis ou portáteis.

   c) Permite o desenvolvimento de aplicações desktop de linha de comando e interfaces gráficas Swing.

   d) Portabilidade dos programas compilados para diversos sistemas operacionais, sem necessidade de recompilação.

   e) Usa conceitos tais como orientação a objetos e multithreading.

Comentários:

O acesso a arrays e strings, e a conversão de tipos são checados em tempo de execução para assegurar a sua validade. O Garbage Collector faz a desalocação automática de memória evitando, erros de referência e desperdício de memória. Finalmente, o recurso de Exception Handling permite o tratamento de erros em tempo de execução, por um mecanismo robusto, análogo ao do C++.

(a) Conforme vimos em aula, ele contém um Garbage Collector para gerenciamento de memória;

Java Micro Edition (Java ME): trata-se do padrão aplicado a dispositivos compactos ou móveis, como smartphones, tablets, controles remotos, etc. Permite o desenvolvimento de softwares embarcados, i.e., aplicações que rodam em um dispositivo de propósito específico, desempenhando alguma tarefa útil. Em geral, possuem limitações de recursos como memória ou processamento.

(b) Conforme vimos em aula, esse é o Java ME (Java Micro Edition);

A Plataforma Java oferece recursos para construção de interfaces gráficas de usuário (GUI), entre eles: AWT (java.awt) e Swing (javax.swing)! O primeiro é um conjunto básico de classes e interfaces que definem os componentes de uma janela desktop. Já o Swing é um conjunto sofisticado de classes e interfaces que definem os componentes visuais necessários para construir uma interface gráfica de usuário.

(c) Conforme vimos em aula, ele permite desenvolvimento de aplicações desktop e de interfaces gráficos (Swing e AWT);

Vocês sabiam que Java é uma Linguagem WORA? Pois é, esse acrônimo significa Write Once, Run Anywhere ou Escreva uma vez, execute em qualquer lugar. Trata-se de um slogan para exemplificar os benefícios multiplataforma da linguagem Java! Idealmente, isso significa que um programa em Java (uma vez compilado em um bytecode) pode rodar em qualquer equipamento que possua uma JVM!

(d) Conforme vimos em aula, é uma linguagem WORA (Write Once, Run Anywhere);

Dessa forma, aplicações funcionam da mesma maneira em qualquer ambiente. Podemos dizer que Java é uma linguagem concorrente ou multithreaded, i.e., pode realizar diversas tarefas assincronamente com o uso de threads, que são suportadas de modo nativo. Java torna a manipulação de threads tão simples quanto trabalhar com qualquer variável.

(e) Conforme vimos em aula, é uma linguagem orientada a objetos e possui suporte nativo a threads; **Gabarito: B**

2. **(FCC - 2010 - TRT - 22ª Região (PI) - Técnico Judiciário - Tecnologia da Informação) A plataforma Java disponibiliza um interpretador que traduz, em tempo de execução, o bytecode para instruções nativas do processador, permitindo, dessa forma, que uma mesma aplicação seja executada em qualquer plataforma computacional que possua essa implementação. Trata-se de:**

a) Java Virtual Machine.

b) Java API.

c) JavaBeans.

d) J2SE.

e) JavaFX.

Comentários:

O bytecode é um código intermediário, que é posteriormente interpretado e executado por uma Java Virtual Machine (JVM). O que é isso, professor? É um programa que carrega e executa os aplicativos Java, convertendo bytecodes em código executável. Lembram que eu falei que Java é uma Linguagem WORA? Pois é, isso ocorre em grande parte por conta do bytecode e da Máquina Virtual Java.

Conforme vimos em aula, trata-se da Java Virtual Machine (JVM)! **Gabarito: A**

3. **(FCC - 2010 - Sergipe Gás S.A. - Analista de Sistemas) É tida como uma das principais linguagens de programação orientada a objeto; tem como característica a compilação para um bytecode e execução por uma máquina virtual. Trata-se da linguagem:**

a) Algol.

b) Delphi.

c) C++.

d) Java.

e) PHP.

Comentários:

Java é também uma linguagem portável e multiplataforma! O Compilador é capaz de gerar um código intermediário (bytecode), que permite que o mesmo programa possa ser

executado em qualquer máquina ou sistema operacional que possua uma JVM. Ademais, busca que todos os aspectos da linguagem sejam independentes de plataforma (Ex: ela especifica o tamanho e comportamento de cada tipo de dado).

Conforme vimos em aula, trata-se da linguagem Java. **Gabarito: D**

4. (FCC - 2010 - TCE-SP - Agente da Fiscalização Financeira - Informática - Suporte de Web) Os aplicativos Java "rodam" em diferentes ambientes. A tradução dos códigos Java (bytecode), para instruções específicas de cada sistema e dispositivo, é uma função do programa:

a) Java Community Process (JCP).

b) Java Virtual Module (JVM).

c) Java Virtual Machine (JVM).

d) Java Comunication Process (JCP).

e) Java Enterprise Machine (JEM).

Comentários:

O bytecode é um código intermediário, que é posteriormente interpretado e executado por uma Java Virtual Machine (JVM). O que é isso, professor? É um programa que carrega e executa os aplicativos Java, convertendo bytecodes em código executável. Lembram que eu falei que Java é uma Linguagem WORA? Pois é, isso ocorre em grande parte por conta do bytecode e da Máquina Virtual Java.

Conforme vimos em aula, trata-se da Java Virtual Machine (JVM)! **Gabarito: C**

5. (FCC - 2009 - TJ-SE - Técnico Judiciário - Programação de Sistemas) Um objeto é instanciado em Java por meio do operador:

a) instanceof.

b) extend.

c) new.

d) this.

e) type.

Comentários:

Palavras	Descrição
Instanceof	Testa se um objeto é uma instância de uma classe específica ou se é null.

Extends	Utilizado para aplicar o conceito de herança para uma classe, onde uma classe receberá os métodos e variáveis de instância da classe chamada de pai.
New	Utilizada para se criar novas instâncias de objetos.
This	Representa a instância que está atualmente sendo executada.

Conforme vimos em aula, trata-se da palavra-reservada new – lembrando que type não é um operador! **Gabarito: C**

6. (FCC - 2009 - TRT - 16ª REGIÃO (MA) - Técnico Judiciário - Tecnologia da Informação) Uma classe Java pode ser instanciada por um comando, cuja sintaxe é:

a) nome_Objeto nome_Classe = new nome_Objeto();

b) nome_Classe nome_Objeto = new nome_Classe();

c) nome_Classe nome_instancia = new nome_Objeto();

d) nome_Instancia nome_Objeto = new nome_Instancia();

e) nome_Instancia nome_Classe = new nome_Instancia();

Comentários:

```
/* 1) Operador NEW é responsável por criar um objeto;
 * 2) NomeClasse() é o construtor da Classe NomeClasse;
 * 3) NomeObjeto é uma variável do Tipo NomeClasse; */

NomeClasse NomeObjeto = new NomeClasse();

/* Observem que é possível atribuir o objeto de uma
 * classe para uma variável de outra classe */
```

Conforme vimos em aula, o operador new cria um novo objeto! Para tal, utiliza-se o construtor da classe que se deseja criar uma instância: Nome_Classe( ). Por fim, ele atribui esse novo objeto a uma variável Nome_Objeto pertencente a classe Nome_Classe. **Gabarito: B**

7. (FCC - 2009 - TRT - 16ª REGIÃO (MA) - Técnico Judiciário - Tecnologia da Informação) A diretiva public é utilizada em Java para aplicar a encapsulação pública:

a) aos métodos e classes, apenas.

b) aos atributos, métodos e classes.

c) às classes, apenas.

d) aos atributos, apenas.

e) aos atributos e classes, apenas.

Comentários:

Pessoal, esses Modificadores de Acesso determinam quão acessíveis são esses elementos. Vamos vê-los agora em mais detalhes:

- <public>: essa instrução indica que a classe, método ou atributo assim declaradas podem ser acessadas em qualquer lugar e a qualquer momento da execução do programa – é o modificador menos restritivo.

Conforme vimos em aula, o public pode ser aplicado a atributos, métodos e classes. **Gabarito: B**

8. (FCC - 2008 - TCE-AL - Programador) Em Java, para alterar a visibilidade do elemento em que se aplica, entre outros, utiliza-se o modificador de acesso:

a) static.

b) abstract.

c) protected.

d) volatile.

e) transient.

Comentários:

Pessoal, esses Modificadores de Acesso determinam quão acessíveis são esses elementos. Vamos vê-los agora em mais detalhes:

- <protected>: essa instrução indica que métodos ou atributos (classes, não) assim declaradas somente podem ser acessadas dentro do pacote em que está contida ou por subclasses no mesmo pacote.

Apenas um desses é um Modificador de Acesso: protected. **Gabarito: C**

9. (FCC - 2007 - MPU - Analista de Informática - Desenvolvimento de Sistemas) Analise os seguintes valores, variáveis e operações usando expressões Java:

```
byte j = 30;
```

```
short k = 54;
int m = 40;
long n = 12L;
long resultado = 0L;
resultado += j;
resultado += k;
resultado /= n;
resultado -= m;
```

Após a última operação, o resultado será igual a:

a) -7.

b) -32.

c) -33.

d) 60.

e) 84.

Comentários:

resultado = resultado + j = 0L + 30 = 30L; //Houve cast implícito

resultado = resultado + k = 30L + 54 = 84L; //Houve cast implícito

resultado = resultado/n = 84L/12L = 7L;

resultado = resultado – m = 7L – 40 = -33L //Houve cast implícito **Gabarito: C**

10. (FCC - 2005 - TRE-MG - Técnico Judiciário - Programação de Sistemas) Os métodos Java que não retornam valores devem possuir no parâmetro tipo-de-retorno a palavra:

a) static.

b) public.

c) void.

d) main.

e) string args.

Comentários:

Palavras	Descrição
	Representa um retorno vazio, i.e., nenhum retorno para esse método.

Void	

Conforme vimos em aula, trata-se da palavra-reservada void. **Gabarito: C**

11. (FCC - 2012 - TST - Analista Judiciário - Análise de Sistemas) Considere o programa abaixo escrito na linguagem Java:

```
public class Programa
{
  public static void main(String args[])
  {
    for(int i=3; i<20; i+=2)
    {
      System.out.print((i%3) + "");
    }
  }
}
```

O resultado a ser informado ao usuário após a execução do programa acima é:

a) 0 0 1 0 0 1 0 0 1

b) 0 1 2 0 1 2 0 1 2

c) 0 1 0 1 0 1 0 1 0

d) 1 2 1 2 1 2 1 2 1

e) 0 2 1 0 2 1 0 2 1

Comentários:

i = 3; 3%3 = 0;

i = 5; 5%3 = 2;

i = 7; 7%3 = 1;

i = 9; 9%3 = 0;

i = 11; 11%3 = 2;

i = 13; 13%3 = 1;

i = 15; 15%3 = 0;

i = 17; 17%3 = 2;

i = 19; 19%3 = 1;

Lembrando que o Operador % oferece o resto da divisão entre dois números. **Gabarito: E**

12. (FCC - 2012 - MPE-AP - Analista Ministerial - Tecnologia da Informação) Analise o código das classes a seguir presentes em um mesmo pacote de um projeto Java:

```
public class NewClassA {
   public double calcular(int x, int y) {
      return x + y;
   }

   public double calcular(double x, double y) {
      return x * y;
   }
}

public class NewClassB extends NewClassA {

}

public class Start {
   public static void main(String[] args) {

   }
}
```

Com base nos códigos apresentados e nos conceitos da orientação a objetos é correto afirmar:

a) No método main da classe Start não é possível instanciar objetos das classes NewClassA e NewClassB, pois essas classes não contêm um construtor válido.

b) Se for digitada a instrução NewClassB c = new NewClassA(); no método main da classe Start será instanciado um objeto da NewClassA.

c) Se for digitada a instrução NewClassA b = new NewClassB(); no método main da classe Start ocorrerá um erro, pois não é possível criar um objeto da NewClassA por meio do construtor da NewClassB.

d) A existência de dois métodos de mesmo nome na NewClassA que recebem a mesma quantidade de parâmetros indica que está ocorrendo uma sobrescrita de métodos.

e) Por meio de um objeto da NewClassB será possível acessar os métodos presentes na NewClassA.

Comentários:

(a) Não, mesmo que não tenha um construtor explícito, o compilador cria um construtor padrão no momento da criação do objeto; (b) Não, não se pode atribuir uma instância de classe-pai a variável de uma classe-filha; (c) Não, não haveria erro! Pode-se atribuir uma instância de classe filha a variável de uma classe-pai; (d) Não, isso é sobrecarga; (e) Perfeito, ela herda tudo da classe-pai. **Gabarito: E**

13. (FCC - 2012 - TCE-SP - Auxiliar de Fiscalização Financeira) Em um programa Java, considere a existência de uma variável do tipo long chamada cod contendo o valor 1234. Para passar o valor contido nessa variável para uma variável do tipo byte chamada codNovo, deve-se fazer casting. Para isso, utiliza-se a instrução: byte codNovo =

a) Byte.valueOf(cod);

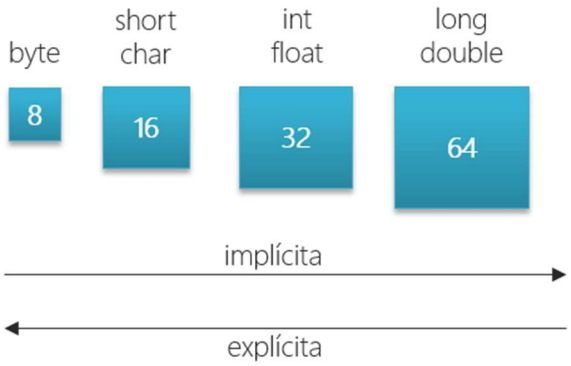
b) (long) cod;

c) Byte.pasreByte(cod);

d) (byte) cod;

e) (cast) cod;

Comentários:



Conforme vimos em aula, estamos indo do valor maior para o valor menor, portanto precisamos fazer um cast explícito: byte codNovo = (byte) cod. **Gabarito: D**

14. (FCC - 2012 - TRE-CE - Técnico Judiciário - Programação de Sistemas) Para chamar o método soma da classe Calculo, e mostrar na tela o retorno desse método, é correto utilizar:

```
public class Calculo {
  public static double soma(double n1, double n2) {
```

```
    return n1 + n2;
  }
  public static double soma(double n1, double n2, double n3) {
    return n1 + n2 + n3;
  }
}
```

a) Calculo c = new Calculo(); System.out.println (c.soma(10, 20, 30)); ou System.out.println (Calculo.soma(10, 20));

b) Exclusivamente as instruções Calculo c = new Calculo(); System.out.println (c.soma(10, 20));

c) Exclusivamente a instrução System.out.println (Calculo.soma(10, 20, 50));

d) Exclusivamente as instruções Calculo c = new Calculo(); double r = c.soma(10, 20); System.out.println(r);

e) Calculo c = Calculo.soma(10,20,30); System.out.println (c); ou System.out.println (Calculo.soma(10, 20));

Comentários:

Vamos lá! Observem que há dois métodos estáticos, i.e., não é necessário criar um objeto dessa classe para ter acesso aos seus métodos. Logo, há duas possibilidades: pode-se instanciar um objeto dessa classe (Calculo c = new Calculo();) e chamar o método pelo objeto (c.soma()) ou fazer o mesmo sem utilizar o objeto (Calculo.soma()). Portanto, as letras B, C, D funcionariam, mas não exclusivamente, porque pode-se acessar o método pelo objeto ou pela classe. **Gabarito: A**

15. (FCC - 2012 - TRE-CE - Técnico Judiciário - Programação de Sistemas) Com relação a herança na programação orientada a objetos com Java, é INCORRETO afirmar:

a) Uma subclasse herda os métodos da superclasse, entretanto, pode ter seus próprios métodos.

b) Quando se instancia um objeto da subclasse, podem ser passados valores para os atributos da superclasse.

c) Um objeto da subclasse pode ser um objeto da superclasse.

d) Em uma superclasse, para acessar métodos da subclasse deve ser usada a instrução super.

e) Para definir que a subclasse herda as características da superclasse utiliza-se a instrução extends na declaração da subclasse.

Comentários:

(a) Perfeito, ela pode criar seus próprios métodos; (b) Perfeito, porque eles são herdados; (c) Perfeito, eles são herdados da superclasse; (d) Não, super é utilizado para acessar métodos da superclasse; (e) Perfeito, esse operador indica que a superclasse será estendida. **Gabarito: D**

16. (FCC - 2011 - TRT - 4ª REGIÃO (RS) - Técnico Judiciário - Tecnologia da Informação) No ambiente de programação Java:

a) uma classe abstrata permite apenas métodos abstratos.

b) o corpo de um método abstrato termina com ponto e vírgula e a declaração é delimitada por chaves.

c) uma interface pode definir tanto métodos abstratos quanto não abstratos.

d) a herança múltipla permite que mais classes sejam estendidas.

e) toda classe é uma subclasse direta ou indireta da classe Object.

Comentários:

Cada classe, na hierarquia de classes, representa uma camada que adiciona diversas capacidades a um objeto. No topo desta hierarquia você sempre vai encontrar uma classe chamada de Object (Objeto). Qualquer classe estende implicitamente (sem necessidade de declarar) a classe Object. Claro que, na maioria das vezes, isso ocorre indiretamente.

(a) Não, ela permite métodos concretos; (b) Não, não tem chaves; (c) Não, todos os métodos são abstratos; (d) Java não suporta herança múltipla; (e) Perfeito, absolutamente todas as classes são filhas da Classe Object. **Gabarito: E**

17. (FCC - 2010 - TRT - 8ª Região (PA e AP) - Analista Judiciário - Tecnologia da Informação) São tipos primitivos da linguagem Java:

a) int, string, long e real.

b) char, int, real e bit.

c) boolean, double, float e byte.

d) real, short, long e char.

e) string, long int, short int e float.

Comentários:

NOME	TIPO	TAMANHO	MÍNIMO	MÁXIMO	DEFAULT
LÓGICO	boolean	-	false	true	false
CARACTERE	char	16 bits	0	$2^{16} - 1$	'\u0000'
INTEIRO	byte	8 bits	-2^7	$2^7 - 1$	0

	short	16 bits	-2^{15}	$2^{15} - 1$	0
	int	32 bits	-2^{31}	$2^{31} - 1$	0
	long	64 bits	-2^{63}	$2^{63} - 1$	0
DECIMAL	float	32 bits	7 Casas Decimais		0.0
	double	64 bits	15 Casas Decimais		0.0

(a) string e real, não; (b) real e bit, não; (c) Perfeito! (d) real, não; (e) apenas float. **Gabarito: C**

18. (FCC - 2010 - AL-SP - Agente Legislativo de Serviços Técnicos e Administrativos - Processamento de Dados) Os tipos de dados primitivos em Java são:

a) char, boolean, byte, short, int, long, float e double.

b) char, boolean, byte, short, int, long, float, double e String.

c) byte, short, int, long, float e double.

d) byte, short, int, long, float, double, String e Date.

e) char, boolean, byte, short, int, long, float, double, String e Date.

Comentários:

NOME	TIPO	TAMANHO	MÍNIMO	MÁXIMO	DEFAULT
LÓGICO	boolean	-	false	true	false
CARACTERE	char	16 bits	0	$2^{16} - 1$	'\u0000'
INTEIRO	byte	8 bits	-2^{7}	$2^{7} - 1$	0
	short	16 bits	-2^{15}	$2^{15} - 1$	0
	int	32 bits	-2^{31}	$2^{31} - 1$	0
	long	64 bits	-2^{63}	$2^{63} - 1$	0
DECIMAL	float	32 bits	7 Casas Decimais		0.0
	double	64 bits	15 Casas Decimais		0.0

São oito tipos: byte, short, int, long, float, double, char e boolean. **Gabarito: A**

19. (FCC - 2009 - TRT - 15ª Região - Analista Judiciário - Tecnologia da Informação) No âmbito da linguagem Java, considere:

I. Edição é a criação do programa, que também é chamado de código Bytecode.

II. Compilação é a geração de um código intermediário chamado fonte, que é um código independente de plataforma.

III. Na interpretação, a máquina virtual Java ou JVM analisa e executa cada instrução do código Bytecode.

IV. Na linguagem Java a interpretação ocorre apenas uma vez e a compilação ocorre a cada vez que o programa é executado.

Está correto o que consta em:

a) I, II, III e IV.

b) II e IV, somente.

c) III e IV, somente.

d) IV, somente.

e) III, somente.

Comentários:

(I) Não, bytecode é o resultado da compilação do código-fonte; (II) Não, o código intermediário é chamado bytecode; (III) Perfeito, é exatamente assim; (IV) Não, a compilação ocorre apenas uma vez e a interpretação a cada vez que o programa é executado. **Gabarito: E**

20. (FCC - 2008 - TCE-AL - Programador) Os três elementos básicos quando contidos num arquivo fonte Java devem obrigatoriamente se apresentar na seguinte ordem:

a) import, package e class.

b) class, package e import.

c) class, import e package.

d) package, class e import.

e) package, import e class.

Comentários:

A diferença entre as duas formas de importação de pacotes é o consumo de recursos do computador. Como o asterisco importa todos os sub-pacotes, o consumo de memória será alto e, muito provavelmente, não usaremos todas as classes de todos os pacotes importados. Por isso, o recomendado é sempre importar apenas o pacote que será utilizado. A ordem é Package, Import e Class.

Conforme vimos em aula, deve-se declarar o pacote; depois, import; por fim, classe. **Gabarito: E**

21. (FCC - 2014 – TRF/3 – Analista de Sistemas) Considere a classe escrita em Java:

```
public class Teste {
   public float multi(float flt, int n) {
         return flt * n + 1;
   }

   public int multi(int dbl, double n) {
         return dbl * (int) n + 2;
   }

   public double multi(double i, double n) {
         return i * n + 3;
   }

   public static void main(String[] args) {
         Teste t = new Teste();

         System.out.println(t.multi(2.5, 2));
   }
}
```

O valor que será impresso na execução do método main é:

a) 6.0

b) 7

c) 8.0

d) 5.0

e) 12.5

Comentários:

Isso é um exemplo de polimorfismo! A questão é: qual método multi será utilizado? Ele envia os valores 2.5 e 2! Ele pode entrar no primeiro método? Não, porque o valor-padrão de número decimal no Java é Double, logo – para entrar no primeiro método – teria que enviar o valor 2.5f, visto que é do tipo Float. Ele pode entrar no segundo método? Não, visto que o valor 2.5 não pode ser inteiro! Ele pode entrar no terceiro método? Sim, visto que o valor 2.5 é Double e o valor 2 é inteiro! Eu posso passar um valor inteiro a uma variável do tipo Double? Sim, o que eu não posso fazer é o contrário. Logo, o valor impresso será 2.5*2+3 = 8.0. **Gabarito: C**

22. (FCC - 2008 - TCE-AL - Programador) Considerando que as variáveis Java X, Y e Z foram todas inicializadas com zero, os resultados das mesmas após as alterações realizadas pelas atribuições X *= 2, Y -= 5 e Z /=3, respectivamente, serão:

a) 0, -5 e 0

b) 0, 5 e 0

c) 1, -5 e 3

d) 2, -5 e 3

e) 2, 5 e 3

Comentários:

X = X*2 = 0*2 = 0;

Y = Y-5 = 0-5 = -5;

Z = Z/3 = 0/3 = 0;

Portanto, a resposta é 0, -5 e 0. **Gabarito: A**

23. (FCC - 2008 - MPE-RS - Técnico em Informática - Área Sistemas) A função Java:

public boolean VerificarCPF (string CPF);

representa um exemplo do conceito de:

a) override.

b) overload.

c) herança.

d) encapsulamento.

e) polimorfismo.

Comentários:

Essa questão é estranha! Ele não especifica exatamente o que ele quer saber, mas vamos lá: por eliminação! (a) Impossível inferir algo sobre isso; (b) Impossível inferir algo sobre isso; (c) Impossível inferir algo sobre isso; (d) Bem, há um modificador de acesso, portanto representa um exemplo do conceito de encapsulamento; (e) Impossível inferir algo sobre isso. **Gabarito: D**

24. (FCC - 2007 - MPU - Analista de Informática - Desenvolvimento de Sistemas) Quanto às variáveis Java, um inteiro de 64 bits em notação de complemento de dois que pode assumir valores entre -263 e 263 -1 é:

a) long.

b) short.

c) float.

d) byte.

e) double.

Comentários:

NOME	TIPO	TAMANHO	MÍNIMO	MÁXIMO	DEFAULT
LÓGICO	boolean	-	false	true	false
CARACTERE	char	16 bits	0	$2^{16} - 1$	'\u0000'

INTEIRO	byte	8 bits	-2^7	$2^7 - 1$	0
	short	16 bits	-2^{15}	$2^{15} - 1$	0
	int	32 bits	-2^{31}	$2^{31} - 1$	0
	long	64 bits	-2^{63}	$2^{63} - 1$	0
DECIMAL	float	32 bits	7 Casas Decimais		0.0
	double	64 bits	15 Casas Decimais		0.0

Conforme vimos em aula, trata-se do tipo long. **Gabarito: A**

25. (FCC - 2005 - TRE-MG - Técnico Judiciário - Programação de Sistemas) A seqüência de etapas para implementação de um programa Java é:

a) interpretação, codificação, execução e compilação.

b) codificação, interpretação, compilação e execução.

c) interpretação, codificação, compilação e execução.

d) codificação, compilação, interpretação e execução.

e) compilação, codificação, execução e interpretação.

Comentários:

O bytecode é um código intermediário, que é posteriormente interpretado e executado por uma Java Virtual Machine (JVM). O que é isso, professor? É um programa que carrega e executa os aplicativos Java, convertendo bytecodes em código executável. Lembram que eu falei que Java é uma Linguagem WORA? Pois é, isso ocorre em grande parte por conta do bytecode e da Máquina Virtual Java.

Conforme vimos em aula, devemos codificar o programa em .java; esse arquivo é compilado em um código intermediário (bytecode) em .class; depois é interpretado em uma JVM; e, por fim, é executado. **Gabarito: D**

26. (FCC - 2012 - TRE-CE - Técnico Judiciário - Programação de Sistemas) Considere a variável idade declarada no método main de uma classe Java, com a seguinte instrução:

int idade=12;

Analise:

I. System.out.println (idade<18?"Menor de idade":"Maior de idade");

II. if(idade<18) {System.out.println("Menor de idade");} else {System.out.println("Maior de idade");}

III. if(idade<18) {System.out.println("Menor de idade");} else if (idade>=18) {System.out.println("Maior de idade");}

IV. switch(idade) {case<18: System.out.println("Menor de idade"); break; case>=18: System.out.println("Maior de idade");}

Contém uma instrução correta que exibirá na tela a frase "Menor de idade":

a) I, II, III e IV.

b) I, II e III, apenas.

c) II e III, apenas.

d) II, apenas.

e) I e III, apenas.

Comentários:

(I) Operador Ternário: 12 é menor que 18, portanto irá imprimir “Menor de Idade”; (II) 12 é menor que 18, portanto irá imprimir “Menor de Idade”; (III) 12 é menor que 18, portanto irá imprimir “Menor de Idade”; (IV) Case não aceita <, >, <=, >=, etc. **Gabarito: B**

27. (FCC - 2012 - TRE-CE - Técnico Judiciário - Programação de Sistemas) Considere a estrutura de repetição seguinte:

```
public static void main(String{} args) {
   int cont=1, r=0;
   cont=1;
   do {
      r=r+cont;
      cont+=4;
   } while(cont<=5);
   System.out.println(r);
   System.out.println(cont);
}
```

A saída na tela será:

a) 15 e 6.

b) 1 e 5.

c) 0 e 1.

d) 6 e 9.

e) 9 e 7.

Comentários:

r = r + cont = 0 + 1 = 1;

cont = cont + 4 = 1 + 4 = 5;

r = r + cont = 1 + 5 = 6;

cont = cont + 4 = 5 + 4 = 9; //Sai do loop

Portanto, r = 6 e cont = 9. **Gabarito: D**

28. (FCC - 2011 - TRE-AP - Técnico Judiciário - Programação de Sistemas) Em relação à plataforma de desenvolvimento JSE, considere:

I. Possibilita o desenvolvimento de aplicações desktop através de linha de comando e através da interface gráfica Swing.

II. É multiplataforma: permite a portabilidade dos programas compilados para diversos sistemas operacionais, sem necessidade de alteração do código ou de recompilação.

III. Faz uso explícito de ponteiros e usa conceitos modernos, tais como, orientação a objetos e suporte a multithreading.

IV. Possui o gerenciamento de memória embutido, por meio do garbage collector.

Está correto o que se afirmar em:

a) I, II e III, somente.

b) I, II e IV, somente.

c) I, III e IV, somente.

d) II, III e IV, somente.

e) I, II, III e IV.

Comentários:

(I) Perfeito, é possível desenvolver aplicações desktop utilizando interface gráfica, como Swing e AWT; (II) Perfeito, é uma linguagem WORA (Write Once, Run Anywhere); (III) Não, não faz uso explícito de ponteiros; (IV) Perfeito, o Garbage Collector realiza o gerenciamento de memória. **Gabarito: B**

29. (FCC - 2011 - INFRAERO - Analista de Sistemas - Desenvolvimento e Manutenção) No Java, um tipo inteiro (int) utiliza quatro bytes para armazenamento. A faixa máxima possível de valores inteiros para se armazenar em uma variável do tipo primitivo int é de:

a) -8388608 a 8388607.

b) -128 a 127.

c) -32768 a 32767.

d) -9223372036854775808 a 9223372036854775807.

e) -2147483648 a 2147483647.

Comentários:

NOME	TIPO	TAMANHO	MÍNIMO	MÁXIMO	DEFAULT
LÓGICO	boolean	-	false	true	false
CARACTERE	char	16 bits	0	$2^{16} - 1$	'\u0000'
INTEIRO	byte	8 bits	-2^7	$2^7 - 1$	0
	short	16 bits	-2^{15}	$2^{15} - 1$	0
	int	32 bits	-2^{31}	$2^{31} - 1$	0
	long	64 bits	-2^{63}	$2^{63} - 1$	0
DECIMAL	float	32 bits	7 Casas Decimais		0.0
	double	64 bits	15 Casas Decimais		0.0

Basta fazer a conta: -231 a 231 – 1 = -2147483648 a 2147483647. **Gabarito: E**

30. (FCC - 2010 - TRT - 9ª REGIÃO (PR) - Técnico Judiciário - Tecnologia da Informação) O JVM mais o núcleo de classes da plataforma Java e os arquivos de suporte formam o:

a) o J2EE.

b) o JDK.

c) o JRE.

d) uma JSP.

e) uma API.

Comentários:

Um programa escrito para a plataforma Java necessita de um ambiente de execução chamado Java Runtime Environment (JRE)! O que tem nesse negócio, professor? Ele contém uma Máquina Virtual (JVM) e Bibiliotecas (APIs). E o Java Development Kit (JDK)? Bem, eles contêm a JRE e outros componentes úteis para executar aplicações (Exemplo: Javac, Javadoc, Jar, Appletviewer, Jconsole, Jstack, Jhat, etc).

Conforme vimos em aula, trata-se da Java Runtime Environment (JRE): ambiente mínimo necessário para rodar aplicativos Java. **Gabarito: C**

31. (FCC - 2014 - TRT - 16ª REGIÃO (MA) - Analista Judiciário - Tecnologia da Informação) Considere as classes a seguir, presentes em uma aplicação Java orientada a objetos:

```
public class Funcionario {
private int id;
private String nome;
private double valorBase;
    public Funcionario() {
    }
    public Funcionario(int id, String nome, double valorBase) {
        this.id = id;
        this.nome = nome;
        this.valorBase=valorBase;
    }
    public double getValorBase() {
        return valorBase;
    }
    public double calcularSalario(){
        return valorBase;
    }
}

public class Mensalista extends Funcionario{
private double descontos;
    public Mensalista(double descontos, int id, String nome, double
    valorBase) {
        super(id, nome, valorBase);
        this.descontos = descontos;
    }
@Override
    public double calcularSalario(){
        return super.getValorBase() - descontos;
    }
}

public class Diarista extends Funcionario {
    private int diasPorSemana;
    public Diarista( int diasPorSemana, int id, String nome, double
    valorBase) {
        super(id, nome, valorBase);
        this.diasPorSemana = diasPorSemana;
    }
    @Override
    public double calcularSalario(){
        return super.getValorBase() * diasPorSemana;
    }
}
```

Em uma classe principal foram digitadas, no interior do método main, as seguintes linhas:

```
double s;
Funcionario f;
f=new Diarista(3,10456,"Ana Maria",90);
s = f.calcularSalario();
System.out.println(s);
f=new Mensalista(298.56,10457,"Pedro Henrique",877.56);
s = f.calcularSalario();
System.out.println(s);
```

As linhas que contêm a instrução s = f.calcularSalario( ); demonstram um conceito da orientação a objetos conhecido como:

a) encapsulamento.

b) sobrecarga de métodos.

c) polimorfismo.

d) sobrescrita de construtores.

e) métodos abstratos.

Comentários:

Primeiro, vamos analisar o código! Saca só... temos três classes: Funcionário, Mensalista e Diarista. Observe que as classes Mensalista e Diarista 'estendem' a classe Funcionário, i.e., são filhas de Funcionário! Agora vamos analisar cada classe:

A classe Funcionário possui dois construtores: Funcionário( ) e Funcionário(int id, String nome, double valorBase), além dos métodos getValorBase( ) e calcularSalario( ). Aí você vai me dizer: professor, tem dois construtores? Sim, com mesmo nome e assinaturas diferentes! Portanto, temos uma: Sobrecarga (Overloading) de Construtores. Bacana?

Já a classe Mensalista possui um construtor Mensalista(double descontos, int id, String nome, double valorBase) e um método calcularSalario( ). Opa, perae... esse método é igual ao método da classe Funcionário, concorda? Mesmo nome e mesma assinatura! Portanto, temos uma Sobrescrita (ou Override) de métodos.

Por fim, a classe Diarista possui um construtor Diarista(int diasPorSemana, int id, String nome, double valorBase) e um método calcularSalario( ). De novo, você vai dizer: professor, esse método também é igual àquele da classe Funcionário e Mensalista. É verdade, portanto temos uma Sobrescrita (ou Override) de métodos.

Agora vamos para o método main! Observe que ele cria uma variável f do tipo Funcionário. Em seguida, ele cria um objeto Diarista e atribui à variável f. Professor, pode isso? Sim, eu posso atribuir um objeto de uma classe-filha para uma variável do tipo da classe-pai - eu não posso é fazer o contrário (pelo menos, sem um casting). Na linha seguinte, ele diz: s = f.calcularSalario( ).

Chegamos ao ponto crucial! Ele chama o método calcularSalario( ), mas você lembra que nós temos 3 métodos com esse nome? Temos um na classe-pai e dois nas classes-filhas, sobrescrevendo o método da classe-pai. E qual desses ele está chamando? Lembra que na linha anterior ele diz que f recebe o objeto da classe Diarista? Pois é, portanto, o método calcularSalario( ) é aquele da classe Diarista. Mais abaixo, ele faz exatamente a mesma coisa, mas atribui o objeto Mensalista() à variável f, portanto, na segunda vez que ele

chama esse método, refere-se ao objeto da classe Mensalista. Entendido? Agora vamos para os itens:

(a) Encapsulamento? Não, não tem nenhum modificador de acesso aí;

(b) Sobrecarga de Métodos? Não! Lá em cima, há sobrecarga de construtores. Nessa linha, há sobrescrita de métodos.

(c) Polimorfismo? Perfeito! Nós temos uma sobrescrita de métodos (que é um tipo de Polimorfismo).

(d) Sobrescrita de Construtores? Não! Lá em cima, há sobrecarga de construtores. Nessa linha, há sobrescrita de métodos.

(e) Métodos Abstratos? Não, não há nenhum método abstrato nessa questão.

Aí você vai me dizer: Professor, mas um construtor é um tipo de método! Logo, se há uma sobrecarga de construtores, há uma sobrecarga de métodos! Sim, concordo contigo! No entanto, temos que lembrar que é uma questão da FCC! Nesse caso, a terceira opção não apresenta problema algum! Já a segunda opção, pode-se dizer que não é exatamente sobrecarga de métodos; é algo mais específico, é uma sobrecarga de construtores. Percebe? Dessa forma, eu recomendo não brigar com a banca e marcar a opção que não gera dúvidas. Agora... se for bastante técnico aqui, você está corretíssimo! A questão possui duas respostas e deveria ser anulada. **Gabarito: C**

32. (FCC - 2010 - AL-SP - Agente Legislativo de Serviços Técnicos e Administrativos - Processamento de Dados) Métodos estáticos em Java são aqueles que:

a) realizam alguma tarefa que é dependente do conteúdo de algum objeto.

b) não podem ser acessados diretamente pelo nome da classe a que pertencem, mas sim por meio de um objeto da classe.

c) realizam alguma tarefa que não é dependente do conteúdo de algum objeto.

d) são acessados por objetos que não necessitam de ser instanciados explicitamente.

e) existem em subclasses de uma herança.

Comentários:

(a) Não, é estático, logo não depende de objetos; (b) Não, é justamente o inverso; (c) Perfeito, não depende de objetos! (d) Na verdade, precisam sim ser instanciados explicitamente; (e) Isso não define um método estático. **Gabarito: C**

33. (FCC - 2008 - TCE-AL - Programador) NÃO são nomes válidos em Java:

a) _Real e $real

b) um1 e dois2

c) 3tres e tres3

d) Codigo e codigo

e) cod_valor e cod$valor

Comentários:

Deve ser a combinação de uma ou mais letras e dígitos UNICODE-16: Letras: A-Z; Letras: a-z; Underscore: _ ; Cifrão: $;Números: 0-9.

1. Não pode ser uma palavra-reservada (palavra-chave);
2. Não pode ser true, false ou null;
3. Não pode começar com números;
4. Não pode conter espaços em branco ou caracteres de formatação;

Conforme vimos em aula, “tres3” é um nome válido, mas “3tres” não! **Gabarito: C**

34. (FCC - 2010 - TCE-SP - Agente da Fiscalização Financeira - Informática - Suporte de Web) A tecnologia Java é, basicamente, dividida em JSE,

a) JEE e JME.

b) JEE e JPE.

c) JDE e JME.

d) JDE e JPE.

e) JEEP e JME.

Comentários:

- Java Standard Edition (Java SE): trata-se de uma ferramenta de desenvolvimento para a Plataforma Java. Ela contém todo o ambiente necessário para a criação e execução de aplicações Java, incluindo a Máquina Virtual (JVM), Compilador (Javac), Bibliotecas (APIs), entre outras ferramentas. Em geral, rodam em computadores pessoais, notebooks, etc.

- Java Enterprise Edition (Java EE): trata-se do padrão para desenvolvimento de sistemas corporativos, voltada para aplicações multicamadas, baseadas em componentes executados em servidores de aplicações – ele inclui o Java SE. Contém bibliotecas para acesso a base de dados, RPC, CORBA, entre outras. As aplicações podem ou não estar na internet.

- Java Micro Edition (Java ME): trata-se do padrão aplicado a dispositivos compactos ou móveis, como smartphones, tablets, controles remotos, etc. Permite o desenvolvimento de softwares embarcados, i.e., aplicações que rodam em um dispositivo de propósito específico, desempenhando alguma tarefa útil. Em geral, possuem limitações de recursos como memória ou processamento.

Conforme vimos em aula, esses são os mais importantes mesmo! **Gabarito: A**

35. (FCC - 2009 - TRT - 16ª REGIÃO (MA) - Técnico Judiciário - Tecnologia da Informação) Uma classe Java pode ser instanciada por um comando, cuja sintaxe é:

a) nome_Objeto nome_Classe = new nome_Objeto();

b) nome_Classe nome_Objeto = new nome_Classe();

c) nome_Classe nome_instancia = new nome_Objeto();

d) nome_Instancia nome_Objeto = new nome_Instancia();

e) nome_Instancia nome_Classe = new nome_Instancia();

Comentários:

```
/* 1) Operador NEW é responsável por criar um objeto;
 * 2) NomeClasse() é o construtor da Classe NomeClasse;
 * 3) NomeObjeto é uma variável do Tipo NomeClasse; */

NomeClasse NomeObjeto = new NomeClasse();

/* Observem que é possível atribuir o objeto de uma
 * classe para uma variável de outra classe */
```

Conforme vimos em aula, trata-se da segunda opção! **Gabarito: B**

36. (FCC - 2005 - TRE-MG - Programador de computador) Os erros gerados durante a execução de um programa Java devem ser controlados com uma estrutura que pode combinar o uso dos blocos:

a) try e finally, somente.

b) try e catch ou try e finally, somente.

c) try, catch e finally, somente.

d) try e catch, somente.

e) try e catch, try e finally ou try, catch e finally.

Comentários:

```
try { //Não vem sozinho: try/catch, try/finally ou try/catch/finally

 //Código a ser executado
```

```
} catch (ClasseDeExceção objDaExceção) { //Não vem sozinho: try/catch ou try/catch/finally


 //Tratamento da exceção
} finally { //Não vem sozinho: try/finally ou try/catch/finally.


 //Código a ser executado mesmo que uma exceção seja lançada
}
```

Conforme vimos em aula, as combinações possíveis são try/catch, try/finally e try/catch/finally. **Gabarito: E**

37. (FCC - 2016 – TRT 14ª RO e AC - Técnico Judiciário - Tecnologia da Informação) Para executar um programa Java deve ocorrer um processo que envolve compilação e interpretação. Quando se compila uma classe com extensão .java é gerado um arquivo com extensão:

a) .class, conhecido como bytecode, que pode ser compilado pela JVM.

b) .jar, conhecido como bytecode, que pode ser lido pela JVM.

c) .class, que instala a classe na memória virtual para ser executada.

d) .jar, que quando executado, cria um arquivo .class, que é interpretado pela JVM.

e) .class, conhecido como bytecode, que pode ser interpretado pela JVM.

Comentários:

Como vimos em aula, o código-fonte .java é compilado. O produto da compilação são arquivos .class que contém bytecodes que são interpretados pela Java Virtual Machine. **Gabarito: E**

38. (FCC - 2014 - SABESP - Tecnólogo – Sistemas – II) O processo híbrido combina a execução eficiente e a portabilidade de programas. A base é a existência de um código intermediário, mais fácil de ser interpretado e não específico de uma plataforma computacional. O método é dividido em duas etapas: compilação para um código intermediário e interpretação desse código. Um exemplo é o Java e a JVM.

Comentários:

O bytecode é um código intermediário, que é posteriormente interpretado e executado por uma Java Virtual Machine (JVM). O que é isso, professor? É um programa que carrega e executa os aplicativos Java, convertendo bytecodes em código executável. Lembram que eu falei que Java é uma Linguagem WORA? Pois é, isso ocorre em grande parte por conta do bytecode e da Máquina Virtual Java.

Conforme vimos em aula, está perfeito! Essa é a conhecida Compilação Híbrida – implementada pela JVM! **Gabarito: C**

39. (FCC – 2003 – TRF/5 - Analista de Sistemas) Um programa Java, que é executado dentro de um browser Web, denomina-se:

a) API.

b) applet.

c) servlet.

d) acriptlet.

e) package.

Comentários:

Em 1995, a tecnologia Java e seu modelo de código móvel conhecido como applet foram criados pela Sun Microsystems. Applets são pequenos programas Java que são instalados em Contêineres Web e referenciados através de Páginas HTML em um navegador. Quando um browser acessa esta página HTML que referencia o applet, ele automaticamente também transfere o código do applet e o executa.

Trata-se, portanto, de uma Applet. **Gabarito: B**

QUESTÕES COMENTADAS – JAVA – CONCEITOS BÁSICOS- MULTIBANCAS

1. **(FUNCAB - 2010 - PRODAM-AM - Analista de TI - Desenvolvimento de Sistemas) Seja a seguinte classe Java:**

```
<mod> public class Xpto
{


}
```

Qual das alternativas a seguir contém um modificador que ao ser usado na declaração acima em substituição ao termo <mod> impedirá que a classe Xpto seja estendida?

a) static

b) const

c) abstract

d) final

e) virtual

Comentários:

```
//Essa classe não pode ser estendida
final class Carro {...}
```

Conforme vimos em aula, trata-se do Final. **Gabarito: D**

2. **(ESAF - 2012 - Receita Federal - Analista Tributário da Receita Federal - Prova 2 - Área Informática) Em programação Java, o comando while:**

a) executa um bloco exclusivamente de comandos de atribuição.

b) executa um bloco de comandos enquanto sua condição for verdadeira.

c) executa um bloco de comandos até que sua condição seja verdadeira.

d) equivale ao comando what-if.

e) é idêntico ao comando do while.

Comentários:

(a) Não, podem ter outros comandos – não é só atribuição; (b) Perfeito, enquanto for verdadeira, continua a iteração; (c) Não, enquanto ela for verdadeira; (d) Não, esse

comando não existe; (e) Não, esse comando entra no bloco e só depois avalia a condição. **Gabarito: B**

3. **(CONSULPLAN – 2006 – Prefeitura de Natal – Analista de Sistemas) Analise as afirmativas abaixo colocando V para as afirmativas Verdadeiras e F para as Falsas. A linguagem JAVA se divide nas seguintes edições:**

   ( ) J2SE (Java 2 Standard Edition) - tecnologia Java para computadores pessoais, notebooks e arquiteturas com poder de processamento e memória consideráveis.

   ( ) J2EE (Java 2 Enterprise Edition) - tecnologia Java para aplicações corporativas que podem estar na internet ou não.

   ( ) J2ME (Java 2 Micro Edition) - tecnologia Java para dispositivos móveis com limitações de memória ou processamento.

   ( ) J2FE (Java 2 Full Edition) - tecnologia Java para aplicações em computadores de grande porte (mainframe).

   A seqüência está correta em:

   a) F, F, F, F

   b) V, V, V, F

   c) V, F, F, V

   d) F, V, F, V

   e) V, V, V, V

Comentários:

- Java Standard Edition (Java SE): trata-se de uma ferramenta de desenvolvimento para a Plataforma Java. Ela contém todo o ambiente necessário para a criação e execução de aplicações Java, incluindo a Máquina Virtual (JVM), Compilador (Javac), Bibliotecas (APIs), entre outras ferramentas. Em geral, rodam em computadores pessoais, notebooks, etc.
- Java Enterprise Edition (Java EE): trata-se do padrão para desenvolvimento de sistemas corporativos, voltada para aplicações multicamadas, baseadas em componentes executados em servidores de aplicações – ele inclui o Java SE. Contém bibliotecas para acesso a base de dados, RPC, CORBA, entre outras. As aplicações podem ou não estar na internet.
- Java Micro Edition (Java ME): trata-se do padrão aplicado a dispositivos compactos ou móveis, como smartphones, tablets, controles remotos, etc. Permite o desenvolvimento de softwares embarcados, i.e., aplicações que rodam em um dispositivo de propósito específico, desempenhando alguma tarefa útil. Em geral, possuem limitações de recursos como memória ou processamento.

Conforme vimos em aula, as três primeiras alternativas estão perfeitas; a última simplesmente não existe. **Gabarito: B**

4. (Instituto Cidades - 2012 - TCM-GO - Auditor de Controle Externo - Informática) Analise:

I. O Java refere-se tanto a uma linguagem de programação quanto a uma plataforma;

II. O Java SE (Standard Edition) é formalmente chamado de J2SE;

III. O J2EE é a edição corporativa do Java. Esta versão inclui o Java Standard Edition além de outras tecnologias como javamail, servlets, JSF e Enterprise Java Beans.

IV. O Java possui uma versão para dispositivos móveis chamada J2ME (Micro Edition).

São verdadeiras as afirmações:

a) I, II e IV, somente;

b) I, III e IV, somente;

c) II, III e IV, somente;

d) I e IV, somente;

e) Todas as afirmações.

Comentários:

Java é tanto uma plataforma quanto uma linguagem de programação orientada a objetos que permite o desenvolvimento de aplicações em diversas plataformas diferentes. Como já foi dito anteriormente, Java está presente desde dispositivos pequenos (Smartphone, Tablet, etc) a máquinas de grande porte (Servidores, Mainframes, etc). A linguagem Java possui quatro ambientes de desenvolvimento:

(a) Conforme vimos em aula, é tanto uma linguagem quanto uma plataforma;

- Java Standard Edition (Java SE): trata-se de uma ferramenta de desenvolvimento para a Plataforma Java. Ela contém todo o ambiente necessário para a criação e execução de aplicações Java, incluindo a Máquina Virtual (JVM), Compilador (Javac), Bibliotecas (APIs), entre outras ferramentas. Em geral, rodam em computadores pessoais, notebooks, etc.

(b) Conforme vimos em aula, é o nome antigo de Java SE;

- Java Enterprise Edition (Java EE): trata-se do padrão para desenvolvimento de sistemas corporativos, voltada para aplicações multicamadas, baseadas em componentes executados em servidores de aplicações – ele inclui o Java SE. Contém bibliotecas para acesso a base de dados, RPC, CORBA, entre outras. As aplicações podem ou não estar na internet.

(c) Conforme vimos em aula, é exatamente isso;

- Java Micro Edition (Java ME): trata-se do padrão aplicado a dispositivos compactos ou móveis, como smartphones, tablets, controles remotos, etc. Permite o desenvolvimento de softwares embarcados, i.e., aplicações que rodam em um

dispositivo de propósito específico, desempenhando alguma tarefa útil. Em geral, possuem limitações de recursos como memória ou processamento.

(d) Conforme vimos em aula, é o Java ME (nome atual). **Gabarito: E**

5. (UFBA - 2012 - UFBA - Técnico de Tecnologia da Informação) O código-fonte de um programa de computador escrito na linguagem Java, é compilado para um formato intermediário conhecido como bytecode.

Comentários:

Java é também uma linguagem portável e multiplataforma! O Compilador é capaz de gerar um código intermediário (bytecode), que permite que o mesmo programa possa ser executado em qualquer máquina ou sistema operacional que possua uma JVM. Ademais, busca que todos os aspectos da linguagem sejam independentes de plataforma (Ex: ela especifica o tamanho e comportamento de cada tipo de dado).

Conforme vimos em aula, a questão está perfeita! No entanto, há uma coisa muito errada nessa questão! O que, professor? Não se separa sujeito do predicado com vírgula! Jamais... Vacilo, UFBA! ;-) **Gabarito: C**

6. (PaqTcPB - 2012 - UEPB - Técnico em Informática - Programador) Em Java, um bloco de código é:

a) Tudo que está entre ( )

b) Tudo que está entre { }

c) Tudo que está entre [ ]

d) Tudo que está entre < >

e) Tudo que está no mesmo nível de indentação.

Comentários:

Blocos de programação são aglomerados de instruções e declarações que têm escopo conjunto. Em outras palavras, as variáveis definidas como locais dentro de um bloco somente serão presentes dentro deste bloco, assim como as instruções ali presentes. Os blocos de programação são delimitados por chaves { } e podem ser aninhados, já os comandos sempre são terminados com ponto-e-vírgula.

Conforme vimos em aula, bloco de código vem delimitado por chaves { }. **Gabarito: B**

7. (CESGRANRIO - 2012 - Petrobrás - Técnico de Exploração de Petróleo Júnior - Informática) Ao escrever o código da Classe PortaDeCofre em Java para que ela atenda a interface Porta, como um programador deve começar a declaração da classe?

a) public class Porta:PortaDeCofre {

b) public class PortaDeCofre :: Porta {

c) public class PortaDeCofre inherits Porta {

d) public class PortaDeCofre extends Porta {

e) public class PortaDeCofre implements Porta {

Comentários:

A sintaxe para implementar uma Interface utiliza a palavra reservada implements:

public class Teste implements FiguraGeometrica

Conforme vimos em aula, utilizamos a palavra-reservada implements. **Gabarito: E**

8. (PaqTcPB - 2012 - UEPB - Técnico em Informática - Programador) Em linguagem de programação, um identificador é o nome que utilizamos para representar variáveis, classes, objetos. etc. Em Java, qual dos itens abaixo não é um identificador válido?

a) falso

b) true

c) maior_valor

d) Mp10

e) xBACON

Comentários:

1. Não pode ser uma palavra-reservada (palavra-chave);
2. Não pode ser true, false ou null;
3. Não pode começar com números;
4. Não pode conter espaços em branco ou caracteres de formatação;

Conforme vimos em aula, não se pode utilizar true. **Gabarito: B**

9. (ESAF - 2008 - CGU - Tecnologia da Informação) Com relação a essa característica, é correto afirmar que:

a) métodos declarados como public em uma superclasse, quando herdados, precisam ser protected em todas as subclasses dessa classe.

b) métodos declarados como protected em uma superclasse, quando herdados, precisam ser protected ou public nas subclasses dessa classe.

c) o nível de acesso protected é mais restritivo do que o nível de acesso default.

d) métodos declarados como public só podem ser acessados a partir dos métodos da própria classe ou de classes derivadas.

e) métodos declarados como default só podem ser acessados a partir dos métodos da própria classe.

Comentários:

(a) Não, essa afirmação não faz qualquer sentido; (b) Sim, as subclasses nunca podem ser mais restritivas que as superclasses; (c) Não, default é mais restritivo; (d) Não, eles podem ser acessados por quaisquer métodos de quaisquer classes ou pacotes; (e) Não, eles podem ser acessados pela própria classe, pelas subclasses e pelas classes do mesmo pacote. **Gabarito: B**

10. (UFBA - 2009 – UFBA – Analista de Sistemas) O bloco finally em uma instrução try catch finally sempre será executado quer ocorra ou não uma exceção no bloco try.

Comentários:

Podemos encadear vários blocos catch, dependendo do número de exceções que podem ser lançadas por uma classe ou método. O bloco catch obtém o erro criando uma instância da exceção. Quando uma exceção é lançada e é necessário que determinada ação seja tomada mesmo após a sua captura, utilizamos a palavra reservada finally – é opcional, mas se existir, sempre será executado.

Conforme vimos em aula, é um bloco opcional. No entanto, caso exista, sempre será executado – com exceção ou não! **Gabarito: C**

11. (CONSULPLAN - 2007 - Chesf - Analista de Sistemas – I) É possível utilizar vários blocos catch para capturar exceções vindas de um único bloco try.

Comentários:

Podemos encadear vários blocos catch, dependendo do número de exceções que podem ser lançadas por uma classe ou método. O bloco catch obtém o erro criando uma instância da exceção. Quando uma exceção é lançada e é necessário que determinada ação seja tomada mesmo após a sua captura, utilizamos a palavra reservada finally – é opcional, mas se existir, sempre será executado.

Conforme vimos em aula, podemos ter vários blocos catch de um único try. **Gabarito: C**

12. (AOCP - 2012 - BRDE - Analista de Sistemas - Desenvolvimento de Sistemas - III) Java threads são objetos que podem cooperar e comunicar-se entre si para compartilhar objetos em memória, a tela, ou outros tipos de recursos e periféricos.

Comentários:

Quando nós executamos essas tarefas em paralelo, estamos usando Threads! Vocês sabem o que essa palavra significa em português? Fios ou Linhas! Em outras palavras, criamos linhas de execução de tarefas paralelas em memória – cada linha responsável por executar alguma coisa simultaneamente e relativamente independentes. Em Java, as Threads são objetos presentes no Pacote java.lang.

Conforme vimos em aula, realmente são objetos que representam recursos compartilhados em memória. **Gabarito: C**

13. (FGV – 2009 – MEC - Analista de Sistemas – D) Swing é um mecanismo simples e consistente para estender a funcionalidade de um servidor web e para acessar existentes sistemas de negócio.

Comentários:

A Plataforma Java oferece recursos para construção de interfaces gráficas de usuário (GUI), entre eles: AWT (java.awt) e Swing (javax.swing)! O primeiro é um conjunto básico de classes e interfaces que definem os componentes de uma janela desktop. Já o Swing é um conjunto sofisticado de classes e interfaces que definem os componentes visuais necessários para construir uma interface gráfica de usuário.

Conforme vimos em aula, a questão não faz o menor sentido – não tem absolutamente nada a ver com Swing. **Gabarito: E**

14. (ESAF – 2008 – CGU - Analista de Sistemas) A linguagem Java possui uma API (Application Program Interface) que disponibiliza pacotes e classes com diversas funcionalidades para auxiliar no desenvolvimento de aplicações. O pacote que contém classes que auxiliam na criação de interfaces de usuário, incluindo tratamento de gráficos e imagens, é denominado:

a) java.util.

b) java.applet.

c) java.graphic.

d) java.image.

e) java.awt.

Comentários:

A Plataforma Java oferece recursos para construção de interfaces gráficas de usuário (GUI), entre eles: AWT (java.awt) e Swing (javax.swing)! O primeiro é um conjunto básico de classes e interfaces que definem os componentes de uma janela desktop. Já o Swing é um conjunto sofisticado de classes e interfaces que definem os componentes visuais necessários para construir uma interface gráfica de usuário.

Conforme vimos em aula, trata-se do pacote java.awt. **Gabarito: E**

15. (FGV – 2015 – PGE/RO – Analista de Sistemas) Na linguagem de programação Java, para indicar que uma classe A é derivada de B, utiliza-se, na declaração de A, o modificador:

a) imports;

b) extends;

c) inherits;

d) subclass;

e) superclass.

Comentários:

A palavra-chave extends faz com que uma subclasse herde (receba) todos os atributos e métodos declarados na classe-pai (desde que ela não seja final), incluindo todas as classes-pai da classe-pai. A classe-filha pode acessar todos os atributos e métodos não-privados. Ela herda, mas não acessa (ao menos diretamente) métodos e atributos privados.

Conforme vimos em aula, trata-se do extends. **Gabarito: B**

16. (FGV – 2015 – PGE/RO – Analista de Sistemas) São tipos primitivos na linguagem de programação Java:

a) int, float, double, char, boolean;

b) int, double, string, char, boolean;

c) integer, real, byte, char, boolean;

d) byte, word, short, integer, char;

e) int, real, char, string, boolean.

Comentários:

Vamos falar agora sobre uma das linguagens mais famosas do mundo! Professor, o que é Java? É uma linguagem de programação orientada a objetos, multiplataforma, robusta, portável, segura, extensível, concorrente e distribuída. E ela é totalmente orientada a objetos? Não! Por que não? Porque nem todos os seus tipos de dados são objetos (possui alguns tipos primitivos: int, float, long, double, char, etc).

Conforme vimos em aula, trata-se do int, float, double, char e boolean. **Gabarito: A**

17. (FGV – 2014 – TJ/GO – Analista de Sistemas) Se uma classe na linguagem Java é declarada com o modificador abstract, então essa classe:

a) não pode ser referenciada;

b) não pode ser estendida;

c) não pode ser instanciada;

d) pode ser instanciada apenas uma vez;

e) não pode possuir métodos estáticos.

Comentários:

Palavras	Descrição
abstract	Aplicado a um método ou classe indica que a implementação completa deste método ou classe é efetuada posteriormente, por uma subclasse. Caso seja uma classe, significa que ela não pode ser instanciada.

Conforme vimos em aula, ela não pode ser instanciada. **Gabarito: C**

18. (FGV – 2014 – TJ/GO – Analista de Sistemas) Na linguagem de programação Java, uma classe declarada com o modificador final:

a) não pode ser instanciada;

b) não pode ser estendida;

c) pode ter o modificador abstract também presente na declaração;

d) não pode ter métodos estáticos;

e) não pode ter métodos de instância.

Comentários:

Portanto para declarar uma classe, deve-se colocar a palavra class seguida de um identificador que irá servir de nome para a classe. O identificador pode ser qualquer palavra, exceto palavras reservadas. Por exemplo: class Conta introduz a declaração de uma nova classe chamada Conta. Note que, por convenção, o nome de uma classe inicia sempre com uma letra maiúscula. A Palavra-Chave é opcional, podendo ser:

```
//Essa classe pode ser acessada por todos
public class Carro {...}

//Essa classe não pode gerar instâncias
abstract class Carro {...}

//Essa classe não pode ser estendida
final class Carro {...}
```

Conforme vimos em aula, ela não pode ser estendida. **Gabarito: B**

19. (FGV – 2010 – BADESC – Analista de Sistemas) Observe o código em Java a seguir, em que se pode verificar a aplicação dos operadores de pré-decremento e pós-decremento.

```
public class Decrementa {
    public static void main (string args {} )
    {
        int m, n = 44;
    m = --n;
    m = n--;
    system.out.println (m);
    system.out.println (n);
    }
}
```

Após a execução do código, as variáveis m e n exibirão, respectivamente, os valores:

a) 42 e 41.

b) 42 e 42.

c) 42 e 43.

d) 43 e 42.

e) 43 e 43.

Comentários:

Vamos analisar essa questão! O examinador queria saber se o aluno conhecia os operadores de pré-incremento e os operadores de pós-incremento. Observem que na linha 5, temos um operador de pré-incremento. Isso significa que a m será atribuído o valor de n-1, ou seja, 43. Ao final dessa linha, m = n = 43. Na linha 6, temos um operador de pós-incremento. Isso significa que a m será atribuído o valor de n, ou seja, 43 e só depois será feita a operação, i.e., m = 43, mas ao final dessa linha, n = 42. Então, temos m = 43 e n = 42. Agora o engraçado é que nada isso será executado porque o comando não é system.out, mas System.out, mas a questão ignorou esse errinho. **Gabarito: D**

20. (FGV – 2015 – TJ/BA – Analista de Sistemas) Em Java, os métodos declarados sem modificadores em uma interface são implicitamente:

a) públicos e estáticos;

b) públicos e abstratos;

c) privados e estáticos;

d) públicos e finais;

e) privados e abstratos.

Comentários:

Galera, se o método é declarado sem modificador dentro de uma interface, então ele é implicitamente público e evidentemente abstrato. **Gabarito: B**

21. (VUNESP – 2009 – CETESB – Analista de Sistemas) Na linguagem de programação Java, o compilador JavaDoc somente compila as tags de documentação que são iniciadas:

a) pela tag #jd#.

b) pela tag <doc></doc>.

c) pelo caractere #.

d) pelo caractere @.

e) pelo caractere <!-- -->.

Comentários:

Ficou fácil! Pelo caractere @. **Gabarito: D**

22. (CESGRANRIO - 2011 – TRANSPETRO - Analista de Sistemas) Muito utilizada para desenvolvimento de aplicativos Web, a tecnologia Java tem como principal característica gerar aplicações que rodam em qualquer dispositivo que tenha acesso a Internet, utilizando, entre outros recursos, o software:

a) JBC (Java Bytecode Console)

b) JDB (Java Developer Builder)

c) JMS (Java Management Server)

d) JAC (Java Application Controler)

e) JVM (Java Virtual Machine)

Comentários:

Por conta deles, programas escritos em Java podem funcionar em qualquer plataforma de hardware e software que possua uma JVM, tornando assim essas aplicações independentes da plataforma, como apresenta a imagem acima (Win32, UNIX e MacOS)! Galera, qualquer plataforma... desde um computador a uma geladeira. A imagem abaixo é similar à anterior, apenas para solidificar!

Conforme vimos em aula, trata-se da JVM (Java Virtual Machine). **Gabarito: E**

23. (CONSULPLAN - 2012 - TSE - Programador de computador) Diferentemente de outras linguagens de programação como C ou Pascal, Java utiliza uma

linguagem intermediária da Java Virtual Machine – JVM. Essa linguagem intermediária denomina-se

a) bytecode.

b) appletcode.

Comentários:

A Linguagem Java tem dois processos de execução de código-fonte: Compilação e Interpretação! Vamos lá... o programador escreve um código em Java em um editor de texto, por exemplo. Ele salva com a extensão .java e passa por um compilador (JavaC)! Esse compilador transforma o arquivo .java em código de máquina e em um arquivo .class, também chamado bytecode – como mostra a imagem abaixo.

Conforme vimos em aula, trata-se do bytecode! **Gabarito: A**

24. (ESAF – 2010 – MPOG - Analista de Sistemas - B) Na linguagem Java uma applet contém unicamente comandos iniciando por /* e terminando por */.

Comentários:

Que loucura é essa? Isso seria a sintaxe para comentários no código-fonte! **Gabarito: E**

25. (ESAF – 2010 – MPOG - Analista de Sistemas - C) Na linguagem Java as applets rodam no navegador web.

Comentários:

Em 1995, a tecnologia Java e seu modelo de código móvel conhecido como applet foram criados pela Sun Microsystems. Applets são pequenos programas Java que são instalados em Contêineres Web e referenciados através de Páginas HTML em um navegador. Quando um browser acessa esta página HTML que referencia o applet, ele automaticamente também transfere o código do applet e o executa.

Perfeito, conforme visto em aula! **Gabarito: C**

Lista de Questões – Java – Conceitos Básicos - Cebraspe

1. **(CESPE - 2013 - SERPRO - Analista - Desenvolvimento de Sistemas) Garbage Collector é a tecnologia que gerencia a memória alocada para o programa, a fim de liberar objetos que não estão sendo utilizados.**

2. **(CESPE - 2012 - TRE-RJ - Técnico Judiciário - Programação de Sistemas) A linguagem de programação Java é muito utilizada por ter como característica gerar um código independente de plataforma que pode ser executado em qualquer arquitetura e sistema operacional que tenha o sistema Java.**

3. **(CESPE - 2012 - Banco da Amazônia - Técnico Científico - Análise de Sistemas) No Java, a JRE possui tudo que é necessário para desenvolver programas em Java.**

4. **(CESPE - 2012 - Banco da Amazônia - Técnico Científico - Análise de Sistemas) Em Java, para toda classe, método e variável de instância que se declara há um controle de acesso, independentemente de o controle ser explicitamente indicado.**

5. **(CESPE - 2012 - Banco da Amazônia - Técnico Científico - Análise de Sistemas) O operador instanceof só pode ser usado para testar valores null.**

6. **(CESPE - 2012 - MPE-PI - Analista Ministerial - Informática - Cargo 6) Em Java, qualquer método de uma classe pode ser sobrescrito por métodos de outra classe.**

7. **(CESPE - 2011 - TJ-ES - Técnico de Informática - Específicos) O JVM (Java Virtual Machine) é um interpretador que atribui portabilidade à linguagem Java, possibilitando, consequentemente, a sua execução em qualquer sistema operacional.**

8. **(CESPE - 2011 - TRE-ES - Técnico - Programação de Sistemas – Específicos) O encapsulamento em Java somente pode ser realizado por meio do modificador de acesso protegido.**

9. **(CESPE - 2010 - TRT - 21ª Região (RN) - Técnico Judiciário - Tecnologia da Informação) A linguagem de programação Java, em razão de sua portabilidade — uma vez que o compilador Java converte o código fonte em bytecodes,**

executados por uma máquina virtual — é bastante utilizada para oferecer conteúdos dinâmicos na Web.

10. (CESPE - 2010 - TRE-BA - Técnico Judiciário - Programação de Sistemas) Em programação orientada a objetos, o pacote tem como função agrupar classes dentro de um grupo. Em Java, o pacote Swing (javax.swing) é composto de várias classes para a implementação de interfaces gráficas em desktop.

11. (CESPE - 2009 - ANAC - Analista Administrativo - Tecnologia da Informação) Pelo uso de polimorfismo, uma chamada de método pode fazer que diferentes ações ocorram, dependendo do tipo do objeto que recebe a chamada.

12. (CESPE - 2010 - EMBASA - Analista de Saneamento - Analista de Tecnologia da Informação - Desenvolvimento) O trecho de código a seguir está incorreto porque uma variável booleana em Java usa a sintaxe == e não =.

```
Public void disconnect() {
    Connected = false;
}
```

13. (CESPE - 2009 - TRE-MA - Técnico Judiciário - Programação de Sistemas) Para definição e manipulação de uma exceção em Java, devem constar no programa, obrigatoriamente, os termos:

a) try e catch.

b) try e finally.

c) finally e catch.

d) finally e retry.

e) try e retry.

14. (CESPE - 2009 - TRT - 17ª Região (ES) - Técnico Judiciário - Tecnologia da Informação) Ao contrário dos tipos primitivos que não são objetos, os tipos de objetos são determinados pela classe de origem.

15. (CESPE - 2009 - TRT - 17ª Região (ES) - Técnico Judiciário - Tecnologia da Informação) Uma classe final indica uma classe que não pode ser estendida. Um método final não pode ser redefinido em classes derivadas.

16. (CESPE - 2008 - SERPRO - Analista - Desenvolvimento de Sistemas) A linguagem Java, orientada a objetos, tem como característica ser compilada em um código executado em máquina virtual.

17. (CESPE - 2008 - TRT - 5ª Região (BA) - Técnico Judiciário - Tecnologia da Informação) A instrução import Java.awt.* indica que o programa irá utilizar componentes gráficos.

18. (CESPE - 2008 - TRT - 5ª Região (BA) - Técnico Judiciário - Tecnologia da Informação) Em Java, os métodos public de uma classe são utilizados pelos clientes da classe para manipular dados armazenados em objetos dessa classe.

19. (CESPE - 2013 - SERPRO - Analista - Desenvolvimento de Sistemas) A tecnologia JSE (Java Small Editon) utilizada pela plataforma iOS permite o desenvolvimento de aplicações para todos os seus dispositivos, como estações gráficas, iPad, iPod, entre outros.

20. (CESPE - 2013 - MPOG - Tecnologia da Informação) O JME foi criado para trabalhar com ambientes de programação multiprocessados em ambientes distribuídos.

21. (CESPE - 2013 - ANTT - Analista Administrativo - Desenvolvimento de Sistemas da Informação) JSE, JME, JEE e JCE são edições da tecnologia Java voltadas para o desenvolvimento de aplicações para desktop/servidores, dispositivos móveis, ambientes corporativos e ambientes em nuvem, respectivamente.

22. (CESPE - 2008 - HEMOBRÁS - Técnico de Informática) O Java dá suporte a programação concorrente (multithreading).

23. (CESPE - 2009 - ANAC - Técnico Administrativo - Informática) A linguagem de programação Java permite operações de bit, como, por exemplo, AND (&) e OR (|).

24. (CESPE - 2010 - MPU - Técnico de Informática) Na linguagem Java, um objeto do tipo Integer pode receber valor nulo, porém uma variável primitiva int não pode.

25. (CESPE - 2013 - TRT - 10ª REGIÃO (DF e TO) - Técnico Judiciário - Tecnologia da Informação) É possível indicar que parte de um código em um método pode gerar uma exceção, por meio da utilização da palavra-chave finally.

26. (CESPE - 2014 - ANATEL - Analista Administrativo - Tecnologia da Informação e Comunicação) A linguagem Java possui checked exceptions que estendem java.lang.Exception, em que o compilador força o programador a capturar tais exceções. Essas exceções devem ser tratadas com um bloco try-catch ou com um throws.

27. (CESPE – 2004 – STJ - Analista Judiciário - Análise de Sistemas) O Java collections framework da API Java J2SE possui um conjunto de interfaces e implementações que define estruturas usadas para manipular coleções de objetos. As interfaces fundamentais do framework estão associadas à identificação de funcionalidades típicas de estruturas de dados clássicas. Assim, a interface java.util.List está ligada a estruturas de listas, a interface java.util.Set está associada a estruturas do tipo conjuntos e a interface java.util.Map refere-se a estruturas do tipo mapas. Set, List e Map possuem a interface abstrata java.util.Collection como superinterface.

28. (CESPE – 2004 – STJ - Analista Judiciário - Análise de Sistemas) As classes java.util.LinkedList e java.util.ArrayList são implementações para listas encadeadas e coleções do tipo arranjos com tamanho modificável, respectivamente, para a interface java.util.List.

29. (CESPE – 2006 – ANCINE – Analista de Sistemas) Os tipos de dados HashSet, ArrayList e HashMap são classes que implementam os tipos de dados Set, List e Map, respectivamente.

30. (CESPE – 2006 – SGA/AC – Analista de Sistemas – C) ArrayList implementa a interface List e Collection.

31. (CESPE – 2008 – MPE/RR - Analista de Sistemas) O pacote java.awt contém diversas classes para criar interfaces gráficas de usuário em aplicações Java.

32. (CESPE – 2008 – MPE/RR - Analista de Sistemas) No AWT, um componente é qualquer classe que possa ser representada em uma tela sem interação com usuário.

33. (CESPE – 2008 – MPE/RR - Analista de Sistemas) O Swing suporta o desenvolvimento de interfaces gráficas para usuários (GUI) com o uso da IDE NetBeans

34. (CESPE – 2011 – MEC - Analista de Sistemas) AWT é um conjunto de componentes GUI, integrante da arquitetura JEE6, constante no pacote java.awt, desenvolvidas para substituir as GUIs do SWING.

35. (CESPE – 2015 – TCU - Analista de Sistemas) No contexto de um código na linguagem Java, o comando a seguir é utilizado com a finalidade específica de passar para o campo de visibilidade pública saldo o valor da variável deposito com o tipo double. public void saldo (double deposito).

36. (CESPE – 2014 – ANATEL - Analista de Sistemas) No JSE (Java Standard Edition) versão 8, é possível utilizar recursos inerentes à programação funcional por meio de uma nova característica da linguagem chamada expressões Lambda, que permitem o tratamento de funções como argumentos de métodos.

37. (CESPE – 2017 – TRE/BA - Analista de Sistemas) No JSE (Java Standard Edition) versão 8, é possível utilizar recursos inerentes à programação funcional por meio de uma nova característica da linguagem chamada expressões Lambda, que permitem o tratamento de funções como argumentos de métodos.

```
1 class abc {
2   String nome;
3   String cpf;
4   double salario;
5   int senha;
6 }
7 class xyz extends abc {
8   int numerodeImoveis;
9 }
```

No trecho de código antecedente, é correto apenas o que se afirma na(s):

a) linhas 7 e 8, pois há a especificação de sobrecarga.

b) linhas 7 e 8, pois há a especificação de sobrescrita.

c) linha 8, pois há especificação de uma variável privada que somente pode ser acessada pela classe xyz.

d) linha 7, pois há uma especificação de herança entre xyz e abc.

e) linha 8, porque há um erro material, pois não pode haver novos atributos nesse tipo de especificação.

38. (CESPE - 2004 – ABIN - Analista de Sistemas) A máquina virtual Java (Java Virtual Machine — JVM) é especificada para interpretar instruções expressas

em bytecodes e compiladas a partir de uma linguagem de programação do tipo Java, mas não necessariamente Java.

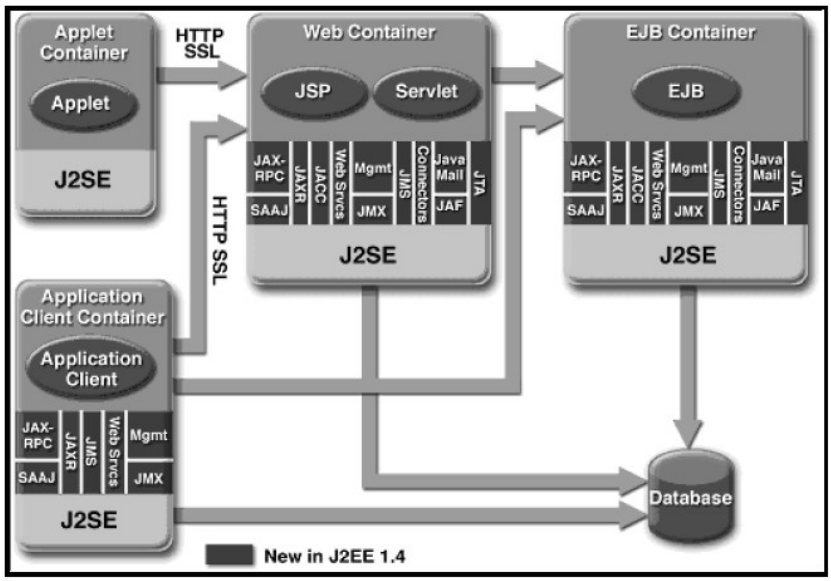
39. (CESPE - 2010 – TRE/MT - Analista de Sistemas – A) JVM é um tipo de linguagem de máquina, resultado da compilação do código-fonte Java, que é interpretado e executado pela máquina virtual Java.

40. (CESPE - 2011 – TJ/ES - Analista de Sistemas) O JVM (Java Virtual Machine) é um interpretador que atribui portabilidade à linguagem Java, possibilitando, consequentemente, a sua execução em qualquer sistema operacional.

41. (CESPE - 2008 - TJ-DF - Analista Judiciário - Tecnologia da Informação) Na linguagem Java, durante a interpretação do código, a JVM (Java Virtual Machine) verifica se o applet faz tentativas de forjar ponteiros, de violar restrições de acesso em membros de classes privadas e de gerar falhas na pilha.

42. (CESPE - 2004 - SEAD - Analista de Sistemas) O programa abaixo escreverá, na tela do computador no qual for executado como aplicação ou Applet, o trecho Ola mundo.

```
import java.applet.Applet;
import java.awt.Graphics;
public class Ola extends Applet {
    public static void main (String[] args) {
System.out.println("Ola mundo");
}
    public void paint(Graphics g) {
        g.drawString("Ola mundo", 50, 25);
    }
}
```



43. (CESPE - 2006 – ANCINE - Analista de Sistemas) Para que um sistema aplicativo funcione nessa arquitetura, é necessário que os contêineres Web e EJB utilizem máquinas com o mesmo sistema operacional, seja ele Windows ou Linux, embora os contêineres Applet e de aplicação cliente possam ser heterogêneos com relação ao sistema operacional.

44. (CESPE - 2006 – DATAPREV - Analista de Sistemas) Os aplicativos Java que executam no Applet Container são instalados originalmente no Web Container.

45. (CESPE - 2006 – DATAPREV - Analista de Sistemas) Um applet pode ser armazenado em um servidor e depois transferido para as máquinas dos usuários. Uma classe que modela um applet deriva da classe Applet e contém um método init executado na carga do applet.

46. (CESPE - 2008 – TJDFT - Analista de Sistemas) Na linguagem Java, durante a interpretação do código, a JVM (Java Virtual Machine) verifica se o applet faz tentativas de forjar ponteiros, de violar restrições de acesso em membros de classes privadas e de gerar falhas na pilha.

47. (CESPE - 2009 – UNIPAMPA - Analista de Sistemas) O trecho de código a seguir é um exemplo de como inserir applet em um documento HTML.

```
<applet
 code=MinhaClasse.class
 width=120
 height=120>
</applet>
```

48. (CESPE - 2008 - TRT - 5ª Região (BA) - Técnico Judiciário - Tecnologia da Informação) Applets são componentes que podem ser executados tanto do lado servidor quanto do lado cliente em qualquer aplicação Java.

GABARITO

1. C
2. C
3. E
4. C
5. E
6. E
7. C
8. E
9. C
10. C
11. C
12. E
13. A
14. C
15. C
16. C
17. C
18. C
19. E
20. E
21. E
22. C
23. C
24. C
25. E
26. C
27. E
28. C
29. C
30. E
31. C
32. E
33. C
34. E
35. E
36. C
37. D
38. C
39. E
40. C
41. C
42. C
43. E
44. C
45. C
46. C
47. C
48. E

Lista de Questões – Java – Conceitos Básicos - FCC

1. **(FCC - 2011 - TRE-RN - Técnico Judiciário - Programação de Sistemas) Em relação ao Java Standard Edition, é INCORRETO afirmar:**

   a) Possui gerenciamento de memória embutido, por meio do coletor de lixo.

   b) Ambiente indicado para o desenvolvimento de aplicativos para dispositivos móveis ou portáteis.

   c) Permite o desenvolvimento de aplicações desktop de linha de comando e interfaces gráficas Swing.

   d) Portabilidade dos programas compilados para diversos sistemas operacionais, sem necessidade de recompilação.

   e) Usa conceitos tais como orientação a objetos e multithreading.

2. **(FCC - 2010 - TRT - 22ª Região (PI) - Técnico Judiciário - Tecnologia da Informação) A plataforma Java disponibiliza um interpretador que traduz, em tempo de execução, o bytecode para instruções nativas do processador, permitindo, dessa forma, que uma mesma aplicação seja executada em qualquer plataforma computacional que possua essa implementação. Trata-se de:**

   a) Java Virtual Machine.

   b) Java API.

   c) JavaBeans.

   d) J2SE.

   e) JavaFX.

3. **(FCC - 2010 - Sergipe Gás S.A. - Analista de Sistemas) É tida como uma das principais linguagens de programação orientada a objeto; tem como característica a compilação para um bytecode e execução por uma máquina virtual. Trata-se da linguagem:**

   a) Algol.

   b) Delphi.

   c) C++.

   d) Java.

   e) PHP.

4. **(FCC - 2010 - TCE-SP - Agente da Fiscalização Financeira - Informática - Suporte de Web) Os aplicativos Java "rodam" em diferentes ambientes. A tradução dos**

códigos Java (bytecode), para instruções específicas de cada sistema e dispositivo, é uma função do programa:

a) Java Community Process (JCP).

b) Java Virtual Module (JVM).

c) Java Virtual Machine (JVM).

d) Java Comunication Process (JCP).

e) Java Enterprise Machine (JEM).

5. **(FCC - 2009 - TJ-SE - Técnico Judiciário - Programação de Sistemas) Um objeto é instanciado em Java por meio do operador:**

a) instanceof.

b) extend.

c) new.

d) this.

e) type.

6. **(FCC - 2009 - TRT - 16ª REGIÃO (MA) - Técnico Judiciário - Tecnologia da Informação) Uma classe Java pode ser instanciada por um comando, cuja sintaxe é:**

a) nome_Objeto nome_Classe = new nome_Objeto();

b) nome_Classe nome_Objeto = new nome_Classe();

c) nome_Classe nome_instancia = new nome_Objeto();

d) nome_Instancia nome_Objeto = new nome_Instancia();

e) nome_Instancia nome_Classe = new nome_Instancia();

7. **(FCC - 2009 - TRT - 16ª REGIÃO (MA) - Técnico Judiciário - Tecnologia da Informação) A diretiva public é utilizada em Java para aplicar a encapsulação pública:**

a) aos métodos e classes, apenas.

b) aos atributos, métodos e classes.

c) às classes, apenas.

d) aos atributos, apenas.

e) aos atributos e classes, apenas.

8. **(FCC - 2008 - TCE-AL - Programador) Em Java, para alterar a visibilidade do elemento em que se aplica, entre outros, utiliza-se o modificador de acesso:**

a) static.

b) abstract.

c) protected.

d) volatile.

e) transient.

9. (FCC - 2007 - MPU - Analista de Informática - Desenvolvimento de Sistemas) Analise os seguintes valores, variáveis e operações usando expressões Java:

```
byte j = 30;
short k = 54;
int m = 40;
long n = 12L;
long resultado = 0L;
resultado += j;
resultado += k;
resultado /= n;
resultado -= m;
```

Após a última operação, o resultado será igual a:

a) -7.

b) -32.

c) -33.

d) 60.

e) 84.

10. (FCC - 2005 - TRE-MG - Técnico Judiciário - Programação de Sistemas) Os métodos Java que não retornam valores devem possuir no parâmetro tipo-de-retorno a palavra:

a) static.

b) public.

c) void.

d) main.

e) string args.

11. (FCC - 2012 - TST - Analista Judiciário - Análise de Sistemas) Considere o programa abaixo escrito na linguagem Java:

```
public class Programa
{
public static void main(String args[])
{
    for(int i=3; i<20 ; i+=2)
    {
            System.out.print((i%3) + " ");
    }
}
}
```

O resultado a ser informado ao usuário após a execução do programa acima é:

a) 0 0 1 0 0 1 0 0 1

b) 0 1 2 0 1 2 0 1 2

c) 0 1 0 1 0 1 0 1 0

d) 1 2 1 2 1 2 1 2 1

e) 0 2 1 0 2 1 0 2 1

12. (FCC - 2012 - MPE-AP - Analista Ministerial - Tecnologia da Informação) Analise o código das classes a seguir presentes em um mesmo pacote de um projeto Java:

```
public class NewClassA {
   public double calcular(int x, int y) {
      return x + y;
   }

   public double calcular(double x, double y) {
      return x * y;
   }
}

public class NewClassB extends NewClassA {

}

public class Start {

   public static void main(String[] args) {

   }
}
```

Com base nos códigos apresentados e nos conceitos da orientação a objetos é correto afirmar:

a) No método main da classe Start não é possível instanciar objetos das classes NewClassA e NewClassB, pois essas classes não contêm um construtor válido.

b) Se for digitada a instrução NewClassB c = new NewClassA(); no método main da classe Start será instanciado um objeto da NewClassA.

c) Se for digitada a instrução NewClassA b = new NewClassB(); no método main da classe Start ocorrerá um erro, pois não é possível criar um objeto da NewClassA por meio do construtor da NewClassB.

d) A existência de dois métodos de mesmo nome na NewClassA que recebem a mesma quantidade de parâmetros indica que está ocorrendo uma sobrescrita de métodos.

e) Por meio de um objeto da NewClassB será possível acessar os métodos presentes na NewClassA.

13. (FCC - 2012 - TCE-SP - Auxiliar de Fiscalização Financeira) Em um programa Java, considere a existência de uma variável do tipo long chamada cod contendo o valor 1234. Para passar o valor contido nessa variável para uma variável do tipo byte chamada codNovo, deve-se fazer casting. Para isso, utiliza-se a instrução: byte codNovo =

a) Byte.valueOf(cod);

b) (long) cod;

c) Byte.pasreByte(cod);

d) (byte) cod;

e) (cast) cod;

14. (FCC - 2012 - TRE-CE - Técnico Judiciário - Programação de Sistemas) Para chamar o método soma da classe Calculo, e mostrar na tela o retorno desse método, é correto utilizar:

```
public class Calculo {
    public static double soma(double n1, double n2){
        return n1 + n2;
    }
    public static double soma(double n1, double n2, double n3){
        return n1 + n2 + n3;
    }
}
```

a) Calculo c = new Calculo(); System.out.println (c.soma(10, 20, 30)); ou System.out.println (Calculo.soma(10, 20));

b) Exclusivamente as instruções Calculo c = new Calculo(); System.out.println (c.soma(10, 20));

c) Exclusivamente a instrução System.out.println (Calculo.soma(10, 20, 50));

d) Exclusivamente as instruções Calculo c = new Calculo(); double r = c.soma(10, 20); System.out.println(r);

e) Calculo c = Calculo.soma(10,20,30); System.out.println (c); ou System.out.println (Calculo.soma(10, 20));

15. (FCC - 2012 - TRE-CE - Técnico Judiciário - Programação de Sistemas) Com relação a herança na programação orientada a objetos com Java, é INCORRETO afirmar:

a) Uma subclasse herda os métodos da superclasse, entretanto, pode ter seus próprios métodos.

b) Quando se instancia um objeto da subclasse, podem ser passados valores para os atributos da superclasse.

c) Um objeto da subclasse pode ser um objeto da superclasse.

d) Em uma superclasse, para acessar métodos da subclasse deve ser usada a instrução super.

e) Para definir que a subclasse herda as características da superclasse utiliza-se a instrução extends na declaração da subclasse.

16. (FCC - 2011 - TRT - 4ª REGIÃO (RS) - Técnico Judiciário - Tecnologia da Informação) No ambiente de programação Java:

a) uma classe abstrata permite apenas métodos abstratos.

b) o corpo de um método abstrato termina com ponto e vírgula e a declaração é delimitada por chaves.

c) uma interface pode definir tanto métodos abstratos quanto não abstratos.

d) a herança múltipla permite que mais classes sejam estendidas.

e) toda classe é uma subclasse direta ou indireta da classe Object.

17. (FCC - 2010 - TRT - 8ª Região (PA e AP) - Analista Judiciário - Tecnologia da Informação) São tipos primitivos da linguagem Java:

a) int, string, long e real.

b) char, int, real e bit.

c) boolean, double, float e byte.

d) real, short, long e char.

e) string, long int, short int e float.

18. (FCC - 2010 - AL-SP - Agente Legislativo de Serviços Técnicos e Administrativos - Processamento de Dados) Os tipos de dados primitivos em Java são:

a) char, boolean, byte, short, int, long, float e double.

b) char, boolean, byte, short, int, long, float, double e String.

c) byte, short, int, long, float e double.

d) byte, short, int, long, float, double, String e Date.

e) char, boolean, byte, short, int, long, float, double, String e Date.

19. (FCC - 2009 - TRT - 15ª Região - Analista Judiciário - Tecnologia da Informação) No âmbito da linguagem Java, considere:

I. Edição é a criação do programa, que também é chamado de código Bytecode.

II. Compilação é a geração de um código intermediário chamado fonte, que é um código independente de plataforma.

III. Na interpretação, a máquina virtual Java ou JVM analisa e executa cada instrução do código Bytecode.

IV. Na linguagem Java a interpretação ocorre apenas uma vez e a compilação ocorre a cada vez que o programa é executado.

Está correto o que consta em:

a) I, II, III e IV.

b) II e IV, somente.

c) III e IV, somente.

d) IV, somente.

e) III, somente.

20. (FCC - 2008 - TCE-AL - Programador) Os três elementos básicos quando contidos num arquivo fonte Java devem obrigatoriamente se apresentar na seguinte ordem:

a) import, package e class.

b) class, package e import.

c) class, import e package.

d) package, class e import.

e) package, import e class.

21. (FCC - 2014 – TRF/3 – Analista de Sistemas) Considere a classe escrita em Java:

```
public class Teste {
   public float multi(float flt, int n) {
         return flt * n + 1;
   }

   public int multi(int dbl, double n) {
         return dbl * (int) n + 2;
   }

   public double multi(double i, double n) {
         return i * n + 3;
   }

   public static void main(String[] args) {
         Teste t = new Teste();

         System.out.println(t.multi(2.5, 2));
   }
}
```

O valor que será impresso na execução do método main é:

a) 6.0

b) 7

c) 8.0

d) 5.0

e) 12.5

22. (FCC - 2008 - TCE-AL - Programador) Considerando que as variáveis Java X, Y e Z foram todas inicializadas com zero, os resultados das mesmas após as alterações realizadas pelas atribuições X *= 2, Y -= 5 e Z /=3, respectivamente, serão:

a) 0, -5 e 0

b) 0, 5 e 0

c) 1, -5 e 3

d) 2, -5 e 3

e) 2, 5 e 3

23. (FCC - 2008 - MPE-RS - Técnico em Informática - Área Sistemas) A função Java:

public boolean VerificarCPF (string CPF);

representa um exemplo do conceito de:

a) override.

b) overload.

c) herança.

d) encapsulamento.

e) polimorfismo.

24. (FCC - 2007 - MPU - Analista de Informática - Desenvolvimento de Sistemas) Quanto às variáveis Java, um inteiro de 64 bits em notação de complemento de dois que pode assumir valores entre -263 e 263 -1 é:

a) long.

b) short.

c) float.

d) byte.

e) double.

25. (FCC - 2005 - TRE-MG - Técnico Judiciário - Programação de Sistemas) A seqüência de etapas para implementação de um programa Java é:

a) interpretação, codificação, execução e compilação.

b) codificação, interpretação, compilação e execução.

c) interpretação, codificação, compilação e execução.

d) codificação, compilação, interpretação e execução.

e) compilação, codificação, execução e interpretação.

26. (FCC - 2012 - TRE-CE - Técnico Judiciário - Programação de Sistemas) Considere a variável idade declarada no método main de uma classe Java, com a seguinte instrução:

int idade=12;

Analise:

I. System.out.println (idade<18?"Menor de idade":"Maior de idade");

II. if(idade<18) {System.out.println("Menor de idade");} else {System.out.println("Maior de idade");}

III. if(idade<18) {System.out.println("Menor de idade");} else if (idade>=18) {System.out.println("Maior de idade");}

IV. switch(idade) {case<18: System.out.println("Menor de idade"); break; case>=18: System.out.println("Maior de idade");}

Contém uma instrução correta que exibirá na tela a frase "Menor de idade":

a) I, II, III e IV.

b) I, II e III, apenas.

c) II e III, apenas.

d) II, apenas.

e) I e III, apenas.

27. (FCC - 2012 - TRE-CE - Técnico Judiciário - Programação de Sistemas) Considere a estrutura de repetição seguinte:

```
public static void main(String[] args) {
    int cont=1,r=0;
    cont=1;
    do {
       r=r+cont;
       cont+=4;
    } while (cont<=5);
    System.out.println(r);
    System.out.println(cont);
}
```

A saída na tela será:

a) 15 e 6.

b) 1 e 5.

c) 0 e 1.

d) 6 e 9.

e) 9 e 7.

28. (FCC - 2011 - TRE-AP - Técnico Judiciário - Programação de Sistemas) Em relação à plataforma de desenvolvimento JSE, considere:

I. Possibilita o desenvolvimento de aplicações desktop através de linha de comando e através da interface gráfica Swing.

II. É multiplataforma: permite a portabilidade dos programas compilados para diversos sistemas operacionais, sem necessidade de alteração do código ou de recompilação.

III. Faz uso explícito de ponteiros e usa conceitos modernos, tais como, orientação a objetos e suporte a multithreading.

IV. Possui o gerenciamento de memória embutido, por meio do garbage collector.

Está correto o que se afirmar em:

a) I, II e III, somente.

b) I, II e IV, somente.

c) I, III e IV, somente.

d) II, III e IV, somente.

e) I, II, III e IV.

29. (FCC - 2011 - INFRAERO - Analista de Sistemas - Desenvolvimento e Manutenção) No Java, um tipo inteiro (int) utiliza quatro bytes para armazenamento. A faixa máxima possível de valores inteiros para se armazenar em uma variável do tipo primitivo int é de:

a) -8388608 a 8388607.

b) -128 a 127.

c) -32768 a 32767.

d) -9223372036854775808 a 9223372036854775807.

e) -2147483648 a 2147483647.

30. (FCC - 2010 - TRT - 9ª REGIÃO (PR) - Técnico Judiciário - Tecnologia da Informação) O JVM mais o núcleo de classes da plataforma Java e os arquivos de suporte formam o:

a) o J2EE.

b) o JDK.

c) o JRE.

d) uma JSP.

e) uma API.

31. (FCC - 2014 - TRT - 16ª REGIÃO (MA) - Analista Judiciário - Tecnologia da Informação) Considere as classes a seguir, presentes em uma aplicação Java orientada a objetos:

```
public class Funcionario {
private int id;
private String nome;
private double valorBase;
    public Funcionario() {
    }
    public Funcionario(int id, String nome, double valorBase) {
        this.id = id;
        this.nome = nome;
        this.valorBase=valorBase;
    }
    public double getValorBase() {
        return valorBase;
    }
    public double calcularSalario(){
        return valorBase;
    }
}

public class Mensalista extends Funcionario{
private double descontos;
    public Mensalista(double descontos, int id, String nome, double
    valorBase) {
        super(id, nome, valorBase);
        this.descontos = descontos;
    }
@Override
    public double calcularSalario(){
        return super.getValorBase() - descontos;
    }
}

public class Diarista extends Funcionario {
    private int diasPorSemana;
    public Diarista( int diasPorSemana, int id, String nome, double
    valorBase) {
        super(id, nome, valorBase);
        this.diasPorSemana = diasPorSemana;
    }
    @Override
    public double calcularSalario(){
        return super.getValorBase() * diasPorSemana;
    }
}
```

Em uma classe principal foram digitadas, no interior do método main, as seguintes linhas:

```
double s;
Funcionario f;
f=new Diarista(3,10456,"Ana Maria",90);
s = f.calcularSalario();
System.out.println(s);
f=new Mensalista(298.56,10457,"Pedro Henrique",877.56);
s = f.calcularSalario();
System.out.println(s);
```

As linhas que contêm a instrução s = f.calcularSalario( ); demonstram um conceito da orientação a objetos conhecido como:

a) encapsulamento.

b) sobrecarga de métodos.

c) polimorfismo.

d) sobrescrita de construtores.

e) métodos abstratos.

32. (FCC - 2010 - AL-SP - Agente Legislativo de Serviços Técnicos e Administrativos - Processamento de Dados) Métodos estáticos em Java são aqueles que:

a) realizam alguma tarefa que é dependente do conteúdo de algum objeto.

b) não podem ser acessados diretamente pelo nome da classe a que pertencem, mas sim por meio de um objeto da classe.

c) realizam alguma tarefa que não é dependente do conteúdo de algum objeto.

d) são acessados por objetos que não necessitam de ser instanciados explicitamente.

e) existem em subclasses de uma herança.

33. (FCC - 2008 - TCE-AL - Programador) NÃO são nomes válidos em Java:

a) _Real e $real

b) um1 e dois2

c) 3tres e tres3

d) Codigo e codigo

e) cod_valor e cod$valor

34. (FCC - 2010 - TCE-SP - Agente da Fiscalização Financeira - Informática - Suporte de Web) A tecnologia Java é, basicamente, dividida em JSE,

a) JEE e JME.

b) JEE e JPE.

c) JDE e JME.

d) JDE e JPE.

e) JEEP e JME.

35. (FCC - 2009 - TRT - 16ª REGIÃO (MA) - Técnico Judiciário - Tecnologia da Informação) Uma classe Java pode ser instanciada por um comando, cuja sintaxe é:

a) nome_Objeto nome_Classe = new nome_Objeto();

b) nome_Classe nome_Objeto = new nome_Classe();

c) nome_Classe nome_instancia = new nome_Objeto();

d) nome_Instancia nome_Objeto = new nome_Instancia();

e) nome_Instancia nome_Classe = new nome_Instancia();

36. (FCC - 2005 - TRE-MG - Programador de computador) Os erros gerados durante a execução de um programa Java devem ser controlados com uma estrutura que pode combinar o uso dos blocos:

a) try e finally, somente.

b) try e catch ou try e finally, somente.

c) try, catch e finally, somente.

d) try e catch, somente.

e) try e catch, try e finally ou try, catch e finally.

37. (FCC - 2016 – TRT 14ª RO e AC - Técnico Judiciário - Tecnologia da Informação) Para executar um programa Java deve ocorrer um processo que envolve compilação e interpretação. Quando se compila uma classe com extensão .java é gerado um arquivo com extensão:

a) .class, conhecido como bytecode, que pode ser compilado pela JVM.

b) .jar, conhecido como bytecode, que pode ser lido pela JVM.

c) .class, que instala a classe na memória virtual para ser executada.

d) .jar, que quando executado, cria um arquivo .class, que é interpretado pela JVM.

e) .class, conhecido como bytecode, que pode ser interpretado pela JVM.

38. FCC - 2014 - SABESP - Tecnólogo – Sistemas – II) O processo híbrido combina a execução eficiente e a portabilidade de programas. A base é a existência de um código intermediário, mais fácil de ser interpretado e não específico de uma plataforma computacional. O método é dividido em duas etapas: compilação para um código intermediário e interpretação desse código. Um exemplo é o Java e a JVM.

39. (FCC – 2003 – TRF/5 - Analista de Sistemas) Um programa Java, que é executado dentro de um browser Web, denomina-se:

a) API.

b) applet.

c) servlet.

d) acriptlet.

e) package.

GABARITO

1. B
2. A
3. D
4. C
5. C
6. B
7. B
8. C
9. C
10. C
11. E
12. E
13. D
14. A
15. D
16. E
17. C
18. A
19. E
20. E
21. C
22. A
23. D
24. A
25. D
26. B
27. D
28. B
29. E
30. C
31. C
32. C
33. C
34. A
35. B
36. E
37. E
38. C
39. B

LISTA DE QUESTÕES – JAVA – CONCEITOS BÁSICOS - MULTIBANCAS

1. **(FUNCAB - 2010 - PRODAM-AM - Analista de TI - Desenvolvimento de Sistemas) Seja a seguinte classe Java:**

   <mod> public class Xpto

   {

   }

   Qual das alternativas a seguir contém um modificador que ao ser usado na declaração acima em substituição ao termo <mod> impedirá que a classe Xpto seja estendida?

   a) static

   b) const

   c) abstract

   d) final

   e) virtual

2. **(ESAF - 2012 - Receita Federal - Analista Tributário da Receita Federal - Prova 2 - Área Informática) Em programação Java, o comando while:**

   a) executa um bloco exclusivamente de comandos de atribuição.

   b) executa um bloco de comandos enquanto sua condição for verdadeira.

   c) executa um bloco de comandos até que sua condição seja verdadeira.

   d) equivale ao comando what-if.

   e) é idêntico ao comando do while.

3. **(CONSULPLAN – 2006 – Prefeitura de Natal – Analista de Sistemas) Analise as afirmativas abaixo colocando V para as afirmativas Verdadeiras e F para as Falsas. A linguagem JAVA se divide nas seguintes edições:**

   ( ) J2SE (Java 2 Standard Edition) - tecnologia Java para computadores pessoais, notebooks e arquiteturas com poder de processamento e memória consideráveis.

   ( ) J2EE (Java 2 Enterprise Edition) - tecnologia Java para aplicações corporativas que podem estar na internet ou não.

( ) J2ME (Java 2 Micro Edition) - tecnologia Java para dispositivos móveis com limitações de memória ou processamento.

( ) J2FE (Java 2 Full Edition) - tecnologia Java para aplicações em computadores de grande porte (mainframe).

A seqüência está correta em:

a) F, F, F, F

b) V, V, V, F

c) V, F, F, V

d) F, V, F, V

e) V, V, V, V

4. **(Instituto Cidades - 2012 - TCM-GO - Auditor de Controle Externo - Informática) Analise:**

I. O Java refere-se tanto a uma linguagem de programação quanto a uma plataforma;

II. O Java SE (Standard Edition) é formalmente chamado de J2SE;

III. O J2EE é a edição corporativa do Java. Esta versão inclui o Java Standard Edition além de outras tecnologias como javamail, servlets, JSF e Enterprise Java Beans.

IV. O Java possui uma versão para dispositivos móveis chamada J2ME (Micro Edition).

São verdadeiras as afirmações:

a) I, II e IV, somente;

b) I, III e IV, somente;

c) II, III e IV, somente;

d) I e IV, somente;

e) Todas as afirmações.

5. **(UFBA - 2012 - UFBA - Técnico de Tecnologia da Informação) O código-fonte de um programa de computador escrito na linguagem Java, é compilado para um formato intermediário conhecido como bytecode.**

6. **(PaqTcPB - 2012 - UEPB - Técnico em Informática - Programador) Em Java, um bloco de código é:**

a) Tudo que está entre ( )

b) Tudo que está entre { }

c) Tudo que está entre [ ]

d) Tudo que está entre < >

e) Tudo que está no mesmo nível de indentação.

7. **(CESGRANRIO - 2012 - Petrobrás - Técnico de Exploração de Petróleo Júnior - Informática) Ao escrever o código da Classe PortaDeCofre em Java para que ela atenda a interface Porta, como um programador deve começar a declaração da classe?**

a) public class Porta:PortaDeCofre {

b) public class PortaDeCofre :: Porta {

c) public class PortaDeCofre inherits Porta {

d) public class PortaDeCofre extends Porta {

e) public class PortaDeCofre implements Porta {

8. **(PaqTcPB - 2012 - UEPB - Técnico em Informática - Programador) Em linguagem de programação, um identificador é o nome que utilizamos para representar variáveis, classes, objetos. etc. Em Java, qual dos itens abaixo não é um identificador válido?**

a) falso

b) true

c) maior_valor

d) Mp10

e) xBACON

9. **(ESAF - 2008 - CGU - Tecnologia da Informação) Com relação a essa característica, é correto afirmar que:**

a) métodos declarados como public em uma superclasse, quando herdados, precisam ser protected em todas as subclasses dessa classe.

b) métodos declarados como protected em uma superclasse, quando herdados, precisam ser protected ou public nas subclasses dessa classe.

c) o nível de acesso protected é mais restritivo do que o nível de acesso default.

d) métodos declarados como public só podem ser acessados a partir dos métodos da própria classe ou de classes derivadas.

e) métodos declarados como default só podem ser acessados a partir dos métodos da própria classe.

10. **(UFBA - 2009 – UFBA – Analista de Sistemas) O bloco finally em uma instrução try catch finally sempre será executado quer ocorra ou não uma exceção no bloco try.**

11. (CONSULPLAN - 2007 - Chesf - Analista de Sistemas – I) É possível utilizar vários blocos catch para capturar exceções vindas de um único bloco try.

12. (AOCP - 2012 - BRDE - Analista de Sistemas - Desenvolvimento de Sistemas - III) Java threads são objetos que podem cooperar e comunicar-se entre si para compartilhar objetos em memória, a tela, ou outros tipos de recursos e periféricos.

13. (FGV – 2009 – MEC - Analista de Sistemas – D) Swing é um mecanismo simples e consistente para estender a funcionalidade de um servidor web e para acessar existentes sistemas de negócio.

14. (ESAF – 2008 – CGU - Analista de Sistemas) A linguagem Java possui uma API (Application Program Interface) que disponibiliza pacotes e classes com diversas funcionalidades para auxiliar no desenvolvimento de aplicações. O pacote que contém classes que auxiliam na criação de interfaces de usuário, incluindo tratamento de gráficos e imagens, é denominado:

a) java.util.

b) java.applet.

c) java.graphic.

d) java.image.

e) java.awt.

15. (FGV – 2015 – PGE/RO – Analista de Sistemas) Na linguagem de programação Java, para indicar que uma classe A é derivada de B, utiliza-se, na declaração de A, o modificador:

a) imports;

b) extends;

c) inherits;

d) subclass;

e) superclass.

16. (FGV – 2015 – PGE/RO – Analista de Sistemas) São tipos primitivos na linguagem de programação Java:

a) int, float, double, char, boolean;

b) int, double, string, char, boolean;

c) integer, real, byte, char, boolean;

d) byte, word, short, integer, char;

e) int, real, char, string, boolean.

17. (FGV – 2014 – TJ/GO – Analista de Sistemas) Se uma classe na linguagem Java é declarada com o modificador abstract, então essa classe:

a) não pode ser referenciada;

b) não pode ser estendida;

c) não pode ser instanciada;

d) pode ser instanciada apenas uma vez;

e) não pode possuir métodos estáticos.

18. (FGV – 2014 – TJ/GO – Analista de Sistemas) Na linguagem de programação Java, uma classe declarada com o modificador final:

a) não pode ser instanciada;

b) não pode ser estendida;

c) pode ter o modificador abstract também presente na declaração;

d) não pode ter métodos estáticos;

e) não pode ter métodos de instância.

19. (FGV – 2010 – BADESC – Analista de Sistemas) Observe o código em Java a seguir, em que se pode verificar a aplicação dos operadores de pré-decremento e pós-decremento.

```
public class Decrementa {
    public static void main (string args {} )
    {
        int m, n = 44;
    m = --n;
    m = n--;
    system.out.println (m);
    system.out.println (n);
    }
}
```

Após a execução do código, as variáveis m e n exibirão, respectivamente, os valores:

a) 42 e 41.

b) 42 e 42.

c) 42 e 43.

d) 43 e 42.

e) 43 e 43.

20. (FGV – 2015 – TJ/BA – Analista de Sistemas) Em Java, os métodos declarados sem modificadores em uma interface são implicitamente:

a) públicos e estáticos;

b) públicos e abstratos;

c) privados e estáticos;

d) públicos e finais;

e) privados e abstratos.

21. (VUNESP – 2009 – CETESB – Analista de Sistemas) Na linguagem de programação Java, o compilador JavaDoc somente compila as tags de documentação que são iniciadas:

a) pela tag #jd#.

b) pela tag <doc></doc>.

c) pelo caractere #.

d) pelo caractere @.

e) pelo caractere <!-- -->.

22. (CESGRANRIO - 2011 – TRANSPETRO - Analista de Sistemas) Muito utilizada para desenvolvimento de aplicativos Web, a tecnologia Java tem como principal característica gerar aplicações que rodam em qualquer dispositivo que tenha acesso a Internet, utilizando, entre outros recursos, o software:

a) JBC (Java Bytecode Console)

b) JDB (Java Developer Builder)

c) JMS (Java Management Server)

d) JAC (Java Application Controler)

e) JVM (Java Virtual Machine)

23. (CONSULPLAN - 2012 - TSE - Programador de computador) Diferentemente de outras linguagens de programação como C ou Pascal, Java utiliza uma linguagem intermediária da Java Virtual Machine – JVM. Essa linguagem intermediária denomina-se

a) bytecode.

b) appletcode.

24. (ESAF – 2010 – MPOG - Analista de Sistemas - B) Na linguagem Java uma applet contém unicamente comandos iniciando por /* e terminando por */.

25. (ESAF – 2010 – MPOG - Analista de Sistemas - C) Na linguagem Java as applets rodam no navegador web.

GABARITO

1. D
2. B
3. B
4. E
5. C
6. B
7. E
8. B
9. B
10. C
11. C
12. C
13. E
14. E
15. B
16. A
17. C
18. B
19. D
20. B
21. D
22. E
23. A
24. E
25. A

ESSA LEI TODO MUNDO CONHECE:
PIRATARIA É CRIME.

Mas é sempre bom revisar o porquê e como você pode ser prejudicado com essa prática.

1. 

Professor investe seu tempo para elaborar os cursos e o site os coloca à venda.

2. 

Pirata divulga ilicitamente (grupos de rateio), utilizando-se do anonimato, nomes falsos ou laranjas (geralmente o pirata se anuncia como formador de "grupos solidários" de rateio que não visam lucro).

3. 

Pirata cria alunos fake praticando falsidade ideológica, comprando cursos do site em nome de pessoas aleatórias (usando nome, CPF, endereço e telefone de terceiros sem autorização).

4. 

Pirata compra, muitas vezes, clonando cartões de crédito (por vezes o sistema anti-fraude não consegue identificar o golpe a tempo).

5. 

Pirata fere os Termos de Uso, adultera as aulas e retira a identificação dos arquivos PDF (justamente porque a atividade é ilegal e ele não quer que seus fakes sejam identificados).

6. 

Pirata revende as aulas protegidas por direitos autorais, praticando concorrência desleal e em flagrante desrespeito à Lei de Direitos Autorais (Lei 9.610/98).

7. 

Concurseiro(a) desinformado participa de rateio, achando que nada disso está acontecendo e esperando se tornar servidor público para exigir o cumprimento das leis.

8. 

O professor que elaborou o curso não ganha nada, o site não recebe nada, e a pessoa que praticou todos os ilícitos anteriores (pirata) fica com o lucro.

Deixando de lado esse mar de sujeira, aproveitamos para agradecer a todos que adquirem os cursos honestamente e permitem que o site continue existindo.