



Módulo 18





BackEnd Java

Rodrigo Pires



A decorative graphic on the left side of the slide. It features a large cyan hexagon with the number '1' inside. Surrounding this central hexagon are several smaller hexagons of varying shades of blue and cyan. Some of these smaller hexagons contain white icons: a lightbulb, a thumbs-up, a smartphone, a magnifying glass, and a gear. There is also a network-like icon with a central node and radiating lines.

1

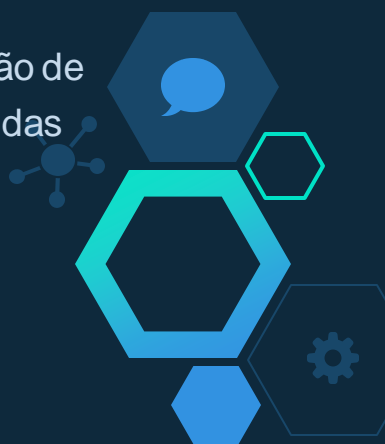
Annotation / Anotação



O que são?

As anotações Java são um mecanismo para adicionar informações de metadados ao nosso código-fonte. Elas foram introduzidas na versão 5 do Java.

Embora possamos anexá-las a classes, interfaces, métodos e campos, as anotações por si mesmas não têm efeito na execução de um programa. É preciso ter uma outra inteligência para a leitura das mesmas, com **Reflection**.





O que são?

Para definir uma anotação no código Java, usamos o símbolo arroba (@) seguido do nome da mesma. Dependendo da categoria da annotation, pode ser necessário incluir dados a ela, no formato de pares nome=valor





Exemplo

```
@Override
public String toString() {
    return "Cliente{" +
        "nome='" + nome + '\'' +
        ", cpf=" + cpf +
        "'}";
}
```





Tipos de anotações:

- Anotações marcadoras
- Anotações de valor único
- Anotações completas





Anotações marcadoras

São aquelas que não possuem membros. São identificadas apenas pelo nome, sem dados adicionais. Por exemplo:

```
@Override
public String toString() {
    return "Cliente{" +
        "nome='" + nome + '\'' +
        ", cpf=" + cpf +
        '}';
}
```






Anotações de valor único

São similares às anteriores, no entanto, possuem um único membro, chamado valor/value. Elas são representadas pelo nome da anotação e um par nome=valor, ou simplesmente com o valor, entre parênteses.

```
@Target(ElementType.ANNOTATION_TYPE)  
public @interface Persistente {
```

```
@Target(value = ElementType.ANNOTATION_TYPE)  
public @interface Persistente {
```






Anotações completas

São aquelas que possuem múltiplos membros. Portanto, neste tipo, devemos usar a sintaxe completa para cada par nome=valor. Neste caso, cada par é informado separado do outro por uma vírgula. Por exemplo, @Version(major=1, minor=0, micro=0)

```
@Deprecated(forRemoval = true, since = "1.2")
public interface Persistente {

    public Long getCodigo();
}
```





Meta-anotações

São anotações utilizadas na criação de anotações ou na marcação.
Estão no pacote **java.lang.annotation**

- @Retention
- @Documented
- @Target
- @Inherited






@Retention

As anotações podem estar presentes apenas no código fonte ou no binário de classes ou interfaces.

Ela suporta três valores:


- **SOURCE**, para indicar que as anotações marcadas não estarão no código binário.
 - **CLASS**, para gravar as anotações no arquivo .class, mas não estarão disponíveis em tempo de execução.
 - **RUNTIME**, para indicar que as anotações estarão disponíveis em tempo de execução.
- 



Exemplo: @Retention

```
@Retention(RetentionPolicy.RUNTIME)  
public @interface ExemploAnotacao {  
}  

```





@Documented


É uma anotação marcadora usada para indicar que os tipos anotação anotados com ela serão incluídos na documentação Javadoc.





Exemplo: @Documented

```
@Documented  
@Retention(RetentionPolicy.RUNTIME)  
public @interface ExemploAnotacao {  
}
```





@Target

Ao criar um tipo de anotação é possível estabelecer qual elementos de uma classe podem ser anotados com ele.

(construtor, variável local, parâmetro de método, método e etc)






@Target

Para obter esse efeito, usamos @Target, a qual suporta os seguintes valores (cada um destinado a definir o elemento que se pretende anotar):


- **TYPE** - Classes ou interfaces
- **FIELD** - Propriedades de classes
- **METHOD** - Métodos
- **PARAMETER** - Parâmetros
- **CONSTRUCTOR** - Construtor
- **LOCAL_VARIABLE** - Variáveis locais





Exemplo: @Target

```
@Documented
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.TYPE)
public @interface ExemploAnotacao {
}
```





@Inherited

Por padrão anotações declaradas em uma classe não são herdadas pelas subclasses. Mas, se for necessário que essa herança ocorra, então o tipo anotação que desejamos que seja herdado deve ser anotado com **@Inherited**


É importante destacar que a utilização desta meta-anotação restringe-se apenas a classes. Por exemplo, anotações em interfaces não são herdadas pelas classes que as implementam.






Exemplo: @Inherited

```
@Documented
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.TYPE)
@Inherited
public @interface ExemploAnotacao {
}
```





Regras para criar anotações

- Por convenção, o nome do único membro em um tipo de anotação com um único elemento deve ser **value**.
 - A declaração de um método em um tipo de anotação não pode ter parâmetro e nem uma cláusula throws, que indica um lançamento de exceção.
 - O método não deve possuir corpo – ele é especificado como um método abstrato;
 - O tipo de retorno do método será o tipo de dado do elemento;
 - O tipo de retorno deve ser um dos seguintes: primitivos, String, Class, enum ou um array cujo tipo seja um dos precedentes.
- 



Exemplo

```
@Documented
@Retention(RetentionPolicy.RUNTIME)
@Target(ElementType.TYPE)
@Inherited
public @interface ExemploAnotacao {


    long value();

    String[] nomes();

    RetentionPolicy[] value2();

    RetentionPolicy value3();

    String nomeDefault() default "Rodrigo";
}
```





Exemplo de uso

```
@ExemploAnotacao(value = 1, nomes = {"Rodrigo"},  
    value2 = {RetentionPolicy.RUNTIME},  
    value3 = RetentionPolicy.RUNTIME)  
  
public class UsandoAnotacao {  
}
```



A decorative pattern of hexagons in various shades of blue and cyan. Some hexagons contain white icons: a lightbulb, a thumbs-up, a smartphone, a magnifying glass, and a gear. A network of dots and lines is also visible on the left side.

2


Exceptions / Exceções



O que são?

Exceptions ou Exceção (Erro) é um evento não esperado que ocorre no sistema quando está em tempo de execução (Runtime). Geralmente quando o sistema captura alguma exceção o fluxo do código fica interrompido.


Para conseguir capturar uma exceção, é preciso fazer antes o tratamento. O uso dos tratamentos é importante nos sistemas porque auxilia em falhas como: comunicação, leitura e escrita de arquivos, entrada de dados inválidos, acesso a elementos fora de índice, entre outros.





Exemplo


```
try {  
    //Código aqui  
} catch (Exception e) {  
    //Captura e tratamento do erro aqui  
}
```





Exemplo


```
try {  
    //Código aqui  
} catch(Exception e) {  
    //Captura e tratamento de erro aqui  
} finally {  
    // Código sempre é executado  
}
```





Exemplo


```
try {  
    //Código aqui  
} catch (NullPointerException e) {  
    //Captura e tratamento do erro aqui  
} catch (Exception e) {  
    //Captura e tratamento do erro aqui  
}
```





Exemplo


```
try {  
    //Código aqui  
} catch (NullPointerException | IndexOutOfBoundsException e) {  
    //Captura e tratamento do erro aqui  
} catch (Exception e) {  
    //Captura e tratamento do erro aqui  
}
```





Exemplo


```
private static void comTratamentoException() {  
    String frase = null;  
    String novaFrase = null;  
    try {  
        novaFrase = frase.toUpperCase();  
    } catch (NullPointerException e) {  
        //TRATAMENTO DA exceção  
        System.out.println("0 frase inicial está nula  
        frase = "Frase vazia";  
        novaFrase = frase.toUpperCase();  
    }  
    System.out.println("Frase antiga: "+frase);  
    System.out.println("Frase nova: "+novaFrase);  
}
```





Exemplo

```
private static void comTratamentoExceptionComFinally() {  
    String frase = null;  
    String novaFrase = null;  
    try {  
        novaFrase = frase.toUpperCase();  
    } catch (NullPointerException e) {  
        //TRATAMENTO DA exceção  
        System.out.println("0 frase inicial está nula, para  
        frase = "Frase vazia");  
    } finally {  
        novaFrase = frase.toUpperCase();  
    }  
    System.out.println("Frase antiga: "+frase);  
    System.out.println("Frase nova: "+novaFrase);  
}
```





Categorias

- Checked Exceptions
- Unchecked Exceptions





Categorias

Quando as checked Exceptions devem ser usadas?

Use checked Exceptions quando houver um erro recuperável ou um requisito de negócio importante.






Categorias

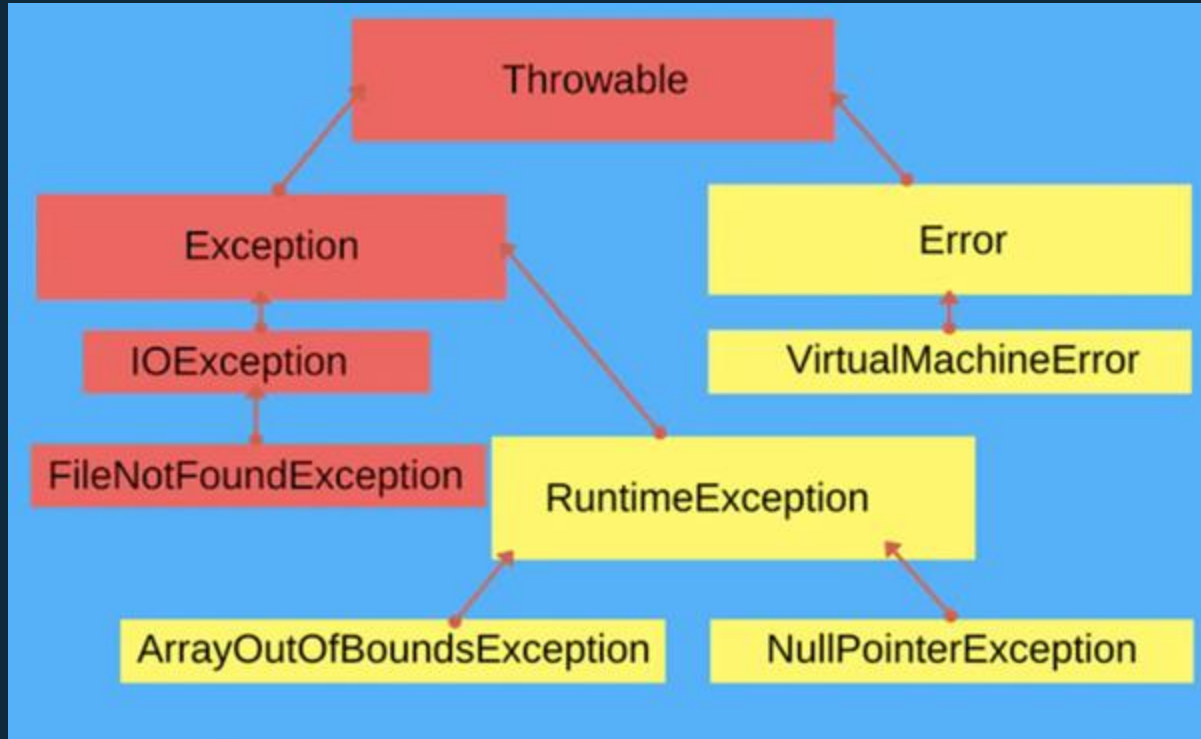
Quando as unchecked Exceptions devem ser usadas?

Use unchecked Exceptions quando não houver recuperação. Por exemplo, quando a memória do servidor é usada em excesso.

RuntimeException é usado para erros quando seu aplicativo não pode recuperar. Por exemplo, NullPointerException e ArrayOutOfBoundsException. Nunca tente tratar exceptions desta categoria.



Hierarquia





Cláusulas throw/throws

As cláusulas throw e throws podem ser entendidas como ações que propagam exceções, ou seja, em alguns momentos existem exceções que não podem ser tratadas no mesmo método que gerou a exceção. Nesses casos, é necessário propagar a exceção para um nível acima na pilha





Cláusulas throw/throws

```
public static void saque(double valor) throws LimiteSaqueException {  
    if(valor > 400) {  
        LimiteSaqueException erro =  
            new LimiteSaqueException("Valor solicitado é maior que seu limite diário.");  
        throw erro;  
    } else {  
        System.out.println("Valor retirado da conta: R$"+valor);  
    }  
}
```


```
private static void exception() {  
    try {  
        ExemploThrow.saque(valor: 500);  
    } catch (LimiteSaqueException e) {  
        System.out.println("ERRO: " + e.getMessage());  
    }  
}
```



Cláusulas throw/throws

```
public static void saqueRuntimeException(double valor) {  
    if(valor > 400) {  
        IllegalArgumentException erro =  
            new IllegalArgumentException("Valor solicitado é maior que seu limite diário.");  
        throw erro;  
    } else {  
        System.out.println("Valor retirado da conta: R$"+valor);  
    }  
}
```

```
private static void runtimeException() {  
    ExemploThrow.saqueRuntimeException(500);  
}
```





Referências

[Exemplos disponíveis no meu github:](#)

<https://github.com/digaomilleniun/backend-java-ebac>

