01

Compartilhando banheiro

Transcrição

Já temos uma boa noção sobre threads. Sabemos como dividir o trabalho entre eles e como inicializá-los. Tudo parece bem, mas claro, como qualquer tecnologia também há perigos. Quais são os perigos dos threads?

Vamos voltar rapidamente ao exemplo de banco de dados que eu mencionei no início do treinamento. Imagine que duas pessoas gostariam de atualizar o mesmo registro ao mesmo tempo. Ou seja, terá dois threads ao mesmo tempo, e para piorar, não temos muito controle sobre quando eles exatamente são executados Nesse momento podem acontecer conflitos na hora da alteração de registro, certo? É preciso ter algum mecanismo para evitar esses conflitos, e no banco de dados temos as transações que ajudam a administrar esses conflitos. No mundo Java, não temos registros e sim objetos, mas o mesmo problema pode acontecer aqui também, dois threads tentando alterar os dados de um objeto ao mesmo tempo. Vamos simular isso dentro de um novo projeto

Um banheiro, vários convidados

O nosso exemplo é bem simples, não é tão real, no entanto muito didático. Imagina que você está fazendo uma festa na sua casa e convidou um monte de amigos. É uma festa bem alegre e com muitas bebidas, mas a sua casa possui apenas um banheiro. O banheiro será o nosso objeto e os convidados serão threads que vão tentar acessar esse banheiro. Vamos tentar implementar isso?

O banheiro será apresentado através de uma nova classe com o mesmo nome. Essa classe terá dois métodos: `fazNumero1()` e `fazNumero2()`.

Então crie o projeto **banheiro** e a classe `Banheiro`, no pacote `br.com.alura.banheiro`:

```
public class Banheiro {

    public void fazNumero1() {

        System.out.println("entrando no banheiro");
        System.out.println("fazendo coisa rapida");

try {
            Thread.sleep(8000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        System.out.println("dando descarga");
        System.out.println("lavando a mao");
        System.out.println("saindo do banheiro");
    }

    public void fazNumero2() {

        System.out.println("entrando no banheiro");
        System.out.println("fazendo coisa demorada");
```

```
        try {
            Thread.sleep(15000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        System.out.println("dando descarga");
        System.out.println("lavando a mao");
        System.out.println("saindo do banheiro");
    }
}
```

A classe não tem mistério, imprime apenas algumas informações no console. Só repare que o método `fazNúmero2()` deixa o thread dormir por 15 segundos, o método `fazNumero1()` por apenas 8 segundos.

Como já falei, na casa tem apenas um banheiro. Então criaremos a classe `Principal` e nela criaremos apenas uma única instância dessa classe, dentro do método `main` :

```
public class Principal {

    public static void main(String[] args) {

        Banheiro banheiro = new Banheiro();

    }
}
```

Implementando os convidados

Agora falta representar os convidados da festa. Vamos simular dois convidados, ou seja dois threads. Cada convidado terá uma tarefa (`Runnable`) associada para ir ao banheiro, fazendo o *número 1* ou *número 2*. Vamos aproveitar e dar um nome ao nosso thread para deixar o nosso código mais legível:

```
public class Principal {

    public static void main(String[] args) {

        Banheiro banheiro = new Banheiro();

        //Passando a tarefa e o nome do Thread
        Thread convidado1 = new Thread(new TarefaNumero1(banheiro), "João");
        Thread convidado2 = new Thread(new TarefaNumero2(banheiro), "Pedro");

        convidado1.start();
        convidado2.start();
    }
}
```

A implementação da tarefa é bem simples. Com o banheiro em mãos basta chamar o método em questão:

```
public class TarefaNumero1 implements Runnable {

    private Banheiro banheiro;

    public TarefaNumero1(Banheiro banheiro) {
        this.banheiro = banheiro;
    }

    @Override
    public void run() {
        this.banheiro.fazNumero1();
    }
}
```

E a classe `TarefaNumero2` :

```
public class TarefaNumero2 implements Runnable {

    private Banheiro banheiro;

    public TarefaNumero2(Banheiro banheiro) {
        this.banheiro = banheiro;
    }

    @Override
    public void run() {
        this.banheiro.fazNumero2();
    }
}
```

Como nomeamos os nossos threads, podemos aproveitar esse nome e imprimir nos métodos do banheiro. Assim nós sabemos qual convidado está acessando o banheiro. Para pegar o nome do thread, basta acessar o thread atual (`Thread.currentThread()`) e chamar o método `getName()` . Por exemplo, no método `fazNumero1()` :

```
public void fazNumero1() {

    String nome = Thread.currentThread().getName();

    System.out.println(nome + " entrando no banheiro");
    System.out.println(nome + " fazendo coisa rapida");

    try {
        Thread.sleep(8000);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }

    System.out.println(nome + " dando descarga");
    System.out.println(nome + " lavando a mao");
    System.out.println(nome + " saindo do banheiro");
}
```

Testando o código

Agora já temos tudo pronto para executar. Vamos rodar a classe principal e logo logo podemos ver que deu problema. Uma possível saída é:

```
João entrando no banheiro
Pedro entrando no banheiro
Pedro fazendo coisa demorada
João fazendo coisa rapida
João dando descarga
João lavando a mao
João saindo do banheiro
Pedro dando descarga
Pedro lavando a mao
Pedro saindo do banheiro
```

Ou seja, dois convidados acessaram o banheiro ao mesmo tempo. Com certeza não é uma boa ideia, né?

Sincronizando o acesso

Como podemos evitar o problema? Como na vida real, é preciso definir que só podemos acessar o banheiro quando temos a chave em mãos. É preciso trancar a porta! A notícia boa é que qualquer objeto possui uma chave e podemos pedir para o convidado/thread pegar essa chave. Sem a chave não será possível acessar e executar esse bloco de código!

No mundo Java, você pega a chave através da palavra chave **synchronized**. Podemos sincronizar o acesso ao objeto, o nosso banheiro! Veja como fica o código do método `fazNumero1()`:

```
public void fazNumero1() {

    String nome = Thread.currentThread().getName();

    System.out.println(nome + " batendo na porta");

    synchronized (this) {

        System.out.println(nome + " entrando no banheiro");
        System.out.println(nome + " fazendo coisa rapida");

        try {
            Thread.sleep(8000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        System.out.println(nome + " dando descarga");
        System.out.println(nome + " lavando a mao");
        System.out.println(nome + " saindo do banheiro");
    }
}
```

E análogo, o `fazNumero2()`:

```
public void fazNumero2() {

    String nome = Thread.currentThread().getName();

    System.out.println(nome + " batendo na porta");

    synchronized (this) {

        System.out.println(nome + " entrando no banheiro");
        System.out.println(nome + " fazendo coisa demorada");

        try {
            Thread.sleep(15000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        System.out.println(nome + " dando descarga");
        System.out.println(nome + " lavando a mao");
        System.out.println(nome + " saindo do banheiro");
    }
}
```

Tudo que está dentro do bloco `syncronized` só poderá ser executado através de um único thread. O `synchronized(this)` significa: "convidado, tente pegar a chave do banheiro"! Sem a chave, o thread fica bloqueado, esperando a liberação.

Ou seja, só um convidado pode entrar no banheiro ao mesmo tempo. Até vários podem bater na porta ao mesmo tempo, mas realmente "fazer o negócio", apenas um.

Vamos testar o código para entender a saída:

```
João batendo na porta
Pedro batendo na porta
João entrando no banheiro
João fazendo coisa rapida
João dando descarga
João lavando a mao
João saindo do banheiro
Pedro entrando no banheiro
Pedro fazendo coisa demorada
Pedro dando descarga
Pedro lavando a mao
Pedro saindo do banheiro
```

Entendendo o synchronized

Lembre-se, não temos controle sobre qual thread realmente começa a execução. Pode ser **João** ou pode ser **Pedro**, mas repare que ambos "bateram na porta". No meu caso, o João foi o primeiro. Ele conseguiu então entrar no banheiro primeiro e a partir desse momento todos os comandos dentro do bloco `synchronized` serão executados de uma vez só. Até pode ser que o Pedro bata na porta nesse mesmo tempo (repare que a impressão do "batendo na porta" está fora do bloco `syncronized`), mas nenhum outro thread pode obter a chave desse objeto para executar o código sincronizado!

Em outras palavras, os comandos dentro do bloco serão executados de maneira atômica. Quando o João sai do banheiro, ou seja, sai do bloco `synchronized`, ele devolve a chave. Nesse momento o Pedro, que ficou bloqueado, pode entrar no banheiro e fazer as coisas que ele precisa fazer.

Aquela chave que usamos para sincronizar o acesso também é chamado de **Mutex**. O *Mutex* é utilizado para executar algum bloco de código de maneira atômica.

Mais convidados

Vamos aumentar um pouco a complexidade e introduzir mais dois convidados:

```
//outros threads
Thread convidado3 = new Thread(new TarefaNumero1(banheiro), "Maria");
Thread convidado4 = new Thread(new TarefaNumero2(banheiro), "Ana");
```

E não podemos esquecer de inicializar os threads :

```
convidado3.start();
convidado4.start();
```

Já podemos executar aplicação, vamos ficar de olho no console. Segue a minha saída no console, mas novamente ela pode variar bastante:

```
Ana batendo na porta
Ana entrando no banheiro  #1
Pedro batendo na porta #2
João batendo na porta   #3
Maria batendo na porta #4
Ana fazendo coisa demorada
Ana dando descarga
Ana lavando a mao
Ana saindo do banheiro
Maria entrando no banheiro
Maria fazendo coisa rapida
Maria dando descarga
Maria lavando a mao
Maria saindo do banheiro
João entrando no banheiro
João fazendo coisa rapida
João dando descarga
João lavando a mao
João saindo do banheiro
Pedro entrando no banheiro
Pedro fazendo coisa demorada
Pedro dando descarga
Pedro lavando a mao
Pedro saindo do banheiro
```

Repare nessa saída, no **#1** a Ana entrou no banheiro. Ela foi a primeira que conseguiu pegar a chave. Nesse momento nenhum outro convidado pode executar algum outro bloco sincronizado do objeto `banheiro`. Mesmo assim, os

convidados podem "bater na porta". Aquele código para "bater na porta" está fora do bloco sincronizado, por isso será executado em paralelo.