01

Esperando limpeza

Transcrição

Nesse capítulo vamos voltar ao nosso exemplo que exemplificou um banheiro. Lembrando que se simulamos uma festa com vários convidados (*threads*), e na casa dessa festa existe apenas um banheiro pra todos os convidados :(

Para não "dar errado no banheiro", sincronizamos o acesso. Apenas um convidado pode entrar nele por vez, no entanto ninguém gostaria de ir usar um banheiro sujo, certo?

Limpando banheiro

Vamos implementar o estado **sujo**. Isso é bem simples, pois basta colocar um atributo dentro da classe `Banheiro`. Vamos utilizar um booleano e definir que o banheiro está sujo no início:

```
public class Banheiro {

    private boolean ehSujo = true;

//outros método omitidos
```

Os nossos convidados são bastante exigentes! Quando eles entram no banheiro e percebem que o banheiro está sujo, eles vão esperar até que alguém o limpe! Mas lembrando, ao entrar no banheiro o convidado pega a chave. Então, para a limpeza entrar, é preciso devolver a sua chave antes de sair. No código, isso é feito através de um método e todas as classes herdam da classe `Object`. Esse método se chama `wait()`.

Quando um thread executa o método `wait()`, ele sabe que tem que devolver a chave e esperar. O thread fica no *estado de espera*. Pense que existe um banco na frente do banheiro, onde os convidados ficam esperando até alguém limpar o banheiro!

Vamos usar o método `wait()` no nosso código. Para simplificar, vamos criar um método auxiliar que se chama `esperaLaFora()`. Dentro da classe `Banheiro`:

```
private void esperaLaFora(String nome) {

    System.out.println(nome + ", eca, banheiro está sujo");
    try {
        this.wait();
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
```

No método `fazNumero1()` vamos testar se o banheiro esteja sujo. E se estiver sujo, vamos mandar o convidado `esperaLaFora`:

```
public void fazNumero1() {

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

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

    synchronized (this) {

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

        if (this.ehSujo) {
            esperaLaFora(nome);
        }

        // restante do código omitido
}
```

Também vamos colocar a mesma condição no método `fazNumero2()`.

Tudo pronto pra testar? Então vamos executar classe `Principal`, com os quatro convidados:

```
Ana batendo na porta
João batendo na porta
Ana entrando no banheiro
Pedro batendo na porta
Maria batendo na porta
Ana, eca, banheiro está sujo
Maria entrando no banheiro
Maria, eca, banheiro está sujo
Pedro entrando no banheiro
Pedro, eca, banheiro está sujo
João entrando no banheiro
João, eca, banheiro está sujo
```

Repare que os convidados bateram na porta, entraram no banheiro e perceberam que o banheiro está sujo. Todos os convidados estão esperando agora e a JVM nunca termina pois ainda há threads esperando!

Visualizando os estado dos threads

Você se lembra da ferramenta `jconsole`? Usamos-a para mostrar os threads da JVM. Com a JVM ainda rodando, vamos subir o `jconsole`.

O interessante é que podemos ver estado de cada thread. Por exemplo, o thread *João* está no estado `WAITING`, igual ao *Pedro*, *Maria* e *Ana*. Que bagunça!

É preciso acordar os nossos convidados/threads. Vamos implementar isso agora!