03. Sincronizando a execução

Date	@24/08/2022
Categoria	Java
Curso	Curso Threads em Java 1: programação paralela

Tópicos

1. Compartilhando banheiro
2. Thread atual
3. Modificador para threads
4. Operação atôminca
5. Resumo da aula
6. Mãos à obra: implementando o banheiro
7. Mãos à obra: Fechando banheiro
8. para saber mais: Obtendos locks de forma explícita
9. Oque apredenmos?

Para saber mais: Obtendos locks de forma explícita

Nesse capítulo vimos o poder do bloco `synchronized` em como ele nos ajuda a trabalhar com tarefas atômicas. Como alternativa, podemos também conseguir esse bloqueio de forma *explícita* (programaticamente) através de uma classe chamada `ReentrantLock`.

E como podemos utilizar essa classe em nosso banheiro?

```
public class Banheiro {

    private Lock lock = new ReentrantLock();

    public void fazNumero1() {

        lock.lock();
            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");
        lock.unlock();
    }

    public void fazNumero2() {

        lock.lock();
            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");
        lock.unlock();
    }
}COPIAR CÓDIGO
```

Pesquise um pouco mais sobre essa classe. Você consegue enxergar alguma diferença de utilizar o bloco `syncronized`?

Certamente há algumas diferenças que podemos examinar. Uma das principais é a possibilidade de se criar um lock com *timeout* usando uma sobrecarga do método `tryLock`.

Por exemplo:

```
boolean locked = lock.tryLock(5, TimeUnit.SECONDS); //5s
```

Com esse método esperamos até cinco segundos e receberemos `true` caso o *lock* for obtido. Caso contrário, receberemos `false`. Como desvantagem, há o fato de o programador ter a responsabilidade de liberar o *lock* (`unlock()`).

O que aprendemos?

- Um Thread pode ter um nome;
- Podemos pegar o `Thread` atual através do método `Thread.currentThread()`;
- Para sincronizar threads, devemos utilizar um bloco `synchronized`;
- O `synchronized` significa que bloqueamos o objeto para outros threads;
- A sincronização é feita através de *mutex*, que nada mais é do que a chave do objeto.

No próximo capítulo vamos ver mais um exemplo de `synchronized`.