05. Espere e notifique

Date	@26/08/2022
Categoria	Java
Curso	Curso Threads em Java 1: programação paralela

Tópicos

1. Esperando limpeza
2. Estado da thread
3. Mãos à obra: Banheiro sujo
4. Limpando banheir
5. Deixa a thread esperar
6. De quem pertence?
7. Esperando fora do synchronized
8. Mãos à obra: Limpando banheiro
9. Voltando a limpar
10. Mãos à obra: Mais limpeza
11. Daemon
12. Para saber mais: Prioridades
13. O que aprendemos?

Daemon

O que é uma Daemon Thread?

São provedores de serviços para outras threads.

Threads *daemon* são como prestadores de serviços para outras threads. Elas são usadas para dar apoio à tarefas e só são necessárias rodar quando as threads "normais" ainda estão sendo executadas. Uma thread *daemon* não impede a JVM de terminar desde que não existem mais threads principais em execução. Um exemplo de uma thread *daemon* é o coletor de lixo da JVM (*Garbage Collector*) ou a nossa limpeza do banheiro :)

Para definir uma thread como *daemon* basta usar o método `setDaemon(boolean)` antes de inicializar:

```
Thread thread = new Thread(runnable);
thread.setDaemon(true);
thread.start();
```

Para saber mais: Prioridadas

Vimos que podemos coordenar a execução de threads, mas quando uma thread realmente executa continuamos não sabendo. Também não sabemos, quando tem mais de uma thread esperando, qual thread realmente continua executando em caso de notificação. Tudo isso é fora do poder do nosso programa e depende do escalonador de threads.

Aí pode vir uma dúvida: Falamos rapidamente que a coleta de lixo (*Garbage Collection*) é executada em uma ou mais threads dentro da JVM. Quando há pouca memória disponível, será que não faz sentido dar o máximo de tempo para essa thread? No final, o trabalho dele possibilita o bom funcionamento da JVM!

A mesma pergunta podemos fazer para a nossa thread de limpeza: Será que não faz sentido dar preferência à limpeza quando tem muitos convidados, sabendo que cada convidado quer um banheiro limpo?

Faz todo sentido e podemos dar uma dica para o escalonador, definindo que há threads com uma prioridade maior do que outras. Isso é apenas uma dica e não temos

garantias, mas muito provável que o escalonador respeite a dica.

Na classe `Thread` existe um método `setPriority` com justamente esse propósito. A prioridade é um valor inteiro entre 1 e 10, sendo 10 a prioridade mais alta. Basta usar o método:

```
Thread limpeza = new Thread(new TarefaLimpeza(banheiro), "Limpeza");
limpeza.setPriority(10);
limpeza.start();COPIAR CÓDIGO
```

Em vez de usar um *integer* diretamente podemos aproveitar alguns constantes criados na classe `Thread`, por exemplo:

```
limpeza.setPriority(Thread.MAX_PRIORITY);COPIAR CÓDIGO
```

ou

```
limpeza.setPriority(Thread.MIN_PRIORITY);COPIAR CÓDIGO
```

E já vem mais uma pergunta: Qual é a prioridade padrão de uma thread?

Até agora não usamos nada de prioridade no nosso código. Qual é a prioridade padrão de uma thread?

Na classe `Thread` temos um atributo público e estático que define a prioridade padrão (`NORM_PRIORITY`):

```
/**
     * The default priority that is assigned to a thread.
     */
    public final static int NORM_PRIORITY = 5;COPIAR CÓDIGO
```

Isso significa que se não usamos nenhuma prioridade explícita, a thread vai assumir o valor **5** para a prioridade.

O que aprendemos?

- ao chamar `object.wait()` a thread fica no estado de espera
- estado de espera significa `WAITING` no mundo de threads
- uma thread esperando pode ser notificada pelo método `object.notifyAll()`
- existem daemon threads para executar tarefas ou serviços secundários
- daemons serão automaticamente desligados quando todas as outras threads terminarem