13

Para saber mais: Prioridades

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();
```

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

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

ou

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

E já vem mais uma pergunta: Qual é a prioridade padrão de uma thread? Tente descobrir (ou clique em continuar) :)