Matéria:

* Semáforos em Java

# – Semáforos

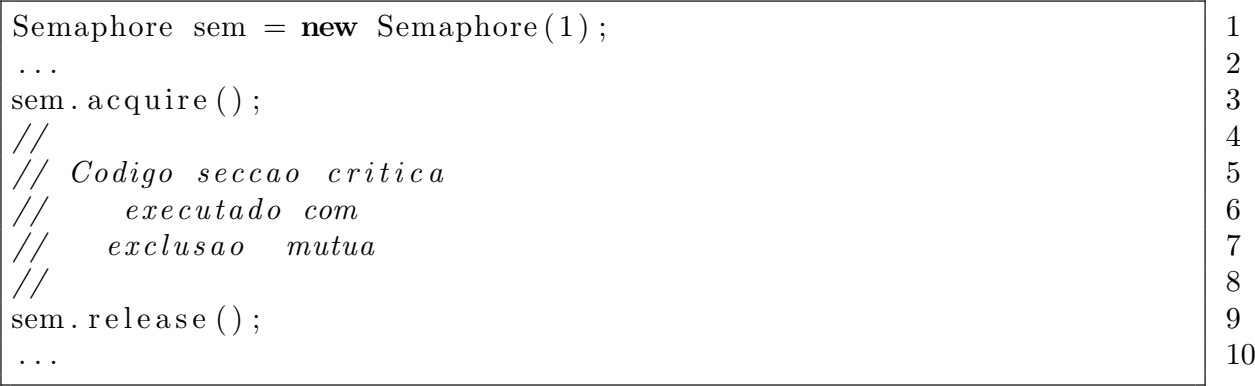
Um semáforo é um mecanismo disponibilizado pela JVM que permite sincronizar a execução de múltiplas *threads*, ou seja, permite resolver problemas decorrentes da partilha de recursos ou de situações de competição. Semáforos em Java são disponibilizados pela classe Semaphore [1](#_bookmark0) que dispõe de dois métodos principais:

* *acquire()*[*2*](#_bookmark1)– Reduz o número de autorizações. Esta operação é bloqueante quando o semáforo não dispõe de autorizações;
* *release()*[*3*](#_bookmark2)– Incrementa o número de autorizações.

O seu funcionamento baseia-se no conceito de quantidade de autorizações. O semáforo deve ser inicializado com um determinado número de autorizações. Cada chamada a *aquire()* reduz este número, cada chamada a *release()* aumenta. Assumindo que um semáforo foi inicializado com N autorizações, tal implica que apenas N *threads* podem chamar *aquire()*, no máximo, antes de ser chamado o *release()*.

# – Proteção de seções críticas

Os semáforos podem ser utilizados para proteger seções críticas de código, bastando para tal inicializar o semáforo com uma autorização. De seguida basta que se obtenha uma autorização do semáforo antes de se executar o código da seção crítica. Após terminar-se o código da seção critica, deve-se libertar a autorização para que outra *thread*, que esteja bloqueada no mesmo semáforo, possa continuar. A Listagem 1 exemplifica este comportamento.



Listagem 1: Utilização de semáforos em seções críticas

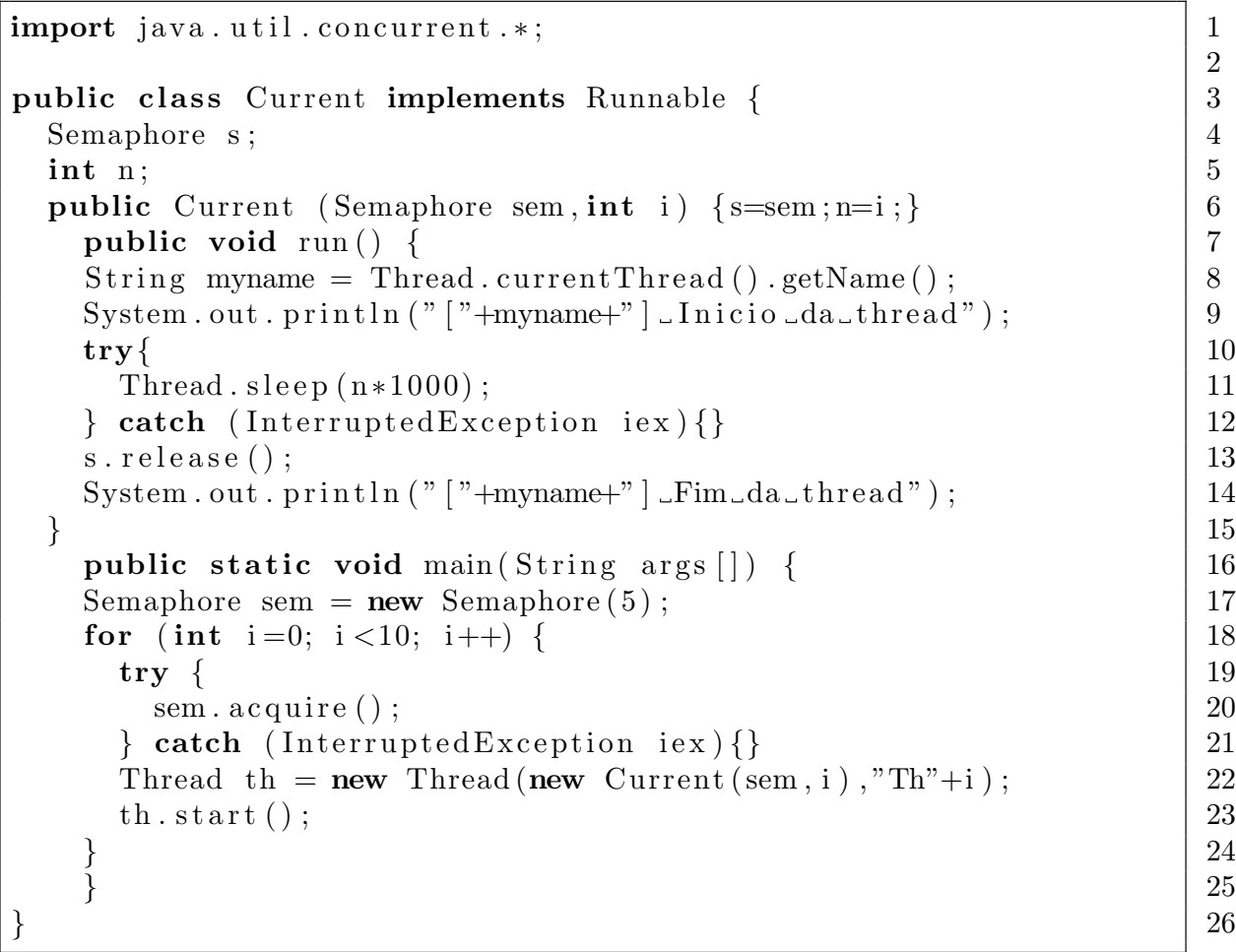
1 Mais informação: <https://docs.oracle.com/javase/7/docs/api/java/util/concurrent/Semaphore.html>

2 Também referenciado como *wait()* ou *down()*

3 Também referenciado como *signal()* ou *up()*

# – Outras situações de sincronização

Além de seções críticas, os semáforos, ao serem usados com um número de autorizações superior a um, podem permitir estratégias de sincronização avançadas. Pode-se inclusive utilizar vários semáforos, com diferentes valores iniciais de autorizações.



Listagem 2: Outras situações de sincronização com semáforos

O excerto de código apresentado na Listagem 2 demonstra como se podem utilizar semáforos para limitar o número de *threads* concorrentes em cada instante de tempo. Neste caso serão criadas 10 *threads* no total, sendo que em cada instante de tempo apenas poderão estar a executar cinco *threads* no máximo.

# – Justiça

A implementação de semáforos do Java não dá garantias de justiça na sua utilização, exceto se tal for explicitamente solicitado. Ou seja, não há garantias de que a primeira *thread* a invocar o método *acquire()* seja a primeira *thread* a obter a permissão.

Listagem 3: Instanciação de um semáforo justo

A justiça pode ser forçada com recurso a um segundo construtor desenvolvido particularmente para este efeito. Contudo, a utilização de semáforos com garantia de justiça implica um custo de desempenho. A sua utilização não é recomendada, a não ser quando realmente necessária. A Listagem 3 demonstra como se pode instanciar um semáforo justo.

# 2– Exercícios

* + 1. Elabore um programa que lance a execução de cinco *thread*. Cada *thread* deve escrever 200 números num ficheiro de texto. Garanta a exclusão mútua nas operações de escrita. O nome do ficheiro deverá ser passado como argumento da linha de comandos;

Uma imagem com texto, captura de ecrã, Tipo de letra, número

Descrição gerada automaticamente

Uma imagem com texto, captura de ecrã, Tipo de letra, número

Descrição gerada automaticamente

* + 1. Elabore um programa que lance a execução simultânea de 20 *thread*. Cada *thread* deve tentar escrever uma linha num ficheiro, aguardar um segundo e que tentar escrever uma segunda linha no mesmo ficheiro. Garanta a exclusão mútua nas operações de escrita. O ficheiro só deverá conter no m´máximo 10 linhas. Utilize dois semáforos. O nome do ficheiro deverá ser passado como argumento da linha de comandos;

Uma imagem com texto, captura de ecrã, Tipo de letra, software

Descrição gerada automaticamente

Uma imagem com texto, captura de ecrã, Tipo de letra, número

Descrição gerada automaticamente

* + 1. Elabore um programa que lance a execução de duas *thread* em simultâneo.

As *thread* devem ser implementadas em classes distintas. Uma *thread* deve escrever "Init" quando iniciar, esperar um número aleatório de segundos (entre 1 e 9) e escrever "End". A outra *thread* deve iniciar após a *thread* inicial escrever "End", escrevendo um número aleatório de linhas (entre 1 e 9);

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente

Uma imagem com texto, captura de ecrã, Tipo de letra, número

Descrição gerada automaticamente

* + 1. Elabore um programa que lance 20 *thread* em simultâneo e que, infinitamente, solicite um número ao utilizador. O número será utilizado para identificar a *thread* a ativar até que seja inserido um novo número. As *thread* devem iniciar suspensas. Cada *thread* deve escrever uma mensagem por segundo para o ecrã sempre que estiver ativa.

Uma imagem com texto, captura de ecrã, número, Tipo de letra

Descrição gerada automaticamente

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamente