Semáforos em JAVA

- Implementado pela classe *Semaphore*

- Composto basicamente de:
  - variável inteira
  - lista de espera

Classe *Semaphore*
Método Construtor

- ***`Semaphore(int permits)`***
- ***`Semaphore(int permits, boolean fair)`***
- Cria um semáforo com o número de permissões fornecido como parâmetro,
- Se o parâmetro "fair" for:
  - *TRUE* -> a lista de espera é uma FIFO,
  - *FALSE* (ou não setado) -> não é possível garantir a ordem da lista de espera.

Classe *Semaphore*
Método *acquire()*

- `acquire() // Adquire uma permissão`
- `acquire(int permits)`
- Se existir ao menos uma (ou *n*) permissão disponível:
  - é liberada uma (ou *n*) permissão para a *thread*,
  - e é reduzido o número de permissões disponíveis no semáforo.
  - Senão: a *thread* fica fora do escalonamento (em estado inativo) até que:
    - alguma outra *thread* realize uma liberação e esta *thread* seja a próxima a conseguir uma (ou *n*) permissão,
    - alguma outra *thread* interrompa a *thread* atual.

Classe *Semaphore*
Método *acquireUninterruptibly()*

- `acquireUninterruptibly() // Semelhante ao acquire()`
- `acquireUninterruptibly(int permits)`

- Se a *thread* atual for interrompida enquanto espera por uma permissão:
  - continua na espera,
  - quando a *thread* retornar deste método ela será interrompida.

Classe *Semaphore*
Método *tryAcquire()*

- `tryAcquire() // Semelhante ao acquire()`
- `tryAcquire(int permits)`

- Se uma (ou *n*) permissão estiver disponível no momento:
  - adquire uma (ou *n*) permissão do semáforo e retorna *TRUE*,
  - senão retorna *FALSE*.
    - Não bloqueia

Classe *Semaphore*
Método *release()*

- `release() // Libera uma permissão`
- `release(int permits)`
- Libera uma (ou *n*) permissão, a qual retorna para o semáforo,
- Número de permissões do semáforo é incrementado,
- Se algumas *threads* estavam na lista de espera, uma delas é escolhida para receber a permissão, a *thread* que receber a permissão retorna para o escalonamento.
    - OBS: Não existe nenhuma obrigação de que a *thread* que realiza a liberação deve ter adquirido a permissão.

Exemplo

```
import java.util.concurrent.Semaphore;

public class Worker implements Runnable {

   private Semaphore sem;
   private String name;

   public Worker(Semaphore sem, String name) {
          this.sem = sem;
          this.name = name;
   }

   public void run() {
          while(true) {
               try {
                    sem.acquire();
                    secaoCritica(name);
                    sem.release();
                    } catch (InterruptedException e)
   { System.out.println("Excecao"); }
          }
   }
}
.... main
Semaphore sem = new Semaphore(1, true);
New thread(new worker(string, sem)).start;
```