Programação Concorrente em Java

Programação Concorrente em Java

Threads em Java

Ciclo de Vida de uma *Thread*

Escalonamento de *Threads*

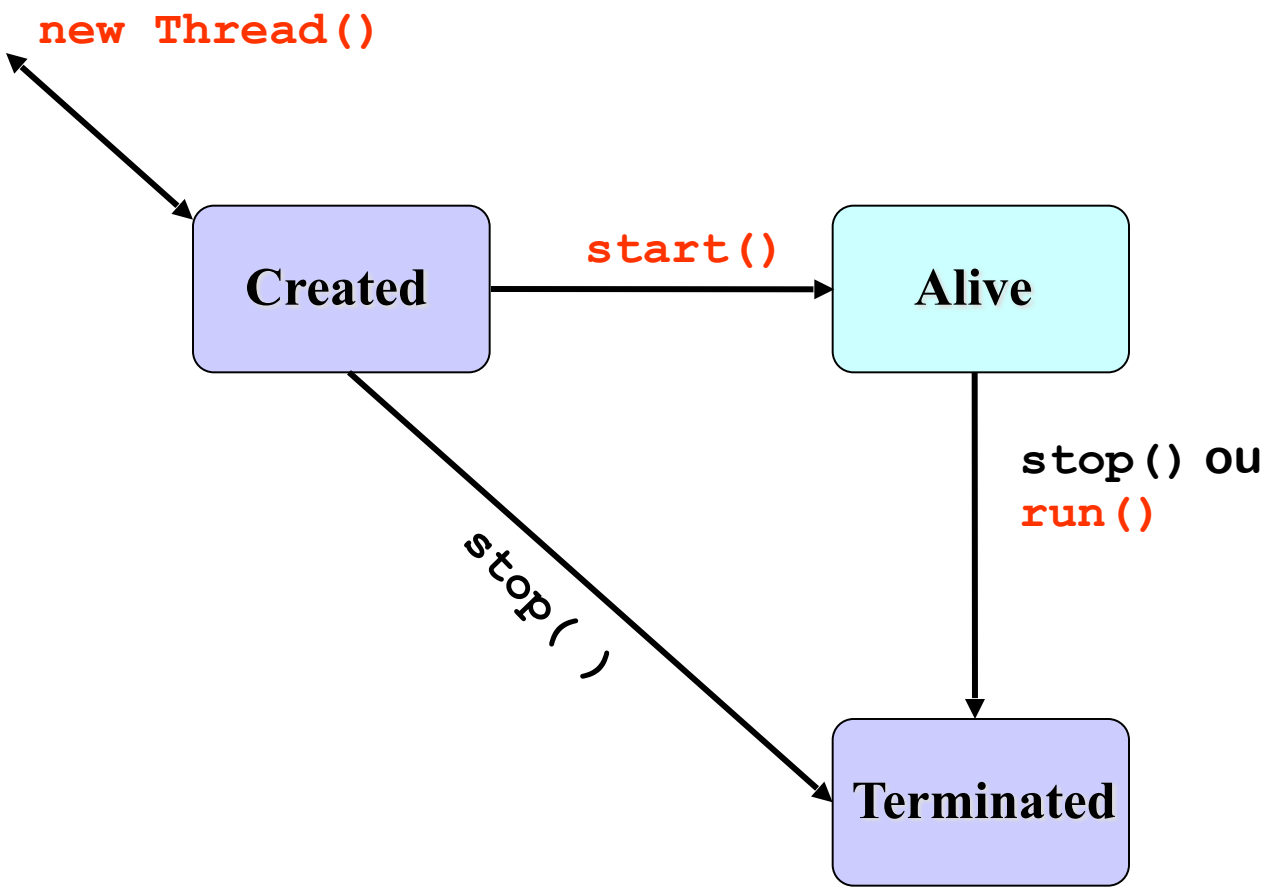
Concorrência de *Threads*

Exemplo *Request-Release*

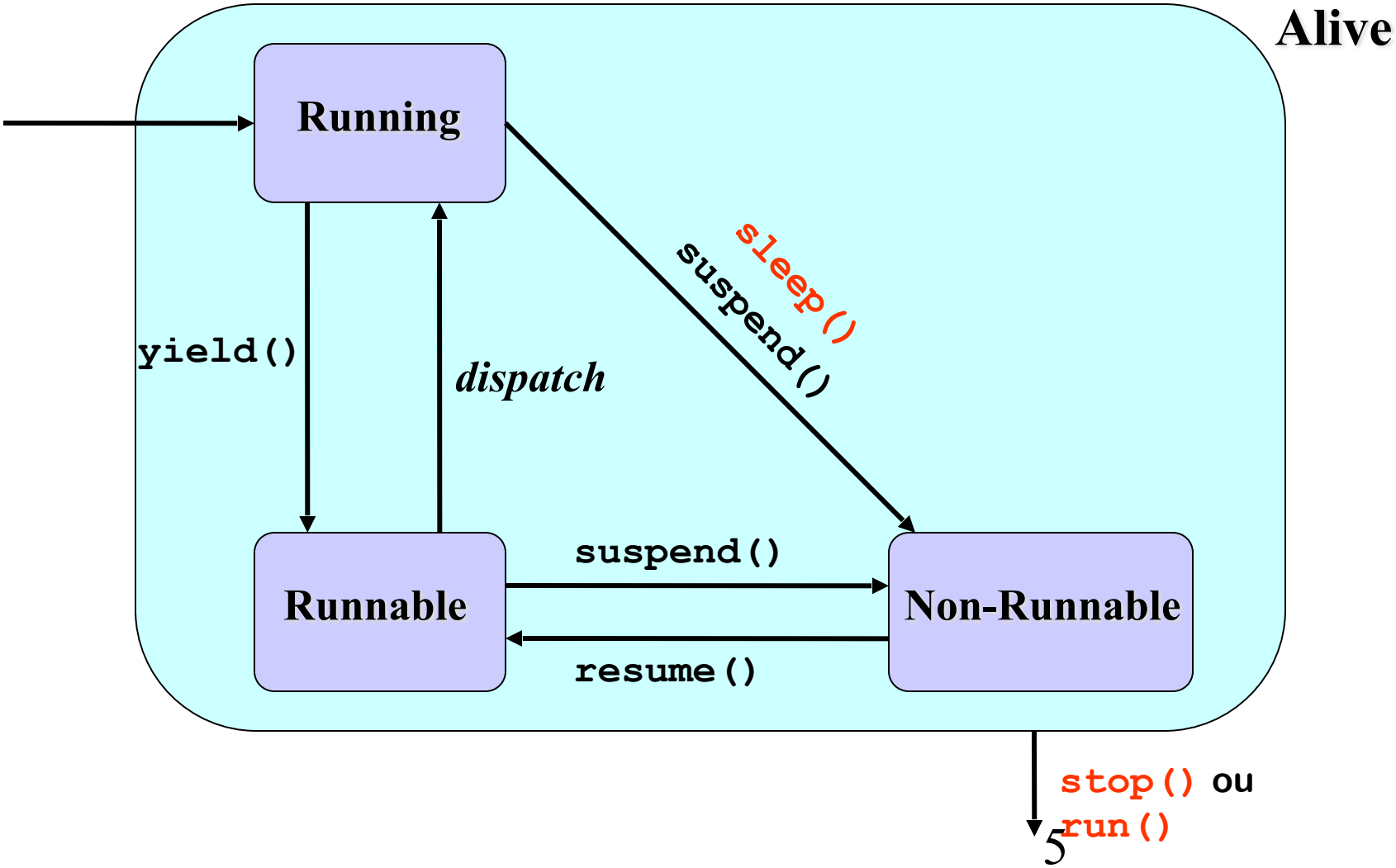
Threads em Java

- **Podem ser criadas, iniciadas e interrompidas**
- **Permitem o tratamento de exclusão mútua (*synchronized*)**
- **Permitem o tratamento de coordenação (*wait* e *notify*)**
- **Têm um ciclo de vida com estados**
- **São escalonadas através de prioridade**

Ciclo de Vida de uma *Thread*



Ciclo de Vida de uma *Thread*



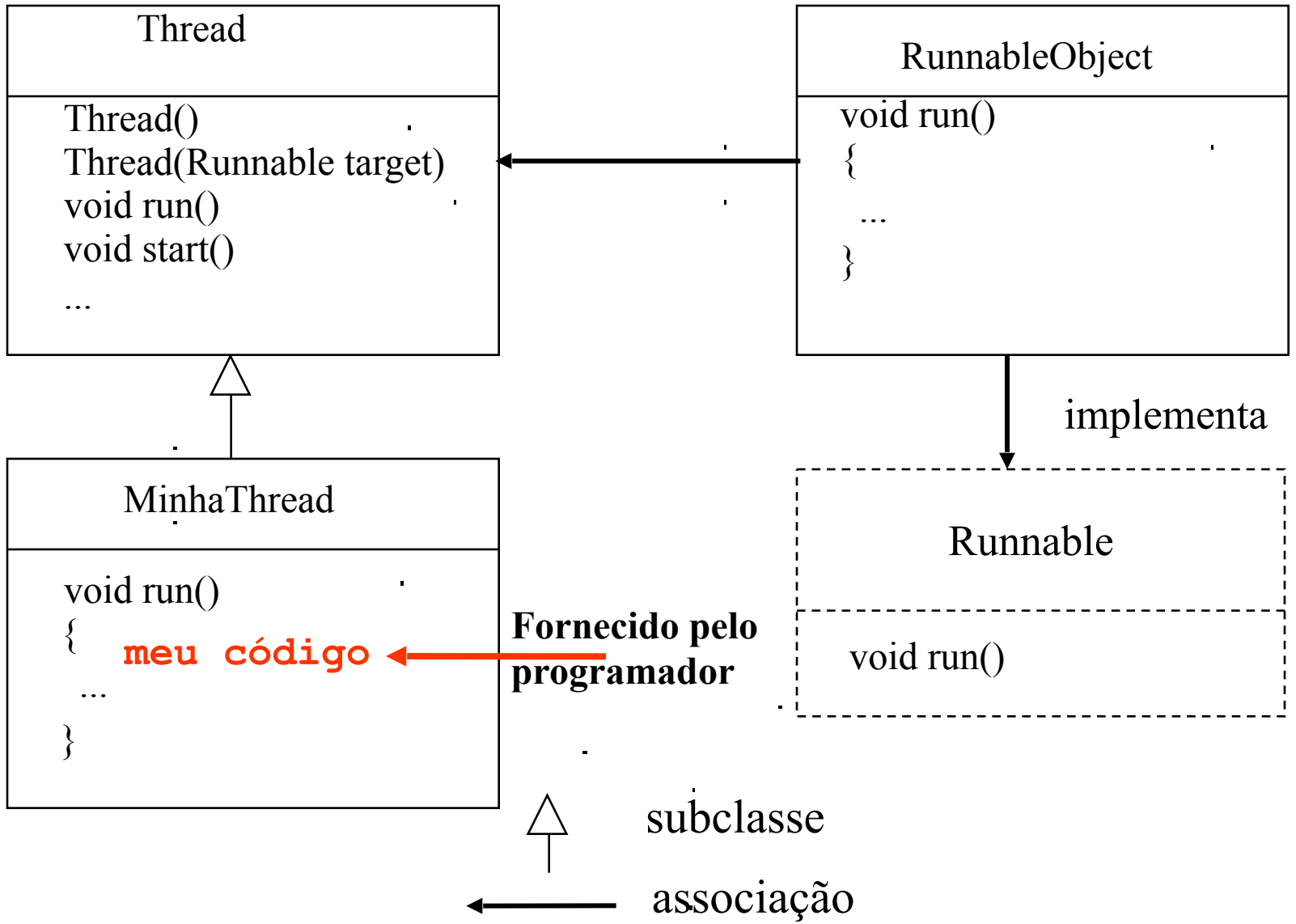
Escalonamento de *Threads*

- **algoritmo de escalonamento de prioridade fixa**

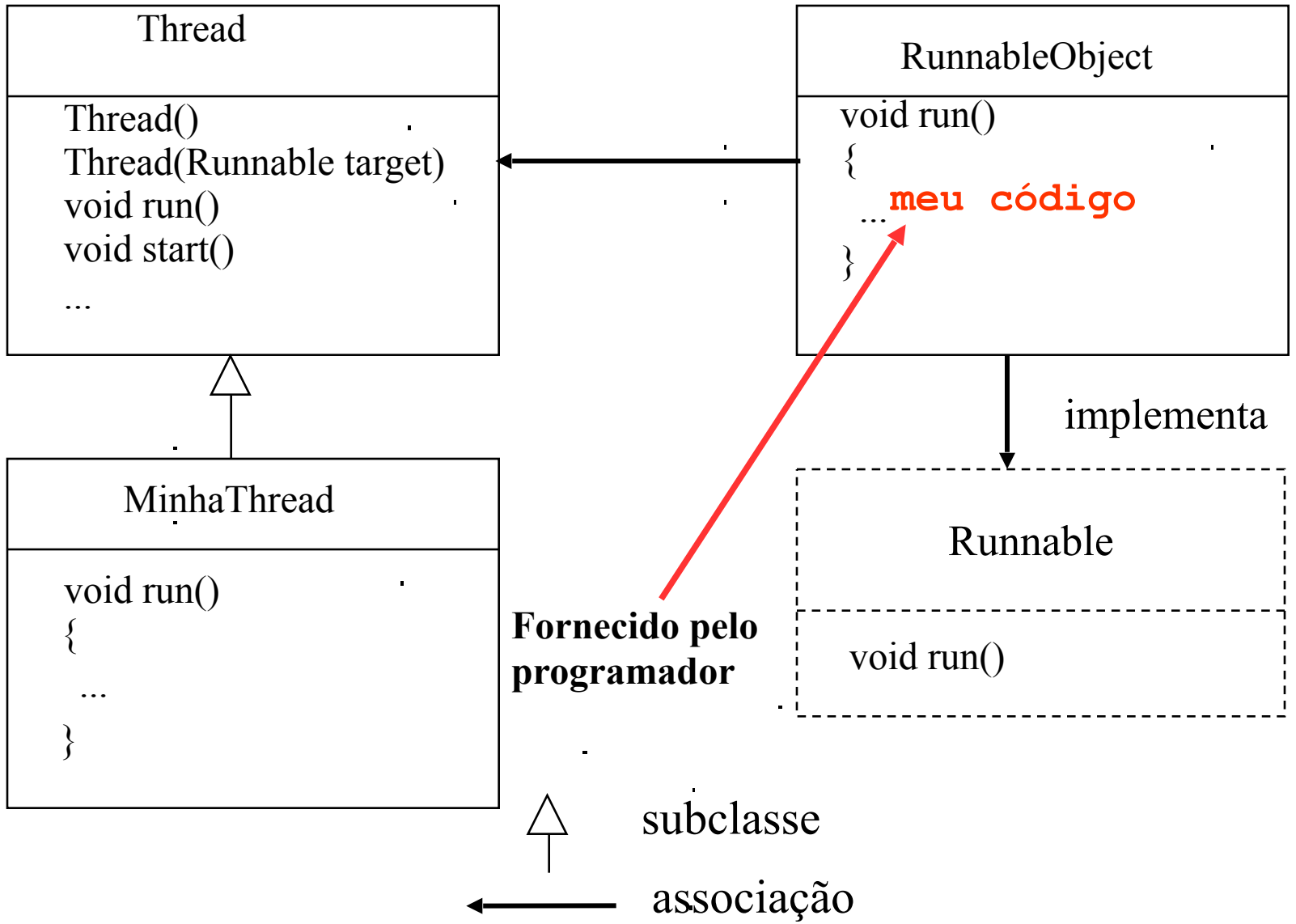
- **condições para suspender a execução de uma *thread*:**
  - **uma *thread* de prioridade mais alta torna-se *Runnable***
  - **a execução de *yield( )* ou o término natural da *thread***
  - **término da fatia de tempo, para sistemas com suporte a *time-slicing***

- **escalonador pode suspender *threads* com prioridade mais baixa para evitar *esfomeação***

modelo


Thread
Thread()
Thread(Runnable target)
void run()
void start()
...
RunnableObject
void run()
{
...
}
implementa
Runnable
void run()
MinhaThread
void run()
{
meu código
...
}
Fornecido pelo programador
subclasse
associação

modelo


Thread
Thread()
Thread(Runnable target)
void run()
void start()
...
RunnableObject
void run()
{
...
}
meu código
implementa
MinhaThread
void run()
{
...
}
Runnable
void run()
Fornecido pelo programador
subclasse
associação

Classe *Thread*

```
public class Thread {
   ...
    sleep (long millis)
    yield()
    resume()
    run(){
         ...
         meu código
         ...
    }
    start()
    stop()
    suspend()
}
```

Fornecido pelo programador

Exemplo: Classe ThreadSimples

```
public class ThreadSimples extends Thread {
    String nomeThread;

    public ThreadSimples(String str) {
        nomeThread = str;
    }

    public void run() {
        for (int i = 0; i < 10; i++) {
            System.out.println(i + " " + nomeThread);
            ...
            }
        }
        System.out.println("Ok! " + nomeThread);
    }
}
```

Exemplo: Criação de *Threads*

```
public class ThreadExemplo {
    public static void main (String[] args) {

      ThreadSimples t01 = new ThreadSimples("Thread01");
      ThreadSimples t02 = new ThreadSimples("Thread02");

      t01.start();
      t02.start();
    }
}
```

Tratamento de Concorrência

Exclusão Mútua — ***synchronized***

x

Coordenação — ***wait( ), notify( ), notifyAll( )***

exclusão mútua

- cada objeto tem um *lock* de exclusão mútua associado
  - não pode ser acessado diretamente pela aplicação
- métodos podem ser definidos com o modificador *synchronized*
  - o acesso ao método só ocorre quando o sistema obtiver o *lock* do objeto
- métodos *synchronized* têm acesso com exclusão mútua aos dados encapsulados no objeto
- métodos sem o modificador *synchronized* não exigem *lock* do objeto

exemplo: compartilhando um número

```
class SharedNumber {

  private int number;

  public synchronized int get() {
    return number;
  }

  public synchronized void set(int number) {
    this.number = number;
  }

  public synchronized void incrementBy(int inc) {
    number += inc;
  }
}
```

exemplo: compartilhando um número

```
class SharedNumber {

  private int number;

  public synchronized int get() {
    return number;
  }

  public synchronized void set(int number) {
    this.number = number;
  }

  public synchronized void incrementBy(int inc) {
    number += inc;
  }
}
```

Todos os métodos que acessam os dados encapsulados devem ter o *synchronized*

coordenação

- a coordenação é feita com métodos da classe Object:
  - `wait()`, `notify()`, `notifyAll()`
- esses métodos só podem ser usados a partir de métodos que têm o *lock* do objeto
  - se forem invocados sem o *lock* uma exceção é disparada

coordenação

- o método `wait()` bloqueia a *thread* e libera o *lock* associado ao objeto

- o método `notify()` acorda uma *thread* em estado de *waiting*
  - Java não define qual *thread* será selecionada

- `notify()` não libera o *lock*
  - A *thread* acordada deve esperar até que obtenha o *lock*

- `notifyAll()` acorda todas as *threads* que estão em estado de *waiting*

Recurso Compartilhado (Tela)

```
public class Tela{
   // Recurso disputado
   String texto;
   public void setTexto(String s){
        texto = s;
   }
   public void mostraTexto( ){
        System.out.println(texto);
   }
}
```

Definição da Classe UserSemControle (*Thread*)

```
public class UserSemControle extends Thread{
    private Tela recurso; // Recurso disputado, sem proteção de acesso
    private String nomeThread; //  Identificacao da thread

    public UserSemControle(String str,Tela r) {
        recurso = r;
        nomeThread = str;
    }

    public void run( ) {
       for (int i=0; i<5; i++) {
          recurso.setTexto(nomeThread); // Seta recurso compartilhado
          try{
            sleep(30);
          }catch(Exception e){ }
          recurso.mostraTexto( );        // Usa recurso compartilhado
       }
    }
}
```

Criação e Execução das *Threads* - 1

```
public class RecursoDesprotegido {
  public static void main(String[] args) {

   Tela recurso = new Tela(); // Criação do recurso a ser compartilhado

//** Criando as threads

    UserSemControle usSem01 = new UserSemControle("Usuario 01",recurso);
    UserSemControle usSem02 = new UserSemControle("Usuario 02",recurso);
    UserSemControle usSem03 = new UserSemControle("Usuario 03",recurso);
    UserSemControle usSem04 = new UserSemControle("Usuario 04",recurso);


//** Executando as threads

    usSem04.start( );
    usSem01.start( );
    usSem03.start( );
    usSem02.start( );
  }
}
```

Monitor ControlaAcesso

```
 public class ControlaAcesso {
 private boolean ocupado = false;//controla se request foi feito
 private Tela recurso;            // recurso do monitor

//** Construtor
public ControlaAcesso(Tela r){
   recurso = r;
}
//** Método para liberar o recurso
public synchronized void release() {
   ocupado = false;
   notifyAll();
}

//** Método para requisitar o recurso
public synchronized void request(){
  while (ocupado) {
   try {
     wait();
   } catch (InterruptedException e) { }
  }
  ocupado = true;
}
```

Monitor ControlaAcesso

```
    public void setRecurso(String s) {
        recurso.setTexto(s);}
    }

    public void usaRecurso() {
        recurso.mostraTexto();
    }
}
```

Definição da Classe Usuário (*Thread*)

```
public class Usuario extends Thread {
    private ControlaAcesso monitor;       // Monitor
    private String nomeThread;            // Identificação da thread

    public Usuario(String str, ControlaAcesso m) {
        monitor = m;
        nomeThread = str;
    }

    public void run() {
       for (int i=0; i<5; i++) {
          monitor.request( ); // Solicita o monitor para usar o recurso
          monitor.setRecurso(nomeThread);
          try{
    sleep(30);
          }catch(Exception e){ }
          monitor.usaRecurso( );
          monitor.release( ); // Libera o monitor para o uso do recurso
       }
    }
}
```

Criação e Execução das *Threads* - 2

```
public class RequestRelease {
    public static void main(String[] args) {

        // Criação do recurso a ser compartilhado
        Tela recurso = new Tela( );

       // Criação do monitor
        ControlaAcesso monitor = new ControlaAcesso(recurso);

//** Criando as threads
         Usuario us01 = new Usuario("Usuario 01",monitor);
         Usuario us02 = new Usuario("Usuario 02",monitor);
         Usuario us03 = new Usuario("Usuario 03",monitor);
         Usuario us04 = new Usuario("Usuario 04",monitor);

//** Executando as threads
         us02.start( );
         us01.start( );
         us04.start( );
         us03.start( );
}}
```

Comparando os resultados

Executando threads que *usam* o monitor para acessar o recurso compartilhado (Tela):

```
Usuario 02    Usuario 03
Usuario 02    Usuario 03
Usuario 02    Usuario 03
Usuario 02    Usuario 03
Usuario 02    Usuario 03
Usuario 04    Usuario 01
Usuario 04    Usuario 01
Usuario 04    Usuario 01
Usuario 04    Usuario 01
Usuario 04    Usuario 01
```

Comparando os resultados

Executando threads que não usam o monitor para acessar o recurso compartilhado (Tela):

Usuario 02	Usuario 03
Usuario 04	Usuario 03
Usuario 03	Usuario 01
Usuario 01	Usuario 03
Usuario 02	Usuario 04
Usuario 04	Usuario 01
Usuario 03	Usuario 01
Usuario 01	Usuario 01
Usuario 02	Usuario 01
Usuario 03	Usuario 02

Programação Concorrente em Java

Sergio T. Carvalho
sergio@inf.ufg.br