Programação Concorrente em Java

Prof. Orlando Loques - IC/UFF

versão 17-04-2001

Referências:

- Programação Concorrente em Java, Sérgio Teixeira de Carvalho & Leonardo N. M. Reis – IC-UFF
- Ousterhout, J.: Why Threads Are A Bad Idea, USENIX Technical Conference,1996, http://www.softpanorama.org/People/Ousterhout/Threads/index.htm

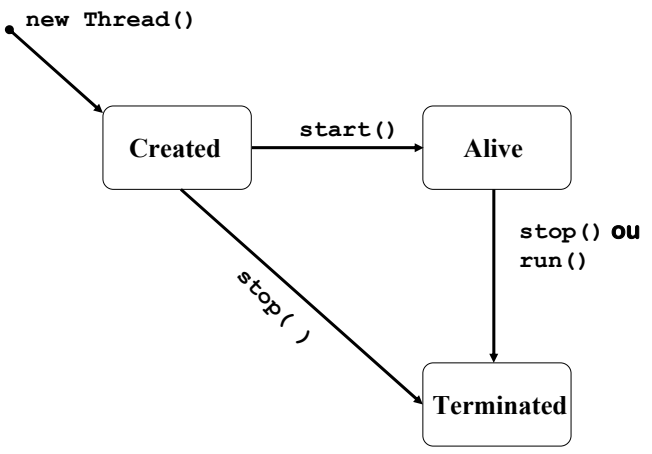
Programação Concorrente em Java

2. Threads em Java
3. Ciclo de Vida de uma Thread
4. Escalonamento de Threads
5. Concorrência de Threads
6. Exemplo Request-Release

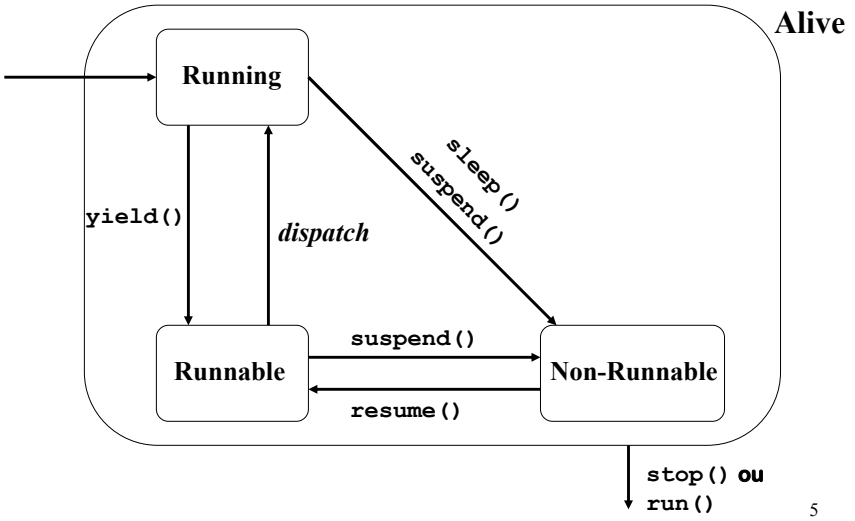
Threads em Java

- **Permitem o tratamento de concorrência (*synchronized*)**
- **Permitem o tratamento de sincronização (*wait* e *notify*)**
- **Podem ser iniciadas e interrompidas**
- **Têm um ciclo de vida com estados**
- **São escalonadas através de prioridade**

Ciclo de Vida de uma Thread



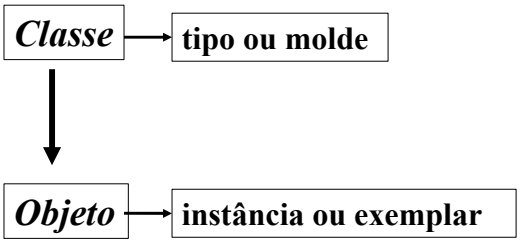
Ciclo de Vida de uma Thread



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***

Classe e Objeto



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( )*

Recurso Compartilhado (Tela)

```
public class Tela{
  // Recurso disputado
  String texto;
  public void
setTexto(String s){
      texto = s;
  }
  public void usa( ){

System.out.println(texto);
  }
}
```

Definição da Classe UserSemControle (Thread)

```
public class UserSemControle extends Thread{
    private Tela recurso;                  //  Recurso disputad
    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 compar
          try{
            sleep(30);
          }catch(Exception e){ }
          recurso.usa( );                              // Usa r
       }
    }
}
```


Criação e Execução das Threads 1

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

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

//** Criando as threads

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


//** Executando as threads

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

Monitor ControlaAcesso

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

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


//** Método para requisitar o recurso
public synchronized void request( ){                 public void
setRecurso(String s){
  while (ocupado) {
recurso.setTexto(s);
      try {                                                }
          wait( );
```


Definição da Classe Usuário (*Thread*)

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

    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 pa
          monitor.setRecurso(nomeThread);
          try{
        sleep(30);
          }catch(Exception e){ }
          monitor.usaRecurso( );
          monitor.release( );              // Libera o monitor
       }
    }
}
```

Criação e Execução das Threads 2

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

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

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
```

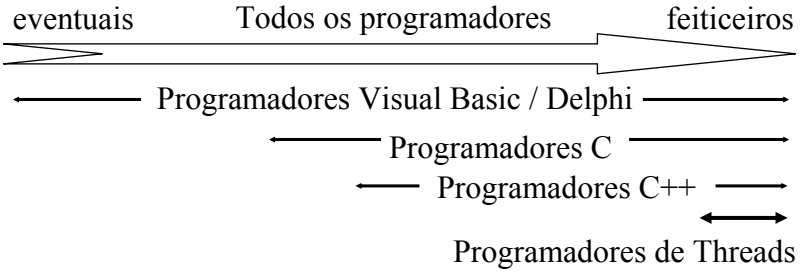
Executando threads que não usam o monitor para acessar o recurso compartilhado (Tel

```
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
```

Implementação de semáforos em Java

```
// The Semaphore Class

// up() is the V operation  = wait
// down() is the P operation = signal

public class Semaphore {

   private int value;

   public Semaphore (int initial) {
      value = initial;
   }

   public  synchronized void  up() {
      ++ value;
      notifyAll ();  // ??? should be notify() but does not work in some browsers
   }

   public  synchronized void  down() throws InterruptedException {
      while (value <= 0) wait ();
      -- value;
   }
}
```


Programação com *threads* ?



- Uso muito complicado para a maioria
- O processo de desenvolvimento é árduo
- *Uso* adequado para situações onde o desempenho for crítico

Monitor ControlaAcesso

```
public class ControlaAcesso {
   private boolean ocupado = false;
   public synchronized void request() {
      while (ocupado) {
         try {
             wait();
         } catch (InterruptedException e) { }
      }
      ocupado = true;
}
   public synchronized void release() {
      ocupado = false;
      notifyAll();
   }
}
```


Definição da Classe Usuário

```
public class Usuario extends Thread {
   private ControlaAcesso monitor;

   public Usuario (ControlaAcesso m) {
      monitor = m;
   }

   public void run() {
      for (int i = 0; i < 5; i++) {
         monitor.request();
                  ... usa recurso ...
         monitor.release();
      }
   }
}
```

Criação das Threads

```
public class RequestRelease {
   public static void main(String[] args) {
      ControlaAcesso monitor = new ControlaAcesso();

      Usuario us_1 = new Usuario(monitor);
      Usuario us_2 = new Usuario(monitor);

      us_2.start();
      us_1.start();
   }
}
```