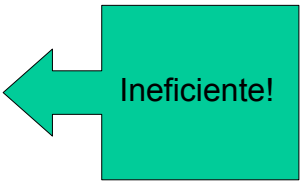
Sistemas Operacionais

Threads em Java

Prof. Fernando Parente Garcia

Aplicação sem threads

```
public class DoAnything {
   public void do() {
     while ( true ) {
          doAnything();
          processAnotherThing();
    }
   }
}
```



Aplicação com threads

```
class DoAnything() extends Thread
    { public void run() { … } }
class DoAnotherThing extends Thread
    { public void run() { … } }
public class DoAnything {
        public void do() {
          DoAnything dan = new DoAnything();
          dan.start();
          DoAnotherThing dat = new DoAnotherThing();
          dat.start();
     }
```

Criando uma Thread

- A classe Thread é a responsável pela implementação de threads em Java.
  - Uma thread em Java é representada por um objeto da classe Thread.
- Há duas maneiras de criar uma thread em Java:
  - Estender a classe Thread e escrever o método run();
  - Implementar a interface Runnable e escrever o método run().
- Na nova classe que estende Thread ou implementa Runnable, deve-se ter um método run() que implementará o comportamento da nova thread.
  - O código no método run() define um caminho de execução independente que termina quando o método finaliza.

Estendendo a Classe Thread

- Procedimento para criar uma thread estendendo Thread:
  - A nova classe, que descende de Thread, deve sobrescrever o método run() para definir o código que será executado pela thread.
  - Esta subclasse de Thread pode chamar os construtores de Thread explicitamente em seus construtores para inicializar a thread.
  - O método start() herdado de Thread é chamado sobre o objeto da nova subclasse para tornar a thread candidata a ser executada.

Estendendo a Classe Thread

```
public class HelloWorldThread extends Thread {
   public void run() {
    while ( true )
    {
       System.out.println( "Hello World of threads!" );
    }
   }
   public static void main( String args[] ) {
    HelloWorldThread t = new HelloWorldThread();
    t.start();
    while ( true )
    {
       System.out.println( "Thread 1" );
    }
   }
}
```

Thread filha

Thread principal

Implementando a Interface Runnable

- Procedimento para criar uma thread implementando Runnable:
  - A nova classe deve prover um método run() que será executado pela thread.
  - Um objeto da classe Thread deve ser criado. Um objeto da nova classe que implementou Runnable deve ser passado como argumento para um construtor da classe Thread.
  - O método start() é chamado sobre o objeto da classe Thread criado no passo acima. Isso ativa a nova thread. O método start() retorna o controle para thread principal imediatamente após a nova thread ter sido disparada.

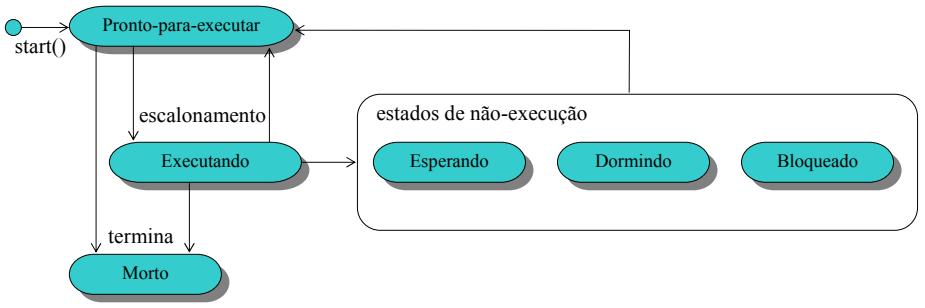
Implementando a Interface Runnable

```
public class HelloWorldThread implements Runnable {
   public void run()
   {
    while ( true )
    {
       System.out.println( "Hello World of threads!" );
    }
   }
   public static void main( String args[] ) {
    HelloWorldThread t = new HelloWorldThread();
    Thread t0 = new Thread( t );
        t0.start();
    while ( true )
    {
       System.out.println( "Thread 1" );
    }
   }
}
```

Thread filha

Thread principal

Estados de uma Thread

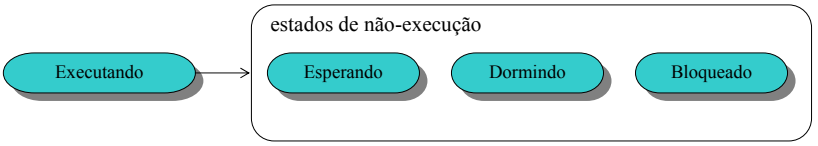


Estados de uma Thread

- Executando:
  - A CPU está realmente executando a thread.
- Pronto-para-executar:
  - Uma thread não vai diretamente para o estado de execução após ser criada ou após deixar algum dos estados de não-execução. Ela primeiro vai para o estado de pronto-para-executar, significando que ela agora é candidata a ser executada.
  - Uma chamada ao método estático yield() da classe Thread suspende a execução da thread corrente, levando-a ao estado de pronto-para-executar, e liberando a CPU para outras threads. Uma vez no estado de pronto-para-executar, ela aguardará sua vez de usar a CPU novamente.

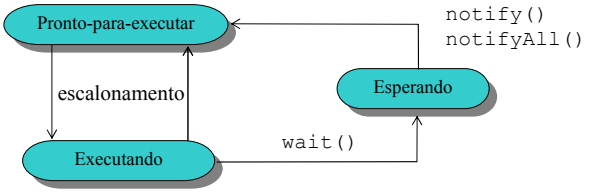
Estados de uma Thread

- Morto:
  - Uma thread chega ao estado de morto ao completar sua execução.
- Não-execução:
  - Uma thread pode sair do estado de execução e entrar em um dos estados de não-execução, dependendo da transição. A thread vai permanecer nesse estado até alguma transição especial movê-la para o estado de pronto-para-executar.



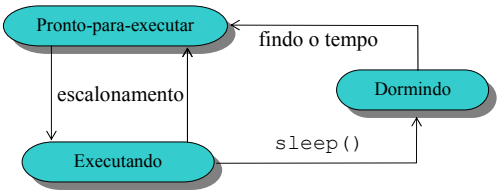
Estados de uma Thread

- Não-execução (cont.):
  - **Esperando:** estando em execução, uma thread pode chamar o método wait() para por a si mesma em estado de espera. Essa thread deve ser notificada através do método notify() por outra thread para mover-se para o estado de pronto-para-executar.



Estados de uma Thread

- Não-execução (cont.):
  - **Dormindo:** Uma chamada ao método estático sleep(tempo) da classe Thread leva a thread corrente ao estado de dormência. Ela irá acordar após transcorrido o tempo determinado de dormência, e migrará para o estado de pronto-para-executar.



- **Bloqueado:** Uma thread em execução, será bloqueada quando houver um I/O.

Outros métodos da classe Thread

- boolean isAlive() - este método pode ser usado para saber se uma *thread* ainda está viva ou não. Pode ser útil para uma *thread* pai saber sobre suas filhas.
- boolean join() - uma chamada a este método irá esperar até que a *thread* tenha terminado.

Outros métodos da classe Thread

```
class Counter extends Thread
{ ... }

public static void main ( String args[] )
{
   Counter ca = new Counter("Counter A");
   Counter cb = new Counter("Counter B");
   try {
      ca.join(); // vai esperar até que a thread A termine
      cb.join(); // vai esperar até que a thread B termine
      if ( !ca.isAlive() ) {
    System.out.println("Counter A not alive.");
    ...
```

Prioridades de uma Thread

- Threads podem ter prioridades que são usadas pelo escalonador para determinar como elas serão tratadas.
- Prioridades são valores inteiros que podem assumir
  - 1 (menor prioridade, indicada pela constante Thread.MIN_PRIORITY)
  - 10 (maior prioridade, indicada pela constante Thread.MAX_PRIORITY)
  - qualquer valor entre Thread.MIN_PRIORITY e Thread.MAX_PRIORITY
  - 5 (se nenhuma prioridade é indicada, a thread assume o valor *default* - Thread.NORM_PRIORITY).
- Uma thread herda a prioriade de seu pai.
- Métodos:
  - setPriority() – altera a prioridade da thread
  - getPriority() – lê a prioridade da thread

Interrompendo Threads

- Às vezes se faz necessário interromper uma thread.
- Exemplo:
  - O usuário clicou no botão fechar de seu aplicativo.
- Use o método interrupt() da classe thread para fazer isso.

- Escreva sua thread de forma que ela trate interrupções:

```
while (!interrupted()
        && mais trabalho a fazer)
{ trabalhe! }
```

Agrupando Threads

- Utilize a classe ThreadGroup para agrupar threads similares.
- Útil quando o seu programa usa muitas threads.
- Exemplo:
  - Cada download de cada imagem de uma página web é feita por um thread diferente em um browser. Se o usuário está tentando fechar o browser, como interromper todas as threads facilmente?

Agrupando Threads

```
ThreadGroup g = new ThreadGroup("imagens");
Thread t = new Thread( g, "imagem01" );
...
// Verificar se ainda existem threads em
// execução de um determinado grupo
if( g.activeCount == 0 )
{ ... }

// Interromper todas as thread de um grupo
g.interrupt();
```