



Execução concorrente em Java: threads

- O que são Threads?
- Criação e métodos de Threads
- Sincronização
- Comunicação entre Threads



Programação concorrente: introdução



- ◆ O mundo real funciona concorrentemente: várias atividades podem ser executadas em paralelo. Exemplo: uma pessoa pode estar
 - respirando, e,
 - falando, e
 - escrevendo, e
 - lendo, etc.
- ◆ Computadores também operam concorrentemente. Exemplo: um computador pode estar
 - compilando um programa, e
 - recebendo uma mensagem, e,
 - imprimindo um arquivo, e,
 - tocando música, etc.



Objetivos da programação concorrente



- ◆ Reduzir o tempo total de processamento
 - múltiplos processadores
- ◆ Aumentar confiabilidade e disponibilidade
 - processadores distribuídos
- ◆ Obter especialização de serviços
 - sistemas operacionais
 - simuladores
- ◆ Implementar aplicações distribuídas
 - correio eletrônico



Aplicações



- ◆ Cada atividade possui uma seqüência própria de operações
- ◆ Programas **independentes** podem se encarregar de
 - Compilador: compilar um programa
 - Navegador: receber uma mensagem e exibir animação
 - Subsistema de e/s: imprimir um arquivo
 - Subsistema de áudio: tocar música
- ◆ **Multiprogramação**: vários programas executam sob controle do sistema operacional

Programação concorrente

- ♦ Uma unidade concorrente é um componente de um programa que não exige a execução seqüencial, ou seja, que sua execução seja realizada antes ou após a execução de outros componentes do programa
- ♦ O termo programação concorrente é usado no sentido abrangente, para designar a programação **paralela** e a programação **distribuída**
- ♦ Concorrência relaciona-se com fluxo de controle: em um programa, existe mais de um fluxo de controle ativo.

Fluxo de execução

Execução seqüencial

- ◆ Comandos de controle de fluxo de execução
 - Seqüencial
 - Condicional
 - Iterativo
- ◆ Requisição de execução de unidades
 - explícita: chamada de métodos
 - implícita: ativação de exceções
- ◆ Programa controla a ordem de execução

Execução concorrente

- ◆ Cada tarefa é uma unidade de execução autônoma (um thread)
- ◆ Tarefas podem ser totalmente independentes
 - Exemplo: execução de um mesmo método sobre dois objetos (da mesma classe)
- ◆ Tarefas podem necessitar comunicação
- ◆ Programa não controla a ordem de execução

Threads: o que são?

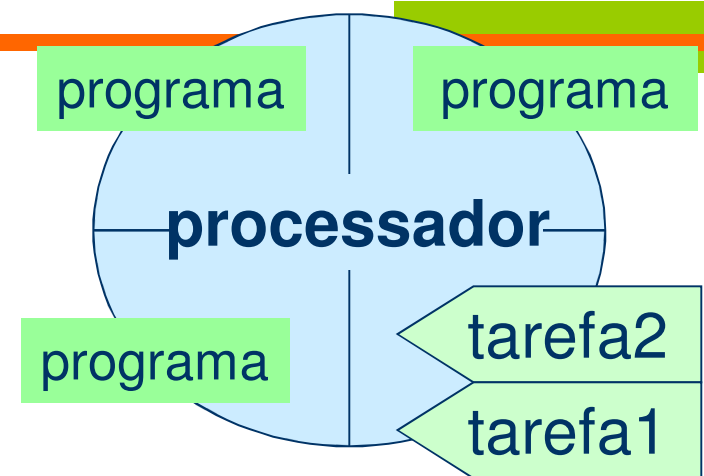
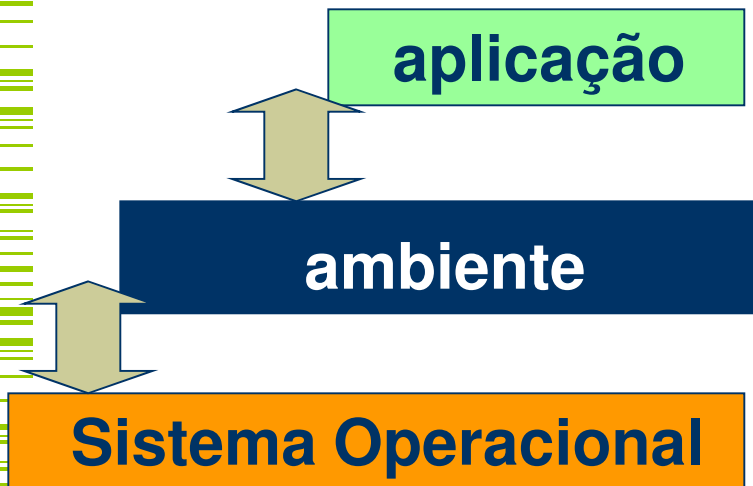
- ♦ **Definição básica:** “Fluxo de controle seqüencial isolado dentro de um programa.”
 - Outra denominação: LightWeight Processes
- ♦ **Programas multithreaded:** Múltiplos threads concorrentes de execução num único programa, realizando várias tarefas “ao mesmo” tempo.
 - Exemplo: programa do usuário + coleta de lixo
- ♦ Diferentes threads podem executar em diferentes processadores, se disponíveis, ou compartilhar um processador único
- ♦ Diferentes threads no mesmo programa compartilham um ambiente global (memória, processador, registradores, etc.)

Algumas aplicações multithreaded

- ♦ Programação Reativa : aplicação responde a eventos de entrada.
 - Exemplo: interfaces com o usuário, onde cada evento corresponde a uma ação
- ♦ Programação Interativa: uma tarefa para fazer alguma interação com o usuário, outra para exibir mensagens, outra para fazer animação, etc..
- ♦ Paralelismo físico/distribuição: para tirar vantagem de múltiplas CPUs centralizadas ou distribuídas

Programação concorrente: níveis

- ♦ Sistema operacional:
 - primitivas de controle
 - forma de escalonamento
- ♦ Ambiente de execução Java depende do SO
 - Windows: time-slice
 - Solaris: prioridade



- ♦ Aplicação Java: biblioteca de classes java. lang
 - Thread, ThreadGroup, ThreadLocal, ThreadDeath

Criação de threads em Java

- ♦ Criar uma subclasse da classe Thread

```
public class MyClass  
    extends Thread { ... }
```

class Thread

class MyClass

- ♦ Implementar a interface Runnable;

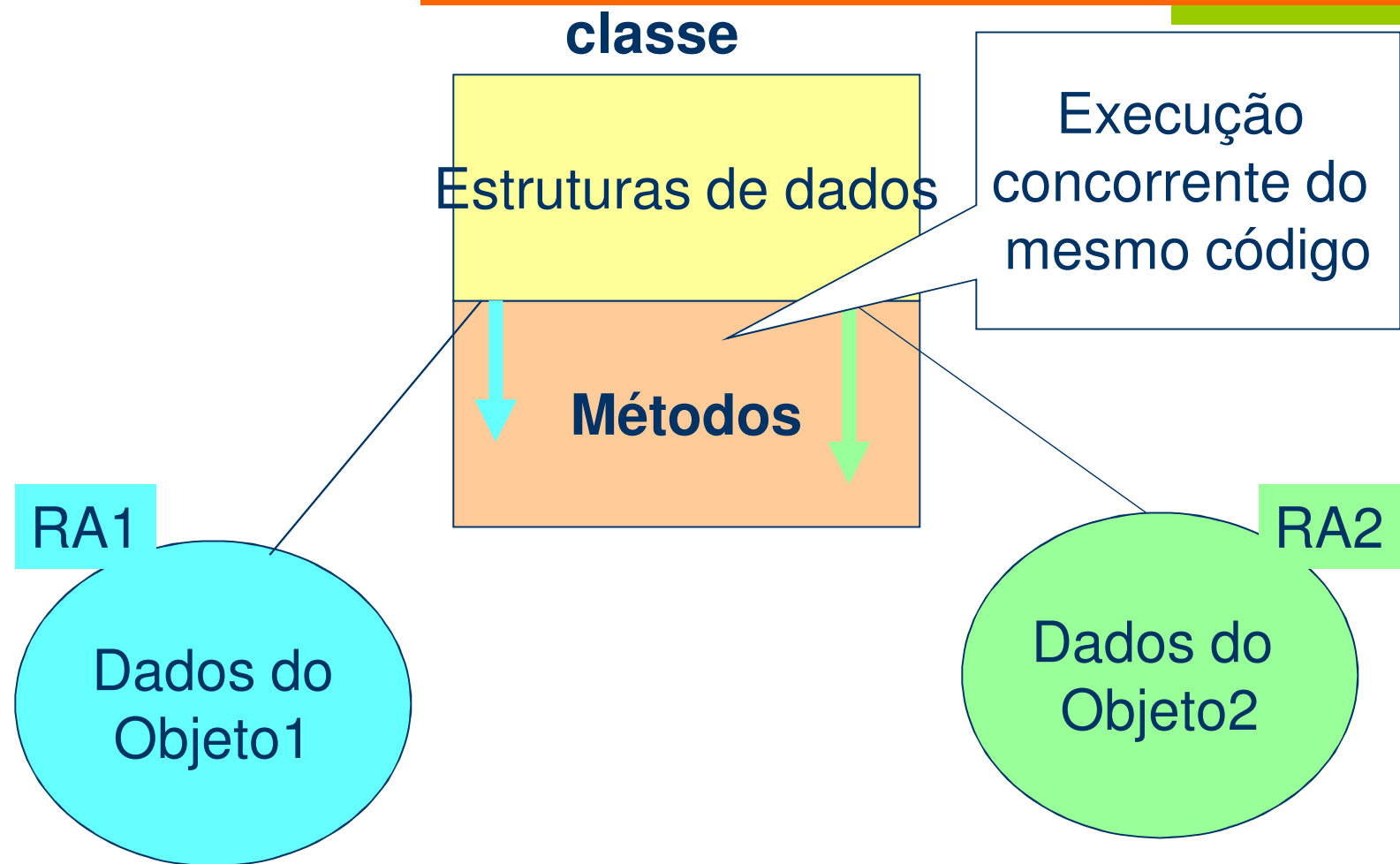
```
public class MyClass  
    extends Applet  
    implements Runnable {  
    ... }
```

class Applet

interface
Runnable


class MyClass

Instanciação de objetos concorrentes




Execução de threads

- ◆ Cada thread possui um método *run()* que define a atividade concorrente

Exemplo: **public void run() {**
O quê?  **for (int count=0; count<1000; count++)**
System.out.println(nome); }

- ◆ A atividade concorrente inicia quando é invocado o método *start()* sobre um objeto.

Exemplo: **public static void main(String[] arg) {....**
Quando?  **um.start();**
dois.start();}

Classe Thread

- ◆ Usada quando a classe a ser executada concorrentemente não deriva de outra classe
- ◆ Contém métodos para controlar a execução
- ◆ Criação e execução:
 - Declarar uma nova classe que seja subclasse da classe Thread
 - Sobrescrever o método run() com o código que será executado pela thread
 - Instanciar a nova classe
 - Invocar seu método start(); o método rodará em seu próprio thread de controle

Contexto da classe Thread

java.lang.Object



java.lang.Thread

```
public class Thread  
    extends Object  
    implements Runnable
```

Alguns métodos da classe Thread

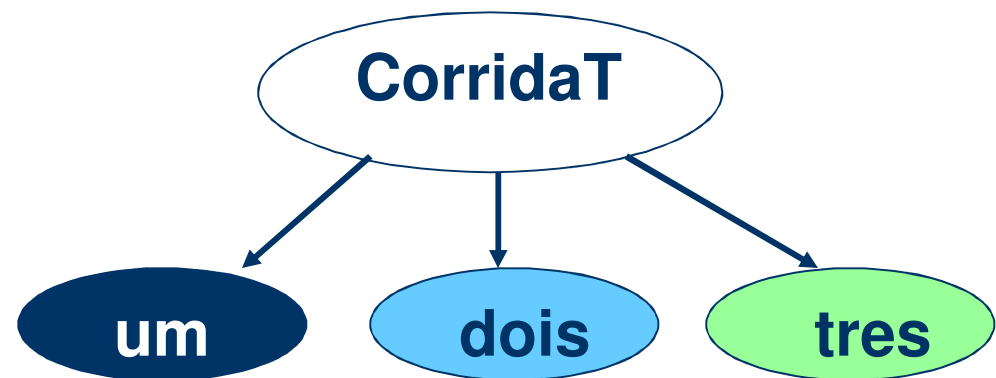
- ⌘ **start()**: inicia a execução do Thread;
- ⌘ **sleep()**: suspende a execução por um determinado tempo (especificado em milisegundos) e automaticamente recomeça a execução;
- ⌘ **destroy()**: destrói esta thread.
- ⌘ Demais métodos: jdk1.2.1/docs/api/java/lang/Thread.html

Exemplo de extensão Thread

```
class Piloto extends Thread{  
    private String nome;  
    public Piloto(String str){  
        nome = str;  
    }  
    public void run(){  
        System.out.println("****LARGADA ****");  
        System.out.println("Primeira volta: " + nome);  
        for(int cont=0; cont<10000; cont++){  
            System.out.println(nome + " -> Terminou a Corrida !!!");  
        }  
    }  
}
```

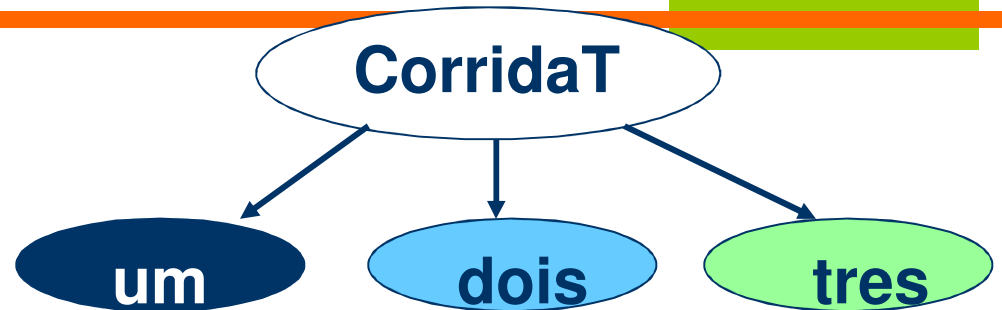
Exemplo de execução

```
public class CorridaT{  
    public static void main(String[] args){  
        Piloto um = new Piloto("Rubinho");  
        Piloto dois = new Piloto("Schumacher");  
        Piloto tres = new Piloto("Raikonnen");  
        um.start();  
        dois.start();  
        tres.start();  
    }  
}
```



Quem terminará antes?

Resultado de uma execução



*** LARGADA ***

*** LARGADA ***

*** LARGADA ***

Primeira volta:Rubinho

Primeira volta:Schumacher

Primeira volta: Raikonnen

Rubinho -> Terminou a Corrida !!!

Raikonnen -> Terminou a Corrida !!!

Schumacher -> Terminou a Corrida !!!

Resultado de outra execução

***** LARGADA *****

Primeira volta:Rubinho

Rubinho -> Terminou a Corrida !!!

***** LARGADA *****

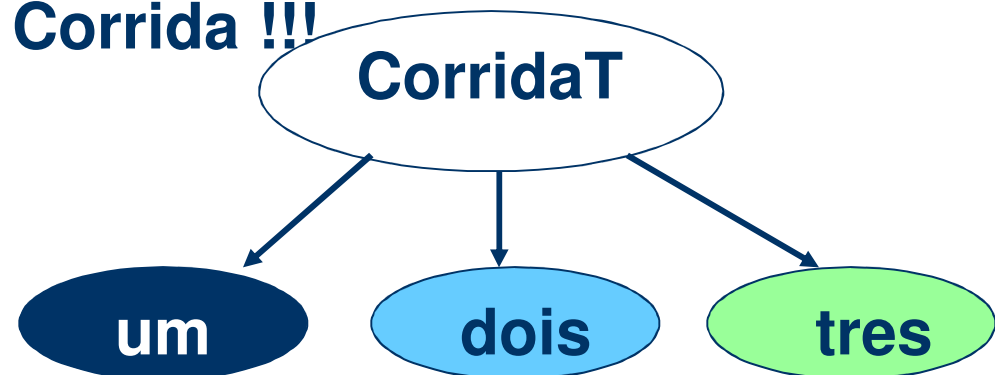
Primeira volta:Schumacher

***** LARGADA *****

Schumacher -> Terminou a Corrida !!!

Primeira volta: Raikonnen

Raikonnen -> Terminou a Corrida !!!



Interface Runnable

- ◆ Usada quando não se pode herdar da classe Thread, pois há necessidade de herança de alguma outra classe
 - possui apenas o método run()
- ◆ Cria-se um objeto da classe base Thread, porém o código a ser executado está descrito na classe do usuário(derivada + ancestrais).
- ◆ Criação e execução:
 - Declarar uma nova classe que implementa a interface Runnable
 - Sobrescrever o método run()
 - Criar um objeto da classe Thread.
 - Exemplo: `Thread um = new Thread(this);`

Exemplo de implementação Runnable (1)

```
class PilotoR implements Runnable{
    private String nome;
    public PilotoR(String str){
        nome = str;
    }
    public void run(){
        System.out.println("*** LARGADA ***");
        System.out.println(" Primeira volta:" + nome);
        for(int cont=0; cont<10000; cont++) {};
        System.out.println(nome + " -> Terminou a Corrida !!!");
    }
}
```

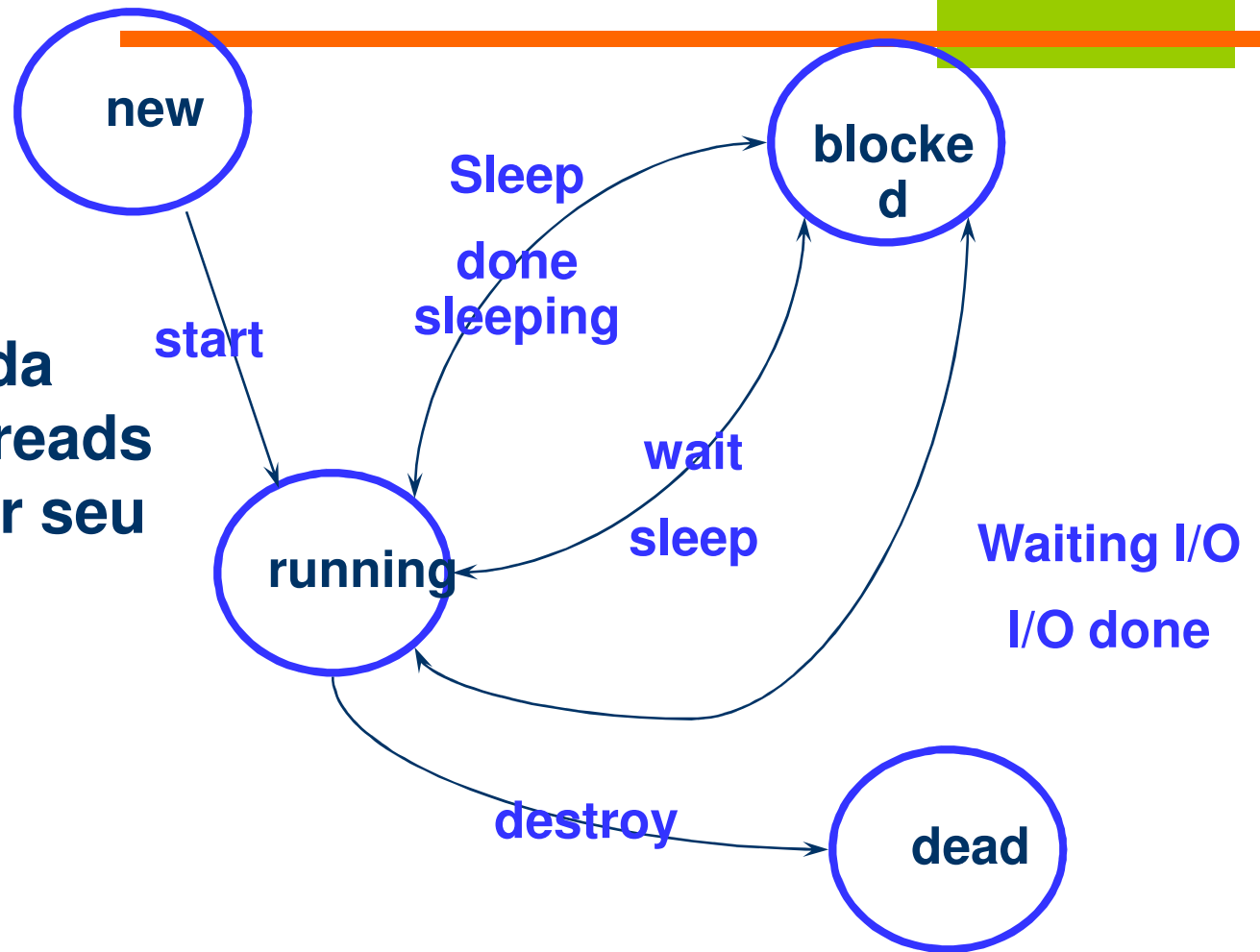
Exemplo de implementação Runnable (2)

```
public class CorridaR{  
    public static void main(String[] args){  
        PilotoR um = new PilotoR("Rubinho");  
        PilotoR dois = new PilotoR("Schumacher");  
        PilotoR tres = new PilotoR(" Raikonnen ");  
        new Thread(um).start();  
        new Thread(dois).start();  
        new Thread(tres).start();  
    }  
}
```

Estados de Threads

No decorrer da execução, threads podem alterar seu estado:

- ativo
- inativo
- encerrado



Exemplo de sleep

Exemplo de Deitel.com

```
class PrintThread extends Thread {  
    private int sleepTime;  
    public PrintThread (String name) { // construtor  
        super(nome); // nome do thread: construtor de Tread  
        sleepTime=(int)(Math.random() * 5000); // 0-5 sec  
        System.out.println("Name: "+ getName() + "; sleep " + sleepTime);  
        public void run ( ){  
            try{  
                System.out.println(getName() + "going to sleep")  
                Thread.sleep (sleepTime) ;}  
            catch ( InterruptedException exception) {  
                System.out.println(exception.toString()); }  
        }  
    }  
}
```

quem?

dorme!

fui interrompida?

Exceção: sleep

```
try {  
    System.out.println(getName() + "going to sleep");  
    Thread.sleep (sleepTime) ;  
}  
catch ( InterruptedException exception)  
{  
    System.out.println(exception.toString());  
}
```

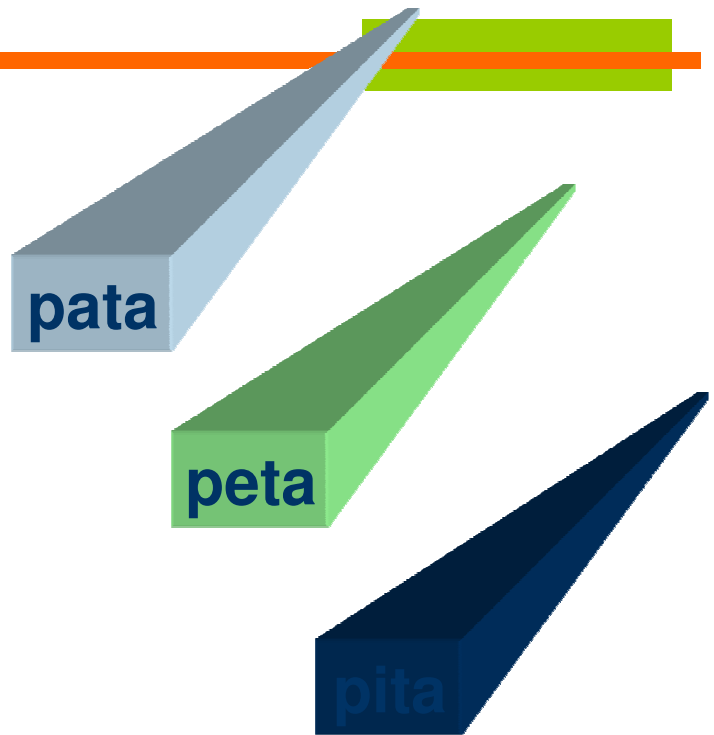
método sleep pode disparar exceção*

nome da exceção

* se outro thread chama o método interrupt durante sleep

Exemplo de programa de teste

```
public class ThreadTester {  
    public static void main(String args[]){  
        PrintThread t1, t2, t3;  
        t1 = new PrintThread("pata");  
        t2 = new PrintThread("peta");  
        t3 = new PrintThread("pita");  
        System.out.println("\n "Iniciando...");  
        t1.start();  
        t2.start();  
        t3.start();  
        System.out.println("\n "Iniciadas !");  
    }  
}
```



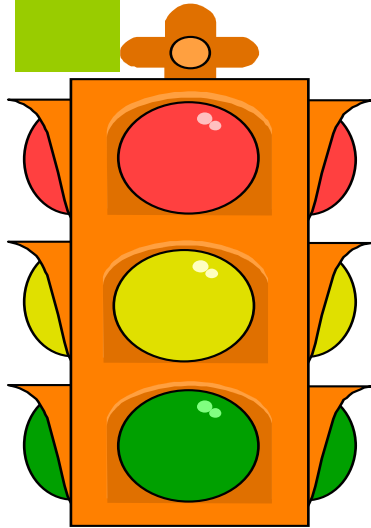
Prioridades

- ◆ Cada thread possui uma prioridade (entre 1 e 10)
- ◆ Default: prioridade = 5
- ◆ Prioridade transmitida por herança
- ◆ Threads de igual prioridade cedem processador por:
 - chamada de yield
 - time-slicing
- ◆ Thread de mais alta prioridade apta a executar:
 - faz com que o thread de menor prioridade ceda o processador
 - seja executada até que termine a sua execução, ou,
 - tenha a sua execução suspensa (sleep/wait...)

Resumo de estados de threads

- ♦ A execução do thread depende do ambiente de execução (sistema operacional)
- ♦ New threads: thread criado com new mas ainda não está rodando
- ♦ Runnable threads: Depois de chamado o método start(), o thread está apto a ser executada (depende da disponibilidade do sistema)
- ♦ Blocked threads: um thread entra no estado bloqueado quando ocorre
 - chamada aos métodos sleep(), suspend() ou wait()
 - espera por I/O
- ♦ Dead threads: execução encerrada (o objeto thread é destruído)

Questões importantes em Multithreading



- ♦ **Segurança:** como sincronizar threads para que elas não interfiram com uma outra
- ♦ **Longevidade:** como evitar situações de deadlock ou livelock para garantir que todos os threads farão progresso

- ♦ **Desempenho:** a mudança de contexto dos diversos threads provoca queda de desempenho (overhead)

