

# Programação Orientada a Objetos

**POO**

**Aula Teórica**

**Threads e Threads em Java**

**Prof. Dirson S. Campos**  
**[dirson\\_campos@ufg.br](mailto:dirson_campos@ufg.br)**

08/01/2023

# Fluxos de Execução

- Um programa sequencial consiste de um único fluxo de execução, o qual realiza uma certa tarefa computacional.
- A maioria dos programas simples tem essa característica: só possuem um único fluxo de execução.
- Por conseguinte, não executam dois trechos de código **“simultaneamente”**.

# Fluxos de Execução

- Grande parte do *software* de maior complexidade escrito hoje em dia faz uso de mais de uma linha de execução.

# Exemplos destes *Softwares*

- Navegador (*browser*)
- Consegue fazer o *download* de vários arquivos ao mesmo tempo, gerenciando as diferentes velocidades de cada servidor e, ainda assim, permitindo que o usuário continue interagindo, mudando de página enquanto os arquivos estão sendo carregados.

# *Threads*

- *Thread* tradução livre em português de “linha”, “filamento”, “fio”.
- *Threads* são fluxos de execução que rodam dentro de um processo.
- Sequencia de instruções a serem executadas dentro de um programa.
- *Thread* é “um processo leve” *“lightweight process”*

# *Threads*

- *Thread* é uma abstração que permite que uma aplicação execute mais de um trecho de código (ex: mais de um método) “simultaneamente”.
- Processos permitem ao Sistema Operacional (S.O.) executar mais de uma aplicação ao mesmo tempo.

# *Threads e Processos*

- Existem duas características fundamentais que são usualmente tratadas de forma independente pelo S.O.:
- Propriedade de recursos (*“resource ownership”*);
- Escalonamento (*“scheduling / dispatching”*).

# Propriedade de recursos

- Trata dos recursos alocados aos processos, e que são necessários para a sua execução.
- Por exemplo, ao rodar um processo de tempos em tempos, o processo pode requerer mais memória, controlar arquivos, requerer dispositivos de E/S, etc.



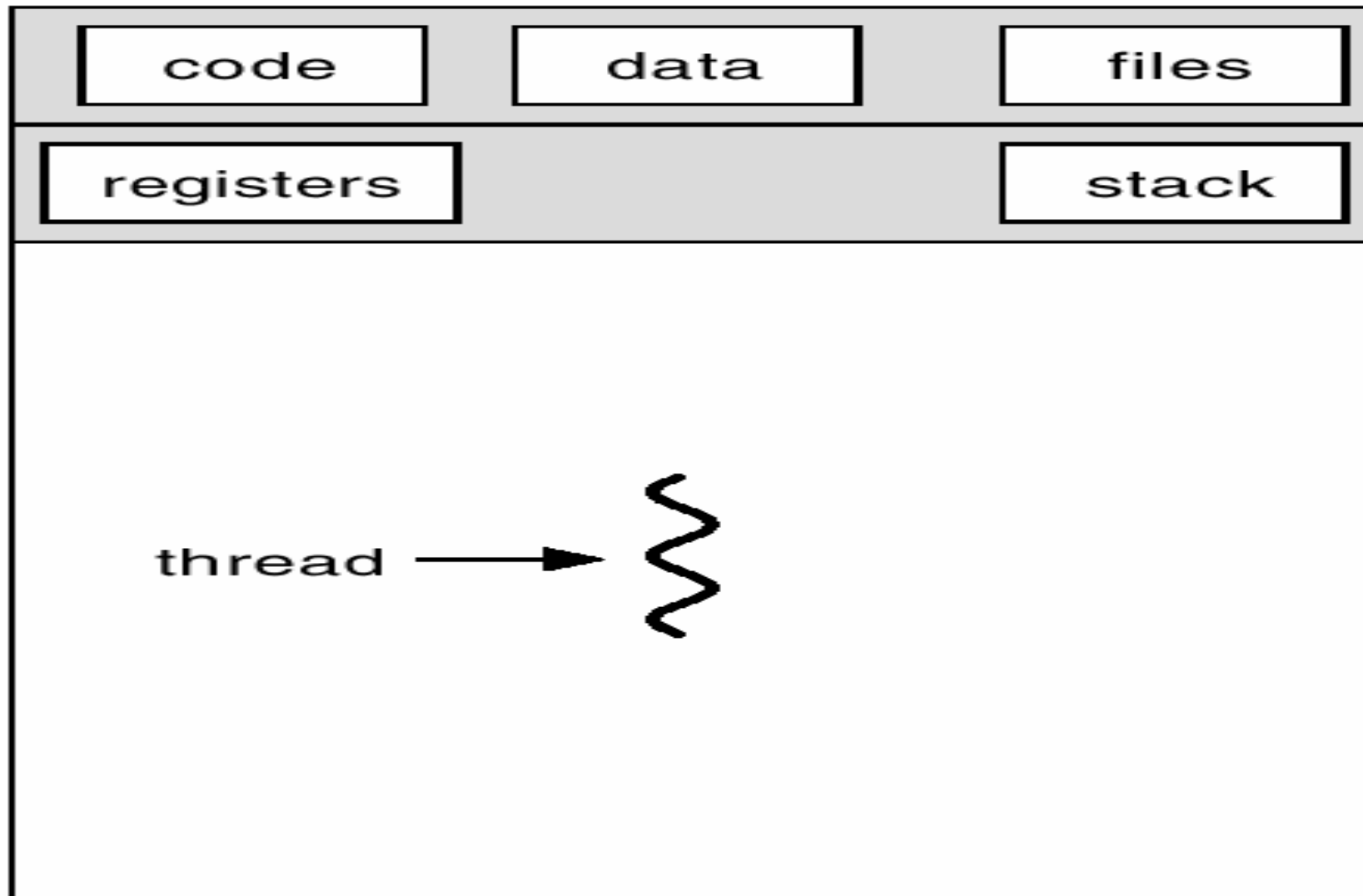
# Escalonamento

- Relacionado à unidade de despacho do S.O.
- Determina o fluxo de execução (trecho de código) que é executado pela CPU.
- Tradicionalmente o processo está associado a:

um programa em execução

um conjunto de recursos

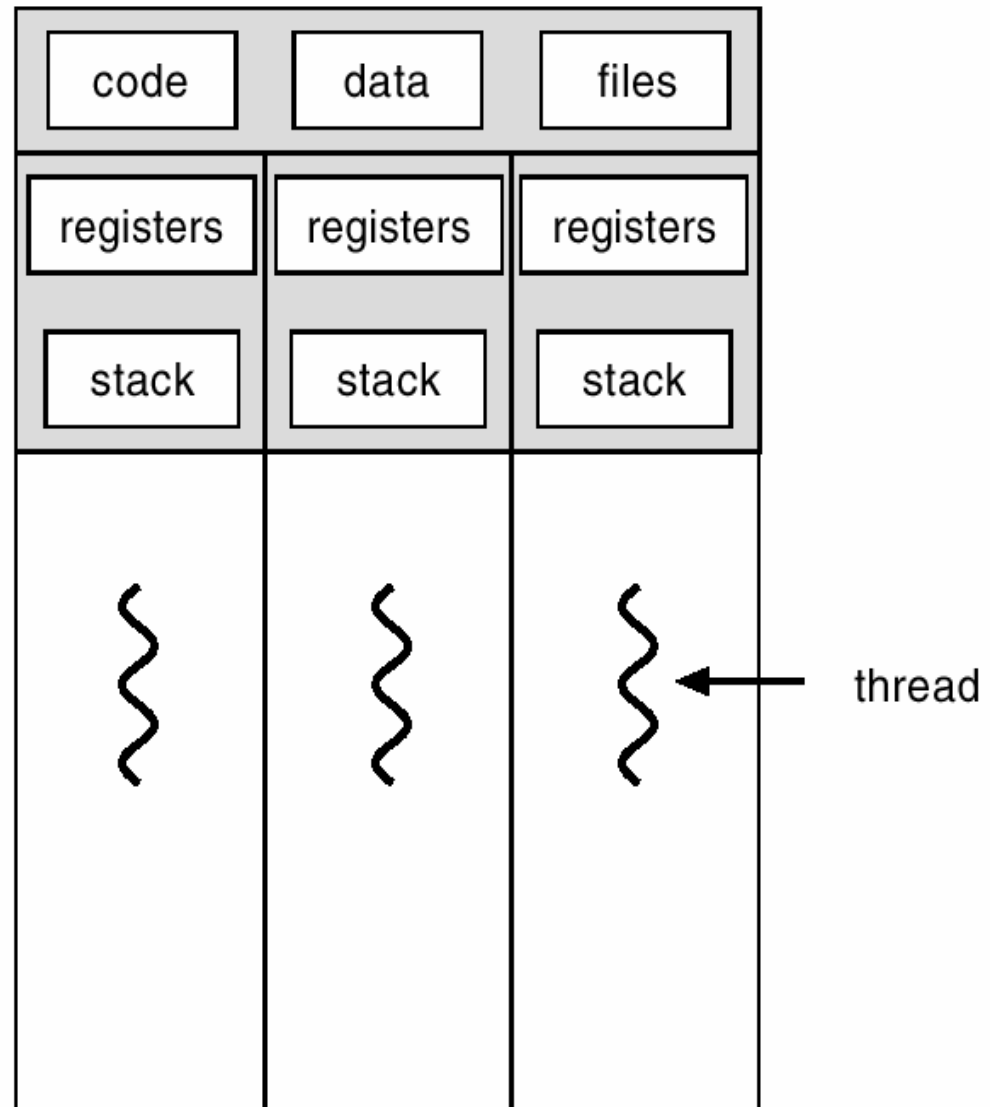
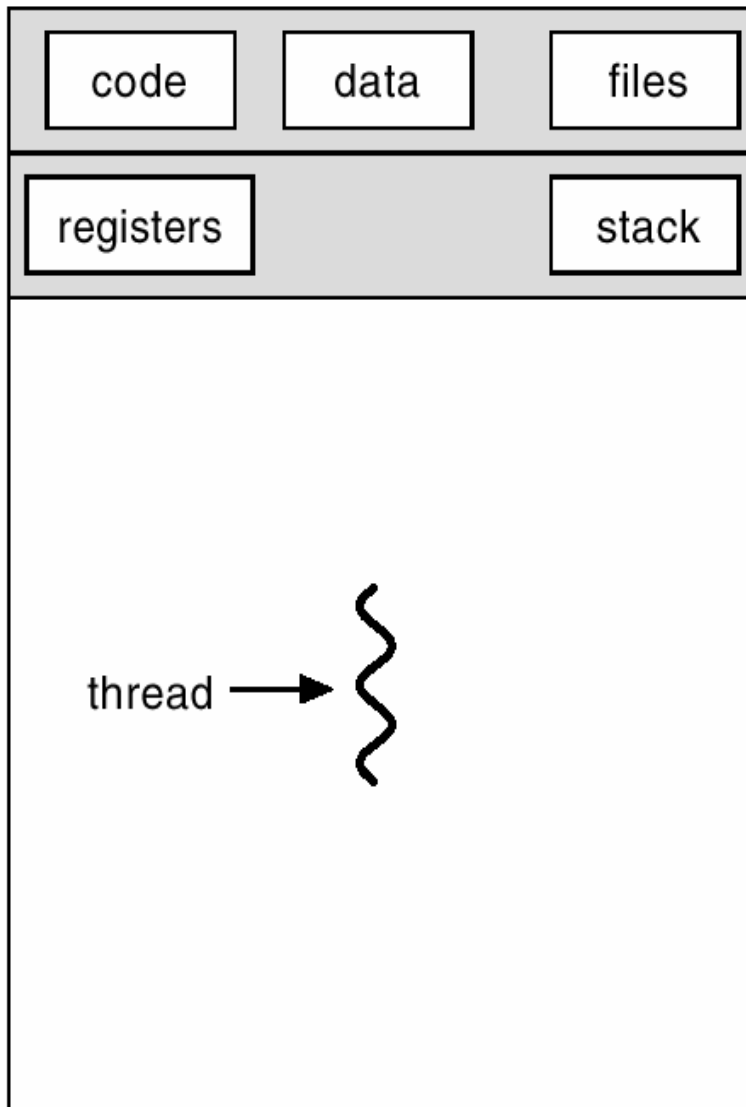
# Escalonamento



# S.O. *multi-threads*

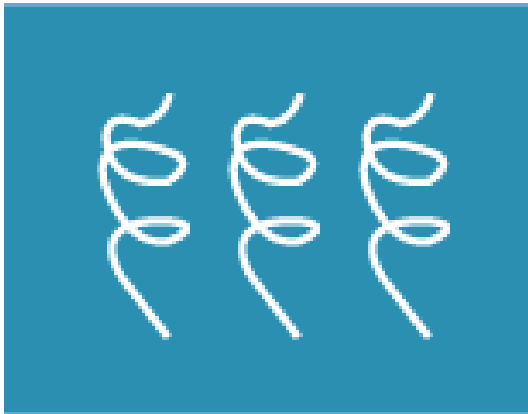
- Processos estão associados somente à propriedade de recursos, e
- *Threads* às atividades de execução (exemplos, *threads* constituem as unidades de despacho).

# S.O. *multi-threads*

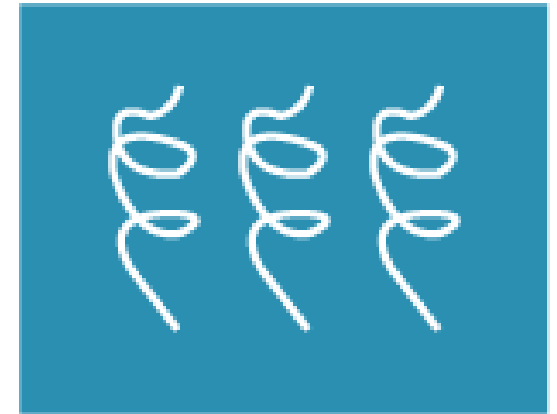
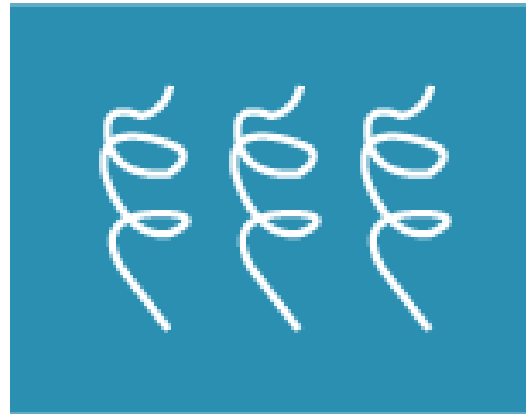


# *Multi-threads*

- *Multithreading* refere-se à habilidade do S.O. em suportar múltiplas *threads* de execução em um mesmo processo.



One Process  
Multiple Threads



Multiple Process Multiple  
Threads per Process

# *Multi-threads*

- Em um ambiente *multithreaded*, um processo:
  - é a unidade de alocação e proteção de recursos;
  - tem um espaço de endereçamento virtual que mantém a imagem do processo;
  - tem acesso controlado a outros processos, arquivos e outros recursos..

# *Multi-threads*

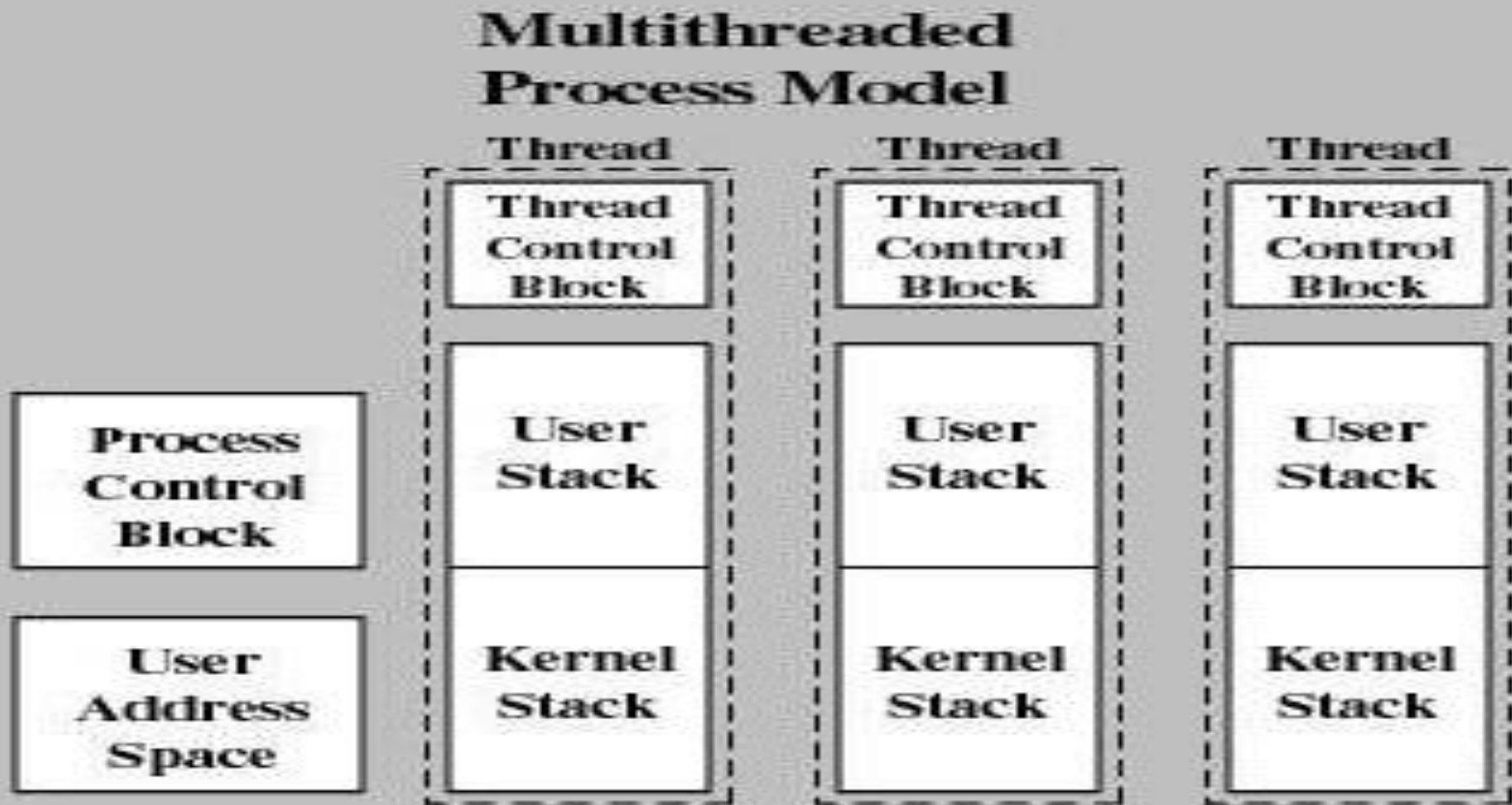
- Já uma *thread* apresenta:
  - um estado de execução (pronta, bloqueada,...).
  - um contexto salvo quando não estiver executando.
  - uma pilha de execução.
  - acesso a variáveis locais próprias.
  - acesso compartilhado com outras *threads* do processo aos recursos do mesmo.

# *Multi-threads*

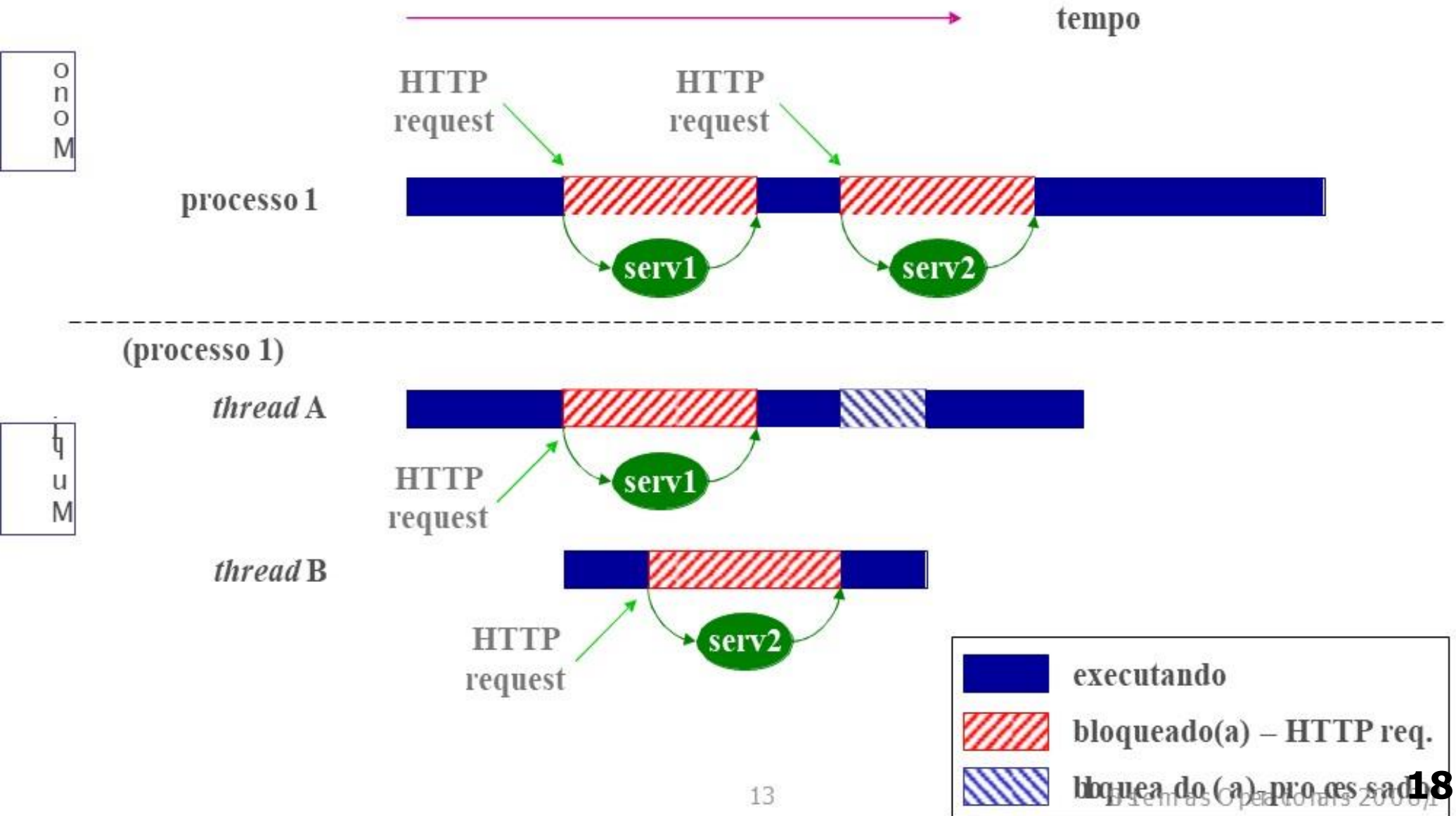
- Já uma *thread* apresenta:
  - um estado de execução (pronta, bloqueada,...).
  - um contexto salvo quando não estiver executando.
  - uma pilha de execução.
  - acesso a variáveis locais próprias.
  - acesso compartilhado com outras *threads* do processo aos recursos do mesmo.



# *Multi-threads*



# Processos mono e multithreaded - Exemplo

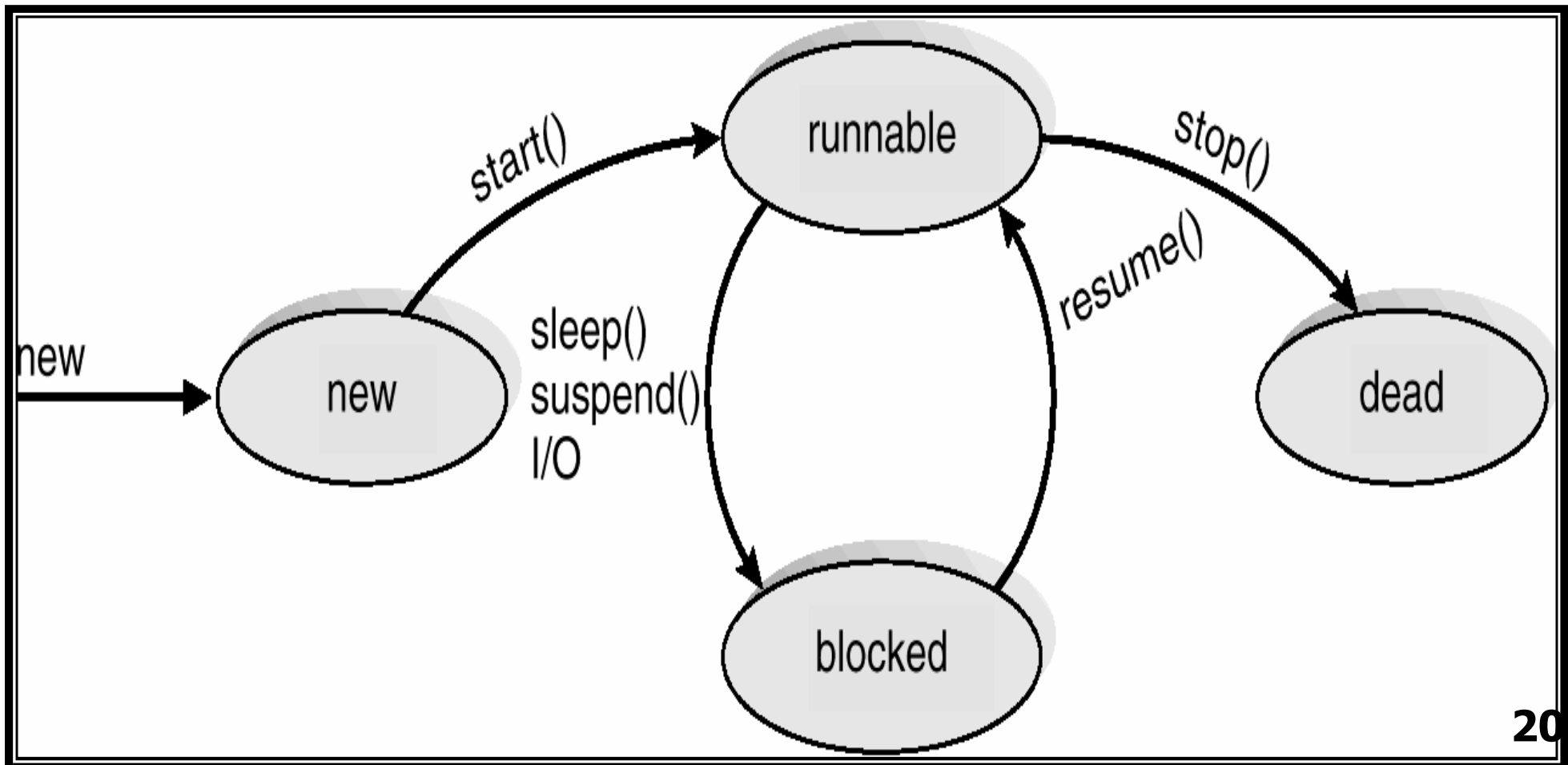


# Benefícios das *Threads*

- É mais rápido criar uma thread do que um processo.
- É mais rápido terminar uma thread que um processo.
- É mais rápido chavear entre threads de um mesmo processo do que entre processos.
- Threads podem se comunicar sem invocar o núcleo já que compartilham memória e arquivos.
- Permite paralelismo de atividades e melhor organização do programa

# Estados de uma Thread

Estados fundamentais: executando, pronta e bloqueada.



# Estados de uma Thread

**Thread** pode bloquear sem bloquear as outras threads do mesmo processo?

Suspender um processo implica em suspender todas as threads deste processo (considerando que todas as threads compartilham o mesmo espaço de endereçamento)?

## *Swapping*

O término de um processo implica no término de todas as threads do processo.

# Tipos de Threads

- A implementação de threads pode ser feita em um dos seguintes níveis:
  - Nível de usuário: user-level threads (ULTs)
  - Nível de kernel: kernel-level threads (KLTs)
  - Apenas em S.O.s c/ suporte a *multithreading*

## User-level Threads <sup>(1)</sup>

O gerenciamento das threads é feito pela aplicação

O núcleo desconhece a existência de threads.

O chaveamento entre threads não requer privilégio de modo kernel

Isso elimina o gasto de dois chaveamentos de modo.

O escalonamento é feito pela aplicação

Um processo pode aplicar sua própria política de escalonamento entre threads

# User-level Threads (2)

São implementadas através de bibliotecas (executam em qualquer S.O.).

Chamada ao sistema bloqueia todas as threads de um processo.

Não aproveita os benefícios do multiprocessamento.

Exemplos:

POSIX Pthreads



## Kernel-level Threads <sup>(1)</sup>

O gerenciamento das threads é feito pelo kernel.  
O kernel mantém a informação de contexto para processo e threads.

O chaveamento das threads é feito pelo kernel (escalonamento "thread-basis").

O bloqueio de uma thread não implica no bloqueio das outras threads do processo

As threads podem aproveitar a capacidade de multiprocessamento

## Kernel-level Threads (2)

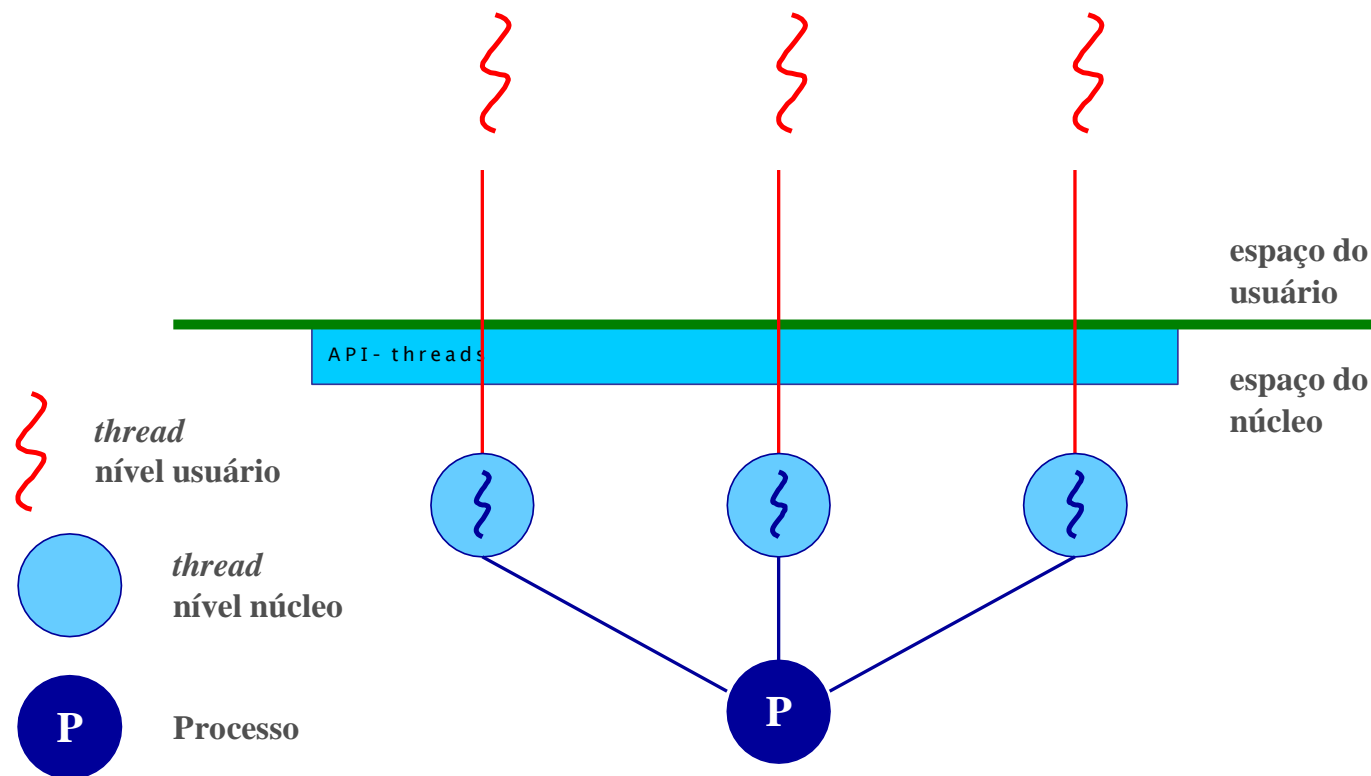
O usuário enxerga uma API para threads do núcleo

Não há código de gerenciamento de threads na área da aplicação

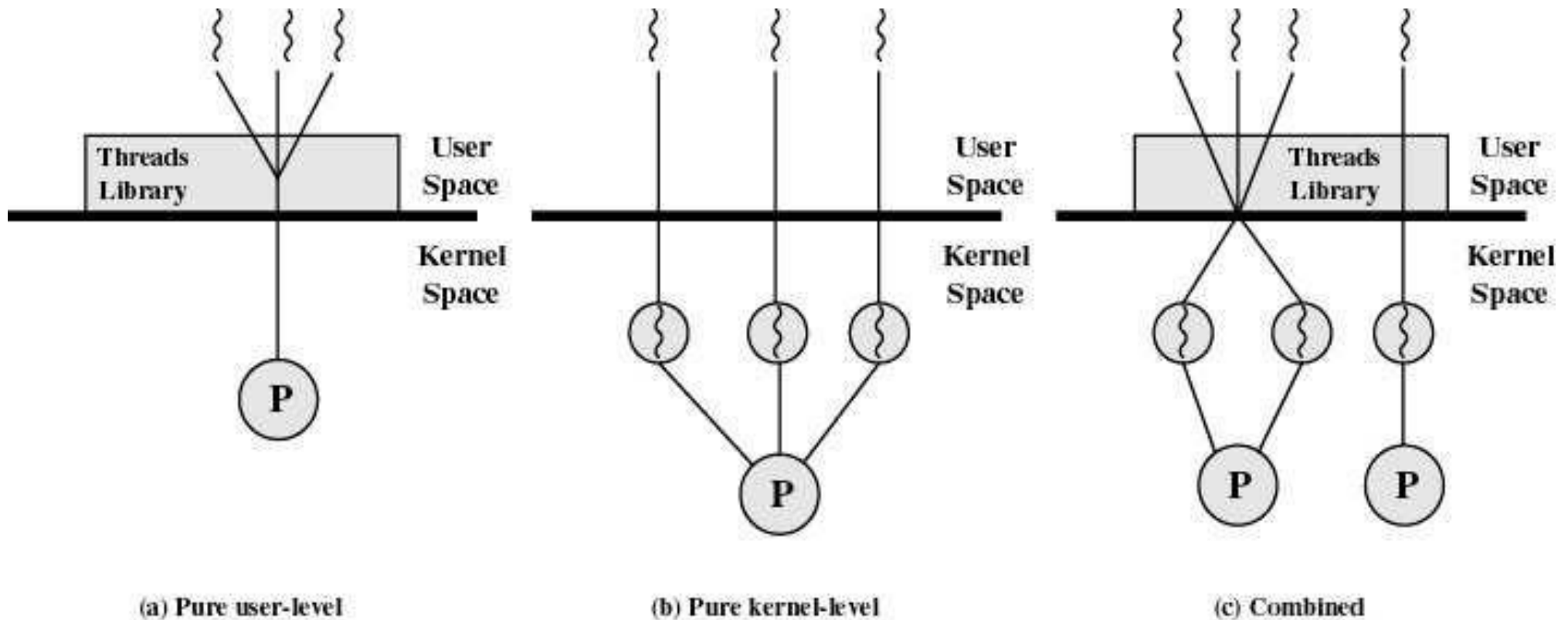
... porém a transferência de controle entre threads de um mesmo processo requer chaveamento para modo kernel.

Windows, Linux são exemplos desta abordagem.

# Kernel-level Threads (3)



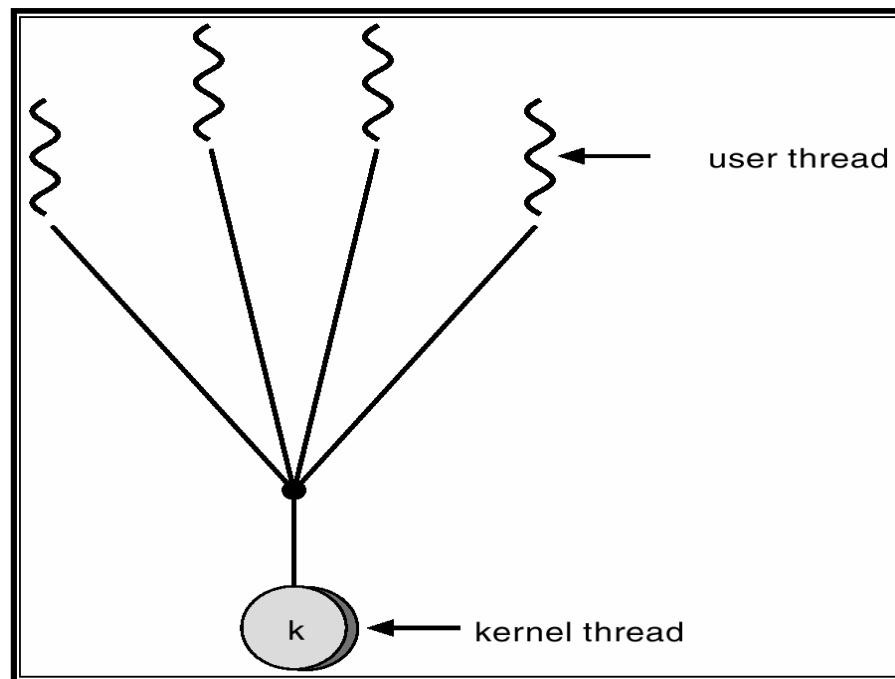
# User-level e Kernel-level Threads



{ User-level thread      (wavy circle) Kernel-level thread      (P) Process

# Modelo Muitos-para-Um

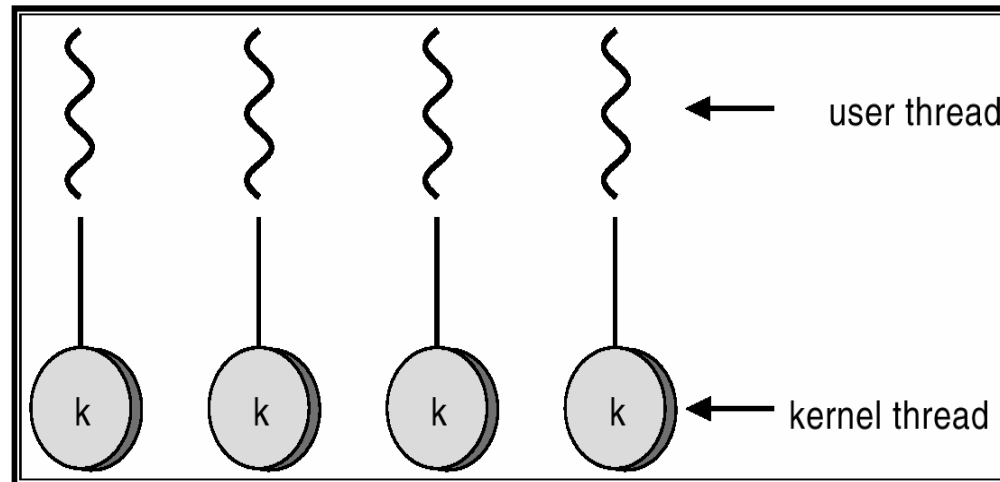
Muitas user-level threads mapeadas em uma única kernel thread. Modelo usado em sistemas que não suportam kernel threads.



# Modelo Um-para-Um

Cada user-level thread é mapeada em uma única kernel thread.

Exemplos: Windows 95/98/NT/2000

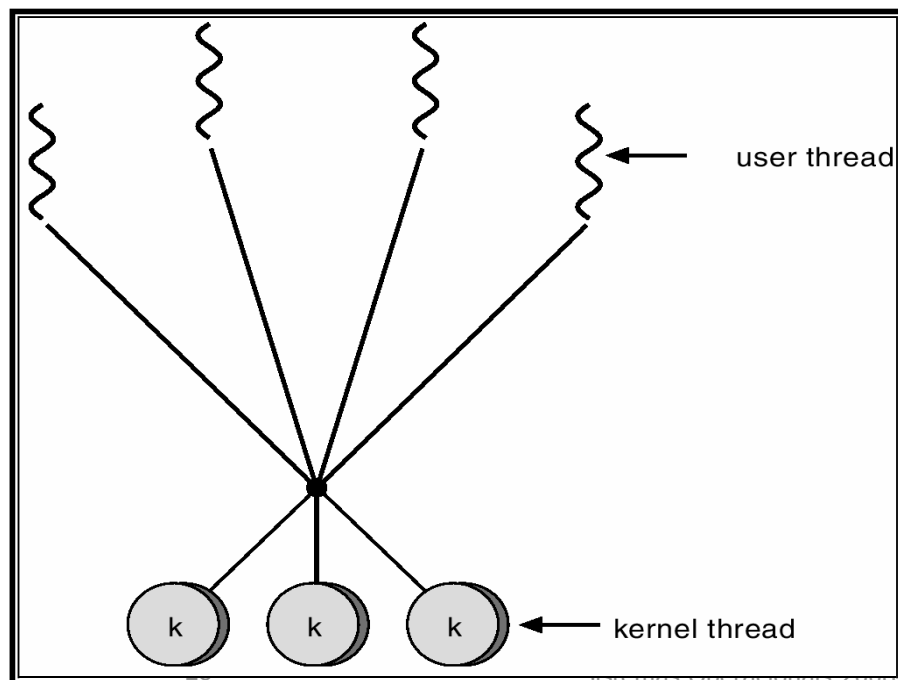


# Modelo Muitos-para-Muitos

Permite que diferentes user-level threads de um processo possam ser mapeadas em kernel threads distintas.

Permite ao S.O. criar um número suficiente de kernel threads.

Exemplos: UNIX's Windows NT/2000 com o ThreadFiber package.



# Pthreads

API padrão POSIX (IEEE 1003.1c) para a criação e sincronização de threads (user-level threads).

A API especifica o comportamento da biblioteca de threads (a implementação é tarefa do desenvolvedor da biblioteca).

Comum em sistemas operacionais UNIX.

Ex: pacote LinuxThreads, que implementa o padrão pthreads -



# Windows Threads

Implementa o modelo de mapeamento um-para-um.

Cada thread contém:

- um thread id
- um register set
- áreas separadas para user stack e kernel stack
- área para armazenamento de dados privados.

# Linux Threads

No Linux as threads são referenciadas como tasks (tarefas).

Implementa o modelo de mapeamento um-para-um.

A criação de threads é feita através da SVC (chamada ao sistema) `clone()`.

`Clone()` permite à tarefa filha compartilhar o mesmo espaço de endereçamento que a tarefa pai (processo).

Na verdade, é criado um novo processo, mas não é feita uma cópia, como no `fork()`;

O novo processo aponta p/ as estruturas de dados do pai

# Java Threads

Difícil de classificar com user thread ou kernel thread

As threads Java são gerenciadas pela JVM.

Threads em Java podem ser criadas das seguintes maneiras:

Fazendo "extend" da classe Thread.

Implementando a interface Runnable.

A JVM só suporta um processo

Criar um novo processo em java implica em criar uma nova JVM p/ rodar o novo processo

# Java

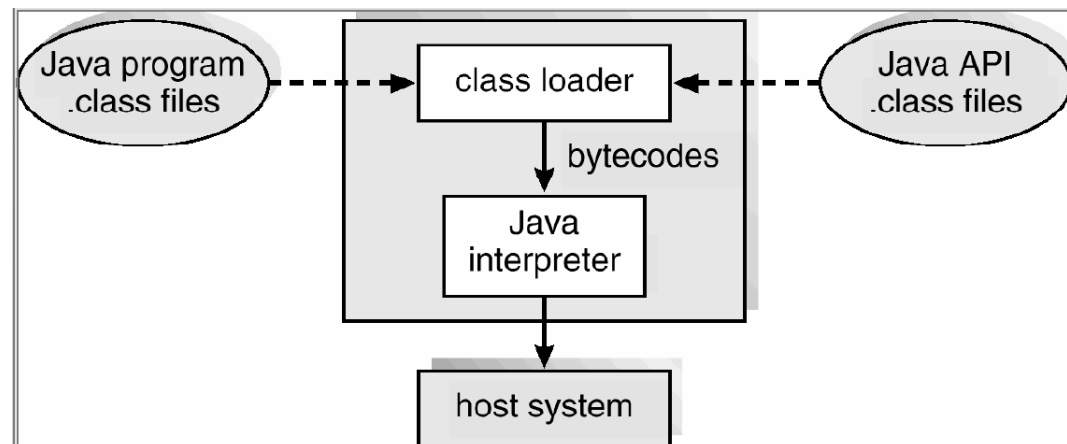
Componentes:

    Especificação da linguagem de programação

    API Java

    JVM: Máquina Virtual Java

JVM:



# JVM

Programas Java compilados são bytecodes independentes de plataforma de execução da JVM

A JVM consiste de:

- Carregador de classes (Class loader)

- Verificador de classes (Class verifier)

- Interpretador runtime (Runtime interpreter)

# Java Threads

Toda aplicação Java tem pelo menos uma thread (sem considerar o system thread)

Do ponto de vista do desenvolvedor, o programa começa com uma thread, chamada de main thread.

A main thread cria novas threads.

## Java Threads (2)

Cada thread é associada com uma instância da classe Thread.

Duas estratégias possíveis para criar uma thread:

- Instanciando a classe Thread;

- Delegando criação/gerência da thread para “executor” (high-level concurrency objects)

## Exemplo1 de criação de Thread em Java pela interface Runnable

- A Classe cria instância de Thread deve fornecer o código a ser executado na *thread* implementando a **interface Runnable**.

Passando um objeto Runnable para o construtor da classe *Thread*.

- A **interface Runnable** deve ser implementada por qualquer classe cujas instâncias sejam executadas por um thread. A classe deve definir um método sem argumentos chamado **run**.



# Exemplo1 de criação de Thread em Java pela interface Runnable

## Código-fonte

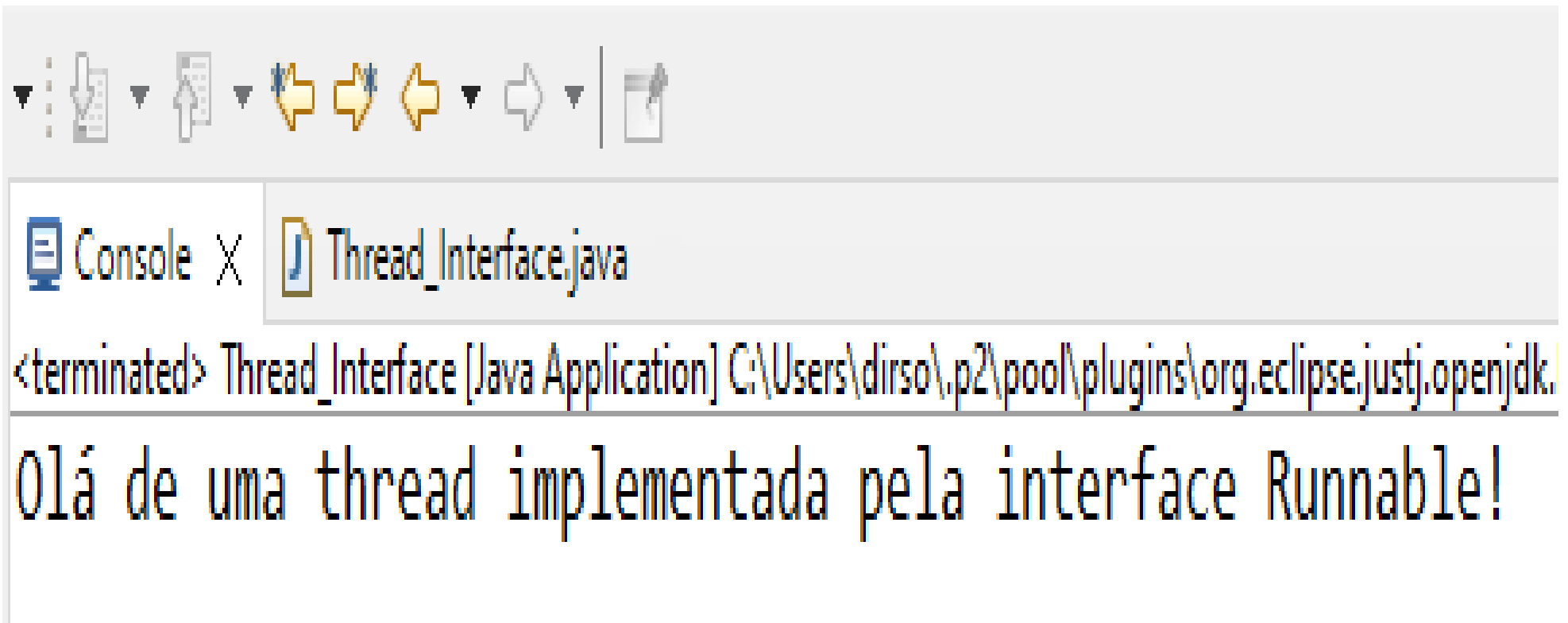
```
package Threads;
/*
 * A aplicação que cria instância de Thread deve fornecer
 * o código a ser executado na thread
 * implementando a classe Runnable
 * Passando um objeto Runnable
 * para o construtor da classe Thread:
 */

public class Thread_Interface implements Runnable {
    public void run() {
        System.out.println("Olá de uma thread implementada pela interface
        Runnable!");
    }

    public static void main(String args[]) {
        (new Thread(new Thread_Interface())).start();
    }
}
```

# Exemplo1 de criação de Thread em Java pela interface Runnable

## Executando o código-fonte



```
<terminated> Thread_Interface [Java Application] C:\Users\dirso\.p2\pool\plugins\org.eclipse.justj.openjdk.  
Olá de uma thread implementada pela interface Runnable!
```

## Exemplo(2) de criação de Thread usando herança em Java

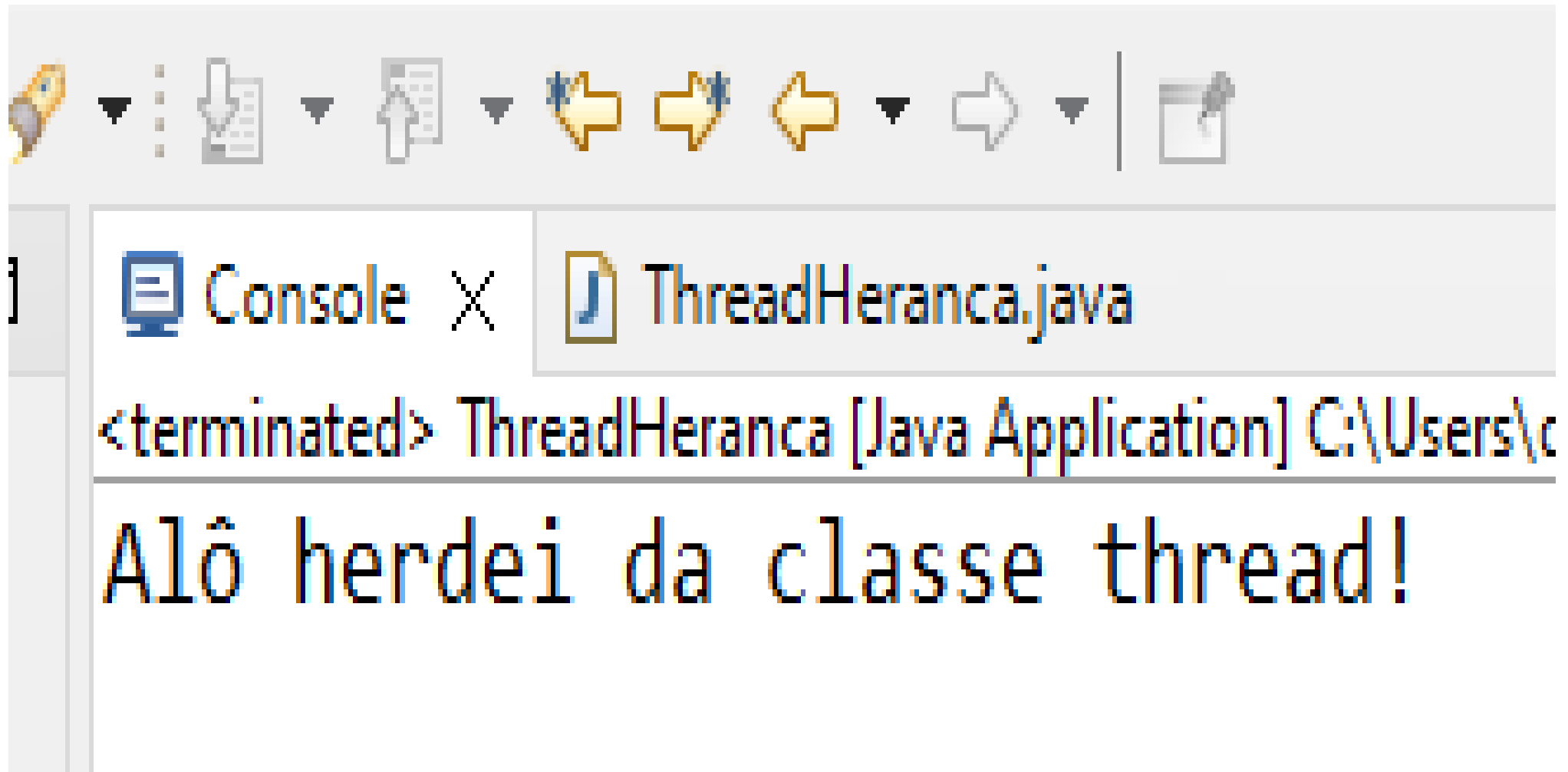
Fazendo subclass de Thread (que também implementa Runnable) usando herança em Java

```
package Threads;
```

```
public class ThreadHeranca extends Thread {  
    public void run() {  
        System.out.println("Alô herdei da classe thread!");  
    }  
}
```

```
public static void main(String args[]) {  
    (new ThreadHeranca()).start();  
}  
}
```

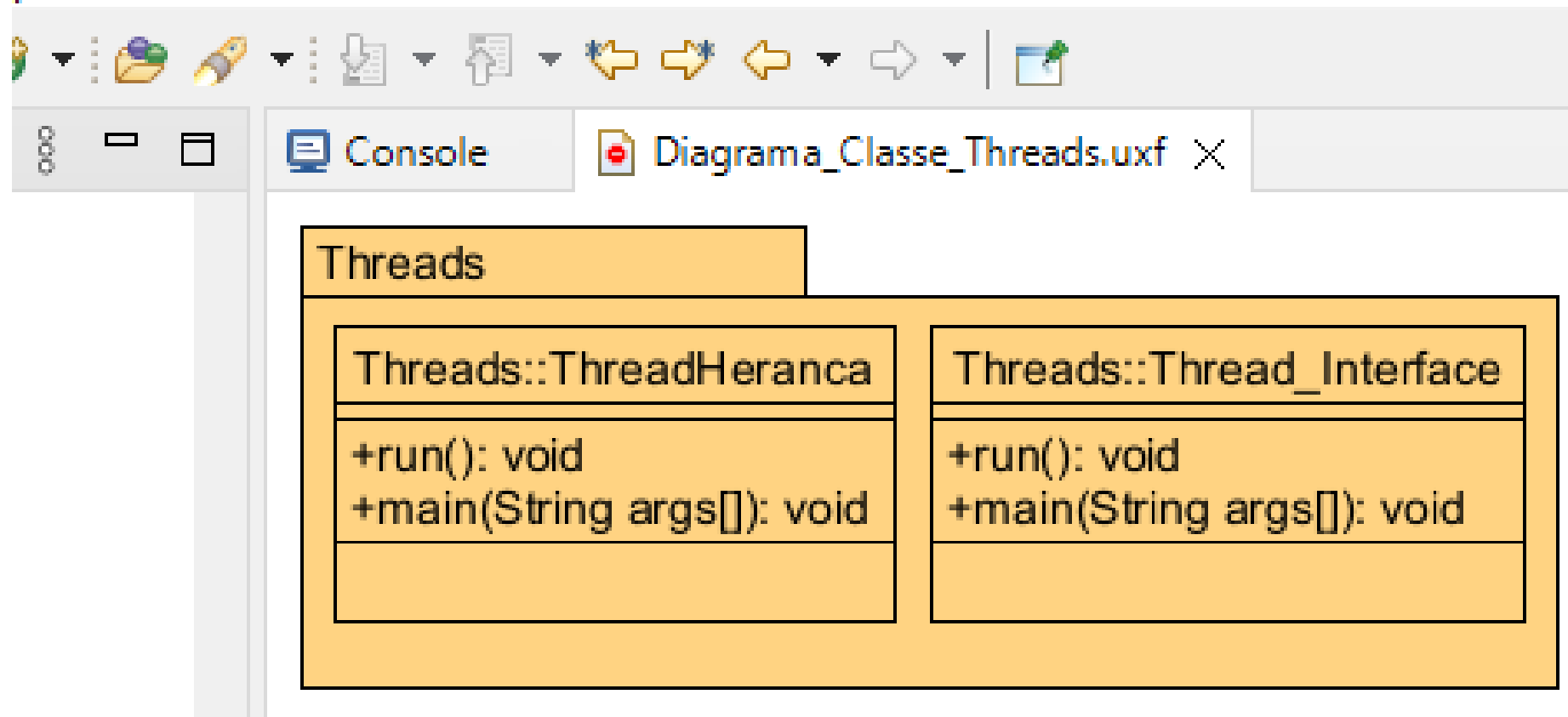
## Exemplo2 de criação de Thread por herança em Java – Rodando o código-fonte



# Criando o Diagrama de Classe UML da Thread do Exemplo 1 e Exemplo 2 usando a UMLet

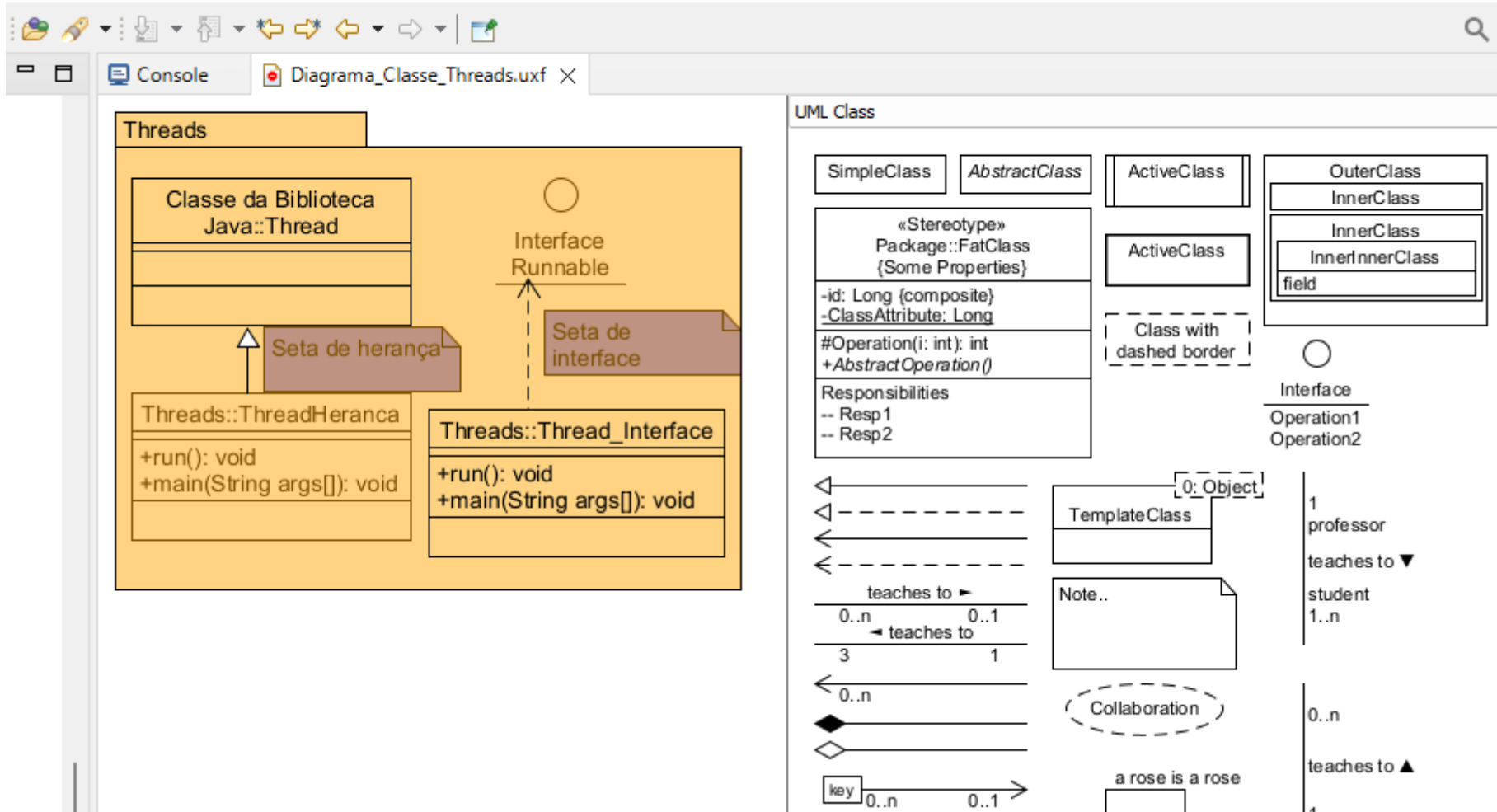
Threads.uxf - Eclipse IDE

p UMLet



**Observação:** A ferramenta UMLet não gerou a classe Thread nem a Interface Runnable por serem respectivamente uma classe e uma interface da linguagem Java e não foram construídas pelo programador.

Criando o Diagrama de Classe UML da Thread do Exemplo 1 e Exemplo 2 usando a UMLet



**Observação:** Criando a classe Thread nem a Interface Runnable usando a Paleta de Ferramenta da UMLet, pois, conforme foi dito elas não são geradas automaticamente pela ferramenta.

# Gerando o Diagrama de Classe UML da Thread do Exemplo 1 e Exemplo 2 usando a UMLet em formato .pdf

Threads.uxf - Eclipse IDE

UMLet

Generate Class Elements from Files or Directory...

Generate Class Element Options...

Zoom to

Export as

Edit Current Palette

Custom Elements

Mail to...

Help

Options...

ThreadHeranca.java

BMP...

EPS...

GIF...

JPG...

PDF...

PNG...

SVG...

Threads::ThreadHeranca

+run(): void  
+main(String args[]): void

Threads::Thread\_Interface

+run(): void  
+main(String args[]): void

# Manipulando Threads em Java

## `Thread.sleep(t)`

Faz com que a thread suspenda a execução por um período.

## `Thread.interrupted()`

Interrompe a execução corrente da thread (para terminar ou para fazer outra coisa)

## `t.join()`

Faz com a thread que esteja executando suspenda para que a thread t execute (até terminar).



# Manipulando Threads em Java

## `Thread.sleep(t)`

Faz com que a thread suspenda a execução por um período.

## `Thread.interrupted()`

Interrompe a execução corrente da thread (para terminar ou para fazer outra coisa)

## `t.join()`

Faz com a thread que esteja executando suspenda para que a thread `t` execute (até terminar).