Lição 9

Threads

Objetivos

Ao final desta lição, o estudante será capaz de:

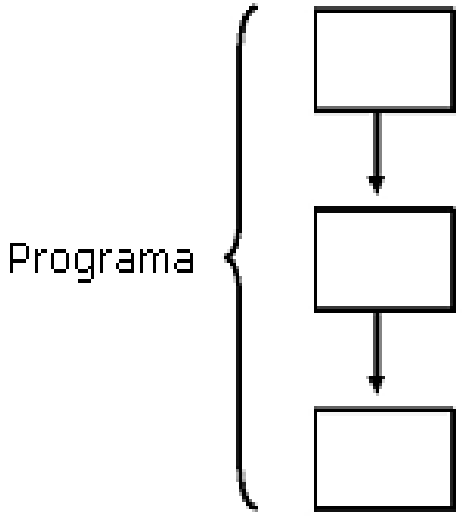
- Definir o que são threads
- Enumerar os diferentes estados de uma thread
- Explicar o conceito de prioridade na thread
- Utilizar os métodos da classe Thread
- Criar suas próprias threads
- Utilizar sincronização para execução de threads concorrentes que sejam independentes umas das outras
- Permitir que as threads se comuniquem umas com as outras concorrentemente em sua execução
- Utilizar as utilidades da concorrência

Threads

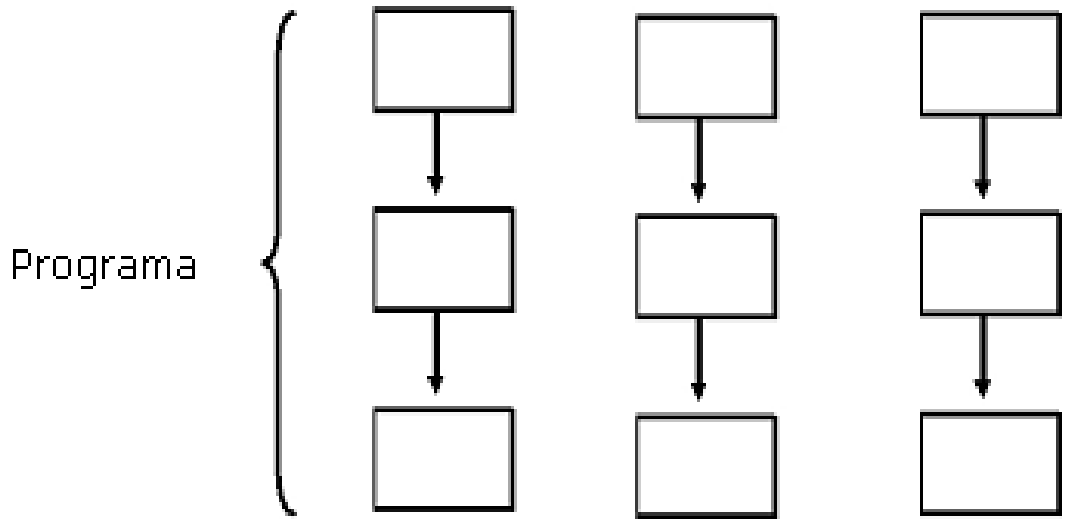
- Fluxo de controle sequencial em um programa
- Pense em *threads* como processos executados por uma classe

Threads

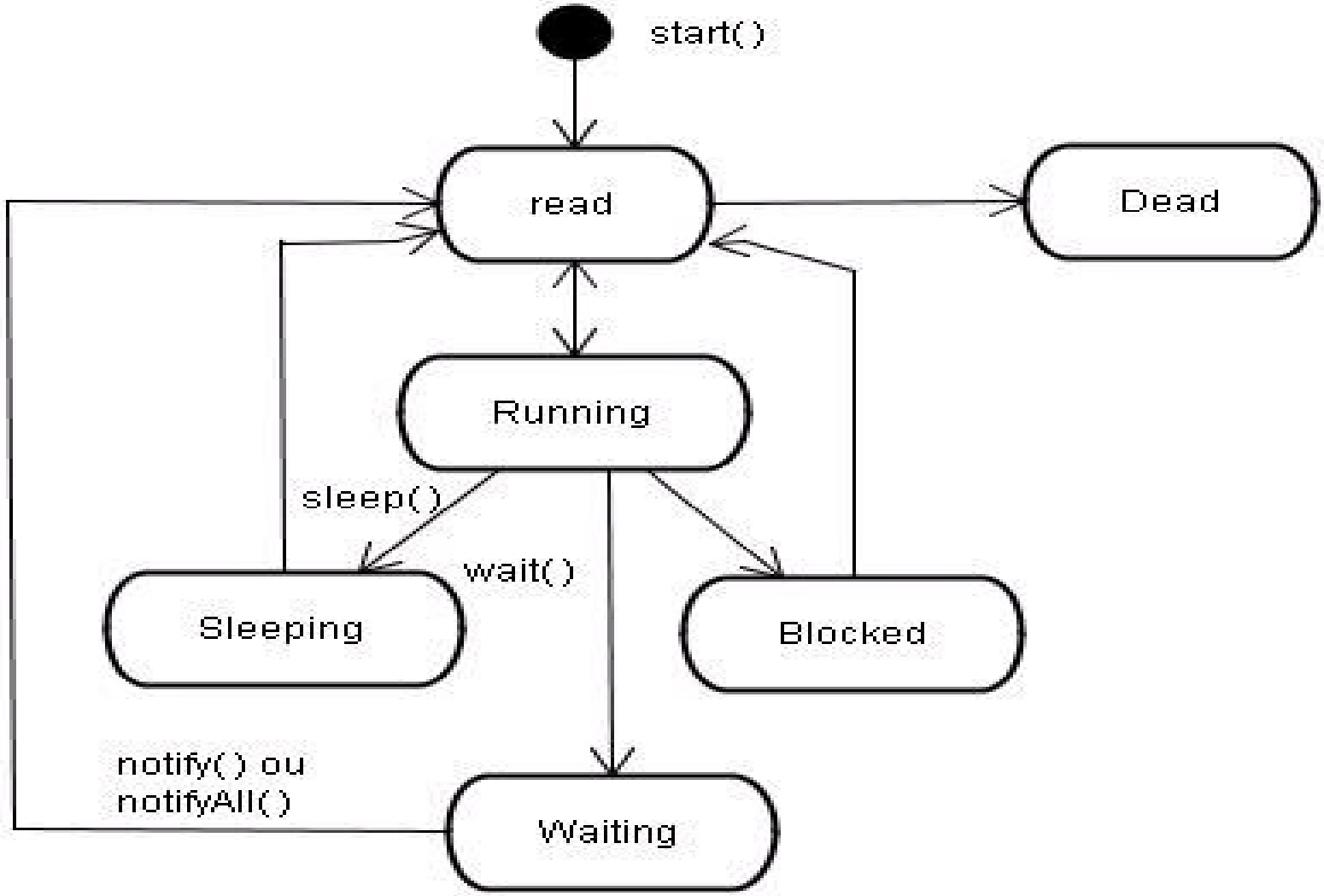
Sequencial



Multi-thread



Estados de *Threads*



Prioridades de *Threads*

- Valor inteiro variando de 1 a 10
- Quanto maior a prioridade do *thread* → Maior a chance de ser executado antes
- Troca de contexto
- Quando mais de um *thread* com a prioridade alta estão prontos

Classe *Thread*: Constantes

```
public final static int MAX_PRIORITY
public final static int MIN_PRIORITY
public final static int NORM_PRIORITY
```

Classe *Thread*: Construtores

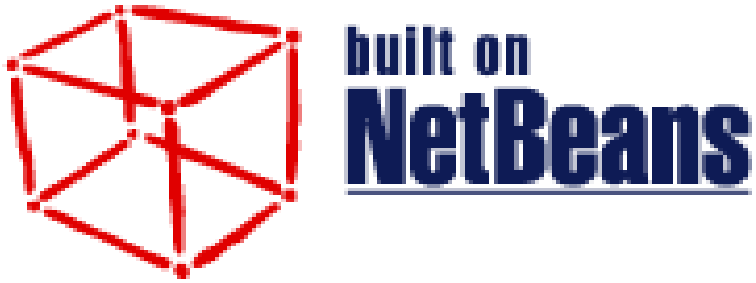
```
Thread()
Thread(String name)
Thread(Runnable target)
Thread(Runnable target, String name)
```

Classe *Thread*: Métodos

```
public static Thread currentThread()
public final String getName()
public final void setName(String name)
public final int getPriority()
public final boolean isAlive()
public final void join([long millis, [int nanos]])
public static void sleep(long millis)
public void run()
public void start()
```

Exemplo de *Thread*

- Passaremos agora para o NetBeans



Criando *Threads*

- Estender a classe *Thread*:

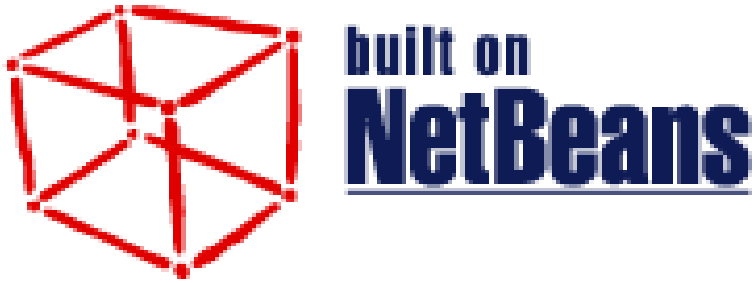
  a classe É UMA thread

- Implementar a interface *Runnable*:

  a classe TEM UMA thread

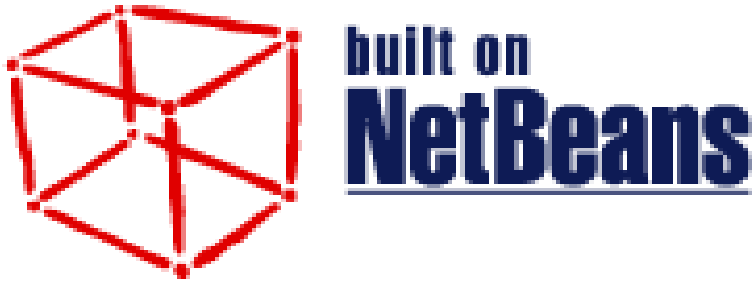
Estendendo a Classe *Thread*

- Passaremos agora para o NetBeans



Implementando a Interface *Runnable*

- Passaremos agora para o NetBeans

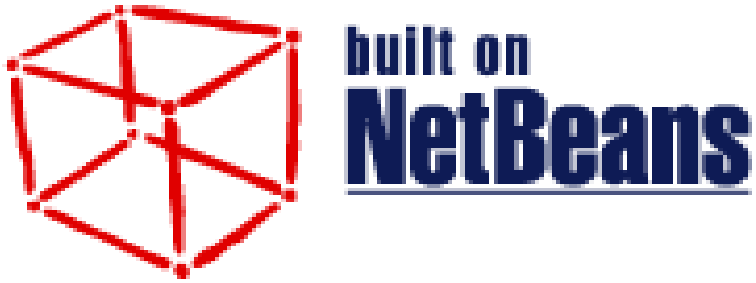


Estender vs. Implementar

- Implementar a interface *Runnable*
- Estender a classe *Thread*
- Escolher entre os dois é uma questão de necessidade

Exemplo: O método *join*

- Passaremos agora para o NetBeans



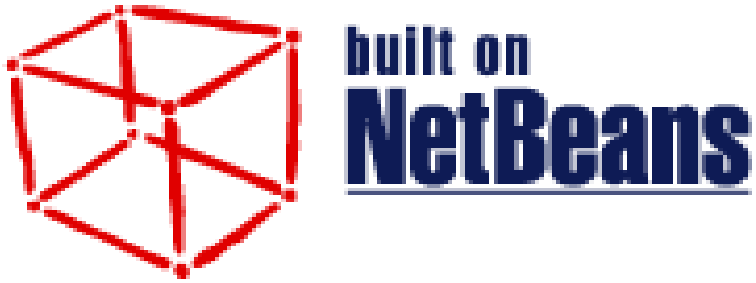
Sincronização

- *Threads* que rodam simultaneamente podem exigir recursos ou métodos externos

- Necessidade de se comunicar com outros *threads* que executam simultaneamente para saber seus *status* e atividades

Um exemplo não-sincronizado

- Passaremos agora para o NetBeans

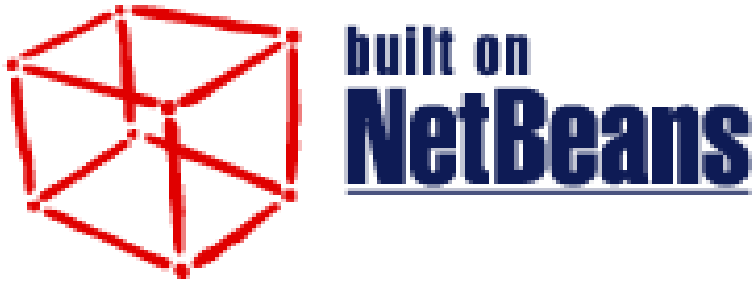


Sincronização: Travando um objeto

- Assegura que apenas um *thread* obtenha acesso a um método específico
- Java permite que você trave objetos com o uso de monitores.

Exemplo Sincronizado

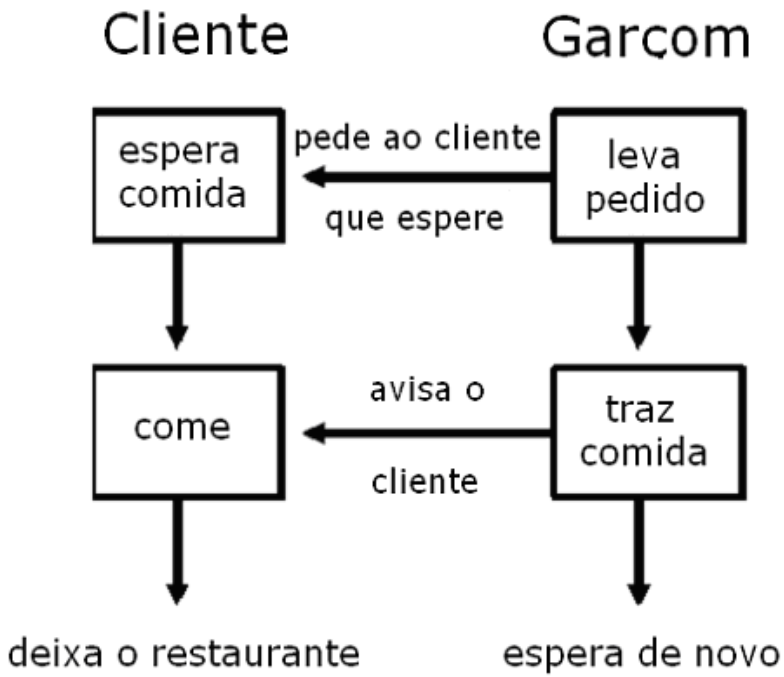
- Passaremos agora para o NetBeans



Comunicação entre *threads*: Métodos

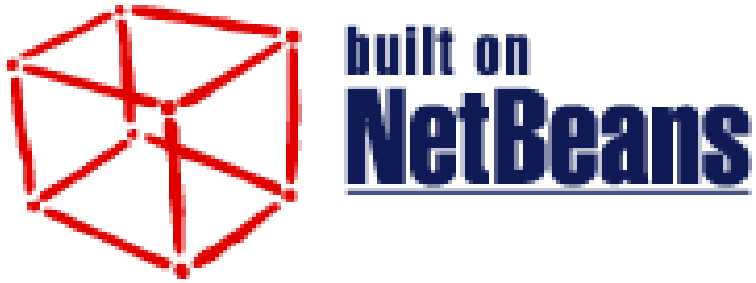
```
public final void wait()
public final void notify()
public final void notifyAll()
```

Comunicação entre *threads*



Exemplo Produtor-Consumidor

- Passaremos agora para o NetBeans



Utilitários de Concorrência

- Introduzido com a J2SE 5.0
- Encontrados no pacote *java.util.concurrent*
- Interface *Executor*
- Interface *Callable*

A Interface *Executor*

- (ANTES) Rodando tarefas executáveis:

```
new Thread(<aRunnableObject>).start();
```

- (DEPOIS) Rodando tarefas executáveis:

```
<anExecutorObject>.execute(<aRunnableObject>);
```

A Interface *Executor*: Problemas

- A criação de Threads é dispendiosa
- Dificuldade no cancelamento e encerramento de threads

A Interface *Executor*

- Desvincula a submissão de tarefas do mecanismo pelo qual cada tarefa executa

- Usando a interface Executor:

```
Executor <executorName> = <anExecutorObject>;
<executorName>.execute(new <RunnableTask1>());
<executorName>.execute(new <RunnableTask2>());
...
```

A Interface *Executor*: Criando objetos

- Não pode ser instanciado
- Pode-se criar uma classe que implemente esta interface
- Podem-se usar métodos fábrica fornecidos pela classe *Executors*
- A classe Executor também fornece métodos fábrica para gerenciamento de pool de *Threads*

Métodos Fábrica da Classe *Executors*

```
public static ExecutorService
  newCachedThreadPool()
public static ExecutorService
  newFixedThreadPool(int nThreads)
public static ScheduledExecutorService
  newScheduledThreadPool(int corePoolSize)
public static ExecutorService
  newSingleThreadExecutor()
public static ScheduledExecutorService
  newSingleThreadScheduledExecutor()
```

A Interface *Executor*

- Controla a execução e a conclusão de tarefas *Runnable*

- Matando *threads*:

```
executor.shutdown();
```

A Interface *Callable*

- É a interface *Runnable* sem seus inconvenientes
- O método *call*

```
V call throws Exception
```

Sumário

- *Threads*
  - Definição, Estados, Prioridades, Construtor, Constantes e Métodos
- Criando *threads*
  - Estendendo a classe *Thread* e implementando a interface *Runnable*
- Sincronização
  - Travando um objeto
  - A palavra-chave *synchronized*
- *Comunicação Interthread*
- Utilitários de Concorrência
  - Interface *Executor* e Interface *Callable*

Parceiros

- **Os seguintes parceiros tornaram JEDI™ possível em Língua Portuguesa:**