## Primeiro Exemplo Básico de Programação com Threads em Java

Mas, antes, analise o exemplo Deitel-Cap 15-Ed.4, mais simples, sobre a estruturação de threads, executando adormecimento e despertar de threads, usando herança direta da classe Thread. Os arquivos abaixo contém texto e código explicativo.

**Tarefa Inicial de Programação - Estruturação de Threads 15.4 Ed 4**

Neste exemplo inicial, a classe **Thread** serve para criar novas classes que suportam *multithreading*. O método run() da classe Thread deve ser sobrescrito para programar as tarefas que serão realizadas concorrentemente.

Porém, se precisamos suportar *multithreading* em uma classe que já é herdada (derivada) de uma classe diferente da classe **Thread**, a estruturação de threads é feita implementando-se a interface **Runnable** nessa classe, porque Java não permite que se herde mais de uma classe ao mesmo tempo. A própria classe Thread implementa **Runnable (**interface no pacote **(java.lang), como no exemplo:**

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

## Segundo Exemplo Básico de Programação com Threads em Java

**Implementação de um Thread** (classe Thread e Runnable)

Analisar um exemplo, sobre a estruturação de threads, executando o adormecimento e despertar de threads, usando a interface **Runnable**.

**Tarefa Inicial de Programação - Estruturação de Threads com Runnable. 23.4 Ed 6**

Cap. 23-Deitel: Java Como Programar, Ed.6
Ciclo de Vida de uma Thread, Métodos executados por threads. Prioridades de Threads
Veja em Capitulo-23-Deitel-Java-Como-Programar-6-Edicao.pdf

Material de Apoio de outros links sobre threads:

O que é um thread?
O ciclo de vida de um thread  Estados e Métodos Básicos de uma Thread

## Exemplos de Threads de Temporização: Uso das classes Timer e TimerTask

## Exemplo com Escalonamento de Threads

## Criando um *pool* de número fixado de threads:

Um *pool* de *threads* é uma coleção de threads que podem ser fixadas, que são escalonadas pelo processador, para executarem tarefas (programadas em threads) que requisitam execução**.**

Para um grande número de threads, um *pools* de *threads*, geralmente provêem:

- Um programa com muitas threads pode sofrer em desempenho.
- Melhor performance quando se executam um grande número de threads.

Para usar pools de threads, instancie uma implementação da interface ExecutorService e entregue tarefas para execução.

Essa implementação da interface **ExecutorService** permite que você estabeleça entre outras coisas:

- O número básico e máximo do pool (número de threads).
- Como criar e terminar threads.
- Como tratar tarefas rejeitadas, isto é, sem poderem ser executadas porque todas as threads no pool estão em uso. Pode-se usar um número fixo de threads e com menos threads do que tarefas.

O método **`newFixedThreadPool(int)`** da classe **Executors,** cria um *pool* com número fixo de threads e fila ilimitada de tarefas.

No exemplo 23-5, threads são utilizadas por um objeto **threadExecutor** da interface **ExecutorService.** Desta forma os *Runnables* que implementam as threads, são executados.

No caso de haver um número fixo de threads, com menos threads do que tarefas, se o método **Execute** for chamado e todas as threads em **ExecutorService** (interface para gerenciar threads) estiverem em uso, **a tarefa (**implementada em Runnable) será colocada numa fila e atribuído à primeira thread que completar sua tarefa.

**Interfaces e Classes usadas:**

interface Runnable,
interface Executor,
interface ExecutorService, para gerenciar threads em um pool de threads.
classe Executors (com *s* no final é uma classe).

Cap. 23-Deitel: Java Como Programar, Ed.6
Ciclo de Vida de uma Thread, Métodos executados por threads. Prioridades de Threads
Veja em Capitulo-23-Deitel-Java-Como-Programar-6-Edicao.pdf

Você pode ver os seguintes links: **Pool de Threads**

**Java Threading: the Executor Framework**
(Um melhor mecanismo para executar threads em Java.**)**

**Sincronização de Threads**

## Threads no Nível de Usuário (onde você está programando)

## Inanição (Starvation)

# Tarefa Inicial de Programação - Produtor-Consumidor com Monitor

## Sincronização de Threads com Monitor

Livro "Java Como Programar", **Edição 6**, Deitel & Deitel - **Cap. 23** - Multithreading
Download de códigos do livro Deitel, Java: Como Programar Ed. 6:

http://www.inf.ufsc.br/~bosco/downloads/Java 6 CD/ch23/fig23_6_10/

## Produtor-Consumidor sem Sincronização

Você deve procurar os arquivos **fig23 6-10**, onde **23.6** (Interface *Buffer*), **23.7** (class *Producer*) e **23.8** (class *Consumer*), que são os exemplos da **seção 23.6** (**Relacionamento entre produtor e consumidor sem sincronização**).

Analise primeiro, os códigos em **23.9** e **23.10**, que tratam da implementação de um buffer não-sincronizado *UnsynchronizedBuffer* e do aplicativo de teste *SharedBufferTest.*

Lembre-se que a corretude do programa é que a thread *Producer* execute primeiro e que cada valor produzido por *Producer* seja consumido exatamente uma vez pelo *Consumer*.

No código 23.10, a implementação *SharedBufferTest*, do aplicativo produtor-consumidor mostra duas threads que manipulam o buffer não-sincronizado *UnsynchronizedBuffer*.

Veja o resultado de sua execução !!!

O que você nota ???
---

Analise, agora, o relacionamento entre **Produtor e Consumidor com Sincronização**, veja agora a seção 23.12 sobre **Monitores em Java**.

Novamente você deve aproveitar os códigos **23.6** do *Buffer* e as classes *Producer* (em **23.7**) e *Consumer* (em **23.8**) do exemplo da seção 23.6.

http://www.inf.ufsc.br/~bosco/downloads/Java 6 CD/ch23/fig23_19_20/

Veja, agora, os códigos em fig23 19-20.

O código que realiza a sincronização é colocado nos métodos *set* e *get* da classe *SynchronizedBuffer* (Fig. **23.19**), que implementa a interface Buffer. A explicação do exemplo é importante para o aprendizado sobre monitores. O código **23.20** *SharedBufferTest2* mostra um aplicativo produto-consumidor que utiliza um buffer sincronizado.

Veja o resultado de sua execução.

O que você nota, agora ???

**Leitores e Escritores** **Leitores e Escritores** (Livro Principles of Concurrent and Distributed Programming, M. Ben Ari)

**Escalonamento de Threads em Java 1**

Livro - Programação com Java: Uma Introdução Abrangente (Ver Capítulo 27, página 1013, Os Utilitários de Concorrência, Usando um Executor) - Hebert Schildt & Dale Skrien