



# Threads

## **T**hreads (encadeamento) em Java

Um dos recursos mais atraentes em Java é o suporte para fácil programação de threads. Java fornece suporte interno para programação multithread. Um programa multithread contém duas ou mais partes que podem ser executadas simultaneamente. Cada parte deste programa é chamada de thread, e cada thread define um caminho de execução separado. Assim, podemos dizer que multithreading é uma forma especializada de multitarefa.

A definição formal de um encadeamento é: Um encadeamento é uma unidade de processamento básica à qual um sistema operacional aloca tempo de processador e mais de um encadeamento pode estar executando código dentro de um processo. Às vezes, um encadeamento é chamado de processo leve ou contexto de execução.

Imagine um aplicativo de reserva de ingressos on-line com muitos recursos complexos. Uma de suas funções é “procurar passagens de trem / voo da origem e do destino”, outra é “verificar preços e disponibilidade” e uma terceira operação demorada é “reservar passagens para vários clientes ao mesmo tempo”.

Em um ambiente de tempo de execução de thread único, essas ações são executadas uma após a outra. A próxima ação pode acontecer apenas quando a anterior for concluída. Se a reserva de um bilhete demorar 10 minutos, outros usuários terão que aguardar a operação de pesquisa ou reserva. Esse tipo de aplicativo resultará em perda de tempo e clientes. Para evitar esse tipo de problema, o java fornece recursos de multithreading onde várias operações podem ocorrer simultaneamente e respostas mais rápidas podem ser obtidas para uma melhor experiência do usuário. O multithreading permite gravar programas muito eficientes

que fazem o máximo uso da CPU, porque o tempo ocioso pode ser reduzido ao mínimo.



## Definindo uma thread

No sentido mais geral, você cria um encadeamento instanciando um objeto do tipo Thread.

Java define duas maneiras pelas quais isso pode ser realizado:

## Implementando Runnable

A maneira mais fácil de criar um encadeamento é criar uma classe que implemente a interface Runnable. Runnable abstrai uma unidade de código executável. Você pode construir uma thread em qualquer objeto que implementa Runnable. Para implementar o Runnable, uma classe precisa apenas implementar um único método chamado run (), que é declarado assim:

```
public void run()
```

Dentro de run (), você definirá o código que constitui o novo thread. É importante entender que run () pode chamar outros métodos, usar outras classes e declarar variáveis, assim como o thread principal. A única diferença é que run () estabelece o ponto de entrada para outro encadeamento simultâneo de execução dentro do seu programa. Esse encadeamento terminará quando o run () retornar.

```
class MinhaRunnableThread implements Runnable {
```

```
    public void run() {
```

```
        System.out.println("Trabalho importante em execução em  
MinhaRunnableThread");
```



```
    }
```

```
}
```

## Estendendo java.lang.Thread

A maneira mais simples de definir o código a ser executado em um encadeamento separado é

A estrutura é:

```
class MinhaThread extends Thread {  
  
    public void run() {  
  
        System.out.println("Trabalho importante sendo executado em  
MinhaThread");  
  
    }  
  
}
```

A limitação dessa abordagem (além de ser uma má escolha de design na maioria dos casos) é que, se você estender o Thread, não poderá estender mais nada. E não é como se você realmente precisasse desse comportamento herdado da classe Thread, pois para usar uma thread, será necessário instanciar um de qualquer maneira.



## Instanciando uma thread

Lembre-se, todo encadeamento de execução começa como uma instância da classe Thread. Independentemente de seu método run () estar em uma subclasse Thread ou em uma classe de implementação Runnable, você ainda precisa de um objeto Thread para fazer o trabalho.

Se você tiver a abordagem dois (estendendo a classe Thread): a instanciação seria simples

```
MinhaThread thread = new MinhaThread();
```

Se você implementar o Runnable, a instanciação será apenas um pouco menos simples.

Para instanciar sua classe Runnable:

```
MinhaRunnableThread MinhaRunnable = new MinhaRunnableThread ();
```

```
Thread thread = new Thread(MinhaRunnable); // Passe seu Runnable para o Thread
```

Atribuir o mesmo destino a vários encadeamentos significa que vários encadeamentos de execução estarão executando o mesmo trabalho (e que o mesmo trabalho será realizado várias vezes).

## Construtores da classe Thread

Construtor padrão - para criar encadeamento com nome e prioridade padrão

Esse construtor criará um encadeamento a partir do objeto executável.



Esse construtor criará um encadeamento a partir do objeto executável com o nome conforme passado no segundo argumento

Este construtor criará um encadeamento com o nome conforme o argumento passado.

Então agora criamos uma instância Thread e ela sabe qual método run () chamar. Mas nada está acontecendo ainda. Neste ponto, tudo o que temos é um objeto Java simples e antigo do tipo Thread. Ainda não é um encadeamento de execução. Para obter um encadeamento real - uma nova pilha de chamadas - ainda precisamos iniciar o encadeamento.

## Iniciando uma thread

Você criou um objeto Thread e ele conhece seu destino (o pass-inRunnable ou ele mesmo se você estendeu a classe Thread). Agora é hora de fazer tudo acontecer - lançar uma nova pilha de chamadas. É tão simples que dificilmente merece sua própria subposição:

```
t.start ();
```

Antes de chamar Start () em uma instância da thread, diz-se que o thread está no novo estado. Existem vários estados de threads que serão estudados mais na frente.

Quando chamamos o método t.start (), acontece o seguinte:

O exemplo a seguir demonstra o que abordamos até agora - definindo, instanciando e iniciando um encadeamento: no programa Java abaixo, não estamos implementando a comunicação ou a sincronização do encadeamento, pois essa

saída pode depender do mecanismo de agendamento do sistema operacional e da versão do JDK.



Estamos criando dois threads t1 e t2 do objeto de classe MinhaRunnable. Iniciando os dois segmentos, cada segmento está imprimindo o nome do segmento no loop.

## Código Java (MinhaRunnable.java)

```
package br.com.java.aula;
```

```
public class MinhaRunnable implements Runnable{
```

```
    @Override
```

```
    public void run() {
```

```
        for(int x =1; x < 10; x++) {
```

```
            System.out.println("MinhaRunnable executando para o  
Nome do Tópico: " + Thread.currentThread().getName());
```

```
        }
```

```
    }
```

```
}
```

# Código Java (TesteMinhaRunnable.java)



```
package br.com.java.aula;
```

```
public class TesteMinhaRunnable {
```

```
    public static void main (String [] args) {
```

```
        MinhaRunnable Minharunnable = new MinhaRunnable();
```

```
        //Passando o objeto Minharunnable para o construtor da classe
```

```
Thread
```

```
        Thread t1 = new Thread(Minharunnable);
```

```
        t1.setName("Teste-1 Thread");
```

```
        //Tópico inicial t1
```

```
        t1.start();
```

```
        Thread t2 = new Thread(Minharunnable);
```

```
        t2.setName("Teste-2 Thread");
```

```
        t2.start();
```

```
    }
```

```
}
```

- Você pode implementar a interface Runnable.
- Você pode estender a classe Thread




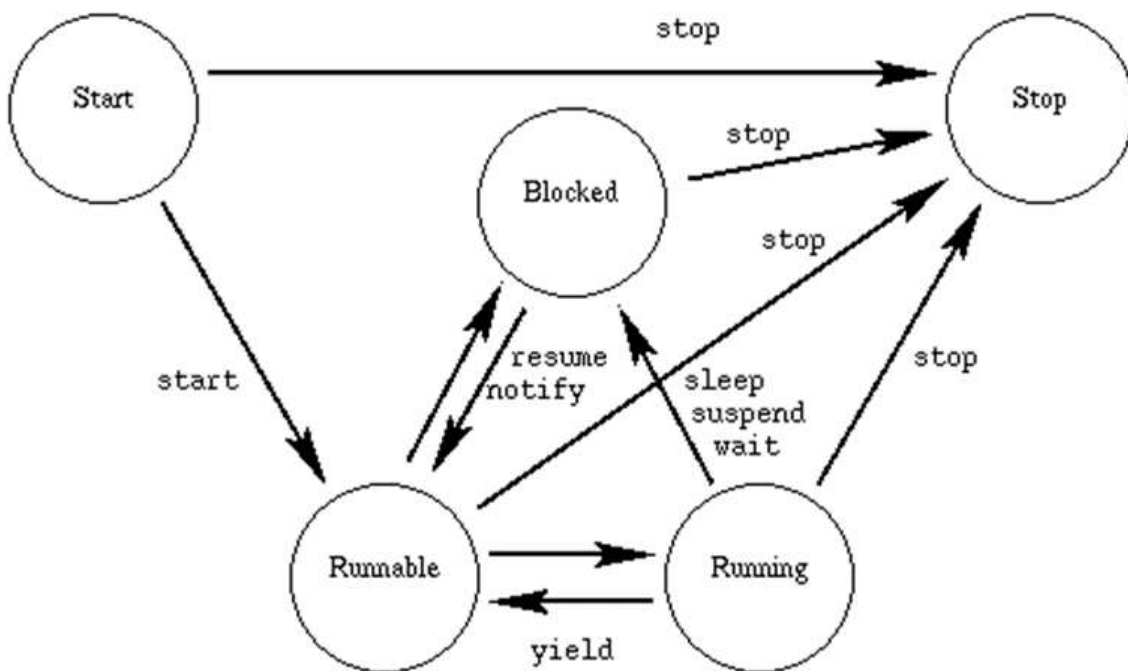




Temos vários métodos que podem ser chamados no objeto de classe Thread. Esses métodos são muito úteis ao escrever um aplicativo multithread. A classe Thread tem os seguintes métodos importantes.

Assinatura do método	Descrição
String getName()	Recupera o nome da thread em execução no contexto atual no formato String
void start()	Este método iniciará um novo encadeamento de execução chamando o método run () do objeto Thread / executável.
void run()	Este método é o ponto de entrada do encadeamento. A execução do encadeamento começa neste método.
void sleep(int sleeptime)	Este método suspende o encadeamento pelo tempo mencionado no argumento (tempo de espera em ms)
void yield()	Invocando esse método, o thread atual pausa sua execução temporariamente e permite que outros threads sejam executados.
void join()	Este método é usado para enfileirar um encadeamento em execução. Uma vez chamado no thread, o thread atual aguardará até que o thread de chamada conclua sua execução
boolean isAlive()	Este método verifica se o encadeamento está vivo ou morto


O trabalho do agendador de thread é mover os encadeamentos para dentro e para fora do estado de execução. Enquanto o planejador de encadeamentos  $\mu$   mover um encadeamento do estado de execução de volta para executável, outros fatores podem fazer com que um encadeamento saia da execução, mas não de volta ao executável. Uma delas é quando o método `run ()` da thread é concluído. Nesse caso, o thread passa do estado de execução diretamente para o estado morto.



New/Start:

Este é o estado em que o encadeamento se encontra após a criação da instância Thread, mas o método `start ()` não foi invocado no encadeamento. É um objeto Thread ativo, mas ainda não é uma thread de execução. Neste ponto, o encadeamento é considerado não ativo.

Runnable

Isso significa que um encadeamento pode ser executado quando o mecanismo de divisão do tempo tiver ciclos de CPU disponíveis para o encadeamento. Portanto,  encadeamento pode ou não estar em execução a qualquer momento, mas não há nada para impedir sua execução, se o planejador puder organizá-lo. Ou seja, não está morto ou bloqueado.



### Running

Esse estado é um estado importante onde está a ação. Esse é o estado em que o encadeamento está quando o agendador de encadeamentos o seleciona (no pool executável) para ser o processo em execução no momento. Um encadeamento pode sair do estado de execução por vários motivos, inclusive porque “o planejador de encadeamentos se sentiu assim”. Existem várias maneiras de chegar ao estado executável, mas apenas uma maneira de chegar ao estado de execução: o planejador escolhe um encadeamento do pool de encadeamentos executável.



### Blocked

O encadeamento pode ser executado, mas algo o impede. Enquanto um encadeamento estiver no estado bloqueado, o agendador simplesmente o ignorará e não fornecerá tempo de CPU. Até que um encadeamento reinsira o estado executável, ele não executará nenhuma operação. O estado bloqueado possui alguns sub-estados, como abaixo,



**Bloqueado na E / S:** O encadeamento aguarda a conclusão da operação de bloqueio. Um encadeamento pode entrar nesse estado devido à espera do recurso de E / S. Nesse caso, o encadeamento retorna ao estado executável após a disponibilidade de recursos.



**Bloqueado para conclusão da junção:** o encadeamento pode entrar nesse estado devido à espera pela conclusão de outro encadeamento.



- **Bloqueado para aquisição de bloqueio:** o encadeamento pode entrar nesse estado devido à espera pela aquisição do bloqueio de um objeto.

- Dead

Um encadeamento no estado morto ou finalizado não é mais agendável e não receberá nenhum tempo de CPU. Sua tarefa está concluída e não pode mais ser executada. Uma maneira de uma tarefa morrer é retornando de seu método run (), mas o encadeamento de uma tarefa também pode ser interrompido, como você verá em breve.

Vamos dar um exemplo de programa Java para demonstrar vários estados e métodos da classe de threads.

Código Java (AnimalRunnable.java)

```
package br.com.java.aula;
```

```
public class AnimalRunnable implements Runnable {
```

```
    @Override
```

```
    public void run() {
```

```
        for (int x = 1; x < 4; x++) {
```

```
            System.out.println("Executado por " +
```

```
Thread.currentThread().getName());
```

```
try {
```



```
    Thread.sleep(1000);
```

```
    } Catch (InterruptedException ex) {
```

```
        ex.printStackTrace();
```

```
    }
```

```
}
```

```
        System.out.println("Estado da Thread: "+  
Thread.currentThread().getName()+ " - "+Thread.currentThread().getState());
```

```
    System.out.println("Saída da Thread: "
```

```
        + Thread.currentThread().getName());
```

```
    }
```

```
}
```

Código Java (Animal/MultiThreadDemo.java): [Vá para o editor](#)

```
public class AnimalMultiThreadDemo {
```

```
    public static void main(String[] args) throws Exception{
```

```
        // Tornar objeto Runnable
```

```
        AnimalRunnable anr = new AnimalRunnable();
```

```
        Thread Gato = new Thread(anr);
```

```
        Gato.setName("Gato");
```

```
Thread Cachorro = new Thread(anr);
```



```
Cachorro.setName("Cachorro");
```

```
Thread Vaca = new Thread(anr);
```

```
Vaca.setName("Vaca");
```

```
System.out.println("Estado da thread Gato antes de iniciar:  
"+Gato.getState());
```

```
Gato.start();
```

```
Cachorro.start();
```

```
Vaca.start();
```

```
System.out.println("Estado da thread Gato no método principal  
antes de dormir: " + Gato.getState());
```

```
System.out.println("Estado da thread Cachorro no método  
principal antes de dormir: " + Cachorro.getState());
```

```
System.out.println("Estado da thread Vaca no método principal  
antes de dormir: " + Vaca.getState());
```

```
Thread.sleep(10000);
```

```
System.out.println("Estado da thread Gato no método principal  
antes de dormir: " + Gato.getState());
```

```
System.out.println("Estado da thread Cachorro no método  
principal antes de dormir: " + Cachorro.getState());
```

```
System.out.println("Estado da thread Vaca no método principal  
antes de dormir: " + Vaca.getState());
```

```
}
```



```
}
```

```
class AnimalRunnable implements Runnable {
```

```
    @Override
```

```
    public void run() {
```

```
        for (int x = 1; x < 4; x++) {
```

```
            System.out.println("Executado por " +  
Thread.currentThread().getName());
```

```
            try {
```

```
                Thread.sleep(1000);
```

```
            } Catch (InterruptedException ex) {
```

```
                ex.printStackTrace();
```

```
            }
```

```
        }
```

```
        System.out.println("Estado da thread: "+  
Thread.currentThread().getName()+" - "+Thread.currentThread().getState());
```

```
        System.out.println("Saída de Thread: "
```

```
            + Thread.currentThread().getName());
```

```
    }
```

```
}
```



## Resultado:

```
Estado da thread Gato antes de iniciar o chamado: NEW
Executado por Gato
Executado por Cachorro
Estado da thread Gato método principal antes de dormir: TIMED_WAITING
Executado por Vaca
Estado da thread Cachorro método principal antes de dormir:
TIMED_WAITING
Estado da thread Vaca método principal antes de dormir: TIMED_WAITING
Executado por Gato
Executado por Vaca
Executado por Cachorro
Executado por Gato
Executado por Cachorro
Executado por Vaca
Estado da thread: Gato - RUNNABLE
Saída de Thread: Gato
Estado da thread: Vaca - RUNNABLE
Estado da thread: Cachorro - RUNNABLE
Saída de Thread: Vaca
Saída de Thread: Cachorro
Estado da thread Gato método principal antes de dormir: TERMINATED
Estado da thread Cachorro método principal antes de dormir: TERMINATED
Estado da thread Vaca método principal antes de dormir: TERMINATED
```

## Atividade extra

Vídeo: “Entendendo o que é Threads”

Link: <https://www.youtube.com/watch?v=T4pylf3OxpU>

## Referências Bibliográficas

BARNES, D. J.; KOLLING, M. **Programação orientada a objetos com java: uma introdução prática usando o bluej**. 4.ed. Pearson: 2009.



FELIX, R. (Org.). **Programação orientada a objetos**. Pearson: 2017.



MEDEIROS, L. F. de. **Banco de dados: princípios e prática**. Intersaberes: 2013;

ORACLE. Java Documentation, 2021. **Documentação oficial da plataforma Java**. Disponível em: <<https://docs.oracle.com/en/java/>>.

**Ir para exercício**