**Thread em JAVA**

**O que é uma Thread?**

Um **thread** é uma **linha de execução** dentro de um programa.  
Cada **thread** representa um fluxo independente de execução, permitindo que várias tarefas sejam realizadas **simultaneamente** (ou quase, **dependendo do processador**).

**Resumindo**:

Um **programa Java** sempre possui **pelo menos um thread**, chamada **main thread**, que executa o **método main().**

Com **threads**, podemos realizar **tarefas concorrentes**, como:

* Fazer download de arquivos enquanto atualiza uma barra de progresso.
* Executar cálculos em paralelo.
* Manter interfaces gráficas responsivas.

**Formas de criar uma Thread em Java**

Existem **duas formas principais**:

**1. Herdando da classe Thread // Package Thread**

class **MinhaThread** **extends** **Thread** {

@Override

**public void run() {**

for (int i = 1; i <= 5; i++) {

System.out.println(getName() + " executando: " + i);

try {

**Thread**.sleep(500); // pausa de 0.5 segundo

} catch (InterruptedException e) {

e.**printStackTrace**();

}

}

}

}

public class **ExemploThread1** {

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

**MinhaThread** t1 = **new** MinhaThread();

**MinhaThread** t2 = **new** MinhaThread();

t1.setName("Thread A");

t2.setName("Thread B");

t1.**start**(); // inicia a execução da **thread** A

t2.**start**(); // inicia a execução da **thread** B

}

}

**Explicação:**

* **start()** inicia a execução da **thread** (nunca chame **run()** diretamente).
* **sleep(ms)** pausa a execução por alguns milissegundos.
* O método **run()** contém o código que será executado em paralelo.

**2. Implementando a interface Runnable**

**class Tarefa implements Runnable {**

**@Override**

**public void run() {**

for (int i = 1; i <= 5; i++) {

System.out.println(Thread.currentThread().getName() + " executando: " + i);

**try** {

**Thread**.sleep(400);

} **catch** (InterruptedException e) {

e.**printStackTrace**();

}

}

}

}

**public class ExemploThread2 {**

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

**Thread** t1 = **new** **Thread**(**new** Tarefa(), "Processo 1");

**Thread** t2 = **new** **Thread**(**new** Tarefa(), "Processo 2");

t1.**start()**;

t2.**start()**;

}

}

**Vantagem:**

Permite que sua classe **herde de outra** (**já que o Java não suporta herança múltipla**).

Além disso, é a forma **mais recomendada** atualmente.

**Conceitos importantes**

| **Método / Conceito** | **Descrição** |
| --- | --- |
| **start()** | Inicia a execução da thread. |
| **run()** | Código executado pela thread. |
| **sleep(ms)** | Faz a thread “dormir” por alguns milissegundos. |
| **join()** | Faz uma thread esperar outra terminar. |
| **isAlive()** | Verifica se a thread ainda está em execução. |
| **synchronized** | Garante que apenas uma thread por vez acesse um recurso crítico. |

**Mini Exercício**

Criar um programa que simule **duas tarefas** sendo executadas em paralelo:

1. Uma **thread** imprime números pares de 0 a 10.
2. Outra **thread** imprime números ímpares de 1 a 9.

**Cada thread** deve pausar 300 ms entre as impressões.

**Solução esperada:**

**class NumerosPares implements Runnable {**

**@Override**

**public void run() {**

for (int i = 0; i <= 10; i += 2) {

System.out.println("Pares: " + i);

**try** {

**Thread**.sleep(300);

} **catch** (InterruptedException e) {

e.**printStackTrace**();

}

}

}

}

**class NumerosImpares implements Runnable {**

**@Override**

**public void run() {**

for (int i = 1; i < 10; i += 2) {

System.out.println("Ímpares: " + i);

**try** {

**Thread**.sleep(300);

} **catch** (InterruptedException e) {

e.**printStackTrace**();

}

}

}

}

**public class ExercicioThreads {**

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

**Thread** t1 = **new** **Thread**(**new** NumerosPares());

**Thread** t2 = **new** **Thread**(**new** NumerosImpares());

t1.**start();**

t2.**start();**

}

}

**Discussão**

* Note que a **ordem de execução** **pode variar a cada execução**.
* Isso ocorre porque as **threads** são **agendadas pelo sistema operacional**, e não temos controle direto da ordem de execução.

**Conclusão**

* **Threads** permitem **execução concorrente** de tarefas.
* Existem duas formas principais de criá-las (**Thread** e **Runnable**).
* É essencial compreender **sincronização** e **concorrência** para evitar problemas como ***race conditions***.
* A partir do Java 8, você pode usar **executors** e **lambda expressions** para simplificar o código **multithread**.