

# Trabalhando com Threads

# Definição

- A thread é um subprocesso de um processo, que permite compartilhar a sua área de dados com o programa ou outras threads.
- Uma thread é similar à um programa com início, sequência de execução e fim. Porém a Thread não é um programa, uma vez que, não pode ser executada diretamente (de forma isolada). Ela deve ser executada dentro de uma aplicação, e esta aplicação poderá possuir vários pontos de execuções, cada um representando uma Thread.
- Threads são usadas para programação paralela.

# Como implementar uma Thread

- Utilizaremos a forma mais flexível para trabalharmos com uma Thread, faremos uso da interface **Runnable**.
- O uso da interface **Runnable** nos obriga a implementação do método **run()**. O método **run()** é o método acionado, na thread, quando chamamos o método **start()** da thread.
- Se as threads gastarem um tempo significativo para serem executadas, serão escalonadas no escalonador de processos do sistema operacional, assim como o processo principal (o programa propriamente dito).
- No slide seguinte apresentaremos um programa que cria duas threads a partir do programa principal.

# Exemplo

```
public class ExemploThread implements Runnable {
```

```
    static Thread thread1;
```

```
    static Thread thread2;
```

```
    long num=0;
```

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

```
        ExemploThread e = new ExemploThread();
```

```
        thread1 = new Thread(e);
```

```
        thread2 = new Thread(e);
```

```
        thread1.start();
```

```
        thread2.start();
```

```
    }
```

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

```
        Thread currentThread = Thread.currentThread();
```

```
        while (true) {
```

```
            if (currentThread == thread1) {
```

```
                try {
```

```
                    thread1();
```

```
                } catch (InterruptedException ex) {
```

```
                } catch (IOException ex) {
```

```
                }
```

```
            } else if (currentThread == thread2) {
```

```
                try {
```

```
                    thread2();
```

```
                } catch (InterruptedException ex) {
```

```
                }
```

```
            }
```

```
        }
```

```
    }
```

# Exemplo

```
private void thread1() throws InterruptedException, IOException {  
    System.out.println("Eu sou a thread 1");  
    num++;  
    Thread.sleep(10);  
}
```

```
private void thread2() throws InterruptedException {  
    System.out.println("Eu sou a thread 2: num vale " + num);  
    Thread.sleep(10);  
}  
}
```

# Outro exemplo

- No próximo exemplo teremos um processo principal e 2 threads trabalhando em conjunto.
- As duas threads serão iguais e ficarão incrementando um atributo interno qt.
- Já o programa principal, através do método `getQt` vai imprimir o valor do atributo qt de cada uma das threads.

# Outro exemplo

```
public class UsandoThread {  
    public static void main(String[] args) throws InterruptedException {  
        Contador cont1 = new Contador("Thread1");  
        Thread thread1 = new Thread(cont1);  
        thread1.start();  
        Contador cont2 = new Contador("Thread2");  
        Thread thread2 = new Thread(cont2);  
        thread2.start();  
        while (true) {  
            Thread.sleep(100);  
            System.out.println("Estou no programa principal: cont1 tem " + cont1.getQt() + " cont2 tem " + cont2.getQt());  
            Thread.sleep(100);  
        }  
    }  
}
```

# Outro exemplo

```
public class Contador implements Runnable {  
  
    private long qt = 0;  
  
    private String nome;  
  
    public Contador(String nome) {  
  
        this.nome = nome;  
  
    }  
  
    public long getQt() {  
  
        return qt;  
  
    }  
  
    public void run() {  
  
        while (true) {  
  
            qt++;  
  
        }  
  
    }  
  
}
```



# Alguns métodos da classe Thread

- ④run(): deve estar presente em todas as threads.
- ④start(): registra a thread no thread scheduler.
- sleep(): Faz com que a thread fique em estado de espera uma quantidade mínima de tempo, em ms, possibilitando a CPU executar outras threads.
- stop(): encerra a thread.

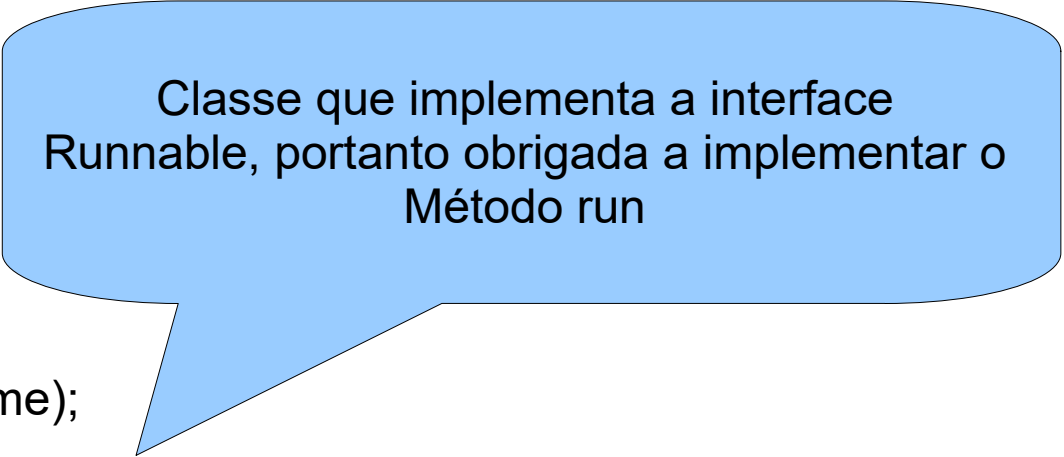
# Funções lambda para Threads

- O conceito de função lambda foi adicionado no Java 8. Esse conceito busca adicionar ao Java técnicas de linguagens funcionais (ex: LISP). A grande vantagem de funções lambda é diminuir a quantidade de código necessária para a escrita de alguns métodos, como por exemplo para Threads.
- Em outras palavras, uma função lambda é uma função sem declaração, isto é, não é necessário colocar um nome, um tipo de retorno e o modificador de acesso.
- No contexto de Threads as funções lambda são usadas principalmente para diminuir a quantidade de código digitada.
- Funções Lambda têm a seguinte sintaxe:

(argumento) -> (corpo)

# ClasseRunnable

```
public class ClasseRunnable implements Runnable {  
    private String nome;  
  
    public ClasseRunnable(String nome) {  
        this.nome = nome;  
    }  
  
    public void run() {  
        try {  
            while (true){  
                Thread.sleep(100);  
                System.out.println(nome);  
                Thread.sleep(100);  
            }  
        } catch (InterruptedException ex) {  
        }  
    }  
}
```



Classe que implementa a interface Runnable, portanto obrigada a implementar o Método run

```
public class FuncoesLambda {  
    public static void main(String[] args) throws InterruptedException {
```

```
    // Classe Runnable (usando implements)
```

```
    ClasseRunnable r1 = new ClasseRunnable("Thread 1");  
    Thread thread1 = new Thread(r1);  
    thread1.start();
```

```
    // Classe Anônima Runnable
```

```
    Runnable r2 = new Runnable() {  
        public void run() {  
            try {  
                while (true) {  
                    Thread.sleep(100);  
                    System.out.println("Thread 2");  
                    Thread.sleep(100);  
                }  
            } catch (InterruptedException ex) {  
            }  
        }  
    };
```

Em amarelo o código economizado  
Na versão que usa função lambda

```
Thread thread2 = new Thread(r2);  
thread2.start();
```

// Função Lambda

```
Runnable r3 = () -> {  
    try {  
        while (true) {  
            Thread.sleep(100);  
            System.out.println("Thread 3");  
            Thread.sleep(100);  
        }  
    } catch (InterruptedException ex) {  
    }  
};
```

```
Thread thread3 = new Thread(r3);  
thread3.start();  
}  
}
```

# Dúvidas?

