

## Módulo 2 - Laboratório 6

### Introdução à programação concorrente em Java

Computação Concorrente (MAB-117)  
Prof. Silvana Rossetto

<sup>1</sup>DCC/IM/UFRJ

#### Introdução

O objetivo deste Laboratório é **introduzir a programação concorrente em Java**. Para cada atividade, siga o roteiro proposto e responda às questões colocadas.

#### Atividade 1

**Objetivo:** Mostrar como criar um programa concorrente em Java. Em Java, a classe `java.lang.Thread` oferece métodos para criar threads, iniciá-las, suspendê-las e esperar pelo seu término.

O primeiro passo para criar uma aplicação concorrente em Java é **criar uma classe que implementa a interface Runnable**. Essa interface define o método `run()`, responsável pelo código que deverá ser executado pela thread.

O segundo passo é **transformar o objeto Runnable em uma thread**, chamando o construtor da classe `java.lang.Thread` com o objeto `Runnable` como argumento.

O terceiro passo é iniciar as threads criadas, usando o método `start()` da classe `Thread`.

Abra o arquivo **HelloThread.java** e siga o roteiro abaixo.

1. Leia o programa e tente entender o que ele faz (acompanhe a explicação da professora na vídeo-aula).
2. Compile o programa fazendo `javac HelloThread.java` no terminal.
3. Execute o programa **várias vezes** (fazendo `java HelloThread`) e observe os resultados impressos na tela. **Há mudanças na ordem de execução das threads? Por que isso ocorre?**
4. Descomente as linhas 43-46 e compile o programa novamente.
5. Execute o programa **várias vezes** e observe os resultados impressos na tela. **Qual alteração na execução da aplicação pode ser observada e por que ela ocorre?**

#### Atividade 2

**Objetivo:** Mostrar outra forma de criar threads em Java, usando herança.

**Roteiro:** Outra forma de criar programas concorrentes em Java é estendendo a classe `Thread`. Abra o arquivo **OlaThread.java** e siga o roteiro abaixo.

1. Primeiro, encontre as principais diferenças em relação ao programa `HelloThread.java` (acompanhe a explicação da professora na vídeo-aula).
2. Compile e execute o programa **várias vezes**, e observe os resultados impressos.

### Atividade 3

**Objetivo:** Mostrar um exemplo de aplicação com threads e memória compartilhada em Java.

**Roteiro:** Abra o arquivo **TIncrementoBase.java** e siga o roteiro abaixo.

1. Leia o programa para entender o que ele faz (acompanhe a explanação da professora na vídeo-aula). **Qual é a seção crítica do código?** Qual saída é esperada para o programa (valor final de `s`)?
2. Compile o programa, execute-o **várias vezes** e observe os resultados impressos na tela. **Os valores impressos foram sempre o valor esperado? Por que?**

### Atividade 4

**Objetivo:** Mostrar como implementar **exclusão mútua** em Java.

**Roteiro:** Ainda no arquivo **TIncrementoBase.java**:

1. Comente as linhas 17-23; e descomente as linhas 27-33.
2. Acompanhe a explanação da professora sobre o uso de `synchronized` em Java.
3. Compile o programa, execute-o **várias vezes** e observe os resultados impressos na tela. **Os valores impressos foram sempre o valor esperado? Por que?**

### Atividade 5

**Objetivo:** Implementar um **programa concorrente em Java para incrementar de 1 todos os elementos de um vetor**. Considere que o número de threads é **menor ou igual** ao número de elementos do vetor. Divida a tarefa entre as threads de forma balanceada.

**Roteiro:**

1. Na thread *main*, preencha e imprima o vetor com valores iniciais, crie e dispare as threads, aguarde todas as threads terminarem e imprima os valores finais do vetor.
2. Teste seu programa.

**Entrega:** Disponibilize o código implementado na **Atividade 5** em um ambiente de acesso remoto (GitHub ou GitLab). **Use o formulário de entrega desse laboratório para enviar o link do repositório do código implementado.**