

# Laboratório 10

## Introdução à programação concorrente em Java e pool de threads

Programação Concorrente (ICP-361)  
Profa. Silvana Rossetto

<sup>1</sup>Instituto de Computação/CCMN/UFRJ

### Introdução

O objetivo deste Laboratório é **introduzir a programação concorrente em Java** e avaliar uma implementação de *pool de threads*. 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**).
2. Traduza 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?**
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**).
2. Traduza 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 explicação da professora**). **Qual é a seção crítica do código?** Qual saída é esperada para o programa (valor final de s)?
2. Traduza o programa, execute-o **várias vezes** e observe os resultados impressos na tela. **Os valores impressos foram sempre o valor esperado? Por que?**
3. **Faça as correções necessárias no código e o avalie novamente.**

### Atividade 4

**Objetivo:** Avaliar uma implementação do padrão **leitor/escritor** em Java.

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

1. Leia o programa para entender o que ele faz (**acompanhe a explicação da professora**).
2. Traduza o programa, execute-o **várias vezes** e observe os resultados impressos na tela. **Os valores impressos foram sempre o valor esperado?**
3. Agora abra o arquivo **verificaLE.py**. Esse arquivo foi escrito para avaliar o log gerado pelo programa Java. Entenda como ele funciona (**acompanhe a explicação da professora**).
4. Execute novamente o programa Java e redirecione a saída para um arquivo .py. Depois execute esse programa em Python e avalie os resultados.

### Atividade 5

**Objetivo:** Compreender como pode ser feita uma implementação de um *pool de threads* em Java.

**Roteiro:**

1. Edite o arquivo **MyPool.java** e compreenda o que faz e como funciona (*acompanhe explicação da professora*).
2. Documente a classe `FilaTarefas`.
3. Execute o programa (**várias vezes**) alterando o tamanho do pool de threads e o número de tarefas executadas. Avalie se o programa funciona como esperado.
4. **Acrescente um novo tipo de tarefa para ser executada no mesmo pool de threads:** dado um número inteiro positivo verifica se ele é primo, como no código mostrado abaixo. Complete a implementação da classe `Primo` e descomente as linhas 93 e 94 do programa.

```
//funcao para determinar se um numero é primo
int ehPrimo(long unsigned int n) {
    int i;
    if(n<=1) return 0;
```

```
    if(n==2) return 1;
    if(n%2==0) return 0;
    for(i=3; i< sqrt(n)+1; i+=2) {
        if(n%i==0) return 0;
    }
    return 1;
}
```