

# INE 5410 - Laboratório

## AULA 09 – JAVA: PROGRAMANDO COM THREADS

---

### 1. INTRODUÇÃO

Um dos pontos interessantes do uso de uma linguagem como Java é, sem dúvida, a possibilidade de desenvolver programas concorrentes com base no uso de Threads. Threads Java são partes de código que podem ser criadas no contexto de um programa e lançadas em execução para executar concorrentemente, cooperando entre si e concorrendo pelo uso dos recursos do ambiente de execução.

### 2. OBJETIVO DA AULA

O objetivo desta aula de laboratório será a experimentação do uso de Threads da linguagem Java, exercitando o desenvolvimento de programas multithreads e observando os problemas que são comuns à programação concorrente.

### 3. CRIANDO THREADS NA LINGUAGEM JAVA

A linguagem de programação Java suporta a definição de programas concorrentes, permitindo que o programador crie um conjunto de threads associadas ao mesmo programa. Para isso, o programador dispõe de duas abordagens possíveis:

- a criação de uma subclasse da classe Thread (classe Java), sendo que a execução da Thread criada é definida pela reescrita do método run(), que vai estabelecer o código a ser executado pela Thread;
- a implementação da interface Runnable, que é utilizada em casos onde a classe já vai derivar de outra superclasse (não podendo, portanto, ser uma subclasse da classe Thread)... neste caso, além de herdar propriedades e métodos de outra classe, a classe criada será também uma Thread que irá executar em conjunto com as demais.

O código a seguir mostra um exemplo de programa onde duas classes são definidas num programa, cada uma seguindo uma das abordagens mencionadas anteriormente.

```
public class MinhaThread extends Thread {
    private int tempo;

    public MinhaThread( String name )
    {
        super( name );
        tempo = (int) ( Math.random() * 5000 );
    }

    public void run()
    {
        while (true) {
            try { Thread.sleep( tempo );
                System.err.println( getName() + " Executando!!!" );
            } catch (InterruptedException erro ){System.out.println( erro);}
        }
    }
} // Fim MinhaThread

public class MinhaRunnable implements Runnable
{
    private String pid;
    private int tempo ;

    public MinhaRunnable (String name )
    {
        this.pid = name;
        tempo = (int) (Math.random() * 5000 );
    }

    public void run()
    {
        while (true){
            try { Thread.sleep ( tempo );
                } catch (InterruptedException erro ){System.out.println( erro);}
            System.out.println(pid + " Executando!!!");
        }
    }
} // Fim MinhaRunnable

public class Main {
    public static void main( String args[] )
    {
        MinhaThread thread1;
        Thread thread2;
        thread1 = new MinhaThread( "thread 1" );
        thread2 = new Thread(new MinhaRunnable("thread 2"));
        System.out.println( "\nIniciando as threads" );

        thread1.start();
        thread2.start();

    }
} // Fim Main
```

#### 4. ATRIBUINDO PRIORIDADES ÀS THREADS

Em determinados sistemas, diferentes tarefas podem assumir diferentes níveis de importância no que diz respeito à sua execução. Os sistemas tempo-real são exemplos disso, onde tarefas associadas à segurança ou ao tratamento de exceções devem ter prioridade sobre as tarefas normais de controle do sistema.

Para isto, as threads em Java podem ser atribuídas a ela diferentes níveis de prioridades, que o escalonador da máquina virtual deverá levar em conta no momento em que tem de escolher qual, dentre as tarefas prontas para executar, deverão entrar em execução num dado momento.

Java suporta 10 níveis de prioridade, representados por números inteiros que variam de 1 a 10. Desta forma, o programador dispõe destes 10 níveis para atribuir às suas threads, em função do nível de importância que é considerado à tarefa realizada por uma dada thread.

A atribuição de prioridades às threads é viabilizada pelo método `SetPriority`, sendo que o argumento deverá ser um número inteiro entre 1 e 10. Quando uma thread inicializa uma outra thread e não há uma atribuição explícita de prioridade à thread criada, esta irá herdar o mesmo nível de prioridade atribuído à thread que a criou. Num programa multithreads, se o programador não fizer uso do conceito de prioridade, todas as threads criadas terão atribuídas a elas a prioridade normal (correspondendo ao valor inteiro 5 na escala das prioridades), passando a competir em igualdade pelo direito à execução.

O código a seguir ilustra um programa que faz uso do conceito de prioridades para que se possa observar a importância da atribuição de diferentes níveis às threads. Neste programa, quatro threads executam a mesma função que é a de preencher os componentes de um vetor de 1000 posições com valores inteiros que correspondem a seus identificadores (valores inteiros entre 1 e 4). Durante sua execução, cada thread vai preenchendo os componentes do vetor com o inteiro correspondente a seu identificador, numa malha que só finaliza quando o vetor estiver completamente preenchido.

Em função do nível de prioridade que se atribui às diferentes threads, é de se esperar que aquelas de mais alto nível executem com mais frequência que as de nível inferior. Do ponto de vista da tarefa a ser executada, que é a mesma para todas as threads, isto vai se refletir na frequência com a qual cada thread preenche os componentes do vetor, ou melhor, quantos valores relativos a seu identificador serão associados a componentes do vetor.

Experimente a execução do programa alterando os valores de prioridades a serem atribuídos às threads, assim como a quantidade de componentes que irão compor o vetor.

```
public class Indice {
    int i, limite;

    public Indice ( int  limite ){
        i  = 0;
        this.limite = limite;
    }

    public synchronized int get(){
        int retorno;
        if ( i < limite ) {
            retorno = i ;
            i++;    } else retorno = -99999;

        return ( retorno );
    }
}
```

```

public class Processo extends Thread {
    int pid, i;
    Indice indice;
    int vetor [];

    public Processo ( int pid, Indice indice, int[] vetor ){
        this.pid    = pid;
        this.indice = indice;
        this.vetor  = vetor;
    }

    public void run(){
        do {
            i = indice.get();
            if ( i >= 0 ) {
                vetor [ i ] = pid;
                for ( int k = 1; k < 500 ; k++ )
                    for ( int l = 1 ; l < 5000; l++ );
            }
        } while ( i >= 0 );
    } }

public class Main {
    public static void main ( String [] s ){
        int i;
        int cont [] = new int[ 5 ];
        int vetor [] = new int[ 1000 ];
        Indice indice = new Indice ( 1000 );
        Processo proc1, proc2, proc3,proc4;
        // Criando as threads
        proc1 = new Processo ( 1 , indice, vetor );
        proc2 = new Processo ( 2 , indice, vetor );
        proc3 = new Processo ( 3 , indice, vetor );
        proc4 = new Processo ( 4 , indice, vetor );
        // Atribuindo prioridades
        proc1.setPriority ( 5 ); proc2.setPriority ( 5 );
        proc3.setPriority ( 5 ); proc4.setPriority ( 5 );
        // Inicializando as threads
        proc1.start(); proc2.start();
        proc3.start(); proc4.start();
        // Aguardando a finalização das threads
        try{ proc1.join();
            proc2.join();
            proc3.join();
            proc4.join();
        } catch ( Exception e ){System.out.println (e);}
        for ( i = 0; i < 1000; i++) cont [ vetor [ i ] ]++;
        System.out.println ( "  NUMERO DE 1: "+ cont [ 1 ] );
        System.out.println ( "  NUMERO DE 2: "+ cont [ 2 ] );
        System.out.println ( "  NUMERO DE 3: "+ cont [ 3 ] );
        System.out.println ( "  NUMERO DE 4: "+ cont [ 4 ] );
    } }

```