→ Threads em Java

Num programa em JAVA é possível definir diferentes sequências de execução independente: **Threads**. Uma Thread é similar a um processo no sentido em que corresponde a um conjunto de instruções que pode ser escalonado para execução num dado processador. No entanto, as Threads definidas num programa partilham o mesmo espaço de endereçamento que o processo principal que lhes deu origem.

⇨ Criar uma Thread

Existem duas formas de criar uma Thread em JAVA. A primeira consiste em criar uma **subclasse da classe Thread** e usar o método start() definido em Thread para iniciar a execução. O método start() invoca por sua vez um método run() a definir pelo utilizador, e que deverá conter a sequência de execução da Thread instanciada.

1 - Implemente e teste o seguinte exemplo:

```
public class MyThread extends Thread {
  public MyThread(){
    super();
  }
  public void run() {
   System.out.println("Hello there, from  " + getName() );
  }}
public class Teste {
public static void main (String[] str){
  MyThread Ta, Tb;
  Ta = new MyThread();
  Tb = new MyThread();
  Ta.start();
  Tb.start();
}}
```

Modifique o método run da Thread juntando mais instruções. Execute o programa várias vezes de forma a verificar que a execução não é determinística.

- Este processo de criar uma Thread não pode ser usado se pretendermos que a classe a criar seja subclasse de alguma outra, por exemplo no caso de querermos construir uma Applet. A segunda forma de criar uma Thread é usar a interface **Runnable,** já definida na linguagem**.** [Recordando … Em termos simples, uma "interface" é a definição de um conjunto de métodos que a classe ou classes que implementam essa interface terão que implementar]

A interface Runnable:

```
package java.lang
public interface Runnable{
  public abstract void run();
}
```

No exemplo abaixo, a classe MyThread2 implementa a interface Runnable, implementado o método run() que é o único método definido na interface. A diferença entre esta classe e a classe MyThread definida anteriormente é que a classe MyThread2 não herda os métodos da classe Thread. Para agora criarmos uma nova Thread temos de passar ao construtor dessa Thread a referência de uma classe que implemente a interface Runnable.

2 - Implemente e estude o exemplo que se segue.

```
public class MyThread2 implements Runnable{
 public MyThread2(){  }
 public void run(){
  System.out.println("Hi there, from " + Thread.currentThread().getName() );
 } }
public class Teste {
 public static void main  (String[] str){
  MyThread2 T = new MyThread2();
  Thread Ta, Tb;
  Ta = new Thread( T );
  Tb = new Thread( T );
  Ta.start();
  Tb.start();
 } }
```

3 – Pretende-se construir um programa que permita adicionar arrays de grande dimensão. Para isso, vamos dividir os arrays a somar em várias porções menores e somar cada porção numa thread independente.
- Construir o código de uma thread, ThreadSoma, que receba como parâmetros as "referências" de 3 arrays, A, B e C e 2 valores inteiros, p e u. A thread deverá fazer a soma dos arrays, A e B, desde a posição p até à posição u-1, colocando o resultado em C.

$$C[i] = A[i] + B[i] \quad \text{com } i = p, p+1, p+2, \ldots u-1$$

a) Construa um programa onde deverá criar dois arrays de inteiros com valores aleatórios. Os arrays, A e B, devem ter a mesma dimensão. Para somar os dois arrays, divida-os em dois e cada metade deverá ser somada por uma instância da classe ThreadSoma que criou na alínea anterior.

b) Queremos agora que no programa anterior, o programa principal calcule a soma de todos os valores do array resultado, C. Essa soma só deve ser feita após ambas as instâncias de

ThreadSoma terminarem a execução. Explore os métodos da classe Thread para ver como pode fazer com que uma Thread espere que outra termine a execução.

c) Compare o tempo de execução de um programa sequencial que some os dois arrays com o tempo de execução da versão multi-threaded. Teste para diferentes dimensões do array.

→ Daemon Threads

Uma Thread "Daemon" é uma Thread, geralmente usada para executar serviços em "background", que tem a particularidade de terminar automaticamente após todas as Threads "não Daemon" terem terminado. Uma Thread transforma-se numa Thread Daemon através do método **setDaemon()**.

```
public class Normal extends Thread{
 public Normal() {
  super( );
 }
 public void run() {
  for (int i=0; i<5; i++){
   try
     { sleep(500);}
   catch (InterruptedException e)
     {...}
   System.out.println (" I' m the normal Thread");
  }
  System.out.println (" The normal Thread is exiting");
 }
}
public class Daemon extends Thread {
public Daemon() {
  super();
  setDaemon( true);
}
public void run(){
   for (int i=0; i<10; i++){
   try
     { sleep(500);}
   catch (InterruptedException e)
     { }
   System.out.println (" I'm a daemon Thread");
}  }
}
```

4 - Depois de implementar as duas classes anteriores, construa uma classe em que teste ambas. Após observar o comportamento do programa, experimente comentar a linha onde o método setDaemon é invocado e observe a diferença de comportamento do programa.

⇨ Grupos de Threads

As Threads de um programa podem ser agrupadas em grupos, tornando possível enviar uma mensagem simultaneamente a um conjunto de Threads.

5 - Considere as classes abaixo. Complete a classe Teste e teste-a.

```
public class MyThread extends Thread{
  public MyThread( String name) {
   super(name);
  }
  public void run (){
    while (true) {
      System.out.println ("Sou a " + this.getName());
      if ( isInterrupted() )
        break;
      Thread.yield();
    } }
}
public class Teste {
  public static void main (String[] arg){
    MyThread Ta, Tb, Tc;
    ThreadGroup this_group;
    this_group = Thread.currentThread().getThreadGroup();
    System.out.println("O nome do grupo é: " + this_group.getName());
    System.out.println("O nº de Threads activas no grupo é " + this_group.activeCount());

    Ta=new MyThread ("Thread A");
    Tb=new MyThread ("Thread B");
    Tc=new MyThread ("Thread C");

    // inicie a execução das Threads
   ...
    // obtenha o nome do grupo e o número de threads activas nesse grupo
   ...
  try
     {Thread.sleep (500);}
    catch (InterruptedException e)
```

```
    {...}

// Pode invocar um método em todas as Threads do grupo:
   this_group.interrupt();
  }
}
```

Um grupo pode ser criado explicitamente:

```
...
ThreadGroup Mygroup = new ThreadGorup (" O meu grupo")
```

Para adicionar uma Thread ao grupo criado deverá criar um construtor da sua subclasse de Thread que inicialize o nome do grupo na superclasse Thread:

```
class MyThread extends Thread {
 public MyThread ( ThreadGroup tg, String name) {
   super(tg, name);
  }

...
```

6 - Teste a criação de dois grupos de threads num mesmo programa.

- **Prioridade de uma Thread**

Java suporta 10 níveis de prioridades. A menor prioridade é definida por Thread.MIN_PRIORITY e a mais alta por Thread.MAX_PRIORITY. Podemos saber a prioridade de uma Thread através do método *int getPriority()* e podemos modificá-la com o método *setPriority(int*).

7 - Verifique qual a prioridade por omissão de uma thread

8 – Construa um programa exemplo que lance 3 ou mais Threads, atribua prioridades diferentes às várias threads e estude o comportamento do programa.

9 – Implemente e estude os exemplos da aula teórica T03a, páginas 19 a 24.