→ Threads

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 dado programa partilham o mesmo espaço de endereçamento que o processo principal que lhes deu origem.

1 - 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.

Exercício 1: - Implemente e teste o seguinte exemplo:

```
public class MyThread extends Thread {
  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();
}}
```

- 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**. 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.

Exercício 2 - Implemente e estude o exemplo que se segue.

```
public class MyThread2 implements Runnable{
 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();
 } }
```

2 - Junção de Threads

O método **join()** definido na classe Thread permite fazer com que uma thread espere que a execução de uma outra termine para que a primeira possa finalmente continuar a sua própria execução. No exemplo abaixo a Thread T2 precisa de esperar que a Thread T1 termine a execução antes de poder continuar.

Exercício 3 : Implemente e estude o exemplo abaixo.

```
public class MyThread1 extends Thread {
 public void run(){
  System.out.println (getName() + "is running...");
  for ( int i=0;i<5; i++){
```

```
   try
    { sleep (500); }
   catch ( InterruptedException e)
    { …}
   System.out.println ("Hello there, from " + getName());
  } }
}

public class MyThread2 extends Thread{
  private Thread wait4me;
  public MyThread2( Thread T) {
    wait4me= T;
  }
 public void run() {
 System.out.println( getName() + " is waiting for " + wait4me.getName() + "...");
 try
  { wait4me.join(); }
 catch (InterruptedException e)
  {}
 System.out.println( wait4me.getName() + "has finished" );
 for ( int i=0;i<5; i++){
   try
    { sleep (500); }
   catch ( InterruptedException e )
    {...}
   System.out.println ("Hello there, from " + getName());
 } }
}

public class Teste {
  public static void main (String[] str){
    MyThread1 T1 = new MyThread1();
    MyThread2 T2 = new MyThread2( T1 );
    System.out.println ("Starting the Threads");
    T1.start();
    T2.start();
  }
}
```

3 – "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 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 Daemom extends Thread ()
public Daemon() {
  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");
} }
}
```

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

4- Grupos de Threads

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

Exercício 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;
      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);
  }
...
```

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

5- 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*).

Exercícios:

7 - Verifique qual a prioridade por omissão de uma thread

8 - Construa uma classe Teste que use a classe MyThread definida na página 1, onde deverá acrescentar a escrita da prioridade de execução da Thread, e comece por:

- atribuir à thread main a prioridade máxima,

- criar três threads,

- forçando a thread main a parar com o método sleep(int) estude o comportamento do programa quando atribui diferentes prioridades às três threads criadas.

→ Sincronização de Threads

A sincronização de Threads em Java é baseada no conceito do Monitor (de Hoare). Cada objecto Java tem associado um monitor (ou "lock") que pode ser activado se a palavra chave **synchronized** for usada na definição de pelo menos um método de instância:

public **synchronized** void metodoX();

O monitor da classe será activado se um método de classe for declarado como **synchronized:**

public static **synchronized** void metodoY();

Se várias Threads invocarem um método declarado como **synchronized** em simultâneo o monitor associado ao objecto garantirá a execução desse método em exclusão mútua.

→ **Monitores de classe e monitores de objecto**

O exemplo abaixo ilustra a sincronização de um método de classe e de um método de objecto ou instância.

Exercício 9: - Depois de o estudar, implemente-o e analise os resultados.

```
public class CriticaUm {

 public synchronized void method_A() {

   System.out.println ( Thread.currentThread().getName() + " Método A");
   try {
     Thread.sleep( (int) Math.round(Math.random()*5000));
   }
   catch (InterruptedException e)
   { … }
   System.out.println ( Thread.currentThread().getName() + " Saindo do Método A");
  }
```

```
  public   static   synchronized void method_B() {
```

```
   System.out.println ( Thread.currentThread().getName() + " Método B");
   try {
     Thread.sleep( (int) Math.round(Math.random() *5000));
   }
   catch (InterruptedException e)
   { … }
   System.out.println ( Thread.currentThread().getName() + " Saindo do Método B");
  }
 }

public class Monitors extends Thread{
  CriticaUm C;

  public Monitors(String nomeObjecto) {
    Thread Thread_a, Thread_b;
    C= new CriticaUm();
    Thread_a = new Thread (this, nomeObjecto + ":Thread a");
    Thread_b = new Thread (this, nomeObjecto + ":Thread b");

    Thread_a.start();
    Thread_b.start();
  }

  public void run (){
    C.method_A();
    C.method_B();
  }
}

public class Teste {
   public static void main (String args[]){
   Monitors M1, M2;
   M1=new Monitors ("Objecto 1");
   M2=new Monitors ("Objecto 2");
   }
}
```

→ Invocação recursiva de métodos sincronizados (código reentrante)

Uma Thread que adquiriu um lock num objecto (ao executar um método sincronizado) pode invocar recursivamente o mesmo método. O lock é readquirido pela Thread.

Exercício 10: - Estude o exemplo abaixo.

```
public class Reentrant {
   static int times =1;
  public synchronized void myMethod() {
    int i = times++;
    System.out.println("myMethod has started " + i + " time(s)" );
    while (times <4)
      myMethod();
    System.out.println("myMethod has exited " + i + " time(s)");
  }
}
public class Teste extends Thread {
 Reentrant R;
 public Teste (){
   R= new Reentrant();
   Thread T = new Thread(this);
   T.start();
 }
public void run(){
    R.myMethod();
 }
public static void main(String args[]) {
     Teste T1 = new Teste();
  }
}
```