BOLETÍN DE ACTIVIDADES

CUESTION 1. Dadas las siguientes afirmaciones, modifique en su caso lo que sea necesario para que las afirmaciones sean **CIERTAS.** *Nota*: Puede haber afirmaciones que ya sean inicialmente ciertas.

1. Un programa concurrente está formado por una colección de actividades (hilos) que se ejecutan en paralelo de forma independiente y sin ningún tipo de comunicación entre sí.
2. Un proceso con un único hilo de ejecución, que se ejecuta en un ordenador con un procesador de cuatro núcleos, es un ejemplo de un programa concurrente.
3. Un programa concurrente requiere ser ejecutado en máquinas con más de un procesador (o con varios núcleos), para permitir que sus tareas sean concurrentes.
4. Una de las ventajas de la programación concurrente es que permite mejorar la depuración de las aplicaciones, respecto a la programación secuencial, ofreciendo una depuración sencilla.
5. La programación concurrente permite programar de una manera más directa (i.e. con mayor hueco semántico) aquellas aplicaciones donde haya múltiples actividades simultáneas.
6. En Java solo se puede crear hilos si se utiliza el paquete *java.util.concurrent*.
7. El código a ejecutar por cada hilo debe estar contenido en su método *start*().
8. La sentencia *t.run()* permite lanzar a ejecución un hilo y ejecutar su método run() concurrentemente.
9. Para definir un hilo de ejecución en Java, debemos definir alguna instancia de una clase que implemente la interfaz *Thread*.
10. Para asignar nombre a los hilos se puede pasar una cadena como argumento en su constructor.
11. *Thread.yield*() hace que un hilo pase del estado 'preparado' al estado 'suspendido'.
12. El método *setName*() de la clase Thread asocia siempre un nombre al hilo que esté en ejecución en ese momento.

CUESTION 2. Considérese la estructura de un programa que debe incluirse en el computador del sistema de control de un coche, que sea capaz de llevar a cabo las siguientes tareas: medir la velocidad cada 200ms (usando para ello 40 ms de CPU), controlar la presión del carburante cada 400ms (usando 100ms de CPU), y controlar la válvula del carburador cada 800 ms (usando 400ms de CPU). Suponga que se dispone de los siguientes métodos:

MV	mide la velocidad
CP	realiza el control de la presión del carburante
CV	realiza el control de la válvula del carburador
Sleep (ms)	suspende la ejecución de quien lo invoca ms milisegundos
delayUntilP(periodo)	suspende la ejecución de quien lo invoca hasta que venza su próximo periodo

Indique cuál será la estructura del código para los siguientes casos:

a) Se aplica una solución secuencial

b) Se aplica una solución concurrente

ACTIVIDAD 1. OBJETIVO: Describir cómo crear y ejecutar hilos en Java

Dado el siguiente programa:

```
public class T extends Thread {
       protected int n;
       public T(int n) {this.n = n;}

       public void delay(int ms) {
      // suspends execution for ms milliseconds
            try {  sleep(ms);
             } catch (InterruptedException ie){ie.printStackTrace();}
       }

       public void run() {
          for (int i=0; i<10; i++) {
                 System.out.println("Thread "+n +" iteration "+i);
                 delay((n+1)*1000);
              }
          System.out.println("End of thread "+n);
       }

       public static void main(String[] argv) {
          System.out.println("---  Begin of execution ---- ");
          for (int i=0; i<6; i++)
               new T(i).start();
          System.out.println("--- End of execution ---- ");
       }
 }
```

1) ¿Cuántos hilos se crean? ¿Se crean usando una "clase con nombre" o bien una "clase anónima"?

2) Reescriba el código anterior, pero en esta ocasión implementando `Runnable` (en lugar de extender `Thread`)

3) ¿Qué ocurriría si en el método *main* se utilizara `T(i).run()` en lugar de `T(i).start()`?

4) El mensaje `"End of execution"` no siempre es el último en escribirse en pantalla. ¿Por qué ocurre esto? ¿Cómo podríamos garantizar que SIEMPRE se escribiese en último lugar?

ACTIVIDAD 2.- OBJETIVO: Describir la gestión de hilos de ejecución en Java.

ENUNCIADO: Dado el siguiente código Java:

```
public class T2 extends Thread {
    private int level;
    public T2(int n){
       level = n;
    }
    public void createThread(int i) {
       T2 h = new T2(i);
       if (i>=1)
          h.start();
       System.out.println("Thread of level "+i+" created.");
    }
    public void run() {
       if (level>0)
           createThread(level-1);
       System.out.println("End of thread. Level:" + level);
    }
    public static void main(String[] argv) {
       for (int i=1; i<3; i++)
           new T2(2).start();
    }
}
```

1) ¿Cuántos hilos se crean? ¿Cuántos de ellos se ejecutan?

2) ¿Hay algún hilo con "nivel" igual a 0? ¿Y con valor 1? En caso afirmativo, indique cuántos hilos hay en cada uno de esos niveles.

3) Muestre una posible traza de este programa, indicando qué se muestra en pantalla.

4) Ejecute varias veces este programa. ¿Se muestran siempre los mismos mensajes? ¿Se muestran en el mismo orden? ¿A qué se debe esto?

5) Si en vez de `h.start()` tuviéramos `h.run()` en el método "createThread", ¿cuál habría sido el resultado? En dicho caso, si ejecutáramos varias veces el programa, ¿obtendríamos siempre el mismo resultado?

ACTIVIDAD 3.- OBJETIVO: Describir la gestión de hilos de ejecución en Java.

ENUNCIADO: Dado el siguiente código Java:

```
public class ExThread {
      public static void main(String[] args) {
          System.out.println(Thread.currentThread().getName());
          for (int i=0; i<10; i++){
             new Thread("MyThread "+i){
                 public void run() {
                    System.out.println("executed by"+
                        Thread.currentThread().getName());
                 }
             }.start();
          }   }   }
```

1) ¿Cuántos hilos se crean? ¿Cuántos de ellos se ejecutan?

2) ¿Para qué se emplea `Thread.currentThread.getName()`?

3) Muestre en una posible traza del programa qué se imprimirá por pantalla.

3) Modifique el programa para que el nombre del hilo se asigne utilizando el método `setName()` de la clase `Thread`.

ACTIVIDAD 4.- OBJETIVO: Describir la gestión de hilos de ejecución en Java.

ENUNCIADO: Dado el siguiente código Java:

```
public class ThreadName extends Thread {
   public void run() {
        for (int i = 0; i < 3; i++)
             printMsg();
   }
   public void printMsg() {
        System.out.println ("name=" +
         Thread.currentThread().getName());
   }
   public static void main(String[] args) {
        for ( int i = 0; i < 10; i++ ) {
             ThreadName tt= new ThreadName();
             tt.setName("MyThread" + i);
             if (i<5) tt.start()} }
   }}}
```

1) Muestre en una posible traza del programa qué se imprimirá por pantalla.

2) ¿Cuántos hilos se han creado? ¿Cuántos hilos se han ejecutado?

3) Modifique el programa de modo que se obtenga una funcionalidad similar, pero utilizando la creación anónima de hilos (es decir, sin definir ninguna instancia de tipo `ThreadName tt`).

ACTIVIDAD 5. OBJETIVO: Conocer el ciclo de vida de los hilos Java. Discutir las diferencias entre la espera activa y la suspensión de hilos. Discutir el uso de los métodos `isAlive` vs. `join`.

ENUNCIADO: Dado el siguiente código:

```
public class CalculateResults extends Thread{
    private String result = "Not calculated";
    public void run(){
        result = calculate();}
    private  String calculate(){
        // Performs a long-time calculation
        try {Thread.sleep(10000);
        } catch(InterruptedException e){};
        System.out.println("Agent thread finishes its calculation");
        return "Calculation done";
    }
    public  String getResults(){
        return result; }
}


class Example_1 {
    public static void main(String[] args){
        CalculateResults agent =new CalculateResults();
        agent.start();
        // It does something during the calculation process
        System.out.println("Main in execution");

       // Employs the result
        System.out.println(agent.getResults());
    }
}
```

1) ¿Qué hace este código? ¿Qué se mostrará en pantalla?

2) Reescriba el método *main()* para que el hilo principal espere a que el hilo *agente* finalice, antes de obtener el resultado calculado por éste. Para implementar la espera utilice **espera activa** en el hilo principal (es decir, el hilo comprobará de forma repetida si la condición es cierta o no).

3) Reescriba el método *main()*, pero esta vez para implementar la espera no utilice espera activa, sino espera suspendida (es decir, suspendiendo al hilo que debe esperar).

ACTIVIDAD 6. OBJETIVO: Comprender la programación concurrente en Java, en concreto el método `Thread.interrupt`.

1) Dado el siguiente código en Java, indique qué se mostrará por pantalla. ¿Para qué se utilizan aquí los métodos `Thread.isAlive()`, `Thread.interrupt()` y `Thread.join()` ?

2) Pruebe con distintos valores de "`patience`", para comprobar el uso del método `interrupt`.

```
public class SimpleThreads {
   // Display a message, preceded by the name of the current thread
    static void threadMessage(String message) {
        String threadName =  Thread.currentThread().getName();
        System.out.format("%s:%s%n",threadName, message);
     }
    private static class MessageLoop  implements Runnable {
        public void run() {
            String importantInfo[] = {"One","Two","Three","Four"};
            try {
               for (int i = 0; i < importantInfo.length; i++) {
                  Thread.sleep(4000); // Pause for 4 seconds
                  threadMessage(importantInfo[i]);  // Print a message
               }
           }catch(InterruptedException e) {threadMessage("I wasn't done!");}
    } }

  public static void main(String args[]) throws InterruptedException {
     // Delay, in milliseconds before we interrupt MessageLoop thread
     long patience = 1000 * 60;  // (default one minute)
     threadMessage("Starting MessageLoop thread");
     long startTime = System.currentTimeMillis();
     Thread t = new Thread(new MessageLoop());
     t.start();
     threadMessage("Waiting for MessageLoop thread to finish");
     // loop until MessageLoop thread exits
     while (t.isAlive()) {
       threadMessage("Still waiting...");
       //Wait maximum of 1 second for MessageLoop thread to finish.
       t.join(1000);
       if  (((System.currentTimeMillis()-startTime)>patience)&&  t.isAlive())
{
           threadMessage("Tired of waiting!");
           t.interrupt();
           // Shouldn't be long now --- wait indefinitely
           t.join();
        } }
     threadMessage("Finally!");
  } }
```

ACTIVIDAD 7. OBJETIVO: Uso de la notación Lambda en la creación de hilos.

ENUNCIADO: Analice el siguiente código y responda a las cuestiones que se plantean.

```
1  public class RunnableLambdaEx {
2   public static void main(String[] args) {
3    System.out.println(Thread.currentThread().getName() + ": RunnableTest");
4
5    // --- Parte 1 ----------------------
6    Runnable task1 = new Runnable(){
7      public void run(){
8        System.out.println(Thread.currentThread().getName() + " is running");
9      }
10   };
11   Thread thread1 = new Thread(task1);
12   thread1.setName("hilo1"); thread1.start();
13   new Thread(task1,"hilo1b").start();
14
15   // --- Parte 2 ---------------
16   Thread thread2 = new Thread(new Runnable() {
17     public void run(){
18       System.out.println(Thread.currentThread().getName() + " is running");
19     }
20   });
21   thread2.setName("hilo2"); thread2.start();
22
23   new Thread(new Runnable() {
24     public void run(){
25       System.out.println(Thread.currentThread().getName() + " is running");
26     }
27   },"hilo2b").start();
28
29   // --- Parte 3 --------------------
30   Runnable task3 = () -> {
31       System.out.println(Thread.currentThread().getName() + " is running");
32   };
33   new Thread(task3,"hilo3b").start();
34
35   Thread thread3 = new Thread(() ->
36       System.out.println(Thread.currentThread().getName() + " is running"));
37   thread3.setName("hilo3"); thread3.start();
38
39   new Thread(() -> System.out.println(Thread.currentThread().getName()+ " is
40 running"),"hilo4").start();
41  }
42 }
```

1. Indique para cada una de las partes cuántos hilos se crearán. ¿Alguna de las sentencias de creación de hilos es incorrecta?

2. Muestre en una posible traza del programa qué se imprimirá por pantalla.

3. Establezca la correspondencia entre las instrucciones de la parte 3, con notación lambda, que son equivalentes en la forma de crear los hilos (salvando los nombres de los objetos) a las de las partes 1 y 2.
   - líneas 30-32 equivalentes a ……..
   - línea 33 equivalente a ……..
   - líneas 35-38 equivalentes a ……..
   - líneas 39-40 equivalentes a ……..

ACTIVIDAD 8. OBJETIVO: Uso de la notación Lambda en la creación de hilos.

ENUNCIADO: Reescriba el código de la actividad 3 utilizando la notación Lambda, para que la clase ExThread proporcione la misma funcionalidad.

```
public class ExThread {
      public static void main(String[] args) {
          System.out.println(Thread.currentThread().getName());
          for (int i=0; i<10; i++){
             new Thread("MyThread "+i){
                 public void run() {
                    System.out.println("executed by"+
                        Thread.currentThread().getName());
                 }
             }.start();
          }   }   }
```