Actividades UD1.-Introducción a la Programación Concurrente

Concurrencia y Sistemas Distribuidos

Cuestión 1

- Indique si las siguientes afirmaciones son V/F, modifique lo necesario para que sean ciertas:

F	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.
F	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.
F	Un programa concurrente requiere ser ejecutado en máquinas con más de un procesador (o con varios núcleos), para permitir que sus tareas sean concurrentes.
F	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.
F	La programación concurrente permite programar de una manera más directa (i.e. con mayor hueco semántico) aquellas aplicaciones donde haya múltiples actividades simultáneas.

Cuestión 1

- Indique si las siguientes afirmaciones son V/F, modifique lo necesario para que sean ciertas:

F	En Java sólo se puede crear hilos si se utiliza el paquete *java.util.concurrent*.
F	El código a ejecutar por cada hilo debe estar contenido en su método *start*().
F	La sentencia *t.run()* permite lanzar a ejecución un hilo y ejecutar su método run() concurrentemente.
F	Para definir un hilo de ejecución en Java, debemos definir alguna instancia de una clase que implemente la interfaz *Thread*.
V	Para asignar nombre a los hilos se puede pasar una cadena como argumento en su constructor.
F	*Thread.yield*() hace que un hilo pase del estado 'preparado' al estado 'suspendido'.
F	El método *setName*() de la clase Thread asocia siempre un nombre al hilo que esté en ejecución en ese momento.

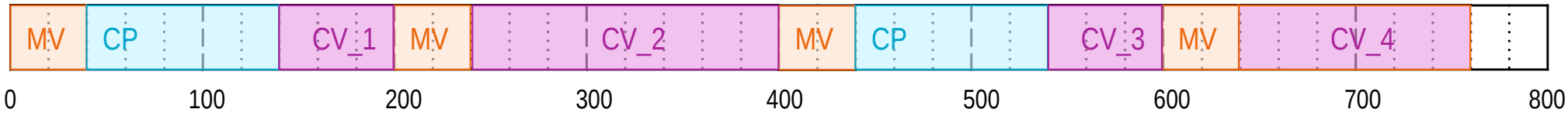
Cuestión 2

- 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).
- Suponer 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

Cuestión 2

- MV: medir la velocidad cada 200ms (usando para ello 40 ms de CPU),
  CP:controlar la presión del carburante cada 400ms (usando 100ms de CPU), y
  CV: controlar la válvula del carburador cada 800 ms (usando 400ms de CPU).
- Solución secuencial:
  - El método CV (400ms) se parte en cuatro métodos CV_1 (60 ms), CV_2 (160 ms), CV_3 (60 ms) y CV_4 (120 ms).



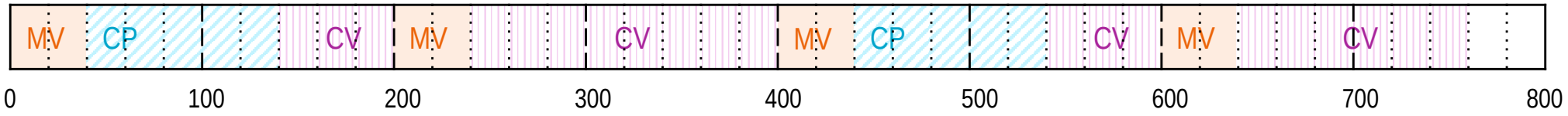
  - La estructura del código sería una secuencia de llamadas a los métodos:
  - ```
    while (TRUE){
     MV();CP();CV_1();MV();CV_2();MV();CP();CV_3();MV();CV_4();sleep(40);
    }
    ```
    ```

Cuestión 2

- MV: medir la velocidad cada 200ms (usando para ello 40 ms de CPU),
  CP:controlar la presión del carburante cada 400ms (usando 100ms de CPU), y
  CV: controlar la válvula del carburador cada 800 ms (usando 400ms de CPU).
- Solución concurrente:
  - Se utilizarían 3 hilos periódicos y les asignaríamos prioridades. Cada hilo invocará el método correspondiente y se suspenderá hasta que venza su siguiente periodo. El planificador se encarga de seleccionar el hilo que pondrá en ejecución.
  - Por ejemplo, la estructura del código del hilo que mide la velocidad sería:

```
while (TRUE){
 MV();
 delayUntilP(200);
}
```

  - Si el planificador es por prioridades expulsivas y la asignación de prioridades es MV la más prioritaria, después CP y por último CV, la ejecución de los hilos hasta el instante 800 será:



Actividad 1: Analizar el código

```
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 ---- ");
        }
 }
```

Actividad 1:

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

```
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 ---- ");
        }
 }
```

Se crean 6 hilos usando una clase con nombre T

Actividad 1:

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

```
public class T Implements Runnable {
        protected int n;
        public T(int n) {this.n = n;}

        public void delay(int ms) {
       // suspends execution for ms milliseconds
             try { Thread.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 Thread(new T(i)).start();
           System.out.println("--- End of execution ---- ");
        }
 }
```

Actividad 1:

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

```
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). run() ;
          System.out.println("--- End of execution ---- ");
       }
 }
```

El hilo `main` ejecutaría 6 veces el método `run` secuencialmente. Veríamos por pantalla, siempre en este orden:

```
Thread 0 iteration 0
Thread 0 iteration 1 …
Thread 0 iteration 9
Thread 1 iteration 0
Thread 1 iteration 1 …
Thread 1 iteration 9
Thread 2 iteration 0
Thread 2 iteration 1 …
Thread 2 iteration 9
 …
Thread 5 iteration 0
Thread 5 iteration 1 …
Thread 5 iteration 9
```

Actividad 1:

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?

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

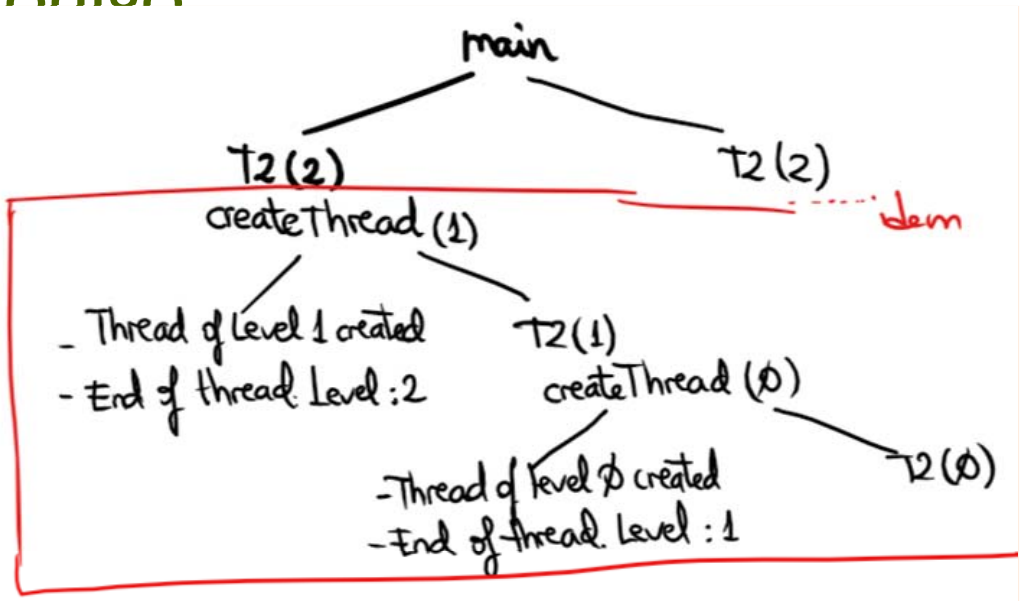
       public void delay(int ms) {
      // suspends execution for ms milliseconds
            try {  sleep(ms);
             } catch (InterruptedException ie){ie.p
       }

       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 ---- ");
         T[] hilo= new T[6];
         for (int i=0; i<6; i++) {
              hilo[i]=new T(i);
              hilo[i].start();
         };
         for (int i=0; i<6; i++)
           try { hilo[i].join();
           } catch (InterruptedException e){};
         System.out.println("--- End of execution ---- ");
 }
```

El hilo `main` se ejecuta concurrentemente con 6 hilos más, y el orden de ejecución depende del **planificador**.
Para garantizar que siempre se escriba en último lugar, main debe esperar la finalización de los hilos con un join().
Para ello ha de tener las referencias a los hilos, que almacenaremos en el vector hilo.

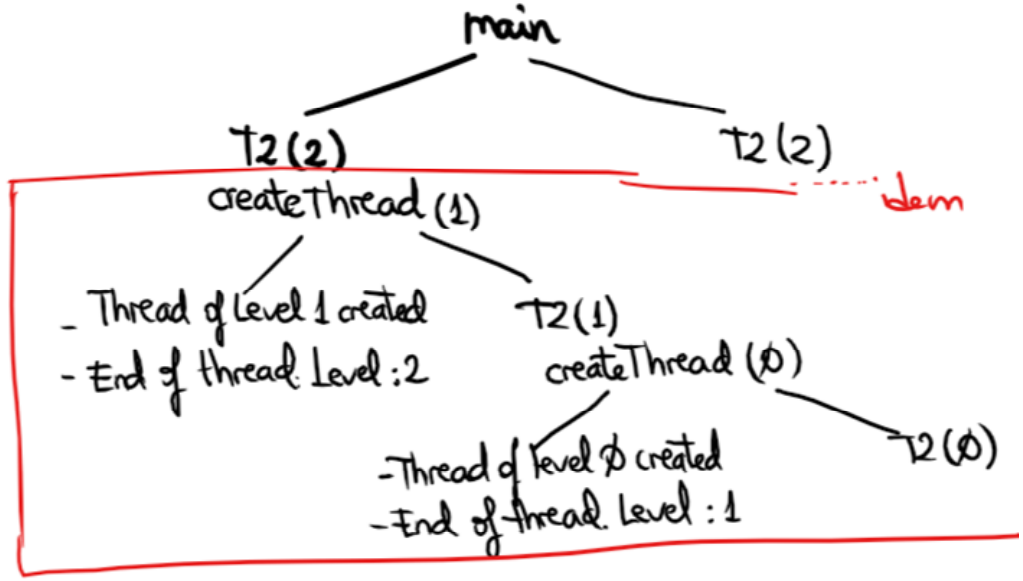
Actividad 2: Analizar el código

```
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();
    }
}
```



Actividad 2:

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

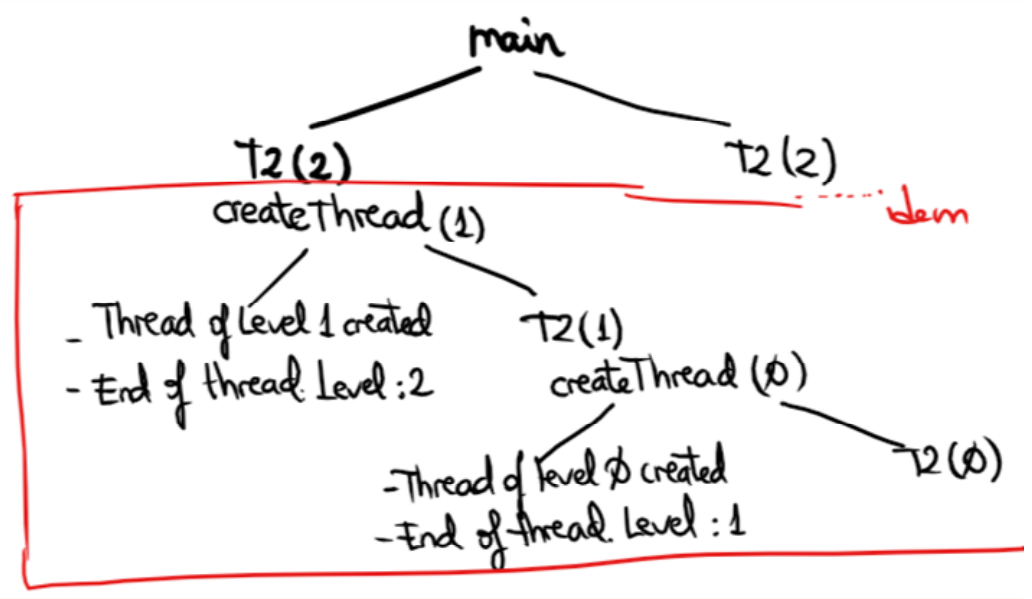



```
public class T2 extends Thread {
    private int level;
    public T2(int n){
       level = n;
    }
    public void createThread(int i) {
       T2 h = new T2(i);
       if (i>=1)
           h.start();
       System.out.println("Thread of level "+i+" created.");
    }
    public void run() {
       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();
    }
}
```

Se crean 6 hilos, pero sólo se ejecutan 4.

Actividad 2:

2) ¿Hay algún hilo con "nivel" igual a 0? ¿Y con valor 1?

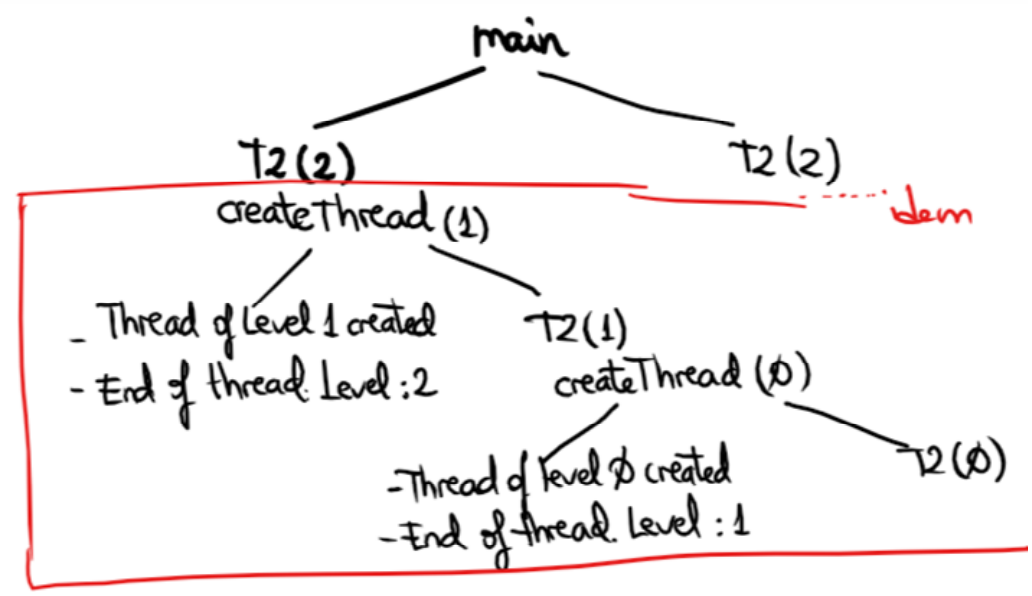



```
public class T2 extends Thread {
    private int level;
    public T2(int n){
       level = n;
    }
    public void createThread(int i) {
       T2 h = new T2(i);
       if (i>=1)
           h.start();
       System.out.println("Thread of level "+i+" created.");
    }
    public void run() {
       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();
    }
}
```

Se crean 2 hilos de nivel 0, pero NO se ejecutan.
Se crean 2 hilos de nivel 1, que SÍ se ejecutan

Actividad 2:

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

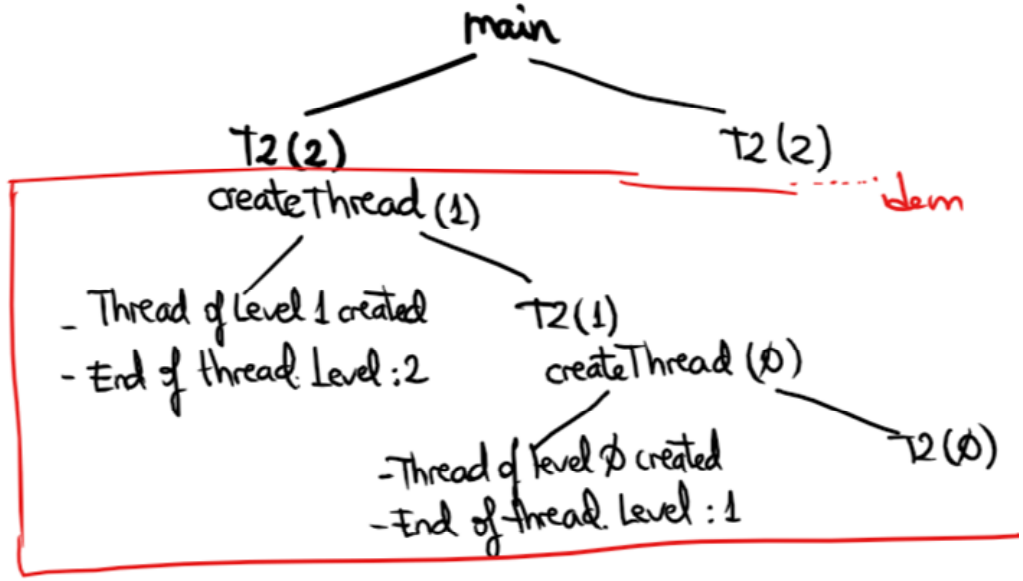



```
public class T2 extends Thread {
    private int level;
    public T2(int n){
       level = n;
    }
    public void createThread(int i) {
       T2 h = new T2(i);
       if (i>=1)
          h.start();
       System.out.println("Thread of level "+i+" created.");
    }
    public void run() {
       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();
    }
}
```

```
Thread of level 0 created.
Thread of level 1 created.
Thread of level 1 created.
End of thread. Level:2
Thread of level 0 created.
End of thread. Level:2
End of thread. Level:1
End of thread. Level:1
```

Actividad 2:

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




```
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();
    }
}
```

Se muestran los **mismos mensajes**, pero cada vez en un **orden distinto**, dependiendo de a qué hilo escoja el **planificador** para ponerlo en ejecución.

Actividad 2:

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?

Sólo se ejecutarían 2 hilos de nivel 2. En la ejecución de cada hilo siempre escribimos en el mismo orden los mensajes:

Thread of level 0 created.
End of thread. Level:1
Thread of level 1 created.
End of thread. Level:2

```
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.run();
       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();
    }
}
```

Como tenemos 2 hilos haciendo lo mismo concurrentemente, el resultado final entremezclaría los mensajes de ambos hilos, por lo que, **en cada ejecución** podríamos observar un **resultado final diferente**.

Actividad 3: Analizar el código

```
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) Para la creación de los hilos, ¿se utiliza una "clase con nombre" o una "clase anónima"? ¿Se extiende a la clase Thread o se implementa la interfaz Runnable?

Se utiliza una clase anónima, ya que se crean directamente con la orden new Thread(...), sin asignarla a ninguna clase determinada.

Actividad 3: Analizar el código

```
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();
          }   }   }
```

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

Se utiliza para consultar el nombre del hilo que se está ejecutando en ese momento.

Actividad 3: Analizar el código

```
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();
          }   }   }
```

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

```
main
executed by  MyThread 0
executed by  MyThread 1
executed by  MyThread 2
executed by  MyThread 8
executed by  MyThread 7
executed by  MyThread 6
executed by  MyThread 5
executed by  MyThread 3
executed by  MyThread 4
executed by  MyThread 9
```

En cualquier orden

Actividad 3: Analizar el código

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

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

Actividad 4: Analizar el código

```
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()} }

}}}
```

Cada hilo escribe en pantalla 3 veces su nombre

name=MyThread4
name=MyThread4
name=MyThread4
name=MyThread3
name=MyThread0
name=MyThread0
name=MyThread0
name=MyThread1
name=MyThread2
name=MyThread2
name=MyThread2
name=MyThread1
name=MyThread3
name=MyThread3
name=MyThread1

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

Actividad 4: Analizar el código

```
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()} }

}}}
```

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

Se crean 10 hilos, pero sólo se ejecutan 5.

Actividad 4: Analizar el código

```
public class ThreadName                    {



   public void printMsg() {
         System.out.println ("name=" +
          Thread.currentThread().getName());
   }
   public static void main(String[] args) {
         for ( int i = 0; i < 10; i++ ) {

            if (i<5)new Thread("MyThread" + i){
               public void run() {
                     for (int i = 0; i < 3; i++) printMsg();}
            }.start();
            else new Thread("MyThread" + i){
               public void run() {
                     for (int i = 0; i < 3; i++) printMsg();}
            };
}}}
```

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:

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

```
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());
    }
}
```

Cuando finalice los cálculos asignará al atributo result el valor “Calculation done”

Se suspende 10 seg.

Creamos el hilo “agent” e iniciamos su ejecución

Imprime el valor del atributo result, que debido a que el hilo agent espera 10 seg. antes de cambiar el valor de result, el hilo principal obtiene “Not calculated”

Actividad 5:

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. Utilice espera activa en el hilo principal.

```
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");
        while (agent.isAlive()) Thread.yield();
        System.out.println(agent.getResults());
    }
}
```

Actividad 5:

3) Reescriba el método main(), pero esta vez para implementar la espera no utilice espera activa, sino espera suspendida.

```
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");

        Try { agent.join(); } catch (InterruptedException e) {};

        System.out.println(agent.getResults());
    }
}
```

Actividad 6:

1) Indique qué se mostrará por pantalla. ¿Para qué se utilizan aquí los métodos Thread.isAlive(), Thread.interrupt() y Thread.join() ?

```
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!");}
    } }
```

El hilo *"t"* espera 4 seg y escribe por pantalla *"nombre de t"*: One …
… espera 4 seg y escribe por pantalla *"nombre de t"*: Four,
Pero si lo interrumpimos antes de que venza el tiempo del sleep, escribirá "I wasn't done!" y acabará porque la captura está fuera del bucle.

Actividad 6:

1) Indique qué se mostrará por pantalla. ¿Para qué se utilizan aquí los métodos Thread.isAlive(), Thread.interrupt() y Thread.join() ?

```
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!");
} }
```

escribe por pantalla *main*: Starting MessageLoop thread

Crea y lanza a ejecución al hilo "t"

escribe por pantalla *main*:
Waiting for MessageLoop thread to finish

Mientras el hilo "t" no finalice

escribe por pantalla *main*: Still waiting

Esperamos máximo 1 seg. a que t finalice

Si sobrepasa mi paciencia y t no ha terminado aún, entonces lo terminamos enviándole una interrupción.
Esperamos a que finalice t, para no volver a entrar en el bucle.

Actividad 6:

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

A. En el código BlueJ proporcionado, se puede pasar como argumento el valor patience, por defecto está puesto muy alto a 1 minuto (60 segundos) y no se llega a interrumpir al hilo t.

B. Si pasamos como argumento "8" le dará tiempo a main a escribir varias veces "Still waiting", al hilo t a escribir algún mensaje One, Two … antes de que main escriba Tired of waiting y t escriba I wasn't done.

A.

```
main:Starting MessageLoop thread
main:Waiting for MessageLoop thread to finish
main:Still waiting...
main:Still waiting...
Thread-15:One
main:Still waiting...
main:Still waiting...
Thread-15:Two
main:Still waiting...
main:Still waiting...
Thread-15:Three
main:Still waiting...
main:Still waiting...
Thread-15:Four
main:Finally!
```

B.

```
main:Starting MessageLoop thread
main:Waiting for MessageLoop thread to finish
main:Still waiting...
main:Still waiting...
Thread-14:One
main:Still waiting...
main:Still waiting...
Thread-14:Two
main:Tired of waiting!
Thread-14:I wasn't done!
main:Finally!
```