UD1. Introducción a la Programación Concurrente

Concurrencia y Sistemas Distribuidos

Objetivos de la Unidad Didáctica

- Revisión del concepto de programación concurrente
  - Conocer algunos ejemplos de aplicaciones típicamente concurrentes
- Conocer un lenguaje de programación que da soporte a la programación concurrente: Concurrencia en JAVA

Contenido

- Concepto de Programación Concurrente
- Programación concurrente en Java

Definición

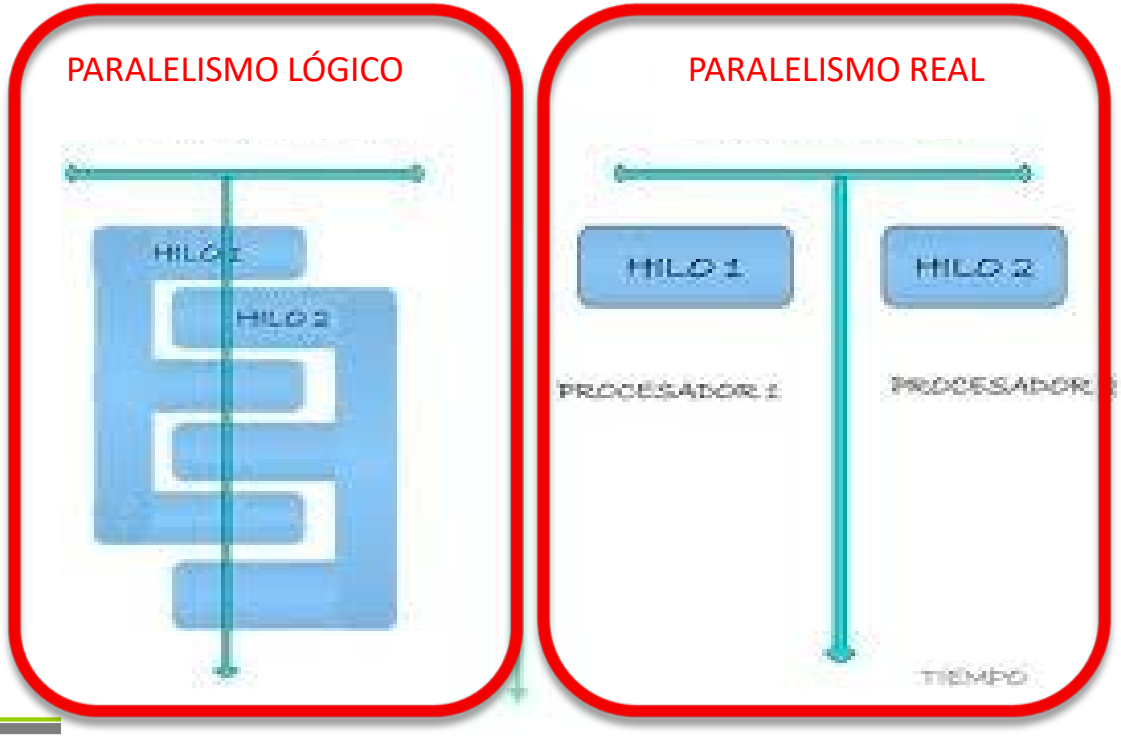
- Programa secuencial
  - Una única actividad (flujo de control único)

- **Programa concurrente**
  - Colección de actividades (hilos) que pueden ejecutarse en paralelo
  - Y ***cooperan*** para llevar a cabo una tarea común

  - Ejemplo: hilos auxiliares (típico en servidores)
    - El hilo principal lanza hilos auxiliares que realizan determinadas tareas de forma concurrente

¿Cómo obtener concurrencia?

- Podemos conseguir concurrencia de dos formas:
  - **Paralelismo lógico:** un procesador con multiprogramación
  - **Paralelismo real:** varios procesadores (ej. varios núcleos)
- Y podemos combinar ambos tipos de paralelismo.

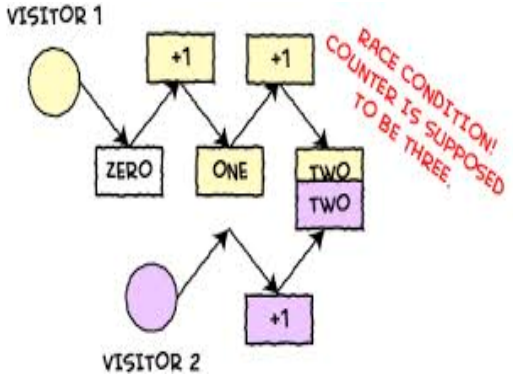


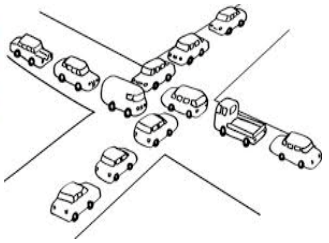
Ventajas e Inconvenientes de la Programación Concurrente

- Ventajas
  - Eficiencia: explota mejor los recursos máquina
  - Escalabilidad: puede extenderse a sistemas distribuidos
  - Gestión de las comunicaciones: explota la red. Ej: facilita el solape entre actividades de red y resto de actividades
  - Flexibilidad: resulta más fácil adaptar el programa a cambios en la especificación
  - Menor hueco semántico: en aquellos problemas que se definen de forma natural como una colección de actividades

Ventajas e Inconvenientes de la Programación Concurrente

- Inconvenientes: la programación concurrente **NO es fácil**
  - Programación delicada
    - Hay que conocer los problemas potenciales
      - Ejemplos:
        - Condiciones de Carrera
        - Interbloqueos





    - Hay que aplicar cierta disciplina en el desarrollo (existen soluciones)
  - Depuración compleja (no determinismo)

Ventajas e Inconvenientes de la Programación Concurrente

- Consecuencia de las ventajas anteriores
  - Mejora prestaciones y tolerancia a fallos. Útil en:
    - Sistemas Operativos
    - Sistemas de Gestión de Bases de Datos
    - Software científico de muy elevadas prestaciones
  - Mejora la interactividad y flexibilidad. Ej.:
    - Sistemas cliente/servidor, uso internet (ej P2P)
    - Dispositivos móviles (teléfonos, tablets, electrónica integrada en automóvil, ..)
    - Modelo de programación cercano al problema real (Sistemas de tiempo real, control de procesos, etc.)

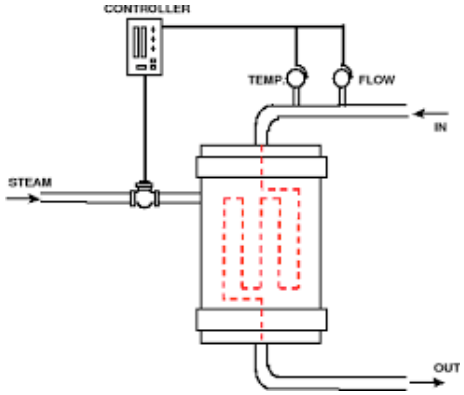
Aplicaciones de la Programación Concurrente

- Útil en prácticamente todos los tipos de aplicación

Una actividad independiente por cada solicitud

Una actividad por cada conexión

Una actividad por cada acción (movimiento, visión….)



Una actividad por cada aspecto a controlar (temperatura, presión…)

Una actividad por cada personaje, escenario, audio, rendering ….

Contenido

- Concepto de Programación Concurrente
- Programación concurrente en Java

Programación Concurrente en Java

- ¿Por qué **Java**?
  - Incorpora construcciones para **Programación Concurrente**
    - Los **hilos** forman parte del modelo del lenguaje
    - Dispone de primitivas para comunicación y sincronización entre hilos
    - Bibliotecas de soporte adicionales (*java.util.concurrent*) para desarrollo de aplicaciones complejas
  - Lenguaje conocido (experiencia, documentación, herramientas)
    - Muy difundido, y demandado por el mercado
    - Independiente de la plataforma (portable)
  - Facilidades para programación en red y distribuida
  - Versiones con soporte para tiempo real

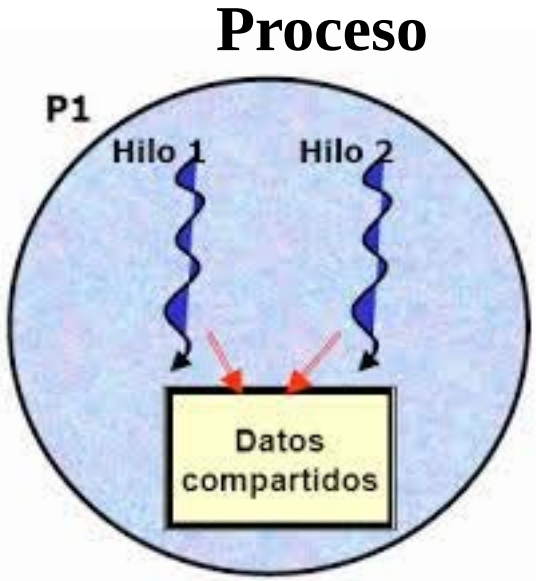
¿Qué es un hilo (en Java)?

- ***Thread*** (hilo) = contexto de ejecución, formado por:

CPU virtual (JVM)

Código (método ***void run()*** de *java.lang.Runnable*)

Datos (Atributos del hilo y de otros objetos que referencie)



¿Cómo crear hilos en Java?

- Alternativas:
  - Implementando la interfaz **Runnable**
    - Define método **run():** contiene el código a ejecutar por el hilo
  - Extendiendo la clase **Thread**
    - Implementa Runnable
    - Ofrece métodos para gestionar los hilos.

```
public class HolaRunnable implements
Runnable {
    public void run()
    { System.out.println("Hola mundo!");    }

   public static void main(String args[])
   { (new Thread(new HolaRunnable())).start();
    }
}
```

```
public class HolaThread extends Thread {
   public void run()
   { System.out.println("Hola mundo!");    }

  public static void main(String args[])
  {  (new HolaThread()).start();
   }
 }
```

¿Cómo crear hilos en Java? → **Opción 1**

- Opción "**clase con nombre**", si se requiere declarar varias instancias:

	Clase con nombre

Implementando Runnable	`public class H implements Runnable {`
	`  public void run() {`
	`  System.out.println("ejecuta hilo");`
	`  }`
	`}`
	`Thread t= new Thread(new H());`
	`t.start();`

Extendiendo Thread	`public class H extends Thread {`
	`  public void run() {`
	`  System.out.println("ejecuta hilo");`
	`  }`
	`}`
	`H t= new H();  t.start();`

¿Cómo crear hilos en Java? → **Opción 2**

- Opción " **clase anónima"**, si sólo se requiere una instancia:

	Clase anónima

Implementando Runnable	`new Thread(new Runnable() {`
	`  public void run() {`
	`  System.out.println("ejecuta hilo");`
	`  }`
	`}).start();`

Extendiendo Thread	`new Thread() {`
	`  public void run() {`
	`  System.out.println("ejecuta hilo");`
	`  }`
	`}.start();`

- **IMPORTANTE**: Si la clase ya extiende de otra, entonces solamente se puede implementar *Runnable* (Java no soporta herencia múltiple)

Java.- ¿Cómo ejecutar hilos en Java?

- La ejecución del hilo se arranca con **start()**
  - Lanza la ejecución del hilo.
  - Ese hilo ejecuta su método *run*()
  - **Error típico:** invocar *run()* en lugar de *start()*

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

   public static void main(String[] argv)
   {
          for (int i=0; i<3; ++i)
                new T(i).start();
    }
   public void run() {
    for (int i=0; i<5; ++i) {
         echo("Hilo "+n +" iteración "+i);
         retardo((n+1)*1000);
         }
    echo("Fin hilo "+n);
   }}
```

Nota: métodos auxiliares usados en estos ejemplos

- Para simplificar el código, asumiremos que hemos definido los métodos siguientes:

```
// suspende la ejecución durante ms milisegundos
void retardo(int ms) {
      try {
            sleep(ms);
      } catch (InterruptedException ie) {
            ie.printStackTrace();
      }
}

// muestra texto en pantalla
void echo (String s) {
      System.out.println(s);
}
```

Java.- ¿Cómo identificar hilos en Java?

- Al crear un hilo le podemos asociar un nombre:
  - El constructor de **Thread** admite un nombre para el hilo

```
new T(i).start();
```

- En cualquier momento podemos asociarle un nombre:
  - Utilizando método **setName(String name)**

```
t.setName("thread" + i);
```

- Identificador accesible con **getName()** sobre cualquier objeto *Thread*

```
t.getName();
```

```
Thread.currentThread().getName();
```

Java.- ¿Cómo identificar hilos en Java?

Ejemplo

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

Java.- A modo de ejemplo….

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

¿Cuántos hilos se crean?

¿Cuántos hilos se ejecutan?

¿Cómo se identifica cada hilo?

Java.- Pausar la ejecución de un hilo con *sleep*

- **Thread.sleep(long millis)**
  - Causa la suspensión de la ejecución del hilo por el tiempo indicado (en milisegundos).
  - Este método lanza la excepción ***InterruptedException*** cuando el hilo suspendido es interrumpido por otro hilo.

```
//Ejemplo: pausar el hilo durante 4 segundos
try {
        Thread.sleep(4000);
    } catch (InterruptedException e) {
        System.out.println("Caught InterruptedException: "
                + e.getMessage() );
    }
```

Java.- Interrumpir un hilo

- **Thread.interrupt()**
  - Operación que reactiva a un hilo que estaba suspendido
  - El hilo interrumpido recibe una **InterruptedException**

```
class Inter extends Thread {

    public void run() {
        System.out.println("Empezando...");
        try {
            sleep(10000); // Esperamos hasta 10 segs.
        } catch (InterruptedException e) {
            System.out.println("Interrumpido.");
        }
        System.out.println("Terminado.");
    }

    public static void main(String[] args) {
        Inter hi = new Inter();
        hi.start();
        System.out.println("Enviando interrupción...");
        hi.interrupt();
        System.out.println("Enviada.");

    }
}
```

TRAZA:

```
Enviando interrupción...
Enviada.
Empezando...
Interrumpido.
Terminado.
```

Java.- Esperar a la terminación de un hilo

- **Thread.join()**
  - Permite a un hilo esperar a la terminación de otro hilo

```
t.join();
```

El hilo actual espera a que el hilo t termine

  - Se puede especificar un tiempo máximo de espera, con **Thread.join(long millis)**

  - Se puede interrumpir al hilo que espera, con el método **Thread.interrupt()**

Java.- Otros métodos de la clase **Thread**

- **Thread.currentThread()**
  - Devuelve referencia al objeto thread que se está ejecutando en ese momento
- **Thread.isAlive()**
  - Devuelve TRUE si el hilo se ha iniciado y todavía no ha terminado; FALSO en caso contrario.
- **Thread.yield()**
  - Abandona voluntariamente el procesador, cediendo la ejecución a otro hilo preparado

Ejemplo

```
class Inter extends Thread {
    public void run() {
        System.out.println("Empezando..." +  currentThread().getName());
        yield();  //cedemos CPU al otro hilo
        try {
            sleep(10000); // Esperamos hasta 10 segs.
            System.out.println("Soy " + currentThread().getName()
                    + "... y sigo vivo?" + currentThread().isAlive());
        } catch (InterruptedException e) {
            System.out.println("Interrumpido." + currentThread().getName() );
        }
        System.out.println("Terminado." + currentThread().getName() );
    }

    public static void main(String[] args) {
        Inter hi1 = new Inter();
        Inter hi2 = new Inter();
        hi1.setName("Worker 1");
        hi2.setName("Worker 2");
        hi1.start();
        hi2.start();
        System.out.println("Enviando interrupción...");
        hi1.interrupt();
        System.out.println("Enviada interrupción.");
        try {
        hi1.join();
        hi2.join();
        } catch(InterruptedException e){
            System.out.println("Hilos interrumpidos mientras esperábamos su finalización"
              + e.getMessage());
        }
    }
}
```

TRAZA:

Enviando interrupción...
Enviada interrupción.
Empezando...Worker 1
Empezando...Worker 2
Interrumpido: Worker 1
Terminado: Worker 1
Soy Worker 2... y sigo vivo?true
Terminado: Worker 2

Java Threads.- Notación Lambda

- Las expresiones Lambda son funciones anónimas, y pueden ser utilizadas donde el tipo aceptado sea una interfaz funcional (interfaz con un solo método abstracto).
  - La interfaz Runnable es una interfaz funcional con un único método abstracto run().
- Su sintaxis básica:

  (parámetros) -> {cuerpo}

  - El operador lambda (->) separa la declaración de parámetros de la declaración del cuerpo de la función.
  - Parámetros:
    - Cuando no se tienen parámetros, o cuando se tienen dos o más, es necesario utilizar paréntesis.
  - Cuerpo:
    - Cuando el cuerpo de la expresión lambda tiene una única línea no es necesario utilizar las llaves.

Java Threads.- Notación Lambda

- Se puede simplificar la sintaxis de la creación de hilos que implementa Runnable, con expresiones Lambda:

```
Runnable task1 = new Runnable(){
  public void run(){
    System.out.println("Hilo #1");
  }
};
```

```
Runnable task1 = () -> { System.out.println("Hilo #1"); };
```

```
Runnable task1 = () -> System.out.println("Hilo #1");
```

```
Thread thread1 = new Thread(task1); thread1.start();
new Thread(task1).start();
```

Java Threads.- Notación Lambda

- Otros ejemplos:

```
Thread thread1 = new Thread(new Runnable()
{
  public void run(){
     System.out.println("Hilo #1");
  }
});
thread1.start();
```

```
Thread thread1 = new Thread(() -> {System.out.println("Hilo #1");});
thread1.start();
```

```
new Thread(() -> System.out.println("Hilo #1")).start();
```

```
new Thread(() ->
System.out.println(Thread.currentThread().getName(),"Hilo #1").start();
```

Resultados de aprendizaje de la Unidad Didáctica

- Al finalizar esta unidad, el alumno deberá ser capaz de:
  - Describir las ventajas e inconvenientes de la programación concurrente frente a la programación secuencial.
  - Describir la gestión de hilos de ejecución en Java que permite implantar programas concurrentes.