# Unit 1.- Introduction to Concurrent Programming

Concurrency and Distributed Systems

# Teaching Unit Objectives

- Understand the concept of concurrent programming
  - Discuss why it is advantageous compared to sequential programming.
  - Identify problems associated with this type of programming.

- Know some examples of typical concurrent applications.

- Know a programming language that supports concurrent programming: Concurrency in Java
  - Get in touch with Java language mechanisms that give support to concurrent programming.

# Bibliography

- Java concurrency:
  - *Processes and Threads; Defining and starting a thread* http://docs.oracle.com/javase/tutorial/essential/concurrency/
  - Reduced version in PoliformaT: Java Concurrency.pdf
- Videos:
  - Standford University - Sequential Programming vs. Concurrent Programming (44:37)
    - http://freevideolectures.com/Course/2260/Computer-Science-III-Programming-Paradigms/14
- In Spanish:
  - Chapter 1, course book ("Concurrencia y Sistemas Distribuidos")

# Content

- What is Concurrent Programming?
- Concurrent Programming in Java

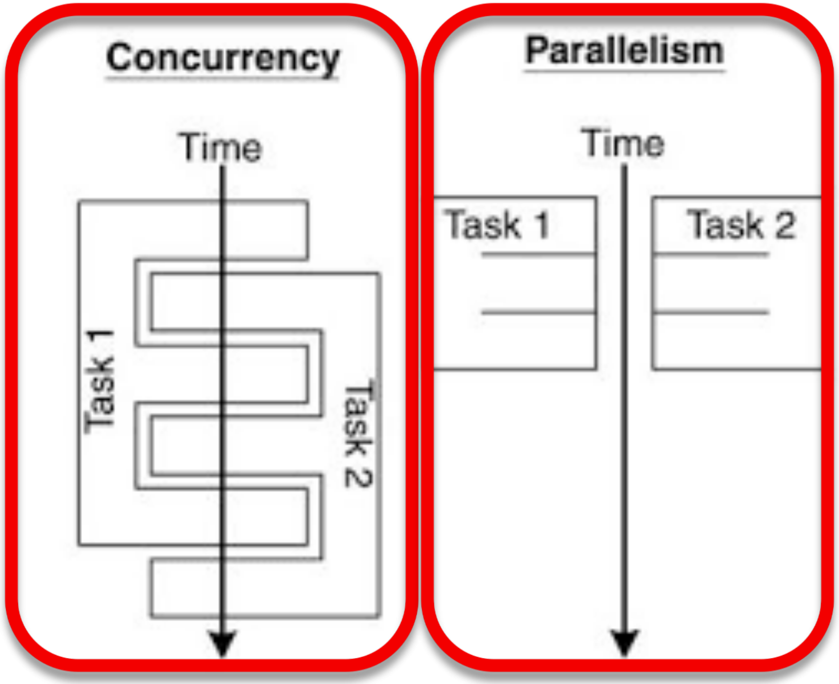
# What is a Concurrent Program?

- **Concurrent Program**
  - Collection of activities (threads) that can run in parallel
  - And **cooperate** to perform a common task

Which one is concurrent?

# How can you obtain concurrency?

- You can achieve concurrency by two means:
  - **Logical Parallelism**: one processor with multiprogramming
  - **Real Parallelism:** several processors (eg. multiple cores)

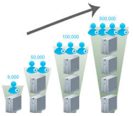


Both types can be combined

# Advantages and Disadvantages of Concurrent Programming

- <u>Advantages</u>

  - Efficiency: exploits better the machine resources



  - Scalability: it can be extended to distributed systems

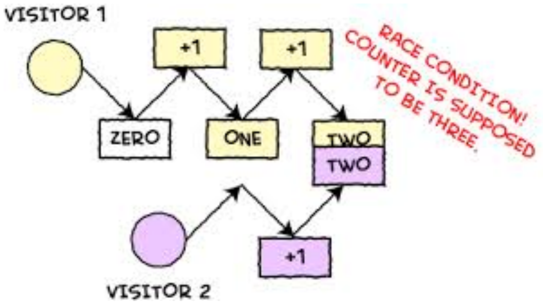
  - Communication management: exploits the network.
    - Facilitates the overlap between network activities and other activities

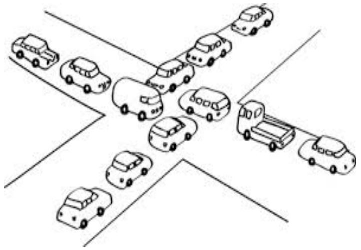
  - Flexibility: it is easier to adapt the program to changes in the specification

  - Minor semantic gap: in those problems that are naturally defined as a collection of activities

# Advantages and Disadvantages of Concurrent Programming

- Disadvantages.- Concurrent Programming **is NOT easy**
  - Makes programming more difficult
    - □ Know the potential problems of Concurrent Programming
    - □ Examples:
      - □ Race-conditions
      - □ Deadlocks

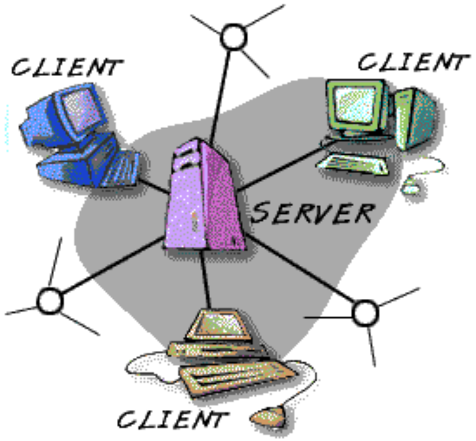




    - □ And apply some discipline in the software development (there are solutions)
  - Complex debugging (non-determinism)

# Applications of Concurrent Programming

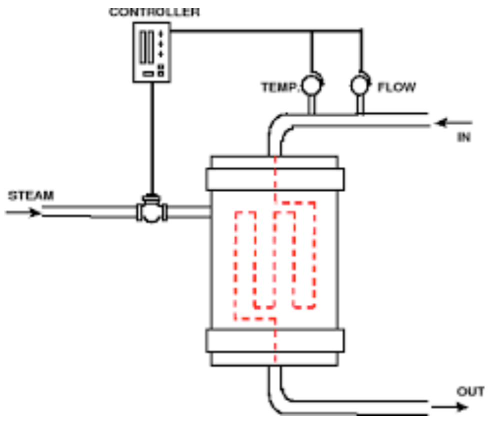
- Useful in practically all types of applications



*An independent activity for each client*

*An activity for each connection*

*An activity for each action (movement, vision….)*



*An activity for each feature to be controlled (temperature, pressure, …)*

*An activity for each character, scene, audio, rendering…*

# Content

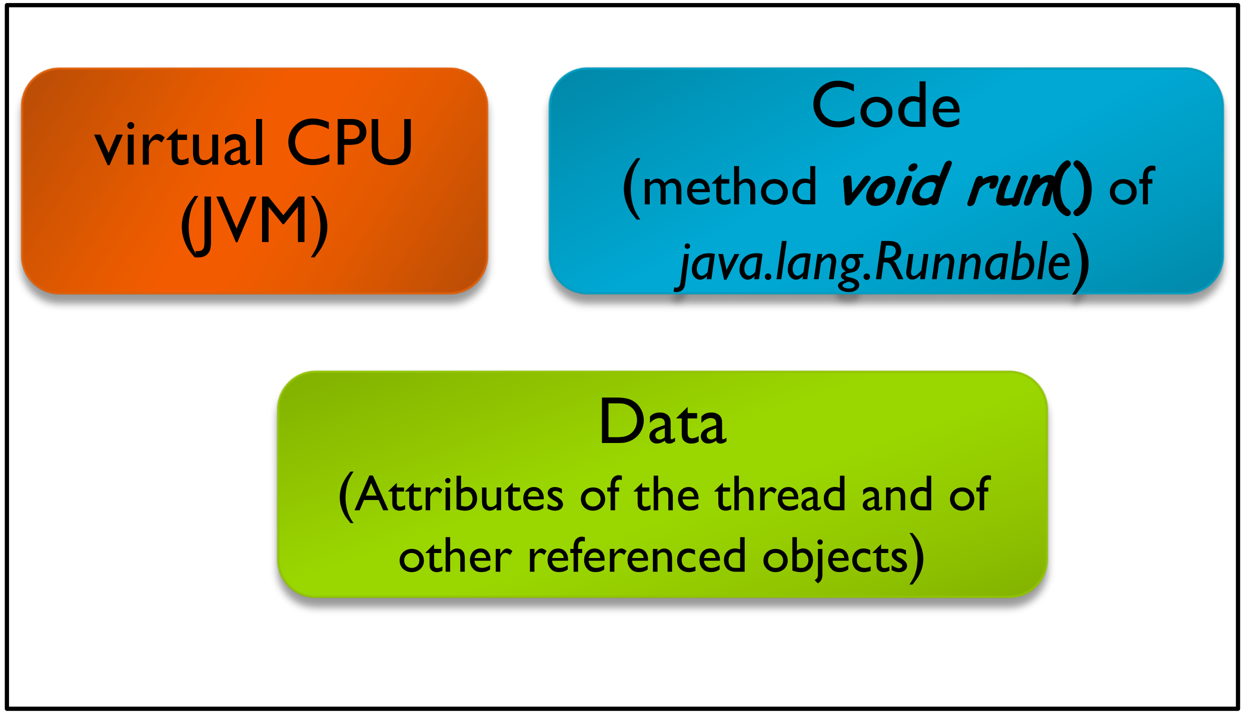
- What is Concurrent Programming?
- Concurrent Programming in Java

# Programming Language that we will use

- Why **Java**?
  - Java includes constructions for **Concurrent Programming**
    - **Threads** are part of the language model
    - Additional support libraries (i.e. *java.util.concurrent*) for developing complex applications
  - Well-known language (experience, documentation, tools)
    - Widespread and demanded by the market
    - Platform independent (portable)
  - Facilities for network and distributed programming
  - Versions with support for real-time programming

# Java.- Threads

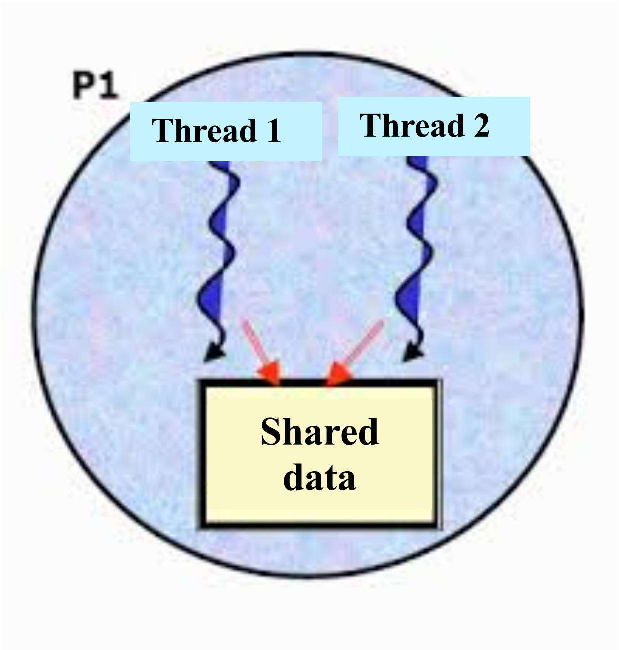
- Thread = execution context, composed of:



virtual CPU (JVM)

Code (method ***void run()*** of *java.lang.Runnable*)

Data (Attributes of the thread and of other referenced objects)



# How can you create threads in Java?

- Alternatives:
  - Implementing **Runnable** interface
    - Defines **run()** method: contains the code to be executed by the thread
  - Extending **Thread** class
    - Implements Runnable
    - It offers methods to manage threads

```
public class HelloRunnable implements
Runnable {
    public void run()
    { System.out.println(”Hello world!");     }

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

```
public class HelloThread extends Thread {
   public void run()
   { System.out.println(”Hello world!");     }

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

# Java.- How to create threads? → Option 1

- Option “**class with name**”, if you need to declare several instances:

| | Class with name |
|---|---|
| **Implementing Runnable** | **public class H implements Runnable**<br>**{ public void** run**() {**<br>System.out.println**(**”execute thread"**);**<br>**}**<br>**}**<br>Thread t**= new** Thread**(new** H**());**<br>**t.start();** |
| **Extending Thread** | **public class H extends Thread {**<br>**public void** run**() {**<br>System.out.println**(**”execute thread"**);**<br>**}**<br>**}**<br>H t**= new** H**(); t.start();** |

# Java.- How to create threads → Option 2

- Option “**anonymous class**”, if you only need one instance:

| | **Anonymous class** |
|---|---|
| **Implementing Runnable** | `new Thread(new Runnable() {`<br>`   public void run() {`<br>`     System.out.println(”execute thread");`<br>`   }`<br>`}).start();` |
| **Extending Thread** | `new Thread() {`<br>`   public void run() {`<br>`      System.out.println(”execute thread");`<br>`   }`<br>`}.start();` |

- **IMPORTANT**: If your class is already extending another, then you can only implement *Runnable* (Java does not support multiple inheritance)

# How can you execute threads in Java?

- Thread execution begins with **t.start**()
  - Then the thread executes its *run*() method.
  - **Typical error**: to invoke *t.run()* instead of *t.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(”Thread"+n +" iteration"+i);
                        delay((n+1)*1000);
                }
                echo(”End of thread "+n);
        }
}
```

# Java.- Auxiliary Methods used

- To simplify the code, we assume that we have defined the following methods:

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

// shows text in the screen
void echo (String s) {
      System.out.println(s);
}
```

# Java.- How can you identify threads in Java?

- When creating a thread, you can give it a name:
  - The **Thread constructor** accepts a name for the thread

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

- At anytime you can give it a name:
  - Using **setName(String name)** *method*

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

- This name is accessible with ***getName*()** method on any *Thread* object

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

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

# Java.- How can you identify threads in Java?

## Example

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

# Java.- Let’s see an example

```
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(”Thread" + i);
          if (i<5) tt.start();
     }
}}
```

How many threads are created?

How many threads are executed?

How are threads identified?

# Java.- Pause execution of a thread with ***sleep***

- **Thread.sleep(long millis)**
  - Causes the suspension of a thread during the given time (in milliseconds)
  - This method launches an ***InterruptedException*** when the suspended thread is interrupted by another thread.

```
//Example: suspends the thread during 4 seconds
try{
        Thread.sleep(4000);
    } catch (InterruptedException e) {
          System.out.println(”Caught InterruptedException: ”
                  + e.getMessage() );
    }
```

# Java.- Interrupt a thread

- **Thread.interrupt()**
  - It reactivates a thread that was suspended.
  - The interrupted thread receives an ***InterruptedException***

```
class Inter extends Thread {

    public void run() {
        System.out.println(”Starting...");
        try {
            sleep(10000); // Wait for 10 seconds
        }  catch (InterruptedException e) {
                System.out.println(”Interrupted.");
        }
        System.out.println(”Finished.");
    }

    public static void main(String[] args) {
        Inter hi = new Inter();
        hi.start();
        System.out.println(”Sending interruption...");
        hi.interrupt();
        System.out.println(”Sent.");
    }
}
```

**TRACE:**

```
Sending interruption...
Sent.
Starting...
Interrupted.
Finished.
```

# Java.- Wait for a thread to finish

- **Thread.join()**
  - Allows a thread to wait the end of another thread
    - The current thread waits until thread t finishes

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

  - You can give a maximum waiting time, using:

    **Thread.join(long millis)**

  - You can interrupt the waiting thread, using:

    **Thread.interrupt()**

# Java.- Other methods of **Thread** class

- **Thread.currentThread()**
  - Returns a reference to the thread object that is currently running
- **Thread.isAlive()**
  - Returns TRUE if the thread has started and has not finished yet; FALSE otherwise.
- **Thread.yield()**
  - Voluntarily leaves the processor, so the scheduler can select another thread for being executed.

# Example

```
class Inter extends Thread {
   public void run() {
      System.out.println(”Starting..." +  currentThread().getName());
      yield();  //cedemos CPU al otro hilo
      try {
          sleep(10000); // Esperamos hasta 10 segs.
          System.out.println(”I am “ + currentThread().getName()
                + “… and still alive? ” + currentThread().isAlive());
       } catch (InterruptedException e) {
          System.out.println("Interrupted:” + currentThread().getName() );
       }
       System.out.println(”Finished:” + 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(”Sending interruption...");
       hi1.interrupt();
       System.out.println(”Interruption sent.");
       try {
       hi1.join();
       hi2.join();
      } catch(InterruptedException e){
          System.out.println(" Threads interrupted while waiting for completion "
            + e.getMessage());
      }
   }
}
```

**TRACE:**

```
Sending interruption...
Interruption sent.
Starting...Worker 1
Starting...Worker 2
Interrupted: Worker 1
Finished: Worker 1
I am Worker 2... and still alive? true
Finished: Worker 2
```

# Learning results of this Teaching Unit

- At the end of this unit, you should be able to:
  - Describe what is concurrent programming
  - Describe its advantages and disadvantages.
  - Describe how to create, execute and identify threads in Java.