# Repaso parte 1 - Concurrencia y multihilo

Los procesadores y los sistemas operativos están diseñados para ejecutar **varios procesos de forma concurrente** (simultáneamente o de forma paralela), y para que cada uno de esos procesos pueda ejecutar **varios hilos.**

- **Un proceso** es un programa o aplicación que se ejecuta en un espacio de memoria propio y de forma independiente. Un sistemas operativo multitarea tiene la capacidad de ejecutar más de un proceso a la vez.
- **Un hilo** es la unidad de procesamiento mínima que puede ser ejecuta en un sistema operativo. Por lo tanto, un proceso tendra ente uno y varios hilos de ejecución.

# Aplicación de un solo hilo

Cuando ejecutamos cualquier aplicación, por pequeña que sea, ya estamos creando un hilo de ejecución.

```
public class Ejemplo{
    public static void main(String[] args) {
        System.out.println("Single Thread");
    }
}
```

Las ventajas de este tipo de aplicaciones son que son **simples de ejecutar**, y que su mantenimiento o **depuración es relativamente facil**.

# Aplicacion con varios hilos

Este tipo de aplicaciones ejecutaran dos o más hilos de forma simultanea y maximizaran el uso de la CPU. El hecho de que dos hilos se ejecuten al mismo tiempo se conoce como **concurrencia**.

En Java, un hilo es una instancia de la clase java.lang.Thread, y puede ser creada de dos formas distintas:

## Método 1 - Extendiendo la clase Thread:

La clase Thread es una implementacion del metodo Runnable y el método run() que implementa esta vacio. Para crear un hilo creamos un objeto que extienda la clase Thread y sobreescribimos su método run().

```
public class MiPrimerHilo extends Thread{
    @Override
    public void run() {
        System.out.println("Soy un hilo");
    }
}
```

La clase Main que ejecuta el hilo:

```
public class Main{
    public static void main(String[] args) {
        MiPrimerHilo miHilo = new MiPrimerHilo();
        miHilo.start();
    }
}
```

La ejecución de este código creara 2 hilos. El primero el de la propia clase Main, que con el comando .start() creara un segundo hilo correspondiente a la clase MiPrimerHilo

## Método 2 - Implementando la interfaz Runnable

Utilizando una instancia de un objeto Runnable para crear un Thread utilizando su constructor. Primero creamos las clase que implementa Runnable:

```
public class MiPrimerRunnable implements Runnable{
    @Override
    public void run() {
        System.out.println("Soy un hilo");
    }
}
```

La clase Main que ejecuta el hilo:

```
public class Main{
    public static void main(String[] args) {
        Thread miHilo= new Thread(new MiPrimerRunnable());
        miHilo.start();
    }
}
```

Notese la diferencia para crear la instancia de un hilo en este método. Seguimos utilizando la clase Thread, pero utilizamos el constructor enviandole la implementación del Runnable que hemos creado.

### ¿Pero cuál de los dos métodos utilizar?

En esta primera unidad podemos usar indistintamente cualquier de los dos métodos. Sin embargo, el objeto Runnable nos proporciona más flexibilidad ya que separa la tarea a realizar del objeto que ejecuta la tarea. Esta separación es útil y aplicable al utilizar APIs o Framework de concurrencia.

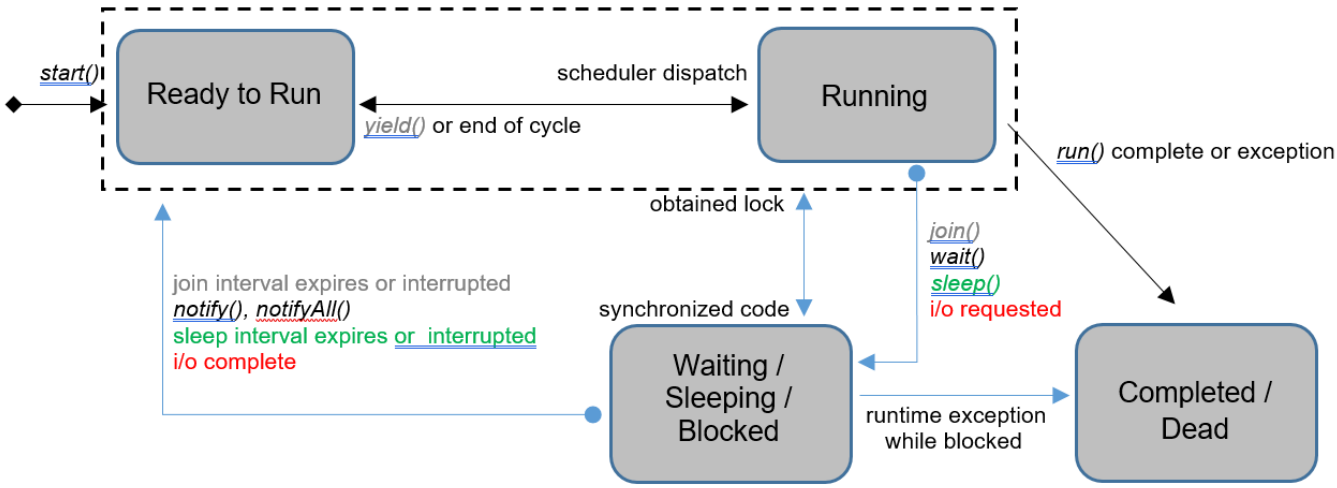
### Como crear un hilo mediante funciones LAMBDA

Podemos simplificar las lineas de codigo para la ejecución de hilos:

```
public class Main{
    public static void main(String[] args) {
        new Thread(() -> {
            System.out.println("Soy un hilo");
        }).start();
    }
}
```

# Ciclo de vida de los hilos

Una vez inicializamos el hilos o hilos con el método .start() **no podemos predecir** en qué momento se ejecutarán las instrucciones definidas. Un hilo detendrá su ejecución cuando se llegue al final de la función run().




***Thread States in Java***

- **Ready to run:** El hilo ha sido creado, pero permanecera en este estado hasta que se invoque el método start().
- **Running/Runnable:** El hilo esta listo para ser ejecutado y su control depende del planificador de tareas que decidira cuando se ejecuta. Depende de la CPU/SO.
- **Waiting/Sleeping/Blocked:** El hilo ha empezado su ejecución pero por alguna razon su ejecución se ha pausado y será despertado para continuar mas adelante.
- **Completed/Dead:** El hilo ha terminado su ejecución.

## Dormir un hilo unos milisegundos.

La ejecución de cualquier aplicación, hasta la de un solo hilo, se puede dormir durante unos milisegundos utilizando sleep();

```
public class Ejemplo{
    public static void main(String[] args) {
```

```
            System.out.println("Single Thread");
            Thread.sleep(1000);
            System.out.println("Ha pasado un segundo.");
        }
    }
```

En la ejecución de este hilo pasara 1 segundo entre un mensaje y otro. Tambien se puede utilizar la función **Thread.wait(1000)** y en este caso tendra un efecto similar, aunque mas adelante aprenderemos las diferencias.

## Dormir un hilo hasta que otro hilo termine

Se puede dormir un hilo hasta que otro hilo termine. Por ejemplo, podríamos crear 2 hilos que realicen una serie de cálculos y hacer que el hilo principal (el que los ha creado) espere hasta que estos 2 hilos terminen, y mostrar el resultado final (el resultado final depende del resultado de los hilos).

Para ello se utiliza el método **join()**, que al igual que sleep() requiere manejar la excepción InterruptedException.

```
public class Main {
    public static void main(String[] args) {
        Thread hilo1 = new Thread(new WaitRunnable());
        Thread hilo2 = new Thread(new WaitRunnable());
        System.out.println("Inicio de la ejecución");
        hilo1.start();    //incio el primer hilo
        hilo2.start();    //inicio el segundo hilo
        try {
            hilo1.join(); //Hasta que no termine el hilo1 el hilo principal parara
            hilo2.join(); //Hasta que no termine el hilo2 el hilo principal parara
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
        System.out.println("Fin de la ejecución"); // Este mensaje se mostrará al
final
    }
```

```
    private static class WaitRunnable implements Runnable {
        @Override
        public void run() {
            try {
                Thread.sleep(5000);
                System.out.println("Soy un hilo");
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
    }
}
```

Sino realizamos esta coordinación con el metodo join() no podriamos garantizar que el mensaje "Fin de la ejecución" fuera el ultimo mensaje en mostrarse.

## Gestión de prioridades

Cada hilo tiene un prioridad (representanda por un numero entero de 1 a 10), y por defecto es heredada del hilo padre que lo crea. El planificador elige qué hilo ejecutar en cada momento utilizando la prioridad. Se pueden utilizar los métodos setPriority() para modificar la prioridad o getPriority() para conocerla. Las constantes:

```
Thread.MIN_PRIORITY //es igual a 1
Thread.NORM_PRIORITY //es igual a 5
Thread.MAX_PRIORITY //es igual a 10
```

A la hora de programar hilos con prioridades tenemos que tener en cuenta que su comportamiento **no está garantizado** y dependerá de más factores.

## ¿Cómo interrumpir o parar un hilo?

No puedes forzar la parada de un hilo, pero puedes interrumpirlo con el método interrupt(). Este método interrumpe el hilo, si el hilo está dormido o esperando (join) la muerte de otro hilo, y lanza la excepción InterruptedException.

Si el hilo está dormido o esperando a otro hilo, el atributo interrupted del hilo sera borrado, y el metodo isInterrupted() devolverá false. Por lo tanto continuará su ejecución. Capturar la InterruptedExcepcion no será suficiente y una solucion podria ser la siguiente:

```
public class EjemploInterrupt implements Runnable {
    @Override
    public void run() {
        while(!Thread.currentThread().isInterrupted()){
            // En esta parte el programa realiza unas operaciones
            try {
                Thread.sleep(5000);
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
            }
            //En este parte el programa realiza otras operaciones
        }
    }
}
```