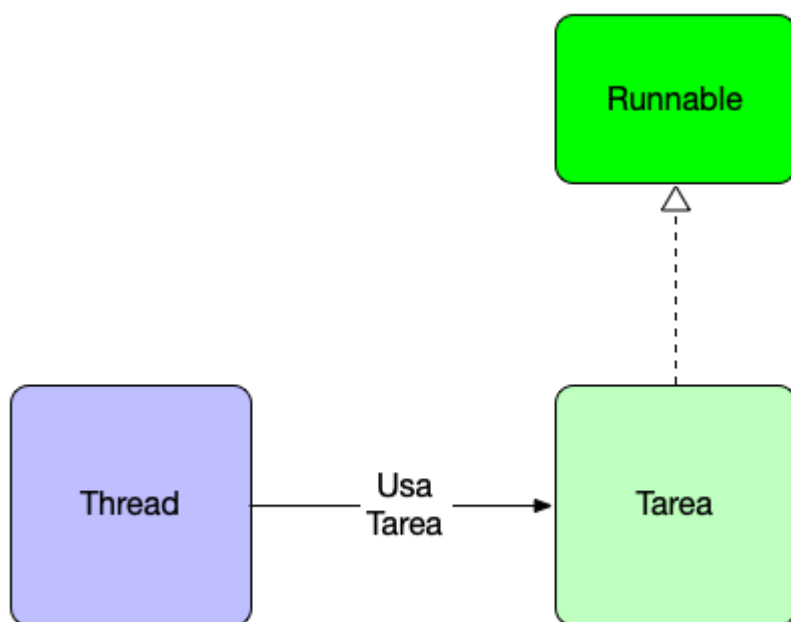


Java Thread es una de las clases más clásicas del API de Java y es la encargada de ejecutar tareas en paralelo creando nuevos hilos de ejecución. Es una clase muy sencilla de utilizar a nivel básico pero en situaciones en los que se necesita temas de concurrencia complejos tiene su miga. Vamos a ver una introducción a esta clase y como usar sus métodos fundamentales. Un Java Thread esta diseñado apoyándose en el concepto de composición y permite recibir una tarea que implemente el interface Runnable.



Una vez tenemos claro la estructura de clases es momento de implementar:

```
package com.arquitecturajava;

public class Tarea implements Runnable {

    private String nombre;
    public String getNombre() {
        return nombre;
    }
}
```

```

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public Tarea(String nombre) {
        super();
        this.nombre = nombre;
    }

    @Override
    public void run() {
        for (int i=0; i<5;i++) {
            System.out.println("tarea "+ nombre);
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
    }
}

```

Java Thread y Tareas

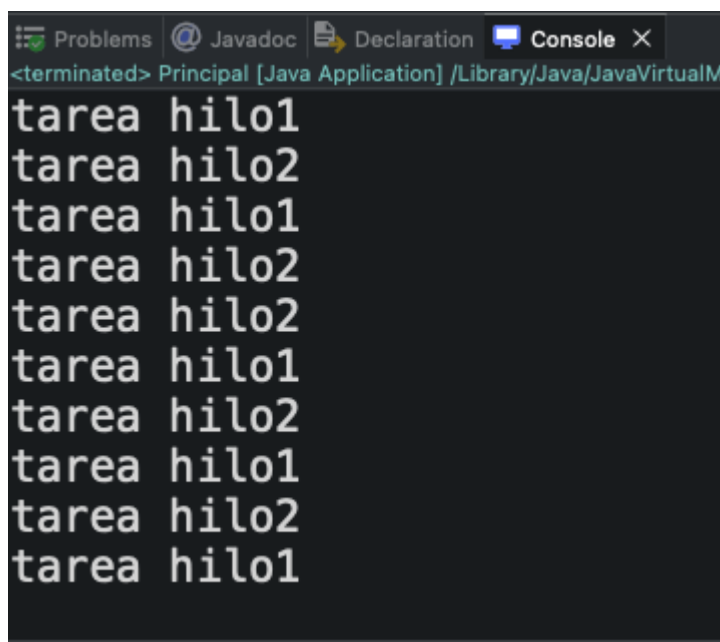
En este caso estamos ante una de las implementaciones más sencillas ya que sobreescribimos el método run y realizamos un bucle for con 5 iteraciones que se para en cada iteración 1 segundo. Además de realizar esta operación la clase dispone de una propiedad nombre que nos puede ser útil para identificar la tarea. No tiene mucho más . Si

construimos el programa principal podemos diseñar dos Java Thread cada uno de ellos ejecutará una tarea diferentes y se irán alternando.

```
package com.arquitecturajava;
```

```
public class Principal {  
  
    public static void main(String[] args) {  
        Thread hilo= new Thread (new Tarea("hilo1"));  
        Thread hilo2= new Thread (new Tarea("hilo2"));  
        hilo.start();  
        hilo2.start();  
    }  
}
```

El resultado de la construcción de estas dos tareas es un programa con dos hilos de ejecución que alterna de forma intermitente cada una de ellas.



```
<terminated> Principal [Java Application] /Library/Java/JavaVirtualM  
tarea hilo1  
tarea hilo2  
tarea hilo1  
tarea hilo2  
tarea hilo2  
tarea hilo1  
tarea hilo2  
tarea hilo1  
tarea hilo2  
tarea hilo1
```

Este es el ejemplo de Hola Mundo de programación concurrente . Podemos añadir más

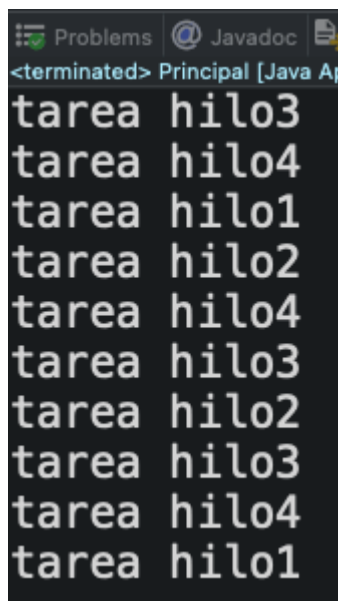
tareas y ver como todas ellas se van alternando.

```
package com.arquitecturajava;

public class Principal {

    public static void main(String[] args) {
        Thread hilo= new Thread (new Tarea("hilo1"));
        Thread hilo2= new Thread (new Tarea("hilo2"));
        Thread hilo3= new Thread (new Tarea("hilo3"));
        Thread hilo4= new Thread (new Tarea("hilo4"));
        hilo.start();
        hilo2.start();
        hilo3.start();
        hilo4.start();
    }
}
```

En este caso las 4 tareas se irán alternando de forma más o menos equitativa aunque no hay nada que lo asegure a nivel de la maquina virtual. Puede ser que la maquina priorice un Java Thread sobre otro.



```
<terminated> Principal [Java Ap...  
tarea hilo3  
tarea hilo4  
tarea hilo1  
tarea hilo2  
tarea hilo4  
tarea hilo3  
tarea hilo2  
tarea hilo3  
tarea hilo4  
tarea hilo1
```

Conclusiones

Acabamos de hacer un sencillo ejemplo de manejo de Java Thread . Recordemos que cualquier ordenador hoy soporta un número elevado de ejecución de tareas concurrentes y el uso de Java Threads nos puede ayudar en el procesado de estas.

Otros artículos relacionados

- [Java Calendar y el manejo de fechas](#)
- [Java ArrayList for y sus opciones](#)
- [Java Array to Stream y sus opciones](#)
- [Java Runnable](#)