

**Instituto Politécnico Nacional**

**Escuela Superior de Cómputo**

**Materia: Sistemas Distribuidos**

**Profesor: Carreto Arellano Chadwick**

**Grupo: 7CM1**

**Práctica 1. Procesos e Hilos**

**Valdés Castillo Diego - 2021630756**

**Fecha de Entrega: 24/02/2025**

# Índice

1. Antecedente (Marco teórico).....	2
2. Planteamiento del Problema .....	3
3. Propuesta de Solución.....	3
4. Materiales y Métodos empleados.....	4
5. Desarrollo de la Solución.....	4
6. Resultados .....	6
7. Conclusiones.....	9
8. Referencias Bibliográficas .....	9

## 1. Antecedente (Marco teórico)

Los hilos de ejecución (“threads”) son un mecanismo fundamental en la programación concurrente. En Java, un hilo es una secuencia de ejecución independiente dentro de un programa. La programación con hilos permite realizar varias tareas de manera simultánea, mejorando la eficiencia y reduciendo los tiempos de espera en procesos complejos.

La clase Thread en Java permite crear y gestionar hilos, facilitando la ejecución de tareas concurrentes. Cada hilo en Java puede ser implementado extendiendo la clase Thread o implementando la interfaz Runnable. Los hilos pueden iniciarse con el método start() y su ejecución puede pausarse o interrumpirse utilizando sleep() o interrupt(). Además, el sistema operativo y la JVM gestionan la asignación de recursos para optimizar la concurrencia.

El uso de hilos es común en simulaciones, videojuegos, aplicaciones en tiempo real y sistemas donde se requiere ejecutar procesos en paralelo. En esta práctica, podemos observar una aplicación de hilos que permite simular una carrera de autos donde cada vehículo avanza de manera independiente sin depender de los demás, ya que cada auto es un hilo de ejecución el cual dependiendo de la entrada que recibe el proceso, mostrará una determinada salida.

## 2. Planteamiento del Problema

El objetivo de esta práctica es recordar el funcionamiento de los hilos de ejecución en Java y comprender cómo, a partir de una entrada, se obtiene una salida determinada mediante procesos concurrentes.

El problema se plantea en la necesidad de simular una carrera de autos donde cada auto avanza en paralelo, respetando tiempos de pausa aleatorios que representan el tiempo que tardaría cada auto en avanzar en un entorno real. La ejecución debe permitir que cada auto corra de manera independiente sin interferencias, garantizando la correcta implementación de hilos.

Entre los desafíos principales está la gestión eficiente de los hilos para evitar bloqueos innecesarios o sobrecargas en la ejecución del programa. También es importante garantizar que la salida refleje el comportamiento esperado de la simulación de una carrera real.

## 3. Propuesta de Solución

Para abordar este problema, se propone el uso de hilos en Java, donde cada auto de la carrera será representado por una instancia de la clase Auto, la cual extiende Thread.

Dentro del método run(), cada auto avanzará en intervalos de 10 metros e imprimirá su progreso en la consola. Para simular el tiempo variable de avance, se empleará el método Thread.sleep(), que generará pausas aleatorias antes de que el auto continúe avanzando. La aleatoriedad permitirá que el resultado de la carrera no sea predecible, reflejando una competición real.

El método start() se usará en el main() para ejecutar cada hilo de manera concurrente, lo que garantizará que todos los autos avancen simultáneamente sin esperar a que los demás terminen.

## 4. Materiales y Métodos empleados

- Hardware: Laptop ASUS y Teléfono iPhone 16.
- Software: Google Chrome, Word y Acrobat Reader.
- Lenguaje de programación: Java
- Entorno de desarrollo: IDE compatible con Java, como Visual Studio Code con las debidas extensiones de depuración y ejecución, además de la terminal con javac y java.
- Clases utilizadas: Thread para la creación y gestión de hilos.
- Métodos empleados:
  - start(): Para iniciar la ejecución de un hilo.
  - run(): Para definir la lógica de ejecución de cada auto.
  - sleep(): Para simular pausas aleatorias en el avance del auto.
  - System.out.println(): Para mostrar el progreso de cada auto en la consola.
- Estrategia de programación: Programación concurrente con hilos, uso de aleatoriedad para la simulación del tiempo de avance de cada auto.

## 5. Desarrollo de la Solución

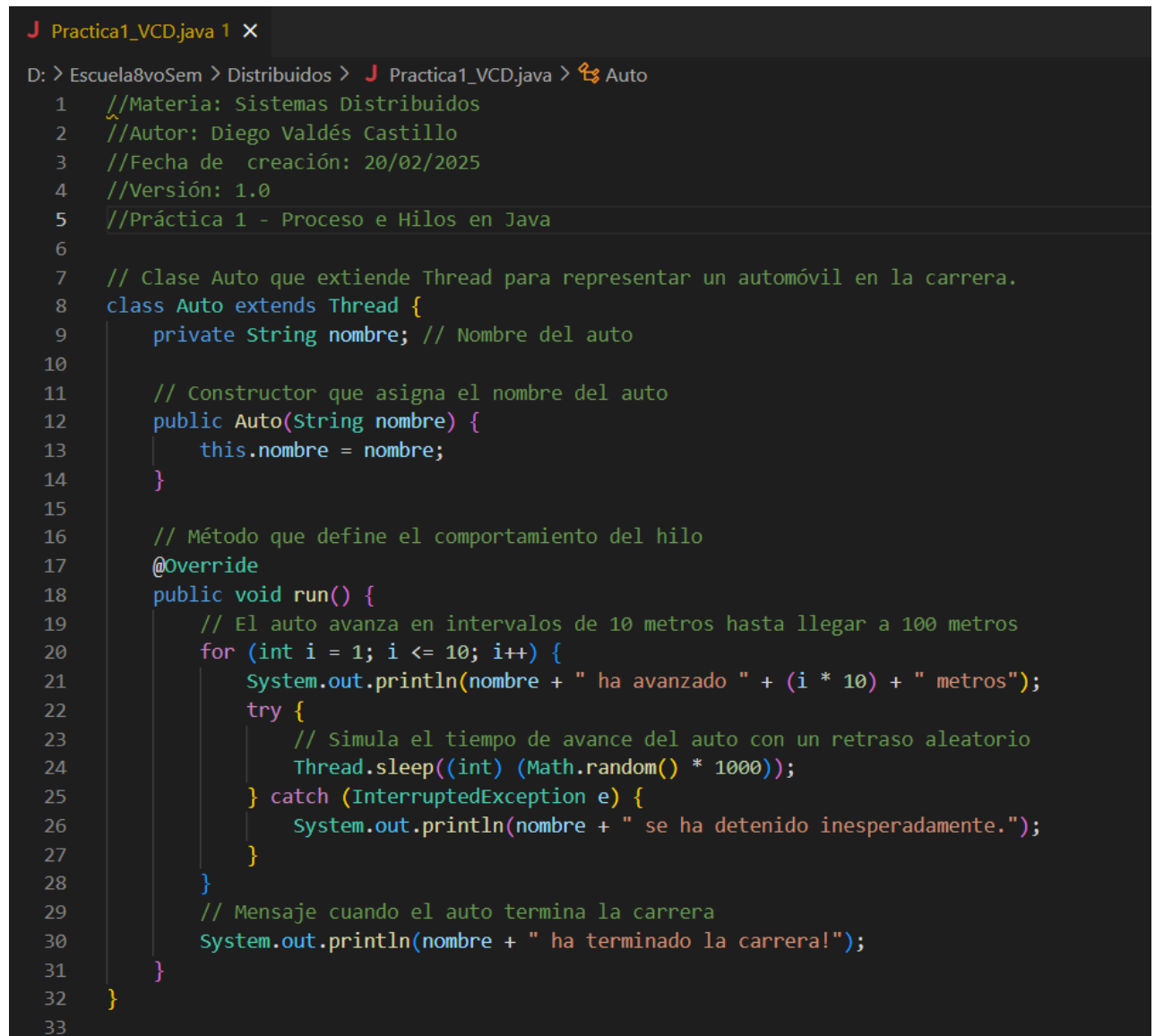
La solución se desarrolla a partir de una clase Auto que extiende Thread. Cada instancia de Auto representa un auto en la carrera y se ejecuta en su propio hilo independiente. Dentro del método run(), se utiliza un bucle for que simula la carrera en intervalos de 10 metros, imprimiendo en consola el avance del auto.

El método Thread.sleep() introduce pausas aleatorias para simular tiempos de respuesta diferentes entre los autos, lo que hace que el resultado no sea predecible. En la clase principal Practica1\_VCD, se crean tres instancias de Auto y se ejecutan con el método start(), lo que permite la ejecución concurrente de los hilos.

Cada auto avanza de manera independiente, y su salida en consola refleja este comportamiento. Al finalizar el ciclo de 10 iteraciones, cada hilo imprime un mensaje

indicando que el auto ha terminado la carrera. De este modo, el programa demuestra la ejecución de procesos concurrentes en Java de manera efectiva.

A continuación se puede ver la solución propuesta en código documentado en la figura 1 y 2:



```
J Practica1_VCD.java 1 X
D: > Escuela8voSem > Distribuidos > J Practica1_VCD.java > Auto
1  //Materia: Sistemas Distribuidos
2  //Autor: Diego Valdés Castillo
3  //Fecha de creación: 20/02/2025
4  //Versión: 1.0
5  //Práctica 1 - Proceso e Hilos en Java
6
7  // Clase Auto que extiende Thread para representar un automóvil en la carrera.
8  class Auto extends Thread {
9      private String nombre; // Nombre del auto
10
11     // Constructor que asigna el nombre del auto
12     public Auto(String nombre) {
13         this.nombre = nombre;
14     }
15
16     // Método que define el comportamiento del hilo
17     @Override
18     public void run() {
19         // El auto avanza en intervalos de 10 metros hasta llegar a 100 metros
20         for (int i = 1; i <= 10; i++) {
21             System.out.println(nombre + " ha avanzado " + (i * 10) + " metros");
22             try {
23                 // Simula el tiempo de avance del auto con un retraso aleatorio
24                 Thread.sleep((int) (Math.random() * 1000));
25             } catch (InterruptedException e) {
26                 System.out.println(nombre + " se ha detenido inesperadamente.");
27             }
28         }
29         // Mensaje cuando el auto termina la carrera
30         System.out.println(nombre + " ha terminado la carrera!");
31     }
32 }
33
```

**Figura 1. Código implementado en Java para simular carrera con hilos - 1.**

```
Practica1_VCD.java 1 X
D: > Escuela8voSem > Distribuidos > J Practica1_VCD.java > Auto
34 // Clase principal que inicia la carrera
35 public class Practica1_VCD {
36     public static void main(String[] args) {
37         // Se crean tres objetos Auto con diferentes nombres
38         Auto auto1 = new Auto(nombre:"Auto Rojo");
39         Auto auto2 = new Auto(nombre:"Auto Azul");
40         Auto auto3 = new Auto(nombre:"Auto Verde");
41
42         // Se inician los hilos para que los autos comiencen la carrera simultáneamente
43         auto1.start();
44         auto2.start();
45         auto3.start();
46     }
47 }
```

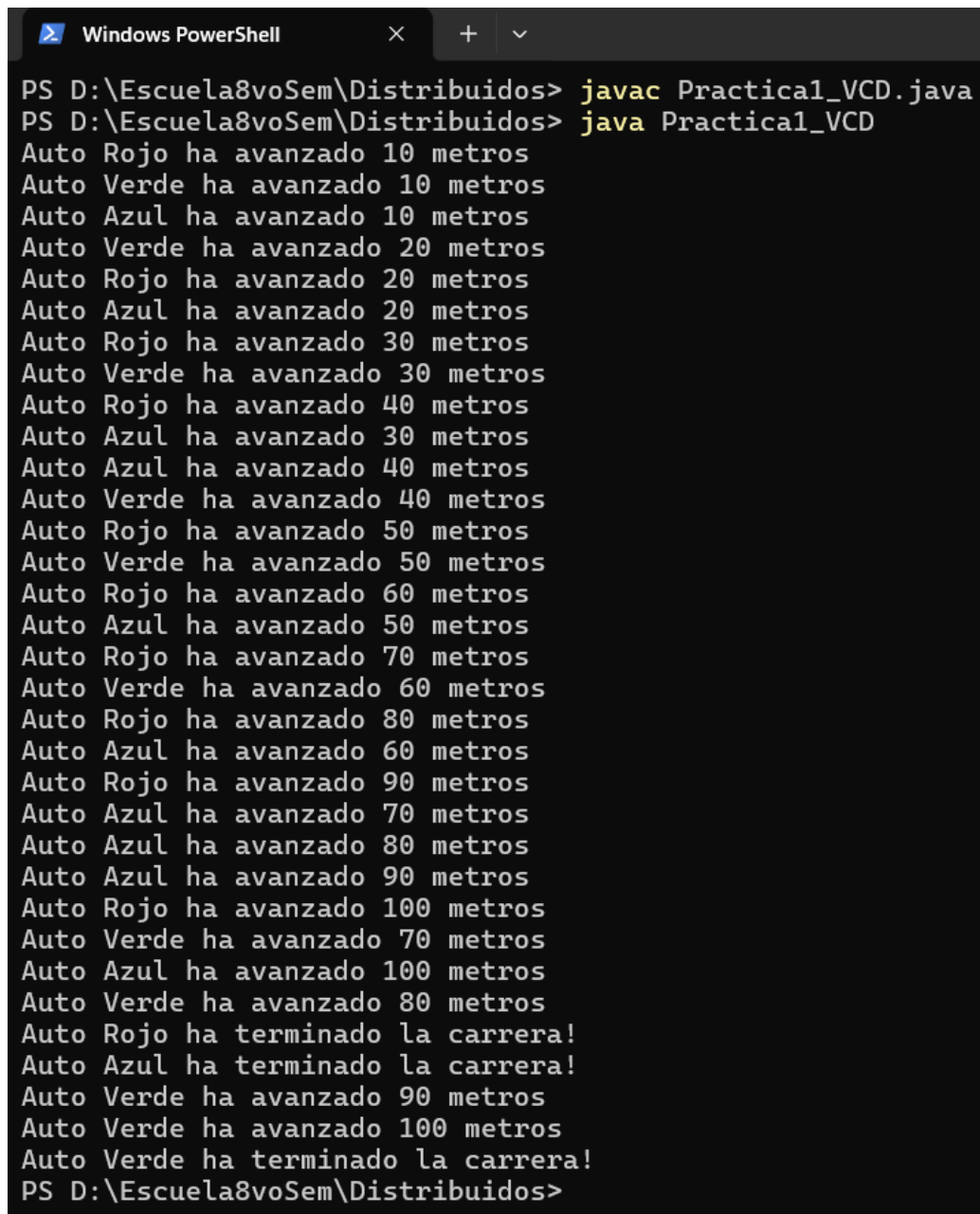
**Figura 2. Código implementado en Java para simular carrera con hilos - 2.**

## 6. Resultados

Los resultados obtenidos muestran que cada auto avanza de manera independiente y en paralelo, reflejando el comportamiento esperado de una carrera con hilos. Al ejecutar el programa varias veces, se observa que el orden en que los autos finalizan la carrera varía, lo que demuestra el impacto de la aleatoriedad introducida con `Thread.sleep()`.

La salida en la consola muestra el progreso de cada auto en diferentes momentos, evidenciando que los hilos se ejecutan simultáneamente. También se verifica que, a pesar de la concurrencia, no hay bloqueos ni errores en la ejecución, lo que indica una correcta gestión de los hilos.

A continuación se puede ver en la figura 3 una prueba de ejecución del código desde la terminal:



```
PS D:\Escuela8voSem\Distribuidos> javac Practica1_VCD.java
PS D:\Escuela8voSem\Distribuidos> java Practica1_VCD
Auto Rojo ha avanzado 10 metros
Auto Verde ha avanzado 10 metros
Auto Azul ha avanzado 10 metros
Auto Verde ha avanzado 20 metros
Auto Rojo ha avanzado 20 metros
Auto Azul ha avanzado 20 metros
Auto Rojo ha avanzado 30 metros
Auto Verde ha avanzado 30 metros
Auto Rojo ha avanzado 40 metros
Auto Azul ha avanzado 30 metros
Auto Azul ha avanzado 40 metros
Auto Verde ha avanzado 40 metros
Auto Rojo ha avanzado 50 metros
Auto Verde ha avanzado 50 metros
Auto Rojo ha avanzado 60 metros
Auto Azul ha avanzado 50 metros
Auto Rojo ha avanzado 70 metros
Auto Verde ha avanzado 60 metros
Auto Rojo ha avanzado 80 metros
Auto Azul ha avanzado 60 metros
Auto Rojo ha avanzado 90 metros
Auto Azul ha avanzado 70 metros
Auto Azul ha avanzado 80 metros
Auto Azul ha avanzado 90 metros
Auto Rojo ha avanzado 100 metros
Auto Verde ha avanzado 70 metros
Auto Azul ha avanzado 100 metros
Auto Verde ha avanzado 80 metros
Auto Rojo ha terminado la carrera!
Auto Azul ha terminado la carrera!
Auto Verde ha avanzado 90 metros
Auto Verde ha avanzado 100 metros
Auto Verde ha terminado la carrera!
PS D:\Escuela8voSem\Distribuidos>
```

**Figura 3. Prueba de Ejecución 1.**

En la figura 3 podemos observar que el código se ejecutó desde la terminal, para este primero se compila el código con el comando *“javac Nombre Practica.java”* y después se ejecuta el programa con el comando *“java Nombre Practica”*, además en esta prueba vemos que gana el auto rojo y azul, ya que terminaron la carrera al mismo tiempo y el que terminó al final fue el auto verde.

A continuación se puede ver en la figura 4 una segunda prueba de ejecución del código desde la terminal:

```
PS D:\Escuela8voSem\Distribuidos> java Practica1_VCD
Auto Rojo ha avanzado 10 metros
Auto Azul ha avanzado 10 metros
Auto Verde ha avanzado 10 metros
Auto Verde ha avanzado 20 metros
Auto Azul ha avanzado 20 metros
Auto Rojo ha avanzado 20 metros
Auto Verde ha avanzado 30 metros
Auto Rojo ha avanzado 30 metros
Auto Verde ha avanzado 40 metros
Auto Rojo ha avanzado 40 metros
Auto Rojo ha avanzado 50 metros
Auto Verde ha avanzado 50 metros
Auto Azul ha avanzado 30 metros
Auto Rojo ha avanzado 60 metros
Auto Verde ha avanzado 60 metros
Auto Azul ha avanzado 40 metros
Auto Azul ha avanzado 50 metros
Auto Rojo ha avanzado 70 metros
Auto Rojo ha avanzado 80 metros
Auto Azul ha avanzado 60 metros
Auto Verde ha avanzado 70 metros
Auto Azul ha avanzado 70 metros
Auto Azul ha avanzado 80 metros
Auto Rojo ha avanzado 90 metros
Auto Azul ha avanzado 90 metros
Auto Verde ha avanzado 80 metros
Auto Rojo ha avanzado 100 metros
Auto Azul ha avanzado 100 metros
Auto Rojo ha terminado la carrera!
Auto Verde ha avanzado 90 metros
Auto Azul ha terminado la carrera!
Auto Verde ha avanzado 100 metros
Auto Verde ha terminado la carrera!
PS D:\Escuela8voSem\Distribuidos>
```

**Figura 4. Prueba de Ejecución 2.**

En la figura 4 podemos observar que el código se ejecutó desde la terminal, para este primero se compila el código con el comando "javac Nombre Practica.java" y después se ejecuta el programa con el comando "java Nombre Practica", además en esta prueba vemos que gana el auto rojo, después termino el auto azul y finalmente el último fue el auto verde.



## 7. Conclusiones

El uso de hilos en Java permite la ejecución simultánea de procesos, optimizando tiempos y recursos. En esta práctica, se ha demostrado cómo los hilos pueden utilizarse para simular eventos independientes en un entorno compartido. La correcta implementación de Thread facilita la gestión de tareas paralelas, evitando bloqueos y mejorando la eficiencia del programa.

El experimento muestra que cada auto progresa de manera autónoma sin esperar a que los demás terminen, gracias al uso de Thread.sleep(). Además, la aleatoriedad introducida permite obtener resultados variados en cada ejecución, lo que simula una carrera más realista.

El aprendizaje clave de esta práctica es comprender la importancia de la programación concurrente y su aplicación en escenarios del mundo real. Los hilos son una herramienta muy importante en la optimización de tareas simultáneas y son fundamentales en el desarrollo de sistemas distribuidos.

## 8. Referencias Bibliográficas

- GeeksforGeeks. (2023). Multithreading in Java. Recuperado el 23 de febrero de 2025, de <https://www.geeksforgeeks.org/multithreading-in-java/>
- Oracle. (s.f.). Thread (Java Platform SE 8). Recuperado el 23 de febrero de 2025, de <https://docs.oracle.com/javase/8/docs/api/java/lang/Thread.html>