



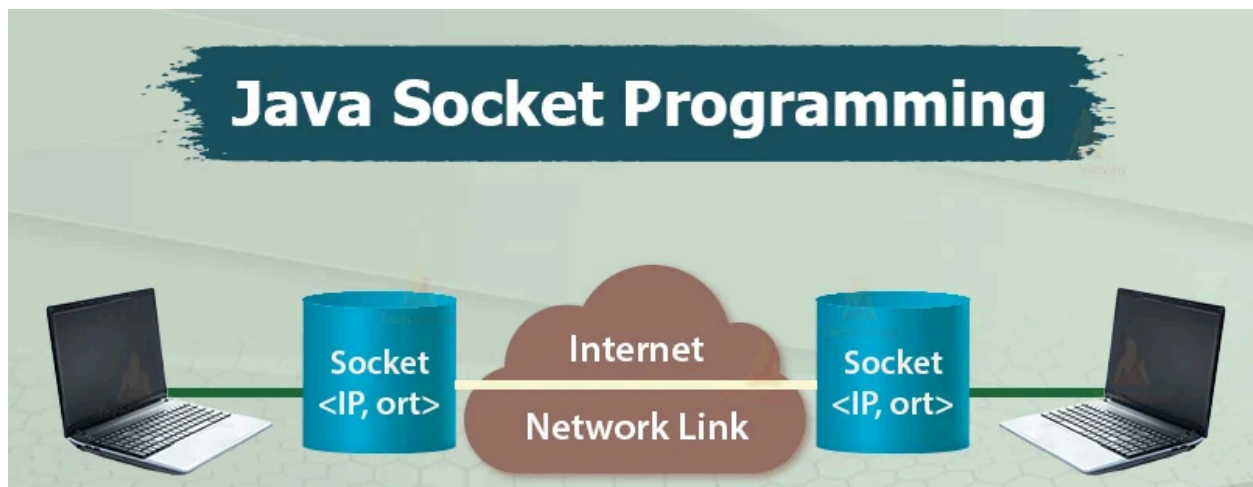
INSTITUTO POLITÉCNICO NACIONAL



ESCUELA SUPERIOR DE CÓMPUTO

Desarrollo de Sistemas Distribuidos

Práctica 2 'Sockets c/s'



Docente:

Carreto Arellano Chadwick

Grupo:

4CM11

Alumnos:

Rodríguez Hernández Marlene Guadalupe

Fecha: martes 4, marzo 2025

Antecedentes

En la práctica anterior vimos los procesos e hilos, ahora para esta práctica analizaremos los sockets c/s, y para esto debemos dejar en claro algunos conceptos sobre la práctica anterior, los cuales son de gran ayuda para continuar con los sockets, tales como:

- Thread: la unidad básica de ejecución en Java, el cual es un flujo de control dentro de un proceso.
- Concurrencia y paralelismo: con la concurrencia es posible manejar múltiples tareas a la vez (aunque estas no sean de manera simultánea), pero los hilos pueden ejecutarse en paralelo (de manera simultánea).
- Creación de hilos: se crean a través de la interfaz Runnable o extendiendo la clase Thread.
- Sincronización: la sincronización es muy importante, ya que, si varios hilos acceden a recursos compartidos es necesario sincronizar el acceso para evitar las race conditions (las condiciones de carrera).

Ya que estamos analizando los sockets del lado del cliente-servidor, los hilos son de gran utilidad puesto que nos permiten que un servidor maneje múltiples conexiones simultáneamente. A diferencia de que si trabajamos sin los hilos, entonces un servidor tendría que manejar cada conexión de cliente de forma secuencial, lo cual haría que el servidor se vuelva más lento y no pueda procesar de manera eficaz las solicitudes de clientes nuevos.

La programación de los sockets en Java es fundamental para crear aplicaciones que puedan comunicarse entre sí mediante la red. Un socket es un punto en una conexión de dos vías entre programas. Facilita el flujo de datos de forma eficaz, con el apoyo de TCP y UDP. [1]

Dicho de otra manera, un socket es un mecanismo que permite la comunicación entre dos dispositivos a través de una red, de esta forma la clase Socket permite que los programas cliente se conecten a un servidor, mientras que la clase ServerSocket es utilizada por el servidor para aceptar conexiones de los clientes.

Clases clave:

- Socket: el cliente crea un socket para conectarse a un servidor y enviar/recibir datos.
- ServerSocket: el servidor escucha en un puerto específico en su máquina y espera las conexiones entrantes de los clientes. [2]

En la siguiente tabla se muestran las ventajas y desventajas

Ventajas	Desventajas
Independencia de la plataforma, ya que, el código de sockets en Java se puede ejecutar en diferentes SO sin modificaciones.	Complejidad, la programación de red suele ser compleja.
Facilidad de usar, puesto que Java proporciona una API sencilla para manejar sockets.	La latencia, puesto que puede provocar retrasos, en especial al transmitir grandes cantidades de datos.
Escalabilidad, ya que, Java puede manejar múltiples conexiones simultáneas.	Uso intensivo de recursos, puesto que manejar múltiples conexiones simultáneas puede requerir una gran cantidad de recursos del sistema.
Seguridad, ya que, Java ofrece soporte para SSL/TLS para conexiones seguras.	Soporte limitado de protocolos, ya que, los sockets están limitados a unos pocos protocolos.
Multithreading, Java permite manejar múltiples conexiones con hilos, mejorando el rendimiento en aplicaciones que manejan muchas conexiones.	

Planteamiento del problema

Retomando el código anterior respecto a los hilos en Java, he propuesto mejorar mi código de la carrera de Tortugas para esto tenemos el Servidor y el lado del cliente (las Tortugas), en el Servidor se escucharán las conexiones de las tortugas (clientes) y se coordinarán, mientras que en el lado del cliente (Tortugas), cada tortuga se conectará al servidor y recibirá las posiciones y responderá a las actualizaciones del servidor. De tal forma que el servidor se mantiene a la escucha en el puerto, y acepta las conexiones de las tortugas y a su vez mantiene una lista de estas, conforme las tortugas se vayan moviendo aleatoriamente este envía su posición actual a cada una de ellas y a diferencia del código anterior de que la carrera terminaba a los diez pasos ahora terminará a los 100. Así mismo, analizando del lado del cliente (las tortugas), cada una se conecta al servidor, y recibe su posición desde el servidor mostrando así su avance en la carrera.

Propuesta de solución

Pese a mi idea de usar el mismo código luego de analizar los sockets c/s, creo que es más fácil interpretar una carrera de esta forma, porque el servidor funciona como el moderador de la carrera, cada una de las tortugas se conecta al servidor y este las enlista (como en una carrera normal) y cuando la carrera da inicio se comienza a ver como cada una de las tortugas avanza y la carrera finaliza cuando una de ellas llega al paso número 100 mismo que el servidor lleva el control y les indica que la carrera ha terminado para que estas puedan detenerse.

Materiales y métodos empleados

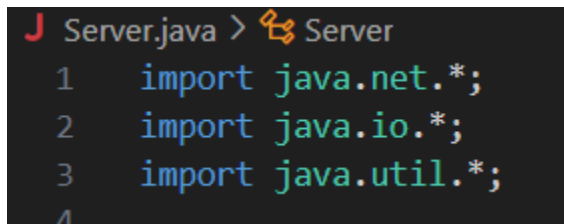
- uso VSCode con la extensión de Java
- librería .net
- librería .io
- compilo haciendo uso del javac Server.java
- ejecuté con java Server para iniciar el servidor
- compilo en otra terminal el javac Tortuga.java
- ejecuté varias veces el java Tortuga para que se unan varias tortugas a la carrera

Desarrollo de la solución

A continuación muestro el desarrollo del código elaborado para esta práctica explicando su funcionamiento.

Archivo Server.java

- se importan las librerías a utilizar

A screenshot of a code editor with a dark background. The text shows the first four lines of a Java file named Server.java. Line 1: import java.net.*; Line 2: import java.io.*; Line 3: import java.util.*; Line 4: (empty line). The text is color-coded: 'import' is blue, 'java' is green, and the package names are white.

```
J Server.java > Server
1  import java.net.*;
2  import java.io.*;
3  import java.util.*;
4
```

- declaramos el puerto, las tortugas en la carrera (5), e iniciamos el servidor

```

5  v public class Server {
6      private static final int NUM_TORTUGAS = 5; // Número de tortugas (clientes)
7      private static final int PUERTO = 5500; // Puerto del servidor
8
9      Run | Debug
10     public static void main(String[] args) {
11         System.out.println(x:"Iniciando servidor de carrera de tortugas...");

```

- el servidor acepta a las tortugas que se vayan uniendo a la carrera y las enlista como Tortuga #1, #2,..., hasta la Tortuga #5, e indica que estas ya están conectadas, asimismo indica que la carrera ya está por iniciar.

```

12  v try (ServerSocket serverSocket = new ServerSocket(PUERTO)) {
13      List<Socket> clientes = new ArrayList<>();
14      List<PrintWriter> escritorClientes = new ArrayList<>();
15
16      // Aceptar conexiones de los clientes
17  v  for (int i = 0; i < NUM_TORTUGAS; i++) {
18      System.out.println("Esperando a tortuga #" + (i + 1) + "...");
19      Socket clienteSocket = serverSocket.accept();
20      clientes.add(clienteSocket);
21      escritorClientes.add(new PrintWriter(clienteSocket.getOutputStream(), autoFlush:true));
22      System.out.println("Tortuga #" + (i + 1) + " conectada.");
23  }
24
25      // Empezamos la carrera
26      System.out.println(x:"¡Las tortugas están listas para la carrera!");
27      int[] posiciones = new int[NUM_TORTUGAS]; // Posición de cada tortuga
28

```

- seguimos teniendo el random para simular que cada una de las tortugas avanza en una velocidad distinta, y a su vez se verifica que la posición de las tortugas llegue a 100 para dar por concluida la carrera todo esto dentro del ciclo.

```

29      // La carrera continúa mientras ninguna tortuga haya llegado al final
30      boolean carreraTerminada = false;
31      while (!carreraTerminada) {
32          // Hacer que todas las tortugas se muevan
33          for (int i = 0; i < NUM_TORTUGAS; i++) {
34              posiciones[i] += (int) (Math.random() * 10); // Cada tortuga avanza a distinta velocidad
35
36              // Enviar posición actual a cada cliente/tortuga
37              escritorClientes.get(i).println("Posición de la tortuga #" + (i + 1) + ": " + posiciones[i]);
38
39              // Comprobar si alguna tortuga ha llegado al final (100 pasos)
40              if (posiciones[i] >= 100) {
41                  carreraTerminada = true;
42                  escritorClientes.get(i).println(x:"¡Has ganado la carrera!");
43              }
44          }
45
46          // Esperar un poco para el siguiente movimiento (simula el paso del tiempo)
47          Thread.sleep(millis:500);
48      }
49

```

- finalmente cuando el ciclo anterior termina significa que la carrera ha llegado a su fin, por lo que se cierran las conexiones con cada una de las tortugas

```

50         // Cerrar las conexiones
51         for (Socket cliente : clientes) {
52             cliente.close();
53         }
54         System.out.println(x:"Carrera terminada.");
55
56     } catch (IOException | InterruptedException e) {
57         e.printStackTrace();
58     }
59 }
60 }

```

Resultados

Primero inicializamos el Server.java, tal como se muestra en la siguiente imagen

```

PS C:\Users\kirit\Downloads\ESCOM\DSD-R\Sockets c-s> javac Server.java
PS C:\Users\kirit\Downloads\ESCOM\DSD-R\Sockets c-s> java Server
Iniciando servidor de carrera de tortugas...
Esperando a tortuga #1...
Tortuga #1 conectada.
Esperando a tortuga #2...
Tortuga #2 conectada.
Esperando a tortuga #3...
Tortuga #3 conectada.
Esperando a tortuga #4...
Tortuga #4 conectada.
Esperando a tortuga #5...
Tortuga #5 conectada.
¡Las tortugas están listas para la carrera!
Carrera terminada.
PS C:\Users\kirit\Downloads\ESCOM\DSD-R\Sockets c-s> 

```

y en otras terminales vamos ejecutando/inicializando el Tortuga.java para que cada una de estas se una a la carrera (se una al servidor), así mismo, en la pantalla del servidor podemos observar que al estar conectadas las 5 tortugas la carrera puede comenzar, y nos indica también el momento en que termina la carrera.

Por otro lado en las siguientes imágenes podemos observar cómo se inicializa cada una de las tortugas para unirse a la carrera (servidor) y cuando comienza la carrera cada una avanza a distinta velocidad, finalmente la carrera termina cuando una de ellas llega al paso 100.

- Tortuga 1

```
PS C:\Users\kirit\Downloads\ESCOM\DSD-R\Sockets c-s> javac Tortuga.java
PS C:\Users\kirit\Downloads\ESCOM\DSD-R\Sockets c-s> java Tortuga
Tortuga conectada al servidor.
Posición de la tortuga #1: 1
Posición de la tortuga #1: 9
Posición de la tortuga #1: 17
Posición de la tortuga #1: 25
Posición de la tortuga #1: 25
Posición de la tortuga #1: 33
Posición de la tortuga #1: 38
Posición de la tortuga #1: 47
Posición de la tortuga #1: 55
Posición de la tortuga #1: 59
Posición de la tortuga #1: 66
Posición de la tortuga #1: 70
Posición de la tortuga #1: 76
Posición de la tortuga #1: 81
Posición de la tortuga #1: 86
Posición de la tortuga #1: 94
Posición de la tortuga #1: 94
Posición de la tortuga #1: 98
Posición de la tortuga #1: 101
¡Has ganado la carrera!
PS C:\Users\kirit\Downloads\ESCOM\DSD-R\Sockets c-s> █
```

- Tortuga 2

```
PS C:\Users\kirit\Downloads\ESCOM\DSD-R\Sockets c-s> javac Tortuga.java
PS C:\Users\kirit\Downloads\ESCOM\DSD-R\Sockets c-s> java Tortuga
Tortuga conectada al servidor.
Posición de la tortuga #2: 7
Posición de la tortuga #2: 8
Posición de la tortuga #2: 12
Posición de la tortuga #2: 15
Posición de la tortuga #2: 20
Posición de la tortuga #2: 23
Posición de la tortuga #2: 32
Posición de la tortuga #2: 34
Posición de la tortuga #2: 39
Posición de la tortuga #2: 45
Posición de la tortuga #2: 48
Posición de la tortuga #2: 48
Posición de la tortuga #2: 50
Posición de la tortuga #2: 57
Posición de la tortuga #2: 66
Posición de la tortuga #2: 69
Posición de la tortuga #2: 76
Posición de la tortuga #2: 82
Posición de la tortuga #2: 83
PS C:\Users\kirit\Downloads\ESCOM\DSD-R\Sockets c-s> █
```

- Tortuga 3

```
PS C:\Users\kirit\Downloads\ESCOM\DSD-R\Sockets c-s> javac Tortuga.java
PS C:\Users\kirit\Downloads\ESCOM\DSD-R\Sockets c-s> java Tortuga
Tortuga conectada al servidor.
Posición de la tortuga #3: 2
Posición de la tortuga #3: 6
Posición de la tortuga #3: 14
Posición de la tortuga #3: 14
Posición de la tortuga #3: 18
Posición de la tortuga #3: 23
Posición de la tortuga #3: 28
Posición de la tortuga #3: 34
Posición de la tortuga #3: 37
Posición de la tortuga #3: 37
Posición de la tortuga #3: 43
Posición de la tortuga #3: 48
Posición de la tortuga #3: 53
Posición de la tortuga #3: 55
Posición de la tortuga #3: 55
Posición de la tortuga #3: 63
Posición de la tortuga #3: 67
Posición de la tortuga #3: 75
Posición de la tortuga #3: 75
PS C:\Users\kirit\Downloads\ESCOM\DSD-R\Sockets c-s> █
```

- Tortuga 4

```
PS C:\Users\kirit\Downloads\ESCOM\DSD-R\Sockets c-s> javac Tortuga.java
PS C:\Users\kirit\Downloads\ESCOM\DSD-R\Sockets c-s> java Tortuga
Tortuga conectada al servidor.
Posición de la tortuga #4: 7
Posición de la tortuga #4: 12
Posición de la tortuga #4: 18
Posición de la tortuga #4: 21
Posición de la tortuga #4: 30
Posición de la tortuga #4: 35
Posición de la tortuga #4: 37
Posición de la tortuga #4: 37
Posición de la tortuga #4: 39
Posición de la tortuga #4: 44
Posición de la tortuga #4: 51
Posición de la tortuga #4: 60
Posición de la tortuga #4: 62
Posición de la tortuga #4: 70
Posición de la tortuga #4: 72
Posición de la tortuga #4: 77
Posición de la tortuga #4: 82
Posición de la tortuga #4: 85
Posición de la tortuga #4: 86
PS C:\Users\kirit\Downloads\ESCOM\DSD-R\Sockets c-s> █
```

- Tortuga 5

```
PS C:\Users\kirit\Downloads\ESCOM\DSD-R\Sockets c-s> java Tortuga
Tortuga conectada al servidor.
Posición de la tortuga #5: 1
Posición de la tortuga #5: 7
Posición de la tortuga #5: 7
Posición de la tortuga #5: 11
Posición de la tortuga #5: 19
Posición de la tortuga #5: 19
Posición de la tortuga #5: 28
Posición de la tortuga #5: 32
Posición de la tortuga #5: 40
Posición de la tortuga #5: 48
Posición de la tortuga #5: 56
Posición de la tortuga #5: 56
Posición de la tortuga #5: 63
Posición de la tortuga #5: 69
Posición de la tortuga #5: 73
Posición de la tortuga #5: 81
Posición de la tortuga #5: 84
Posición de la tortuga #5: 91
Posición de la tortuga #5: 100
¡Has ganado la carrera!
PS C:\Users\kirit\Downloads\ESCOM\DSD-R\Sockets c-s> █
```

Conclusiones

Considero que analizar el desarrollo de este código (mi carrera de tortugas) desde sockets c/s fue todavía un poco más fácil que analizarlo con hilos o que incluso tratar de hacerlo mediante el uso de procesos, aunque entiendo que cada una de estas cosas nos va a llevar a una simplicidad después, cabe destacar que si me fue un poco complicado entender cómo conectar cada uno de estos procesos o hilos...o bueno lograr entender cómo conectar cada una de mis tortugas al servidor fue un tanto complejo pero me ayudo bastante analizar mas la teoria y hacer muchos ejemplos para comprender mejor el funcionamiento de estos.

Enlace a la práctica en GitHub

<https://github.com/Marlene0807/Desarrollo-de-Sistemas-Distribuidos.git>

Referencias

- [1] [Programación de Sockets en Java: Comunicación Básica de Red](#)
- [2] ▷ [Sockets en Java → 【 Tutorial de JAVA 】](#)
- [3] [Programación de Sockets en Java: Comunicación Básica de Red](#)
- [4] [Sockets en Java \(UDP y TCP\) – Programación](#)
- [5] [Sockets en Java: Un sistema cliente-servidor con sockets](#)
- [6] [Socket \(Java Platform SE 8 \)](#)