

## **Ejercicio Multihilos**

PSP - Andrés Cuevas  
Rodríguez



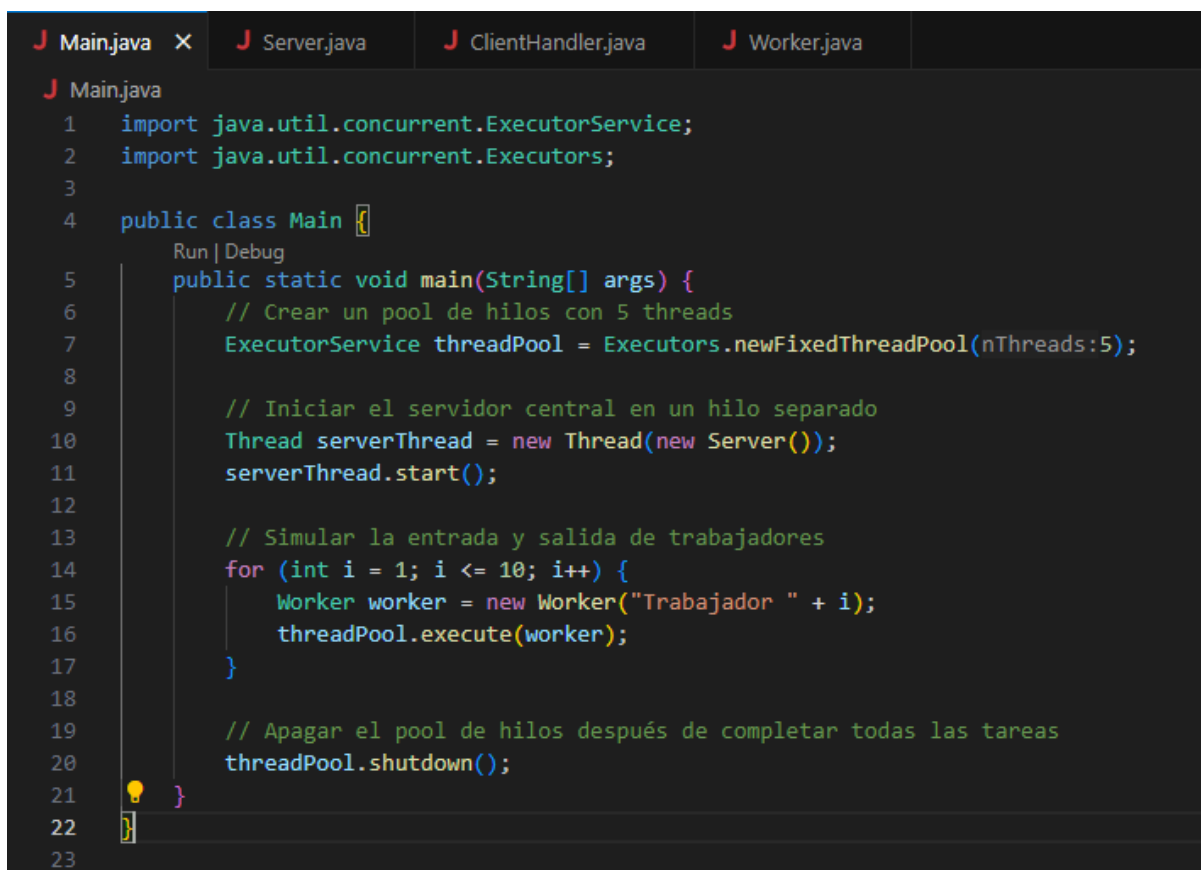
## Índice

<b>Ejercicio Multihilos</b>	<b>1</b>
<b>1. Creación clase Main</b>	<b>2</b>
<b>2. Creación clases Worker, Server y ClientHandler</b>	<b>3</b>
<b>3. Pruebas funcionales</b>	<b>5</b>



# 1. Creación clase Main

Para empezar, creamos una clase Main.java para runnear la aplicación:



```
J Main.java X J Server.java J ClientHandler.java J Worker.java
J Main.java
1 import java.util.concurrent.ExecutorService;
2 import java.util.concurrent.Executors;
3
4 public class Main {
5     Run | Debug
6     public static void main(String[] args) {
7         // Crear un pool de hilos con 5 threads
8         ExecutorService threadPool = Executors.newFixedThreadPool(nThreads:5);
9
10        // Iniciar el servidor central en un hilo separado
11        Thread serverThread = new Thread(new Server());
12        serverThread.start();
13
14        // Simular la entrada y salida de trabajadores
15        for (int i = 1; i <= 10; i++) {
16            Worker worker = new Worker("Trabajador " + i);
17            threadPool.execute(worker);
18        }
19
20        // Apagar el pool de hilos después de completar todas las tareas
21        threadPool.shutdown();
22    }
23 }
```

En el método main, crearemos un pool de hilos con 5 threads, una función para iniciar el servidor central en un hilo separado, un bucle for para simular la entrada y salida de trabajadores y un comando para apagar el pool luego de completar todas las tareas.



## 2. Creación clases Worker, Server y ClientHandler

Clase Worker.java, que simulará la entrada y salida de los trabajadores:

```
J Main.java J Server.java J ClientHandler.java J Worker.java X
J Worker.java > Worker > run()
1  import java.util.Date;
2
3  public class Worker implements Runnable {
4      private String name;
5
6      public Worker(String name) {
7          this.name = name;
8      }
9
10     @Override
11     public void run() {
12         // Simular la entrada del trabajador
13         logEvent(eventType:"Entrada");
14
15         // Simular alguna actividad
16         try {
17             Thread.sleep((long) (Math.random() * 3000));
18         } catch (InterruptedException e) {
19             e.printStackTrace();
20         }
21
22         // Simular la salida del trabajador
23         logEvent(eventType:"Salida");
24     }
25
26     private void logEvent(String eventType) {
27         String logMessage = String.format(format:"[%s] %s - %s", new Date(), name, eventType);
28         System.out.println(logMessage);
29     }
30 }
31
32
```



Clase Server.java, que funcionará como servidor central que recibirá registros:

```
J Main.java J Server.java X J ClientHandler.java J Worker.java
J Server.java > Server
1  import java.io.IOException;
2  import java.net.ServerSocket;
3  import java.net.Socket;
4
5  public class Server implements Runnable {
6      @Override
7      public void run() {
8          try (ServerSocket serverSocket = new ServerSocket(port:12345)) {
9              while (true) {
10                 // Esperar a que llegue una conexión
11                 Socket clientSocket = serverSocket.accept();
12
13                 // Procesar la conexión en un hilo separado
14                 Thread clientThread = new Thread(new ClientHandler(clientSocket));
15                 clientThread.start();
16             }
17         } catch (IOException e) {
18             e.printStackTrace();
19         }
20     }
21 }
22
```

Clase ClientHandler:

```
Main.java J Server.java J ClientHandler.java X J Worker.java
J ClientHandler.java > ...
1  import java.io.BufferedReader;
2  import java.io.IOException;
3  import java.io.InputStreamReader;
4  import java.net.Socket;
5
6  public class ClientHandler implements Runnable {
7      private Socket clientSocket;
8
9      public ClientHandler(Socket clientSocket) {
10         this.clientSocket = clientSocket;
11     }
12
13     @Override
14     public void run() {
15         try {
16             // Configurar un lector de entrada para recibir datos del cliente
17             BufferedReader reader = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
18
19             // Leer los registros de entrada y procesarlos
20             String line;
21             while ((line = reader.readLine()) != null) {
22                 System.out.println("Registro recibido en el servidor: " + line);
23                 // Aquí puedes realizar el procesamiento y almacenamiento real de los registros.
24             }
25
26             // Cerrar la conexión después de leer todos los registros
27             clientSocket.close();
28         } catch (IOException e) {
29             e.printStackTrace();
30         }
31     }
32 }
```



### 3. Pruebas funcionales

Compilamos los programas abriendo una terminal y escribiendo “javac Main.java”, esto compilará los 4 programas.

```
1 import java.util.concurrent.ExecutorService;
2 import java.util.concurrent.Executors;
3
4 public class Main {
5     public static void main(String[] args) {
6         // Crear un pool de hilos con 5 threads
7         ExecutorService threadPool = Executors.newFixedThreadPool(5);
8
9         // Iniciar el servidor central en un hilo separado
10        Thread serverThread = new Thread(new Server());
11        serverThread.start();
12
13        // Simular la entrada y salida de trabajadores
14        for (int i = 1; i <= 10; i++) {
15            Worker worker = new Worker("Trabajador " + i);
16            threadPool.execute(worker);
17        }
18
19        // Apagar el pool de hilos después de completar todas las tareas
20        threadPool.shutdown();
21    }
22 }
```

Luego ejecutamos con “java Main”:

```
PS C:\Users\Alumno1\Documents\Cuevas\2ndo-DAM\Programación de Servicios y Procesos\Tarea Final> java Main
[Mon Feb 12 11:07:57 CET 2024] Trabajador 1 - Entrada
[Mon Feb 12 11:07:57 CET 2024] Trabajador 3 - Entrada
[Mon Feb 12 11:07:57 CET 2024] Trabajador 2 - Entrada
[Mon Feb 12 11:07:57 CET 2024] Trabajador 4 - Entrada
[Mon Feb 12 11:07:57 CET 2024] Trabajador 5 - Entrada
[Mon Feb 12 11:07:58 CET 2024] Trabajador 3 - Salida
[Mon Feb 12 11:07:58 CET 2024] Trabajador 6 - Entrada
[Mon Feb 12 11:07:58 CET 2024] Trabajador 2 - Salida
[Mon Feb 12 11:07:58 CET 2024] Trabajador 7 - Entrada
[Mon Feb 12 11:07:59 CET 2024] Trabajador 5 - Salida
[Mon Feb 12 11:07:59 CET 2024] Trabajador 8 - Entrada
[Mon Feb 12 11:07:59 CET 2024] Trabajador 1 - Salida
[Mon Feb 12 11:07:59 CET 2024] Trabajador 9 - Entrada
[Mon Feb 12 11:08:00 CET 2024] Trabajador 4 - Salida
[Mon Feb 12 11:08:00 CET 2024] Trabajador 10 - Entrada
[Mon Feb 12 11:08:00 CET 2024] Trabajador 6 - Salida
[Mon Feb 12 11:08:01 CET 2024] Trabajador 7 - Salida
[Mon Feb 12 11:08:02 CET 2024] Trabajador 9 - Salida
[Mon Feb 12 11:08:02 CET 2024] Trabajador 10 - Salida
PS C:\Users\Alumno1\Documents\Cuevas\2ndo-DAM\Programación de Servicios y Procesos\Tarea Final> java Main
```

La simulación y registros funcionan correctamente y el server se para cuando acaba el bucle.