Ejercicio 1 SingleThreadTCPFramework

Inciso 1:

Refactorizar el método SingleThreadTCPFramework::handleClient(Socket) para convertirlo en un Template Method. Este método debe incluir métodos hook opcionales, es decir, métodos hooks que pueden ser implementados por las subclases o no.

El metodo handleClient de la clase SingleThreadTCPFramework, luego de aplicar refactoring para convertirlo en un template method me quedo asi:

```
private final void handleClient(Socket clientSocket) {
    try (
            PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);
            BufferedReader in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));) {
        String inputLine;
        while ((inputLine = in.readLine()) != null) {

            // primer refactoring
            this.beforeCommunication(clientSocket, inputLine);

            // segundo refactoring
            if (this.exitConnection(inputLine))
            {
                break;// Client requested to close the connection
            }

            handleMessage(inputLine, out);
        }

            // tercer refactoring
        this.afterConnectionClosed(clientSocket);

    } catch (IOException e) {
        System.err.println("Problem with communication with client: " + e.getMessage());
    } finally {
        try {
            clientSocket.close();
        } catch (IOException e) {
            System.err.println("Error closing socket: " + e.getMessage());
        }
    }
}
```

Los metodos protected (hooks) son los siguientes:

```java
// primer hook
protected void beforeCommunication (Socket clientSocket, String inputLine)
{
     System.out.println("Received message: " + inputLine + " from "
               + clientSocket.getInetAddress().getHostAddress() + ":" + clientSocket.getPort());
}

// segundo hook
protected boolean exitConnection(String inputLine)
{
    return inputLine.equalsIgnoreCase("");
}

// tercer hook
protected void afterConnectionClosed(Socket clientSocket)
{
    System.out.println("Connection closed with " + clientSocket.getInetAddress().getHostAddress() + ":"
            + clientSocket.getPort());
}
```

Inciso 2:

Extienda el framework para permitir que la “palabra” que produce el cierre de la sesión con un cliente sea configurable. Evalúe las siguientes cuatro alternativas e implemente la que considere más adecuada:

1. Una variable en SingleThreadTCPServer que se configura desde el método main() de las subclases.
2. Un método (hook) que retorna un booleano resultado de evaluar la condición.
3. Un método (hook) que retorna un String que es la palabra de término de sesión.
4. Una jerarquía de Strategies que implementan cada una de las condiciones de cierre de sesión.

Para ayudarlo en definir cuál sería la alternativa que considera más apropiada, puede usar los siguientes aspectos (pero no limitarse a):

- Esfuerzo de implementación dentro del framework
- Facilidad de uso para los programadores usuarios del framework
- Limitaciones de la solución; es decir, que tan flexible es la alternativa elegida, ¿que casos de uso permite abarcar y cuales no podría?

Vamos a definir los pros y las contras de cada alternativa recomendada:

Alternativa	Flexibilidad	Facilidad de uso	Esfuerzo	Escalable
1. Variable configurable	Baja	Alta	Muy bajo	No
2. Hook boolean	Alta	Media	Bajo	Sí
3. Hook `String` con comparación fija	Media	Alta	Bajo	Sí

Alternativa	Flexibilidad	Facilidad de uso	Esfuerzo	Escalable
4. Strategy	Muy alta	Media/Baja	Medio/Alto	Sí

Solucion:

- Implementé una combinación de la **opción 1 (variable configurable)** y **opción 2 (método hook para evaluar cierre)** para aprovechar la simplicidad y la flexibilidad.
  Lo que pasa que la opcion 2 ya la tenia en mi refactoring del punto anterior.
- La variable `exitKeyWord` permite configurar la palabra de cierre desde la subclase, idealmente desde el `main`.
- El método `checkCloseConnection` es un hook que permite redefinir la condición de cierre con lógica personalizada si se desea.

```
protected String exitKeyWord = "exit";

protected void setExitKeyWord(String exitKeyWord) {
    this.exitKeyWord = exitKeyWord;
}
```

```
private boolean exitConnection(String inputLine)
{
    return this.checkCloseConnection(inputLine);
}

protected boolean checkCloseConnection(String inputLine)
{
    return inputLine.equalsIgnoreCase(this.exitKeyWord);
}
```

El metodo exitConnection seria un frozenSpot, mientras que el metodo checkCloseConnection es un hook que permite a las subclases poder redefinirlo.

Ventajas y desventajas de esta solucion:

- Ventajas:
  - Flexibilidad: Ya que permite dos formas de cerrar la conexion
    - (1)Modificando la variable 'exitKeyWord'
    - (2)Redefiniendo el metodo 'checkCloseConnection'
    - Esta solucion se adapta tanto a casos simples (una palabra fija), como a casos mas complejos (condiciones personalizadas) donde se podria redefinir el metodo 'checkCloseConnection'
  - Reutilizacion sin modificar el framework:

- Las subclases pueden extender el comportamiento realizando minimo esfuerzo.

Inciso 3: Resuelto en el codigo.

Inciso 4: Resuelto en el codigo.