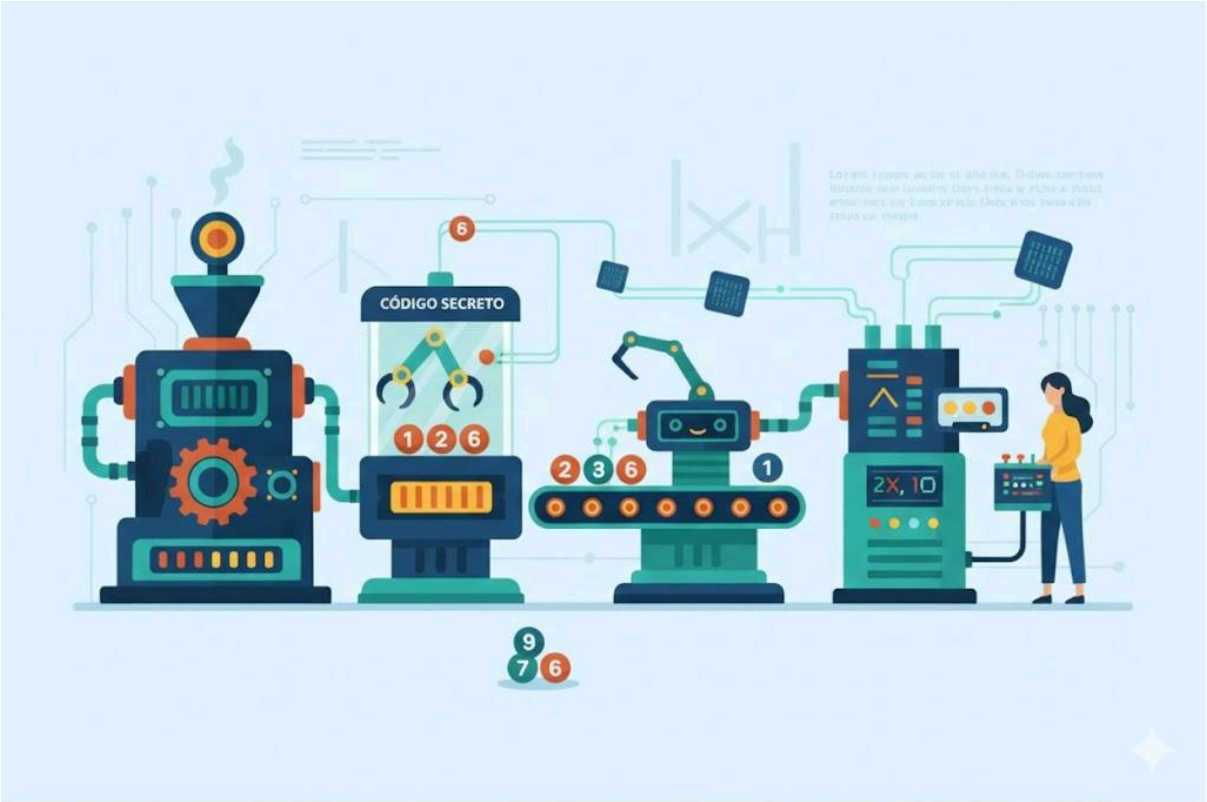
Memoria del Proyecto
Mastermind
RA 3



Aitor Jury Rodríguez. 2º DAM.
Programación de Servicios y Procesos - RA3.

Índice

1. Introducción

En este proyecto se ha desarrollado una versión digital y multijugador del clásico juego de lógica Mastermind utilizando la arquitectura Cliente-Servidor en Java. El sistema permite que múltiples usuarios se conecten simultáneamente a través de sockets TCP para intentar adivinar un código secreto de 4 dígitos generado por el servidor.

El proyecto aplica conceptos fundamentales de red, como la gestión de puertos, flujos de datos (Streams), serialización de mensajes mediante un protocolo propio y concurrencia para el manejo de múltiples hilos de ejecución.

2. Objetivo del Proyecto

El objetivo principal es establecer una comunicación robusta y segura entre procesos situados en diferentes terminales. Los objetivos específicos son:

- Implementar un Servidor TCP capaz de gestionar partidas independientes de forma concurrente.
- Desarrollar un Cliente TCP con una interfaz de consola intuitiva para el usuario.
- Diseñar un Protocolo de Capa de Aplicación para el intercambio de información (intentos, pistas y resultados).
- Garantizar la persistencia de datos mediante un sistema de ranking que almacene los mejores resultados en un fichero físico.

3. Herramientas

Para el desarrollo del proyecto se han utilizado las siguientes herramientas:

- Lenguaje de programación: Java SE 11.
- IDE recomendado: IntelliJ IDEA / Eclipse.
- Librerías: java.net.* (Sockets), java.io.* (Streams), java.util.* (Colecciones y Random).
- Concurrencia: Thread y Runnable para el soporte multijugador.
- Sincronización: ConcurrentHashMap y bloques synchronized para la seguridad de los datos del ranking.
- Protocolo: Mensajería basada en texto plano con delimitadores (split("|")).

4. Especificación de clases

El programa se organiza en varias clases principales:

4.1. Clase Server

Es la clase principal del lado servidor.

- Inicia el ServerSocket en el puerto 5000.
- Contiene el método main con un bucle infinito para aceptar clientes.
- Gestiona el ranking global y la lectura/escritura del fichero ranking.txt.

4.2. Clase Client (en el Servidor, implementa Runnable)

Es una clase interna dentro de Server que actúa como el "manejador" de cada jugador.

- Cada instancia corre en un hilo separado.
- Contiene la lógica de juego: genera el código secreto del jugador y evalúa cada intento (X y O).
- Se comunica con el cliente físico a través de BufferedReader y PrintWriter.

4.3. Clase Client (Proyecto Cliente)

Representa la aplicación que utiliza el usuario.

- Se conecta al host y puerto especificados.
- Solicita el nombre del usuario y envía los intentos.
- Interpreta los mensajes del servidor para mostrar las pistas o el ranking final.

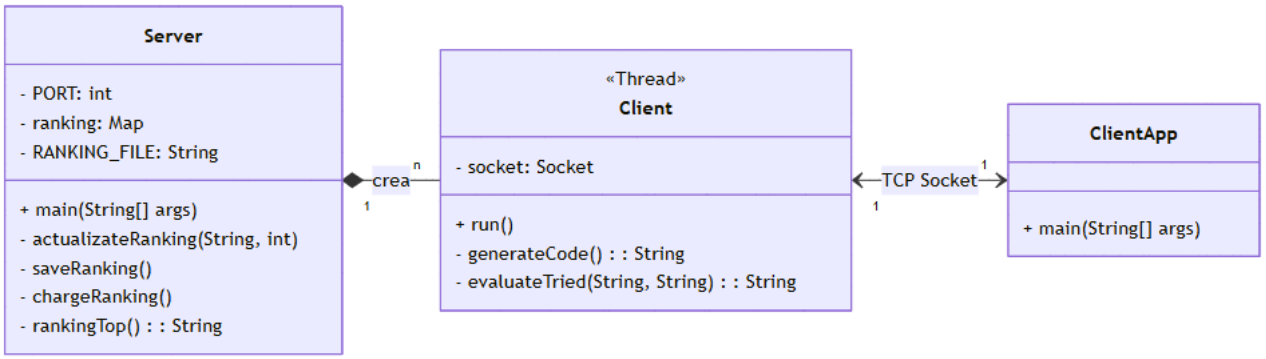
5. Diseño

5.1. Organización física del proyecto

El proyecto se organiza de la siguiente manera:

```
Mastermind/
 |
 |—src/
 |        |—Server.java
 |        |—Client.java
 |
 |—ranking.txt (Si está creado)
```

5.2. UML



5.3. Casos de uso

1. Conexión y Registro:
    - Actor: Jugador (Cliente).
    - Descripción: El usuario inicia el cliente y establece conexión con el servidor. Se le solicita un nombre que el servidor recibe para identificar su sesión y su futuro récord en el ranking.
2. Adivinar Combinación:
    - Actor: Jugador (Cliente) e Hilo de Juego (Servidor).
    - Descripción: El jugador introduce 4 dígitos. El servidor valida la combinación contra el código secreto generado para esa sesión específica.
    - Resultado: El servidor devuelve el número de aciertos exactos (X) y parciales (O).
3. Gestión de Ranking y Persistencia:
    - Actor: Servidor.
    - Descripción: Al detectar una victoria (4X), el servidor comprueba si la puntuación del jugador entra en el Top 3 o mejora su marca anterior.
    - Resultado: Se actualiza el mapa en memoria y se vuelca inmediatamente al archivo ranking.txt.
4. Finalización de Sesión:
    - Actor: Jugador / Servidor.

- Descripción: El juego termina por victoria o por desconexión del cliente (null check). El servidor cierra los flujos de datos y libera el hilo para mantener la eficiencia del sistema.

5.4 Protocolo de Comunicación (Capa de Aplicación)

Se ha diseñado un protocolo simple para que ambos extremos entiendan la información:

- Cliente → Servidor: Envía el nombre del jugador o un String de 4 dígitos (ej: "1234").
- Servidor → Cliente:
    - FEEDBACK|pistas: Envía el resultado del intento (ej: FEEDBACK|1X, 2O).
    - WINNER|intentos|ranking: Indica la victoria, los intentos totales y el Top 3 actual.

6. Funcionamiento

El funcionamiento del proyecto se presenta de la siguiente forma:

1. El Servidor se inicia y carga los datos previos de ranking.txt.
2. El Cliente se conecta y envía su nombre. El servidor le asigna un hilo y un código secreto (ej: 4216).
3. El Cliente introduce 1234. El servidor responde FEEDBACK|1X, 1O.
4. Este ciclo se repite hasta que el servidor detecta 4X.
5. El servidor actualiza el ranking si el jugador ha mejorado su marca personal y envía la lista de los 3 mejores.

7. Mejoras implementadas

Durante el desarrollo del proyecto, además de cumplir con los requisitos mínimos, se implementaron mejoras para hacer la simulación más completa y controlada. Estas mejoras son:

1. Sistema Multijugador Real:
   Mediante el uso de Threads, el servidor no se bloquea. Pueden jugar 10 personas a la vez, cada una con un código secreto distinto.
2. Ranking de Mejores Resultados:
   El servidor detecta automáticamente quiénes son los 3 jugadores que han necesitado menos intentos para ganar.
3. Persistencia (Guardado de partidas):
   El ranking no se borra al apagar el servidor. Se guarda en ranking.txt y se recupera al iniciar.
4. Tratamiento de Errores de Formato:
   El cliente valida que la entrada sea de 4 dígitos y entre 1-6 antes de enviarla, optimizando el uso de la red.
5. Sincronización de Datos:
   Se ha utilizado ConcurrentHashMap y métodos sincronizados para evitar que dos hilos corrompan el ranking al escribir simultáneamente.

8. Conclusión

Este proyecto ha permitido poner en práctica la teoría de sockets TCP. Se ha logrado una comunicación bidireccional fluida y se han resuelto problemas complejos de concurrencia. La implementación de mejoras como el ranking y la persistencia dota al software de una utilidad real, simulando un entorno de servidor de juegos profesional donde la integridad de los datos y la atención a múltiples usuarios son críticas.