

**DISEÑO Y MANTENIMIENTO DE SOFTWARE**

DESCRIPCIÓN DETALLADA DE LA ARQUITECTURA Y DISEÑO ESCOGIDOS

Diana Bringas Ochoa

Lisa Cané Sáiz

Willow Maui García Moreno

Julen Rostan Saez

Rute Catarina Teixeira Dos Santos

Contenido

[Planteamiento inicial 3](#_Toc25142341)

[Diseño final 3](#_Toc25142342)

# Planteamiento inicial

El primer planteamiento que propusimos para el diseño fue aplicar el patrón Método Fabrica, de forma que el encargado de crear (instanciar) el Árbitro fuera el Jugador (aunque realmente lo instanciase el Servidor siendo el Jugador el encargado de enviarle un mensaje para ello), y además, aplicar el patrón Singleton a la clase Árbitro y el patrón Prototipo a la clase Tablero. Por lo que el diseño inicial quedaba de la siguiente forma:

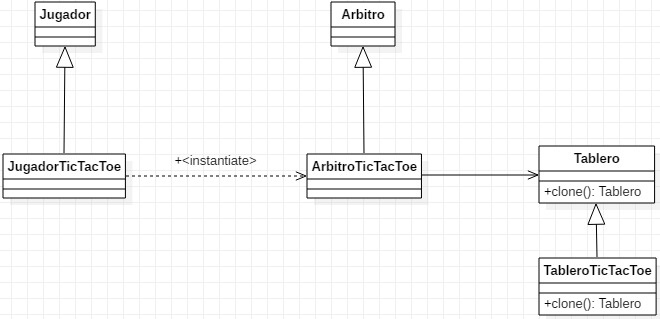


Ilustración 1 Diagrama de clases del planteamiento inicial

Descartamos la opción de que el Árbitro implementase el patrón Singleton ya que si se plantea la posibilidad de que se puedan jugar varias partidas simultáneas, habría que instanciar varios árbitros.

También descartamos la opción de que el Tablero implementase el patrón Prototipo ya que realmente no ganábamos nada con ello y estábamos forzados a mantener una instancia del tablero.

Y finalmente descartamos la opción de implementar el Método Fábrica con Jugador y Árbitro ya que con el planteamiento de juego que tenemos nos es suficiente con crear dos clases normales sin aplicar patrones.

Además, en nuestro planteamiento inicial la lógica del juego y de los mensajes la planteamos dentro de las clases Jugador y Árbitro, lo que tuvimos que cambiar casi completamente.

# Diseño final

El diseño final que hemos planteado para el juego es el siguiente:

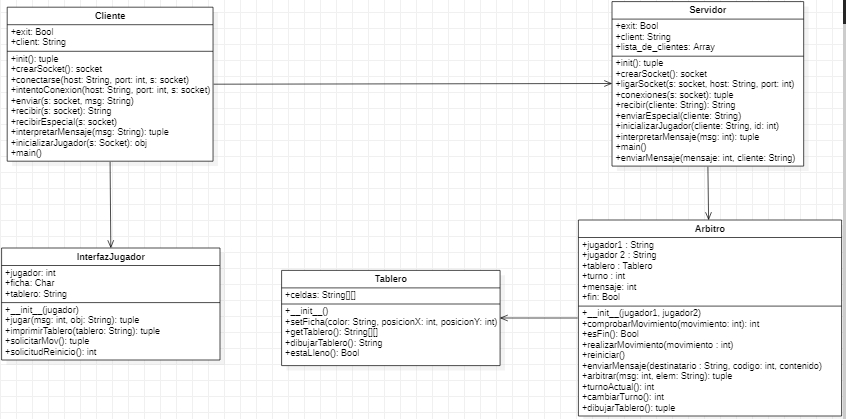


Ilustración 2 Diagrama de clases del planteamiento final

## Clase Cliente

La clase Cliente es la encargada de realizar la conexión y las configuraciones relativas al cliente del juego. Además, llamará a la función “*jugar(msg, obj*)” de InterfazJugador.

Dispone de un método “***ini()***” que se encarga de inicializar el puerto y el host, y devolverlos.

El método “***crearSocket()***” que devuelve un nuevo socket siguiendo el esquema del protocolo TCP.

El método “***conectarse(host, port, s)***” se encarga de conectar el socket con el puerto y host pasados.

El método “***intentoConexion(host, port, s)***” se encarga de, si el puerto no está siendo usado y la dirección pasada es correcta, conectarse al servidor. Si no encuentra un servidor, lo reintenta tras 5 segundos.

El método “***enviar(s,msg)***” realiza el envío de mensajes de Cliente a Servidor. En caso de no poder enviarlo, lo reintenta tras 5 segundos.

El método “***recibir(s)***” se encarga de gestionar los mensajes recibidos del Servidor.

El método “***recibirId (s)***” recibe el número asignado al cliente y se le asigna.

El método “***interpretarMensaje(msg)***” interpreta el mensaje pasado.

El método “***inicializarJugador(s)***” inicializa un jugador, para lo que requiere de un cliente y de su id.

El método “***\_\_init\_\_()***” se encarga de inicializar las variables de la clase.

En el método “***main()***” se realizan las llamadas a las funciones relativas a la configuración del Cliente.

Finalmente, el método “***\_\_del\_\_()***” destruye los socket que no son nulos al final de la ejecución.

## Clase InterfazJugador

La clase InterfazJugador es la encargada de, cuando sea el turno del jugador, mostrarle el tablero actualizado (que ha obtenido del Árbitro a través del Servidor), solicitarle el movimiento que desea realizar, y una vez ha finalizado el juego hacerle la solicitud de reinicio para saber si quiere volver a jugar o no.

Dispone de un método **“*\_\_init\_\_(jugador)***” que inicializa la variable “*jugador*” de la clase con el jugador que se le ha pasado y la variable “*ficha*” en función de si es el jugador 1, que tendrá la ficha negra (“*X*”), o si es el jugador 2, que tendrá la ficha blanca (“*O*”), y muestra un mensaje por pantalla al jugador indicándole qué jugador es y de qué ficha dispone.

El método principal de la clase es “***jugar(msg, obj*)**”, que llamará al resto de funciones de la misma en función del mensaje que el Cliente ha recibido del Servidor.

* REDACTAR cuando acabemos

El método “***mostrarResultado(obj)***” interpretará el objeto recibido y en función de éste mostrará un mensaje al jugador con el resultado de la partida. Si se trata de un “0” significa que los jugadores han empatado, si no se mostrará el mensaje correspondiente al jugador que ha ganado y perdido.

El método “***imprimirTablero(tablero)***” se encarga de mostrar por pantalla la representación del tablero actualizado al jugador.

El método “***solicitarMov()***” solicita el movimiento al jugador mostrándole un mensaje por pantalla con la solicitud primero de la posición de las “*x*” y luego de la posición de las “*y*”, almacenando dichos *inputs* en un string “*mov*” el cual se devolverá junto con un código de mensaje “104”.

## Clase Servidor

Dispone de un método “***ini()***” que se encarga de inicializar el puerto y el host, y devolverlos.

El método “***crearSocket()***” devuelve un nuevo socket siguiendo el esquema del protocolo TCP.

El método “***ligarSocket(s, host, port)***” relaciona un socket con el puerto y el host.

El método “***conexiones(s)***” espera a la conexión de clientes y devuelve el cliente y la dirección.

El método “***recibir(cliente)***” gestiona los mensajes recibidos de los clientes y los devuelve. Si no responde, lo reintenta tras 5 segundos.

El método “***enviarId(cliente)***” se asigna un número al cliente pasado y se le envía.

El método “***enviar\_Mensaje(mensaje,cliente)***” realiza el envío de mensajes de Cliente a Servidor. En caso de no poder enviarlo, lo reintenta tras 5 segundos.

El método “***interpretarMensaje(msg)***” interpreta el mensaje pasado.

El método “***inicializarJugador(cliente, id)***” inicializa un jugador, para lo que requiere un cliente y su id.

El método “***\_\_init\_\_()***” se encarga de inicializar las variables de la clase.

En el método “***main()***” se realizan las llamadas a las funciones relativas a la configuración del Servidor.

El método “***\_\_del\_\_()***” destruye los socket que no son nulos al final de la ejecución.

## Clase Arbitro

Dispone de un método **“*\_\_init\_\_(jugador)***” que inicializa la variable “*jugador*” de la clase con el jugador que se le ha pasado y la variable “*ficha*” en función de si es el jugador 1, que tendrá la ficha negra (“*X*”), o si es el jugador 2, que tendrá la ficha blanca (“*O*”), y muestra un mensaje por pantalla al jugador indicándole qué jugador es y de qué ficha dispone.

Dispone de un método “***\_\_init\_\_(jugador1, jugador2)***” que inicializa las variables de los jugadores con los parámetros pasados, instancia el tablero e inicializa la variable turno a 1 (por defecto el jugador que empieza es el 1).

El método “***arbitrar(msg, elem)***”

* REDACTAR cuando acabemos

El método “***realizarMovimiento(mov)***” comprueba si el movimiento pasado por parámetro es válido con “*comprobarMovimiento(mov)*”, **TERMINAR**

El método “***comprobarMovimiento(movimiento)***” comprueba si el movimiento pasado por parámetro es válido, para ello tiene en cuenta el jugador que ha enviado el movimiento y el estado del tablero. Devuelve un entero que será “1” si el movimiento es correcto o “2” si el movimiento es incorrecto, para que el Servidor envíe un mensaje al Cliente y éste último vuelva a solicitar el movimiento. **TERMINAR**

El método “***reiniciar()***” reinicia el tablero (lo vuelve a instanciar) y establece el turno actual al de por defecto (jugador 1).

El método “***esFin()***” obtiene el tablero y busca jugadas ganadoras o si el tablero está lleno. Devuelve un entero con el turno del jugador que ha ganado, un “0” en caso de que el tablero esté lleno (empate) o un “-1” si no ha acabado el juego.

El método “***turnoActual()***” devuelve el turno actual.

El método “***cambiarTurno()***” se encarga de cambia el turno actual.

El método “***dibujarTablero()***” devuelve un mensaje y la representación del tablero.

## Clase Tablero

La clase Tablero contiene el tablero de la partida, el cual se irá actualizando, se podrá obtener (por el árbitro), imprimir por pantalla y comprobar si está lleno.

Dispone de un método “***\_\_init\_\_()***” que inicializa el string del tablero con “0” en todas las posiciones, lo que indica que están todas vacías.

Con el método “***setFicha(color, posicionX, posicionY)***” se irá actualizando el string del tablero colocando en la posición pasada por parámetro la ficha correspondiente. Si se trata de una ficha del jugador 1 se pondrá un 1 en la posición y sino un 2.

El método “***getTablero()***” está diseñado para que el Árbitro pueda obtener el tablero actualizado. El Cliente nunca obtendrá el tablero tal cual, sólo podrá tener su representación.

El método “***dibujarTablero()***” devuelve la representación del tablero para poder mostrársela al jugador por pantalla.

El método “***estaLleno()***” devuelve un booleano indicando si el tablero está lleno o no.