ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

Titulación :

INGENIERO TÉCNICO DE TELECOMUNICACIÓN, ESPECIALIDAD EN SONIDO E IMAGEN

Título del proyecto:

PROGRAMACIÓN JAVA

Xabier Martínez de Morentin

Jesús Villadangos Alonso

Pamplona, Enero 2014

ÍNDICE

1. INTRODUCCIÓN AL PROYECTO

Este proyecto trata de informar y dar una base sobre Java, así como los programas a los cuales se les da uso, para facilitar la programación en el mundo laboral.
Ya sean programas de gestión de datos como *Assembla* y *Tortoise* o de desarrollo de aplicaciones, como *Eclipse* o *NetBeans*.

Trata de llevar al ámbito más profesional, la realización de una aplicación Java. Para ello se respetarán los convenios a la hora de denominaciones de clases, así como los métodos, etc., y la realización de estas de una forma elaborada.

Para plasmar esto se ha realizado una primera aplicación *CONECTA4, en la que* se ha procurado crear una estructura lo más encapsulada posible. Esta será objeto de estudio de temas como el diseño de una interfaz gráfica o el uso del teclado para comunicarse con el juego. Este juego será para 2 jugadores y se jugará en el mismo ordenador.

Por otra parte se ha programado una segunda aplicación *HundirLaFlota*, la cual será estudio de otros puntos de interés como el diseño de hilos de comunicación así como la implementación de clases de testeo automáticos gracias al uso de JUnit. Este juego esta diseñado para 2 jugadores, pero en este caso en *hosts* diferentes ya que está pensado para su uso en red.

2. INTRODUCCIÓN A JAVA

2.1 Historia de Java

En 1991, trece personas dirigidas por James Gosling, ingeniero de Sun Microsystems, trabajaron durante 18 meses sin descanso en Menlo Park, en el desarrollo de una nueva herramienta de programación. Este proyecto se denominó "The Green Project" y a ellos "Green Team".

El objetivo era crear una tecnología para programar una nueva generación de dispositivos inteligentes. Crear un lenguaje de programación fácil de aprender y de usar.

En un principio basaron sus ideas en el lenguaje ya existente C y C++, pero creían que este lenguaje era inadecuado, por lo que intentaron extenderlo y modificarlo, añadiendo y eliminando características a C++. No funcionó. Los lenguajes existentes eran demasiado complicados y abandonaron este camino para crear un nuevo lenguaje desde cero, lo más sencillo posible, con la idea de que se pudiera adaptar más fácilmente a cualquier entorno de ejecución [1].

Este nuevo programa recibió el nombre de "Oak" (roble en inglés) aunque tuvieron que cambiarlo ya que esta marca comercial ya estaba registrada por otra empresa (se dice que este nombre se debía a un roble que había cerca del despacho de Gosling). Pasó a llamarse "Green" y al final se quedó con "Java". Aunque se ha hablado mucho de posibles acrónimos para Java, la hipótesis que más fuerza tiene es la de que Java debe su nombre a un tipo de café asiático disponible en una cafetería cercana, de ahí que el icono de Java sea una taza de café caliente. Un pequeño signo que da fuerza a esta teoría es que los 4 primeros bytes de los archivos .class que genera el compilador, son en hexadecimal, 0xCAFEBABE.

En 1992, se aplicó por primera vez Java a "The Green Project" [2]. Consistía en un sistema de control completo de los aparatos tecnológicos y el entorno de un hogar. Para este fin, se construyó un prototipo denominado *7 (Star Seven), dispositivo parecido a una PDA, con una interfaz basada en la representación de la casa de forma animada.

Continuaron con el desarrollo estableciendo “The Green Project” como una subsidiaria de Sun Microsystems. A finales de 1992, se termina con su desarrollo y con ello “The Green Project”.

Más tarde, Java se aplicó a otros proyectos (VOD, Video On Demand), pero ninguno de estos se convirtió nunca en un sistema comercial, aunque fueron enteramente desarrollados en un Java primitivo.

En 1994, Java dió un giro, y el equipo reorientó la plataforma hacia la Web. Crearon un prototipo de navegador al que llamaron WebRunner, conocido más tarde como HotJava. Los ejecutivos de Sun Microsystems reconocieron su potencial y aceptaron el proyecto.

En esos momentos, Java1.0a ya podía descargarse, pero fue durante las conferencias del SunWorld de 1995 donde John Cage, Director Científico de Sun Microsystems, dió a conocer públicamente Java y HotJava, el navegador Web. Además, Marc Andreessen, cofundador y vicepresidente ejecutivo de Netscape, anunció que Java 1.0a sería soportado en sus navegadores. Con la segunda alpha de Java, se añadió el soporte para Windows NT, y con la tercera, para Windows 95.

En 1996, Sun Microsystems formó la empresa JavaSoft, para su desarrollo tecnológico, pudiendo crear también aplicaciones, herramientas, sistemas de plataforma y servicios para aumentar las capacidades del lenguaje. No pasaron ni tres semanas y la primera versión de Java JDK 1.0 ya fue publicada.

El punto clave para que Java se hiciera famoso y se diera a conocer, se llama “Java Applets” que es un código Java que se utiliza para el trabajo de páginas web. El entorno de ejecución era estable y relativamente seguro, y los principales navegadores web pronto incorporaron la posibilidad de ejecutar Java Applets incrustadas en las páginas web.

Muy pronto las licencias y permisos de Java mostraban firmas de gran prestigio como IBM, Microsoft, Symantec, Silicon Graphics, Oracle, Toshiba y Novell.

Desde su versión primigenia JDK 1.0, Java ha experimentado numerosos cambios. Además, ha habido un enorme incremento en el número de clases y paquetes que componen la biblioteca estándar.

La evolución de Java, ha sido [1];

- 1996 – JDK 1.0
- 1997 – JDK 1.1
- 1998 – J2SE 1.2 / Playground
- 2000 – J2SE 1.3 / Kestrel
- 2002 – J2SE 1.4 / Merlin
- 2004 – J2SE 5.0 / Tiger
- 2006 – Java SE 6 / Mustang
- 2011 – Java SE 7 / Dolphin

Hoy en día, la tecnología Java se puede encontrar en todas partes, desde portátiles o dispositivos móviles, juegos o internet, hasta superordenadores científicos o simuladores de mercado en Wall Street.

“ Write Once, Run Anywhere”

(Escríbelo una vez,
ejecútalo en cualquier lugar)

– James Gosling -

2.2 Características

La característica más notoria de Java es, que se trata de un lenguaje orientado a objetos [3]. En sus inicios, sus creadores intentaron partir de lenguajes ya existentes como C++. Al no funcionar, Java fue diseñado partiendo de cero, sin derivar de ningún otro lenguaje y con ninguna compatibilidad con ellos. Las características claves que diferencian a Java de otros entornos software desde un punto de vista tecnológico son;

- *es un lenguaje de programación con un diseño orientado a objetos;* además, se conservan elementos "no objetos" como números, caracteres y otros tipos de datos simples.

- *en un entorno robusto;* Java realiza una detección de la mayor parte de errores en la fase de compilación. Verifica su código al mismo tiempo que el programador lo escribe, y una vez más antes de ejecutarse, consiguiendo de esta forma un alto margen de codificación sin errores. Además, posee una gestión de memoria avanzada (recolector de basura) que permite una fácil creación y eliminación de objetos y mayor seguridad.

- *que ofrece un conjunto de clases potente y flexible;* la biblioteca de clases de Java proporciona un conjunto único de protocolos de Internet.

- *de fácil utilización y aprendizaje;* se ha creado un lenguaje simple (eficaz y expresivo) para una buena comprensión de los conceptos básicos en el lenguaje de programación orientada a objetos, requerimiento base para un buen aprendizaje del lenguaje Java.

- *muy bueno para incluir aplicaciones directamente en páginas Web (Applets);* Java hace posible la incorporación de aplicaciones interactivas y especializadas en páginas Web, como texto, logotipos o figuras animadas. Además, la utilización de *applets* (miniaplicaciones) supuso una gran novedad, presentando el programa un comportamiento inteligente a través de Internet, pudiendo reaccionar a la entrada de un usuario y cambiar de forma dinámica. Esto hizo posible a los programadores ejercer un control sobre los programas ejecutables de Java que en otros lenguajes no se encuentran.

2.3 Programación Orientada a Objetos(POO)

2.3.1 Introducción

La POO tiene como principal objetivo, la reutilización de código, así como facilitar la manipulación de este para su actualización o modificación. Para ello, en este tipo de programación se utilizan clases. Estas poseen atributos para determinar sus propiedades y métodos para determinar las funciones que le podemos pedir.

Para ilustrar esto con un ejemplo podemos poner un ejemplo de clase **`Casa;`**

- Esta tendrá por ejemplo los atributos de; **`tamaño, pisos, color,`** etc.
- También tendrá por ejemplo métodos como; **`agrandar , añadir piso, cambiar color,`** etc.
- De estas clases se llamarían instancias para crear casas, pero estas ya serían concretas y tendrán valores en sus atributos, por ejemplo; **`grande, 5,verde,`** etc.

De este modo con crear la clase Casa, tenemos un molde con el que crearemos, dando atributos, todas las casas que queramos, sin tener que incrementar el código para cada una de las casas.

Los métodos son comunes para todas las instancias creadas, de modo que no se necesita incrementar el código incluyendo estas funciones para cada casa creada.

Como se apreciará en las aplicaciones realizadas en el proyecto, al contrario de otras formas de programación, la programación orientada a objetos trata de poner como entidades algo cercano a la realidad.

2.3.2 Características principales

- **Abstracción;** La abstracción determina la forma de trabajo a seguir, para ello hay que dividir en subgrupos según su función o como se desee dividir el trabajo a realizar [3]. Sigamos con el ejemplo de la **`Casa`**, en este caso la abstracción consiste en dividir en grupos de código mas pequeños lo que la **`Casa`** es. Tendríamos por ejemplo; el **`tejado`**, la **`pared`**, el **`suelo`**, etc. A su vez, estos objetos serán separados según su función, siendo subdivididos nuevamente en grupos de menor tamaño de código. Por ejemplo; la **`pared`** se dividiría en **`muro`**, **`ventana`**, etc. Así continuamente la ventana se podría subdividir de ser necesario en; marco, cristal, etc.

  De este modo podemos dar atributos y métodos a cada objeto de forma singular conformando finalmente la casa en su totalidad.

- **Encapsulación;** La encapsulación es la característica por la cual, el programador puede diseñar una serie de clases en su aplicación con el objetivo de dejar o no al usuario interactuar con ellas, mediante sus métodos [3]. Esto significa que el programador, remitiéndome al ejemplo de la **`Casa`**, puede dejar al usuario que añada un piso a esta, por ejemplo, mediante un método público. De igual manera el programador puede considerar que el usuario no quiere, no necesita o no se desea que sepa como se añade un nuevo piso a la **`Casa`** de manera interna.

  Esta forma de programación también se usa para evitar que otras personas puedan alterar el contenido del código de la aplicación si no se desea, aunque existen formas para hacerlo, por ejemplo, creando una clase pública de una clase privada, podremos acceder a los métodos que posea.

  También favorece la realización de cambios en la programación interna o funcionamiento de la aplicación, ya que si dejamos la interfaz con la que el usuario se comunica con ella, igual no sabrá de la realización de estos cambios.

- **Herencia;** Desde mi punto de vista, esta es la característica más interesante y que más posibilidades da al programador. Mediante ella se puede ahorrar una gran cantidad de líneas de código repetidas, entre clases que gocen de herencia de otras.

  Siguiendo con el ejemplo de la clase **`Casa`**, el programador puede desear construir una clase **`Universidad.`** Esta clase **`Universidad`** puede extender de la clase **`Casa`**, de modo que herede sus atributos y métodos [3]. Esto nos ahorra tener que programarlos de nuevo, teniendo solo que añadir los nuevos que conciernen a la clase **`Universidad`**. Los métodos privados también se heredan, pero sólo se podrá acceder a ellos a través de otros públicos, ya que no pertenecen a la clase.

2.4 Nociones Básicas en Lenguaje Java

La estructura de Java se define por los tokens [4]. Estos son los elementos más pequeños que el compilador extrae al analizar un programa Java [1]. Se dividen en 5 categorías;

- **Identificadores;** token que representa el nombre que se le asigna a las variables, clases, paquetes, métodos y constantes entendibles por el programador e identificables por el compilador;
    - diferencian entre mayúsculas y minúsculas (Case Sensitive)
    - deben comenzar con una letra, un guión bajo "_" o el símbolo del dólar "$"
    - no se pueden usar *palabras clave* de Java

- **Palabras Clave;** palabras reservadas por Java que se usan de forma limitada, específica y para un objeto determinado.

abstract	boolean	break	byte	byvalue
case	cast	catch	char	class
const	continue	default	do	double
else	extends	false	final	finally
float	for	future	generic	goto
if	implements	import	inner	instanceof
int	interface	long	native	new
null	operator	outer	package	private
protected	public	rest	return	short
static	super	switch	syncroniced	this
throw	throws	transient	true	try
var	void	volatile	while	

Tabla 1: Palabras reservadas Java

- **Literales y Constantes;**
    - Literales; valores que puede tomar una variable
    - Constante; variable de valor fijo que no puede ser modificada

- **Operadores u Operandos;** Conjuntos de signos que indican una evaluación sobre un identificador o constante. Existen unarios o binarios.

+	^	<=	++	%=
>>>=	-	~	>=	-
&=	.	*	&&	<<
==	<<=	[	/	||
>>	+=	^=	]	%
!	>>>	=	!=	(
&	<	*=	)	|
>	?!!	/=	>>	

Tabla 2: Operadores Java

- **Separadores;** Ubican los elementos del código, son; { } , ; :

Los espacios en blanco, tabuladores, comentarios, y retornos de carro son reconocidos y eliminados por Java, por lo que no forman parte de los tokens.

3. PLATAFORMA DE DESARROLLO JAVA

3.1 Subversion

3.1.1 Utilidades

Subversion es una herramienta de código abierto, gratuita. Esta herramienta actúa parecido a un diario, de modo que siempre que se realice un cambio en un fichero o un directorio, éste quedará reflejado. No solo permite la función explicada. Una de las aplicaciones más importantes que se le da a esta herramienta es la de permitir saber, cuando más de un usuario está interactuando o modificando un fichero, en este caso un .java. Y no sólo queda reflejado, si no que además informará si los cambios realizados por estos usuarios requieren de una modificación, debido a posibles incompatibilidades de los códigos añadidos por cada uno de ellos.

Al quedar los cambios reflejados también se puede retomar el proyecto desde un punto previo al que se encontraba, si esto fuera necesario.

3.1.2 Assembla.com

Esta pagina Web, proporciona un servicio, el cual permite crear y gestionar un repositorio para el control de versiones, con subversión [5]. No obstante, también facilita la opción de utilizar Git. Además, Assembla utiliza el protocolo WebDAV a un servidor Apache configurado con Subversión pero con encriptación SSL.

De este modo deja la base de datos, fuera del terminal utilizado y así se puede recrear el modo de trabajo en una empresa, haciendo uso del terminal personal o del portátil, de modo que el repositorio lo diferencie como 2 usuarios diferentes.

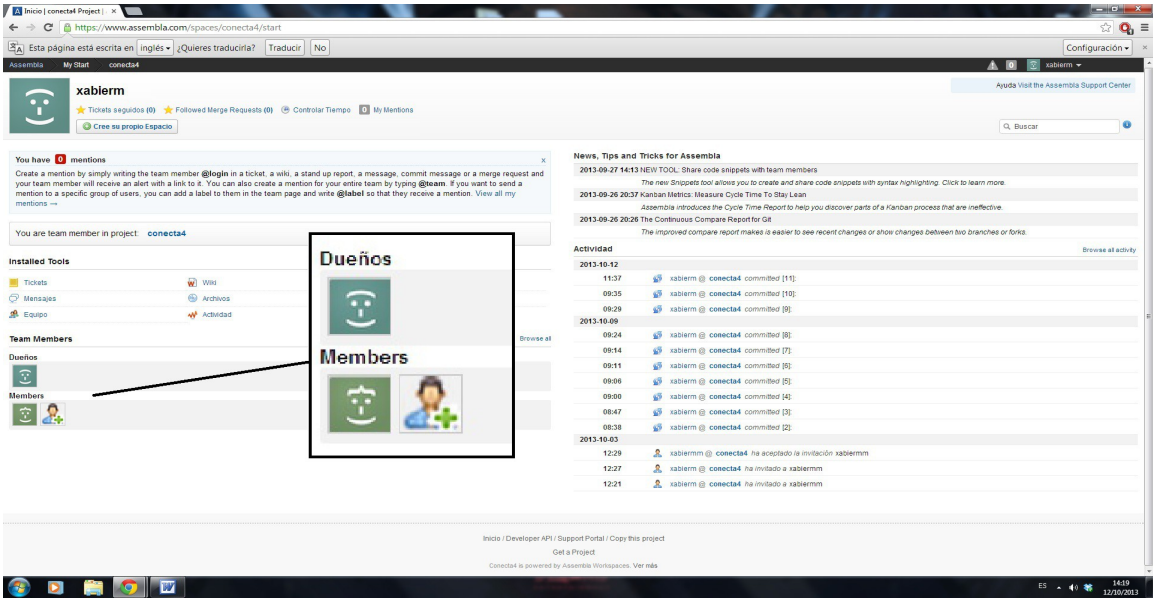


Imagen 1

Inicialmente existen tres carpetas, entre las cuales están, la carpeta branches, tags y trunk. La carpeta branches se utiliza para guardar una copia del proyecto original, cuando se quiera realizar cambios que puedan ser de riesgo para el proyecto. La carpeta tags contiene las diferentes versiones de una aplicación. Si acaso se quiere mejorar la aplicación en un futuro, aquí se encontrará todas las versiones, desde la primitiva. La carpeta trunk, es en la que trabajamos y donde tenemos el proyecto en desarrollo.

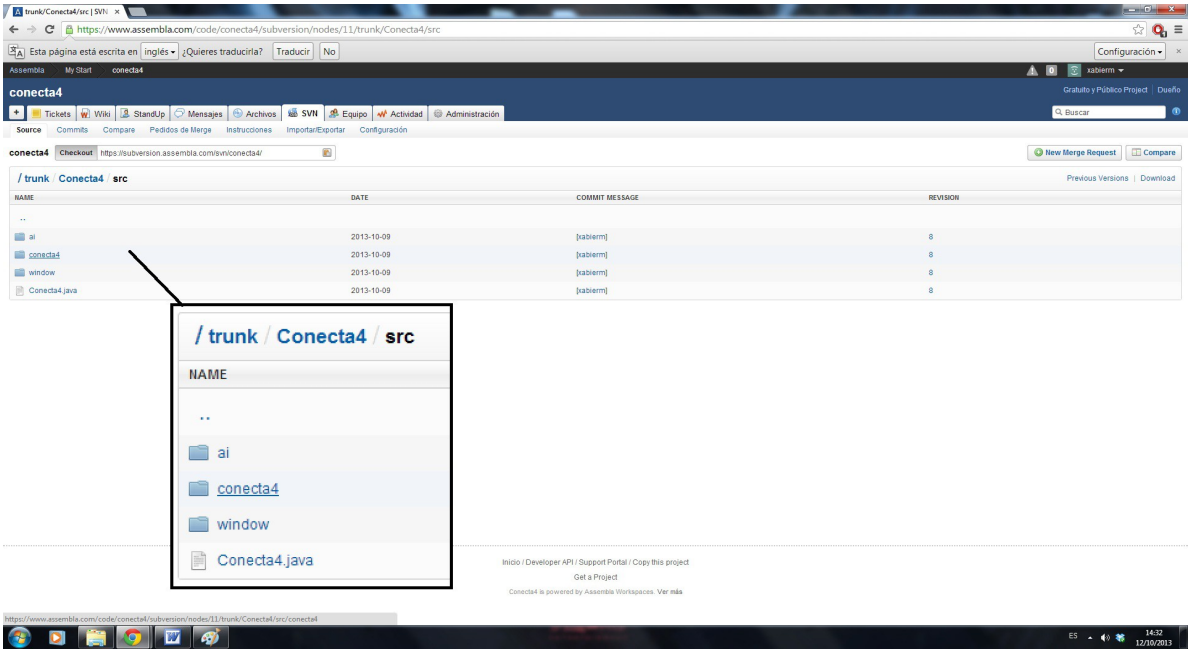


Imagen 2

3.2 Tortoise SVN

3.2.1 Utilidades

Esta segunda herramienta permite controlar el sistema Subversion, de control de versiones [6]. Así mismo también es gratuita y de código abierto.

Se encuentra integrada en el explorador y con un sencillo 'click', en el escritorio del terminal, se observan las opciones disponibles, como en cualquier aplicación de un ordenador.

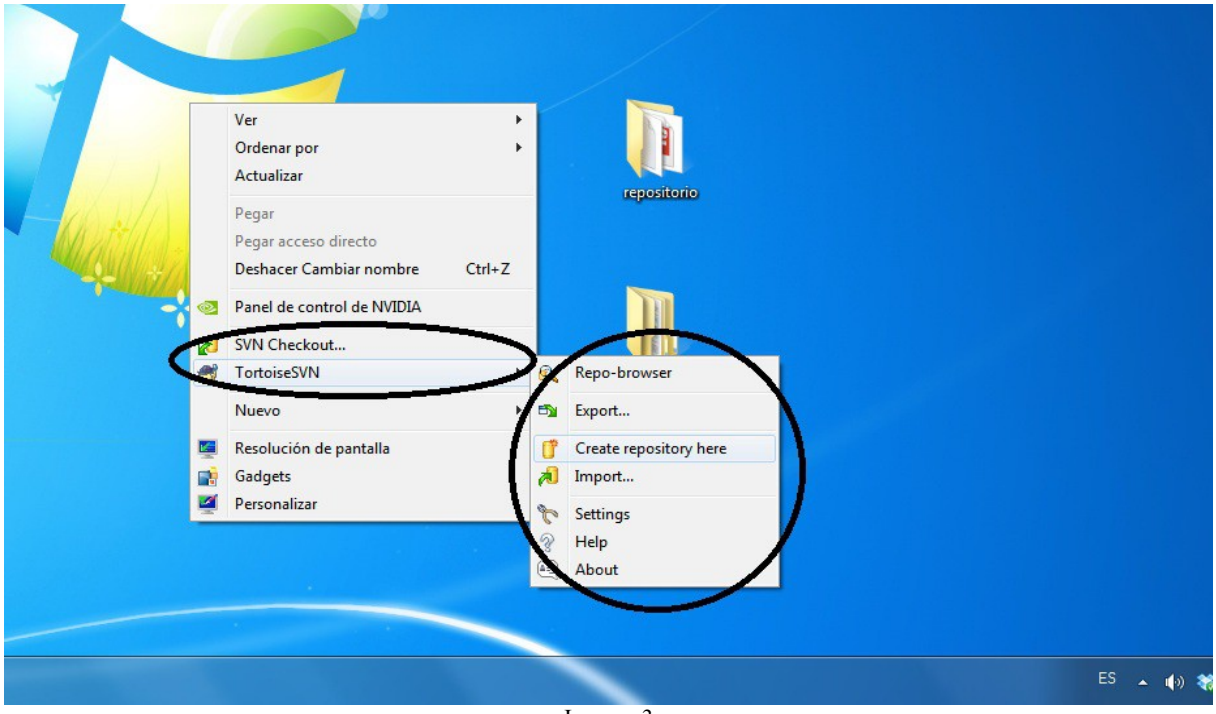


Imagen 3

Como se observa en la Imagen 3, hay una serie de opciones. La opción 'SVN checkout', permite vincular un directorio creado en el terminal con un repositorio. En este caso mediante una URL, lo hemos vinculado con Assembla.com. Se puede ver también la opción de crear un repositorio en el propio terminal, en el cual con la opción 'Import' se puede colocar el proyecto.

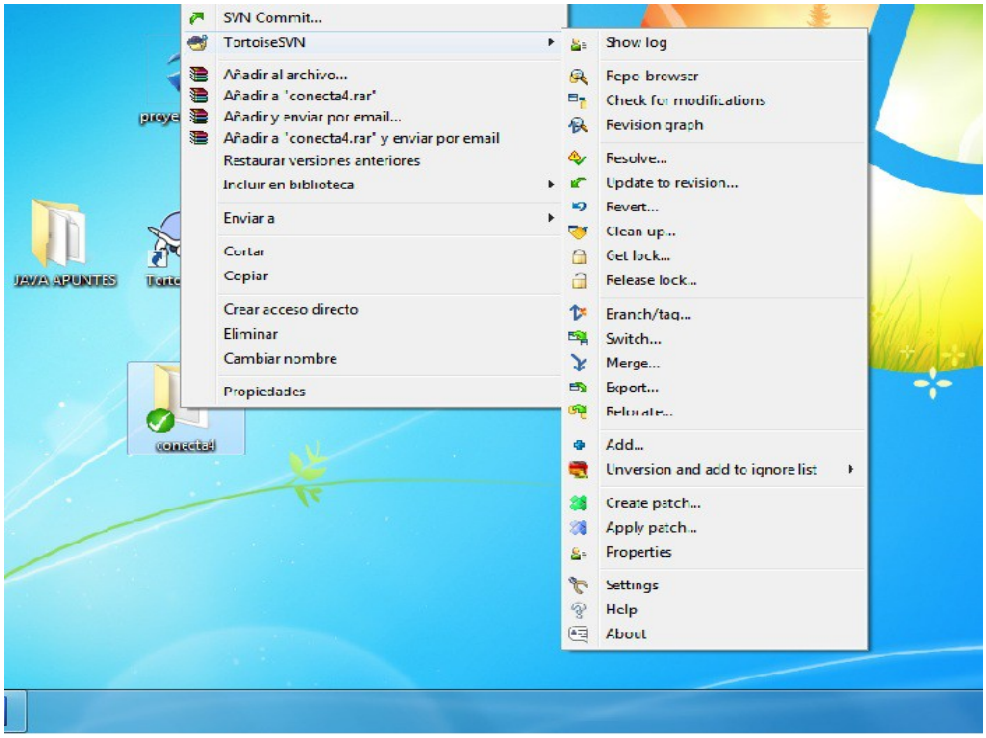


Imagen 4

No hay que olvidar que no se modificará el proyecto en el propio repositorio, sino que es en el directorio que se ha creado y en el que está el proyecto, en el que se irá programando el proyecto Java.

En este caso, el directorio es ‘Conecta4’. Se puede ver que en él, se dá la opción de hacer ‘SVN update’, así como ‘SVN commit’. Con ello se comunicaran los cambios realizados, al propio repositorio.

Se observa además, sin necesidad de acceder al repositorio Web, el contenido de este desde el navegador, gracias a la opción ‘repo-browser’.

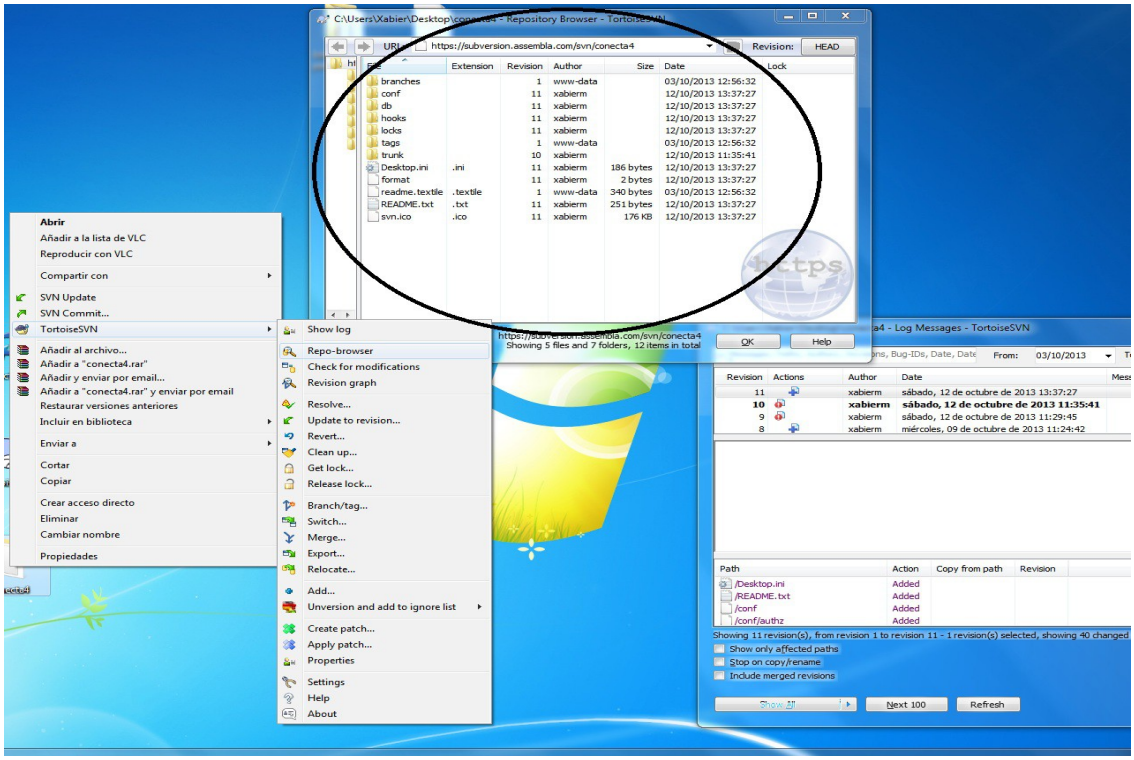


Imagen 5

3.2.2 Modo de manejo de datos

Existen dos modelos [7] o formas de tratar los proyectos.

- Modelo de copiar/modificar/fundir (modelo no bloqueante)

- Modelo de bloquear/modificar/desbloquear (modelo bloqueante)

Son dos modelos completamente diferentes. Mientras que el primer modelo no bloquea a ningún posible usuario que intente acceder al proyecto Java, el segundo modelo si lo hace.

Es por tanto el modelo no bloqueante, un modelo que mejora la productividad a nivel de trabajo, ya que múltiples usuarios pueden estar modificando el fichero al mismo tiempo.

Por otra parte el modelo bloqueante, asegura que nadie mas estará modificando el proyecto en cuestión, mientras se esté editando y no haya sido desbloqueado por el usuario actual.

- El funcionamiento del **modo bloqueante** será:

1º. Actualizar el directorio de trabajo, para tener disponible la última versión del proyecto.

2º. Bloquear los datos o fichero Java.

3º. Modificar el fichero Java.

4º. Mediante el comando 'SVN commit' subir los cambios al repositorio y quitar el bloqueo.

Este modelo trae consigo una serie de inconvenientes. En primer lugar el hecho de que sea algo restrictivo y no permita la modificación de nuevos usuarios es un problema, ya que no siempre tienen por que entrar en conflicto los datos añadidos por ambos usuarios.

El hecho de la necesidad de desbloquear el fichero una vez se ha trabajado con él, puede traer problemas, ya que uno puede olvidarse de ello, dejándolo bloqueado permanentemente.

El problema más importante quizás, es la falsa sensación de aislamiento, ya que el hecho de tener un fichero bloqueado, no significa que otro fichero del proyecto no esté siendo modificado, y con ello, pueda suceder un error, que haga que la aplicación Java deje de funcionar.

- El funcionamiento del **modo no bloqueante** será:

1º. Actualizar el directorio de trabajo, para tener disponible la última versión del proyecto.

2º. Modificar el fichero Java.

3º. Mediante el comando 'SVN commit' subir los cambios al repositorio.

Este modelo, como ya se ha citado, permite la modificación de un fichero, a partir de uno o más usuarios, simultáneamente. Es lógico que el primer usuario en actualizar su proyecto, no tenga ningún problema en subir su nueva versión al repositorio.

Por otra parte los consecutivos usuarios tendrán un error de conflicto, ya que el fichero actual ya no es el original, sobre el que ellos han modificado. En este caso, SVN dará la opción de revisar la versión original, la modificada por el primer usuario y la suya,

para resolver este conflicto. De este modo la fusión de cada uno de los ficheros privados de cada usuario, conformará el nuevo proyecto subiendo éste al repositorio.

En la actualidad este modo es extremadamente eficiente, mejorando al modelo bloqueante. Únicamente con ficheros de imagen, da mejor resultado el modelo bloqueante, ya que este tipo de ficheros no es fusionable.

3.3 Eclipse

3.3.1 Introducción

Plataforma creada por IBM usada para desarrollar "Entornos de Desarrollo Integrados" (IDE, Integrated Development Enviroment). Es de código abierto y extensible. Esta plataforma, típicamente ha sido usada para desarrollar entornos de desarrollo integrado, como el IDE de desarrollo de Java llamado JDT(Java Development Toolkit) y el compilador (ECG) que se entrega como parte de Eclipse [8]. Sin embargo, también se puede usar para otros tipos de aplicaciones cliente, como BitTorrent.

3.3.2 Utilidades

Para la creación de un proyecto se debe se debe seguir el siguiente orden en la barra de menú, *FILE→NEW→JAVAPROYECT,* a continuación se selecciona la versión de la librería JRE utilizada, en este caso se utiliza la versión JavaSE 1.7. Por ultimo *FINISH*, con ello ya se tiene el proyecto creado y ya se dispone de la carpeta *SRC*.

Aquí se crean las diferentes clases y packages necesarias para confeccionar el proyecto, estas se mostraran en la ventada Package Explorer.

Para abrir el proyecto finalizado se debe acceder al menú *FILE→IMPORT*, aquí se elige la opción *general* y dentro de esta la opción Existing Proyect Into WorkSpace, una vez seleccionado, se elige el proyecto a abrir.

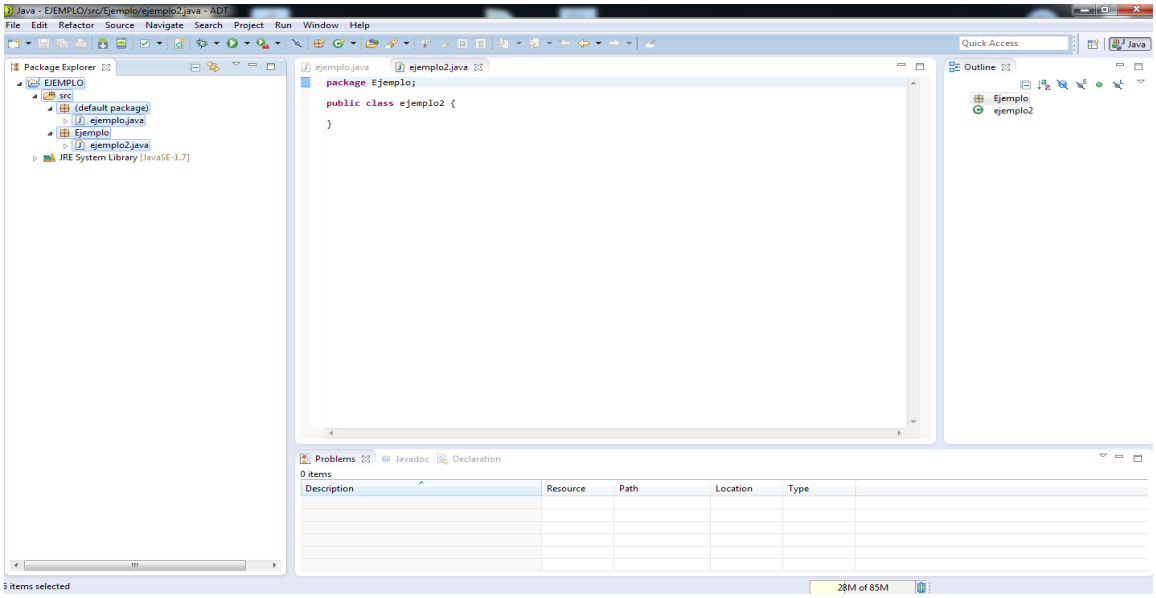


Imagen 6

3.4 Junit

3.4.1 ¿Qué es?

JUnit es un conjunto de bibliotecas creadas por Erich Gamma y Kent Beck. JUnit se conforma por un conjunto de clases(test.java) [9] que permiten ejecutar las clases de java para poder evaluar si funcionan los métodos de cada clase de forma correcta, es decir como es esperado, de no devolver el valor esperado se devolverá un fallo por parte de JUnit.

En el caso de las plataformas Eclipse y NetBeans se dispone de la opción mediante un plug-in [10] de generar de forma automática una plantilla, la cual luego será rellenada con los métodos pertinentes.

Todas las clases de testeo deben heredar de la clase TestCase.

3.5 NetBeans

3.5.1 Introducción

NetBeans es un entorno de desarrollo integrado libre y gratuito [11], su uso esta enfocado por mucho a JAVA.

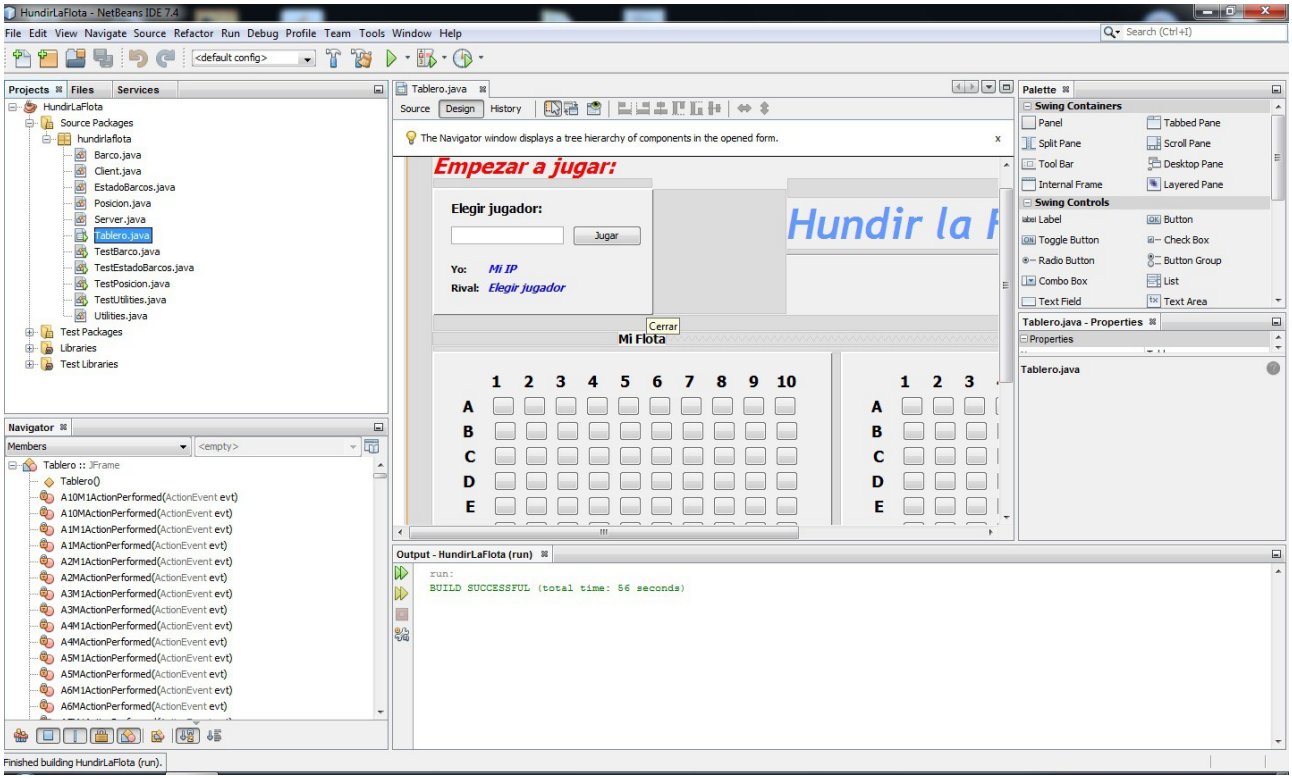


Imagen 7

Sun MicroSystems fundó el proyecto NetBeans en el año 2000, en la actualidad sigue liderando la mayoría de sus proyectos.

Las aplicaciones en NetBeans son desarrolladas a partir de un conjunto de componentes llamados módulos [12]. Un módulo es un archivo de JAVA que contiene diferentes clases de java, estas interaccionan con las APIs de NetBeans y con un archivo llamado manifest que lo identifica como módulo.

A una aplicación formada por estos módulos se le puede añadir nuevos módulos y por tanto es fácil que sean extendidas por otros desarrolladores de software.

Una de las peculiaridades de NetBeans es la del editor gráfico [11]. Pese a que otras plataformas incorporan paquetes pasa dar este servicio, NetBeans lo posee en su versión de descarga, sin necesidad de añadirlo posteriormente. En la barra de menú se podrá diseñas la interfaz gráfica accediendo a ella a través de *Design*, se hace uso para ello de los elementos encontrados en la ventana Swing Containers. La IDE de NetBeans utilizada sera la versión 7.4.

4. PROYECTO JAVA, CONECTA 4

4.1 Default Package

4.1.1 Conecta4.java

Como se puede ver en toda clase `.java`, inicialmente se importan las clases deseadas de java ya existentes, en este caso utilizamos `Swing` que es básicamente un subproceso de trabajo que tiene una cola FIFO de objetos de eventos, los cuales son ejecutados uno tras otro.

Primero se creará la clase de java, la cual se llamara `'Conecta4'`. En este caso no tendrá atributos, estos serán explicados en futuras clases de Java. Esta clase será pública y no requerirá de crear un objeto `Conecta4` ya que tiene el modificador `static`. Al ser de tipo `void`, no requiere el retorno de información.

Conecta4.java

```
import javax.swing.SwingUtilities;

public class Conecta4 {

    public static void main(String[] args) {
        new Conecta4();
    }

    public Conecta4() {
        // El Frame
        try {
            SwingUtilities.invokeAndWait(
                new Runnable() {
                    public void run() {
                        new ControlJuego();
                    }
                }
            );
        } catch (Exception e) { };
    }
}
```

Imagen 8

En segundo lugar se llamará al constructor de Java para crear los objetos que necesitemos. Este siempre sera público, de otro modo no podríamos crearlos ya que no tendríamos acceso a este. Este primer constructor tendrá como `args` un array de `strings`, que es obligatorio como argumento del método `main` de un programa de Java. Gracias a este podemos comunicarnos con el programa. De este modo creamos `Conecta4`.

En tercer lugar se llamará a un segundo constructor cuyo identificador sera el mismo que se le ha puesto a la clase, en este caso `Conecta4.` Esto es de obligado cumplimiento. En este caso el constructor no necesita parámetros a la hora de ejecutarse, por ello el paréntesis esta vacío. Como se aprecia luego utilizaremos `invokeAndWait` ya que cuando se dispara un evento que afecta a la interfaz gráfica, se desea que espere a que esta exista para poder continuar con el siguiente evento.

Se utiliza la interfaz `runnable`, la cual únicamente pide un método `run` el cual tiene la tarea a ejecutar, en este caso crear el control del juego en un hilo de ejecución separado. Como se puede apreciar la estructura en java consiste en contructores, los cuales disponen de diferentes métodos que utilizaremos para definir nuestros objetos.

4.2 Conecta4 Package

4.2.1 Constantes.java

En esta clase se encuentra una nueva línea de comando al comienzo de la clase, esta es de carácter organizativo, y sirve para separar las clases en diferentes *packages*. Esta clase sirve para albergar diferentes constantes que se utilizan en diferentes clases de la aplicación.

De este modo su utilización sera mas clara y organizada. Se utiliza el modificador `final` seguido del tipo de variable a guardar, de este modo, determinaremos que las variables son constantes y por tanto los valores guardados no se pueden modificar.

4.2.2 ControlEventos.java

En esta clase se importará una clase de java tipo `awt.event.*`, la razón es que nos interesa un oyente de sucesos tipo `ActioListener`.

Al crear la clase en este caso utilizaremos la palabra clave `extends`. Esta palabra esta reservada en Java para indicar la herencia de Java con respecto a otra clase. Es una de las herramientas mas potentes de Java. De este modo tendremos los métodos propios de la clase `KeyAdapter` y unos nuevos que añadiremos nosotros.

Además se indica también la palabra clave `implements` para que la clase implemente los interfaces de `ActionListener`.

Al crear la clase, se utilizará un atributo que será de tipo `private`. De este modo solo podrá ser utilizado en el interior de esta clase y no por otras, este será `ControJuego`.

Se llamará al constructor, en este caso utilizamos `super`, ya que la clase hereda de otra, y esta tiene su propio constructor. De este modo llamaremos al constructor de la clase de la que se hereda.

En último lugar tenemos tres métodos para esta clase. En el segundo método `keyPressed` se puede ver como asignamos para los dos jugadores humanos, los botones a utilizar desde el 1 al 7 del teclado, los cuales corresponderán a las siete columnas del juego, por el que caerán las fichas.

```java
import java.awt.event.*;

public class ControlEventos extends KeyAdapter implements ActionListener {
    private ControlJuego controlJuego;

    public ControlEventos() {
        super();
    }

    public void setControlJuego(ControlJuego c) {
        controlJuego = c;
    }


    @Override
    public void keyPressed(KeyEvent e) {
        Player p;
        int num = -1;
        if (e.getKeyCode() == KeyEvent.VK_1) { num = 1; };
        if (e.getKeyCode() == KeyEvent.VK_2) { num = 2; };
        if (e.getKeyCode() == KeyEvent.VK_3) { num = 3; };
        if (e.getKeyCode() == KeyEvent.VK_4) { num = 4; };
        if (e.getKeyCode() == KeyEvent.VK_5) { num = 5; };
        if (e.getKeyCode() == KeyEvent.VK_6) { num = 6; };
        if (e.getKeyCode() == KeyEvent.VK_7) { num = 7; };
        if (num > 0) {
            p = controlJuego.player1; if (p instanceof HumanPlayer) { ((HumanPlayer) p).setKey(num); }
            p = controlJuego.player2; if (p instanceof HumanPlayer) { ((HumanPlayer) p).setKey(num); }
        }
    }
```

Imagen 9

4.2.3 ControlJuego.java

En este caso esta clase interesa que extienda, de un `timertask` y por tanto herede sus métodos. Esto se debe a que interesa que esta tarea se ejecute más de una vez dado un `timer`, así tendremos los sucesivos `Frames`.

Determinaremos en la creación de la clase los atributos que necesitaremos en esta clase. Estos atributos, solo serán utilizados por ella. Algunos de estos parámetros pasarán a existir creándolos con el constructor mediante la palabra clave, `new`. Así mismo se inicializan los atributos dándoles un valor.

Con el método `'setControlSelection'` el cual es de tipo `void`, ya que no tiene que devolver ninguna respuesta elegiremos los jugadores y el color del jugador. Así mismo pasaremos a estar en el estado `THINKING`, donde se espera que elijamos la posición

para tirar la ficha, ya que inicialmente nos encontrábamos en el estado `CONTROLS` inicializado de este modo en el constructor.

En el siguiente método según el estado en el que se encuentre se realizarán diferentes operaciones.

En primera instancia se encuentra en estado `THINKING`. Se comprueba si el jugador actual esta pensando, si no es así se establece la posición por la que debe ser la caída de la ficha, a menos que el número elegido no sea válido en cuyo caso se tendrá que elegir uno que sí lo sea.

```
if (state == State.THINKING) {
    if (!currentPlayer.isThinking()) {
        move = currentPlayer.getMove();
        ok = posicion.meterFicha(move);
        if (!ok) { currentPlayer.think(posicion); return; };    // Y no podemos hacer más... ya
        fila = posicion.getLastFila();
        controlGrafico.tirarFicha(move, fila, currentPlayer == player1);
        state = State.TIRAR;
    }
}

if (state == State.TIRAR) {
    controlGrafico.moveAndDraw();
    if (!controlGrafico.isEnMovimiento()) {
        if (posicion.getWinner() == Valor.PLAYER1) {
            winsPlayer1++;
            if (player2 instanceof Player);
        }
        if (posicion.getWinner() == Valor.PLAYER2) {
            winsPlayer2++;
            if (player1 instanceof Player);
        }
        if ((posicion.getWinner() != Valor.NONE) || (posicion.isDraw())) {
            System.out.println("PLAYER1   " + winsPlayer1 + " - " + winsPlayer2 + "   PLAYER2")
            tick = 0;
            state = State.ENDGAME;
        } else {
            currentPlayer = (currentPlayer == player1 ? player2 : player1);
            currentPlayer.think(posicion);
            state = State.THINKING;
         }
```

Imagen 10

El estado pasa a ser `TIRAR`. En el estado `TIRAR` es donde se valora si el estado es de no caída de fichas. Es decir si la ficha a hecho que uno de los jugadores gane. Si la situación es así, se presentará por pantalla un rótulo que indica la puntuación de victorias de los jugadores. Se pasará al estado `ENDGAME` y posteriormente al estado `VACIAR` donde

se vaciará el tablero de fichas y se volverá a la situación inicial de estado `CONTROLS` donde se volverá a elegir jugador para iniciar una nueva partida. Si no es así y la ficha no hace que ninguno de los dos jugadores gane, se volverá a pasar al estado de `THINKING` donde se deberá elegir una nueva posición para la caída de la ficha.

4.2.4 Player.java

Se importará la clase `Posición` ya que será necesaria para esta clase. Esta clase es de tipo abstracta, ya que no se puede crear un objeto `Player`, pero si ser usada como `superclase`. Sus atributos, de tipo `protected`, pueden ser utilizados por cualquier clase dentro del paquete en el que se encuentra.

Se establecerán unos métodos para esta clase `Player` que nos servirán posteriormente, según el estado en el que se encuentre el jugador. En primer lugar en el constructor se inicializan los parámetros. De esta forma el jugador puede estar pensando, en la posición a elegir para dejar caer la ficha. Pero también puede estar moviendo la ficha si a dejado de pensar, en este caso, devolverá el valor de `move`.

4.2.5 HumanPlayer.java

Esta clase, nos sirve para definir un tipo especial de `Player`, y por tanto esta clase es una extensión de `Player`. Heredará por tanto los métodos y atributos, pero no el constructor de su predecesor. Con esta clase se definirá a los jugadores, cuando sean necesarios como `player1` y `player2`.

4.3 Conecta4.Tablero Package

4.3.1 EvaluadorPosicion.java

Esta clase es utilizada para comprobar en todo el tablero, ya sea horizontal, vertical u oblicuamente, la existencia de un cuatro en raya.

Mediante el método `getWinner` se barre el tablero completo en busca de un cuatro en raya satisfactorio. De ser así retornará el valor de la posición (`x,y`). Si no hay cuatro en raya, no retornará ningún valor.

```
public Valor getWinner() {
    for (int f = 0; f < Constantes.FILAS_TABLERO; f++) {
        for (int c = 0; c < Constantes.COLUMNAS_TABLERO; c++) {
            if (hay4EnFila(f,c)) {
                return posicion.tablero[f][c];
            }
        }
    }
    return Valor.NONE;
}
```

Imagen 11

Mediante el método `isDraw` se plantea el caso de que no haya cuatro en raya y el tablero no tenga todas las posiciones con fichas, en este caso seguirá la partida. Si el tablero está completo y no hay cuatro en línea, entonces se considerará un `draw`. También en el caso de que se haya rellenado todo el tablero, pero sí se consiga cuatro en linea, se considerará que se ha terminado la partida con una victoria y no un `draw`.

```java
public boolean isDraw() {
    // Si no está completo, no hay tablas
    for (int c = 0; c < Constantes.COLUMNAS_TABLERO; c++) {
        if (posicion.tablero[0][c] == Valor.NONE) { return false; };
    }
    // Si hay ganador, aunque esté completo, tampoco son tablas
    if (getWinner() != Valor.NONE) { return false; };
    return true;
}

private boolean hay4EnFila(int f, int c, Direccion dir) {
    int dx = dir.getDx();
    int dy = dir.getDy();
    Valor valor = posicion.tablero[f][c];
    // Si el primer valor es de "casilla vacia": no hay
    if (valor == Valor.NONE) { return false; };
    // Barremos los 3 siguientes
    for (int i = 0; i < 3; i++) {
        f += dy; c += dx;
        // Si se sale del tablero: no hay
        if ((f < 0) || (c < 0)) { return false; };
        if ((f == Constantes.FILAS_TABLERO) || (c == Constantes.COLUMNAS_TABLERO)) { return false;
        // Si no es el mismo valor: no hay
        if (posicion.tablero[f][c] != valor) { return false; };
    }
    // Si pasa todos los tests: OK, 4 en raya
    return true;
}
```

Imagen 12

El método `hay4EnFila` es el utilizado para establecer un cuatro en linea satisfactorio. Para ello se estudiarán varios casos que se deben cumplir, para considerar un cuatro en raya.

- Primero debe haber una ficha en la casilla inicial, de no ser así, se retornara `false.`
- Si efectivamente había una ficha, se estudiarán las tres posiciones consecutivas a ella, en las direcciones comentadas con anterioridad, con el método `Direccion.` Si alguna de estas posiciones, por cercanía a un borde, queda fuera de una posición válida del tablero, se retornará un `false.`
- Si se encontraban dentro del tablero, se comprobará que las fichas tenían el mismo valor, es decir que pertenecían al mismo jugador. Estos valores quedan recogidos en la clase `Valor` y pueden ser; `NONE`, `PLAYER1`, `PLAYER2`. De no ser así también se retornará `false`.

Si todas estas premisas son cumplidas, se retornará `true`, y por tanto se verifica que estamos en presencia de un cuatro en raya.

4.3.2 Posicion.java

En esta clase se puede ver un nuevo tipo de atributo, `Valor[ ][ ]tablero`, este sirve para crear un array de valores, para las posiciones del tablero. Como de costumbre estos serán de tipo `private`, para hacer uso de estos, únicamente en esta clase, o las que hereden de esta.

Inicialmente mediante el constructor `Posicion`, se tiene el valor de el `Player1` y se crean e inicializan los valores de todas las posiciones del tablero a `NONE`.

Mediante los métodos de esta clase, tendremos un valor para cada posición, que podremos preguntar con el método `getValor`, así sabremos si esta posición ya tiene una ficha o no, es decir si tiene un valor `NONE` o no.

El método `meterFicha` será utilizado para comprobar, si el valor de la fila y la columna seleccionados son válidos. De ser así, siempre que el valor de la posición sea `NONE`, se podrá meter una ficha en esta posición, a través de la columna seleccionada.
Si el turno tenía como valor al `PLAYER1`, ahora sera `PLAYER2` o viceversa.

El valor de `lastFila`, pasará a ser el de fila, esto servirá, para saber la fila en la que deberá caer la próxima ficha.
Se dispone de otros métodos para preguntar el valor del evaluador de la posición.

4.4 Window Package

4.4.1 MainFrame.java

Esta clase extiende de un `Jframe` con el objetivo claro de crear una ventana o contenedor, en la cual se puede a continuación introducir botones, cajas de texto, etc., añadiéndole unos métodos que interesan en nuestro caso.
Se crearán una serie de atributos para los tres botones horizontales de la parte superior, de modo que les daremos una posición y una dimensión, para determinar donde y de que tamaño serán. Estos serán de tipo `static`, para ser compartidos por los objetos de la clase y `final`, para no ser modificados.
Se crearán también los atributos para las opciones de los botones, ya que deberán ser unas opciones predefinidas y por tanto estos serán de tipo `enum`.

En este caso concreto, ya que se hereda de una clase padre y el constructor tiene el mismo nombre cosa que suele ser así por elegancia, se debe usar la palabra clave `super` para llamar al constructor y así poder distinguirlo.

Se establece algunos parámetros para el objeto a crear por el constructor, como son el tamaño de la ventana, el hecho de no ser *resizable* y volverla visible, etc., este deberá ser el último de ellos y solo la entrega del foco se realizará posteriormente.

A continuación, se tienen métodos como `setControlEventos` para la espera y respuesta, en función de si sucede un evento o se selecciona una `key` del teclado.

Para el botón de elección de jugador se tiene otro método y de igual modo para la elección de color. Este elegirá la opción por defecto de `VSplayer` y `Red` respectivamente a menos que no sean elegidas ellas. En tal caso se jugará `VScomputer` o `Yellow` si la selección se cambia.

También se tiene dos métodos para habilitar y deshabilitar los controles de los botones, el objetivo es no poder cambiar de jugador o color para hacer trampas. De este modo se tiene la opción de poder pedir que al comenzar la partida los controles sobre estos botones se deshabiliten.

Se tiene también un método para crear un `PanelControl`, este será explicado en una clase posterior. Además de ello, se le añadirán los botones desplegables de la elección de jugador, color y el botón `start`.
Por supuesto los métodos de los botones. Se utilizarán los atributos explicados, para determinar el nombre, la posición y el tamaño de estos elementos.
Anteponiendo a todos estos métodos tenemos el método `getPanelDibujo`, para obtenerlo de su correspondiente clase, la cual explicaremos posteriormente. Es necesario ya que necesitaremos tener el `PanelDibujo` cada vez que añadamos una nueva ficha a nuestro tablero.
Por último tenemos el método `crearMainPanel` con el que además de crear un nuevo `MainPanel`, añadiremos a este un `panelControl` y un `panelDibujo`, componiendo así nuestro `frame`.

Se puede decir de forma analítica, que creamos un `MainPanel`, que dispone de una zona de trabajo (800 ancho x 600 alto) y un menú de botones (800 ancho x 40 alto). Sobre este añadimos el `PanelControl` el cual añadirá nuestro tablero en color azul. Ademas sobre este mismo añadirá nuestro `PanelDibujo` que marcará un telón de fondo de color negro. Ambos dos sobre la zona de trabajo de 800x600.

4.4.2 MainPanel.java

Esta clase extiende de un `Jpanel`, igual que la clase `MainFrame`. Como sucede en esta necesitaremos utilizar la palabra clave `super` en el constructor para poder diferenciarlo de la clase padre.

En este caso el objeto creado tendrá unas dimensiones de 800 píxeles de ancho x 640 píxeles de alto. Así quedará definido la totalidad de nuestra ventana, u objeto a visualizar. Se determiná la localización de la ventana en el punto 0,0 o punto central del monitor.
Asignamos el valor de `layaut` a `null`, ya que seremos nosotros mismos los que determinaremos dónde y cómo aparecerán los botones en el panel, ya que no se desea que se haga de forma automática por java.

```java
import java.awt.Dimension;

@SuppressWarnings("serial")
public class MainPanel extends JPanel {

    public MainPanel() {
        super();
        setLayout(null);
        setSize(new Dimension(Constantes.WIDTH,Constantes.HEIGHT+Constantes.HEIGHT_CONTROLS));
        setLocation(0,0);
    }

}
```

Imagen 13

4.4.3 PanelControl.java

Esta clase tiene la misma estructura que `MainPanel` salvo por el tamaño de esta y su color. Determinaremos en su constructor un tamaño de 800 píxeles de ancho por 600 de alto y un color azul, ya que así se desea que sea el tablero.

4.4.4 PanelDibujo.java

Esta clase tiene la misma estructura que el resto de las clases de el package. Determinaremos en ella el color negro de fondo para nuestro tablero, así como el resto de parámetros en su construcción.

Se hace uso de la clase `graphics` para dibujar la imagen completa, el parámetro de salida será *`void`* ya que no necesita retornar ningún valor.

```java
import java.awt.*;


@SuppressWarnings("serial")
public class PanelDibujo extends JPanel {
    private BufferedImage image;

    public PanelDibujo() {
        super();
        setFocusable(true);
        setLayout(null);
        setSize(new Dimension(Constantes.WIDTH, Constantes.HEIGHT));
        setLocation(0,Constantes.HEIGHT_CONTROLS);
        setBackground(Color.black);
        image = new BufferedImage(800, 600, BufferedImage.TYPE_INT_ARGB);
    }

    public BufferedImage getImage() {
        return image;
    }

    public void paint(Graphics g) {
        g.drawImage(image, 0, 0, null);
    }
}
```

Imagen 14

4.5 Window.Gráficos Package

4.5.1 ControlGrafico.java

En esta clase se crearán los atributos necesarios para los diferentes elementos del juego; tablero, fichas, etc...
A continuación se crearán los diferentes elementos en el constructor. Así como las fichas de cada jugador, donde serán asignadas por colores.

Con el método `moveAndDraw` dibujamos el tablero con sus elementos, de modo que el movimiento de las fichas resulte en una correcta visualización de este. Para ello primero se moverán los elementos móviles, después se dibujará el fondo para que quede correcto, los números de igual modo, las fichas presentes y el tablero en último lugar, para que las fichas queden detrás de este, como debe verse y no por delante.

```
public void moveAndDraw() {
    Graphics g = buffer.getGraphics();
    // Movemos los elementos móviles
    for (Ficha f:fichas) { f.mover(); }
    // Dibujamos el fondo
    g.setColor(new Color(0,0,0));
    g.fillRect(0, 0, Constantes.WIDTH, Constantes.HEIGHT);
    // Dibujamos los números
    numeros.draw(g);
    // Dibujamos las fichas
    for (Ficha f:fichas) { f.draw(g); }
    // Dibujamos el tablero
    tablero.draw(g);
}

public void tirarFicha(int col, int filaDest, boolean player1) {
    Ficha ficha = null;
    Ficha[] fichasPlayer = (player1 ? fichasPlayer1 : fichasPlayer2);
    // Color de los números
    numeros.setSelectedMove(col, player1);
    // Buscamos una ficha libre para tirar
    for (Ficha f:fichasPlayer) { if (!f.isVisible()) { ficha = f; break; }; };
    // Calculamos coordenadas
    int x = tablero.getXCol(col);
    int y0 = tablero.getYStart();
    int y1 = tablero.getYFila(filaDest);
    // Tiramos la ficha
    ficha.caer(x, y0, y1);
}
```

Imagen 15

Con el siguiente método `tirarFicha` se determina la ficha a tirar, primero será la del `player1` y posteriormente la del `player2`. Con ello se determina el color en el que debe iluminarse el número correspondiente a la columna por la que posteriormente se dejará caer la ficha. Este proceso se llevará a cabo hasta que se terminen las fichas o algún jugador gane. Antes de dejar caer la ficha se determinan las coordenadas y posteriormente deja caer la ficha.

Al finalizar la partida se vaciará el tablero dejando caer la totalidad de las fichas, antes de poder comenzar una nueva partida.

4.5.2 Ficha.java

Determinaremos en primer lugar, los atributos de la ficha como su posición, su color, si esta en movimiento o no, si es visible, etc., y posteriormente los crearemos.

En el método `caer` determinaremos nuestro `tick` de tiempo en función de la frecuencia de refresco necesaria para ser vista de forma fluida por el ser humano (25FPS). Además si la ficha esta en movimiento siempre será visible. Se guardarán los valores de la componente `Y`, ya que será necesaria para saber hasta qué posición de la columna debe caer la ficha.

```
public void caer(int x, int y0, int y1) {
    // La colocamos en posicion de partida
    pos.setX(x); pos.setY(y0);
    enMovimiento = visible = true;
    this.y0 = y0;
    this.y1 = y1;
    ticks = (int) Math.round(-0.2 * Constantes.FPS);
}

public void draw(Graphics g) {
    if (!visible) { return; };
    g.setColor(color);
    g.fillOval(pos.getX(), pos.getY(), Constantes.DIAMETRO_FICHA, Constantes.DIAMETRO_FICHA);
}

public void mover() {
    if (!enMovimiento) { return; };
    ticks++; if (ticks < 0) { return; };
    double y = (Constantes.ACELERACION * ticks * ticks / 2.0) + y0;
    pos.setY(y);
    if (pos.getY() >= y1) { pos.setY(y1); enMovimiento = false; };

}
```

Imagen 16

En el método `mover` se determina cómo la ficha se mueve por la columna seleccionada.

En cada `tick` de tiempo hasta que estos llegan a 0, siendo incrementado desde -5, se calcula la posición de `Y` en función a una caída realista basada en la fórmula de la aceleración.

Si `tick` está en un valor menor de 0, significará que la columna está llena de fichas.

Así las fichas irán siendo colocadas una encima de otra hasta completar la columna.

4.5.3 Numeros.java

En esta clase se seleccionará el tipo de letra, así como su tamaño y su color de todos los números correspondientes a las diferentes columnas del juego.

Inicialmente se fijará el turno del `player1`, por tanto el número elegido que corresponderá a la primera ficha será de color rojo, y posteriormente amarillo para el `player2`.

Una vez elegidos los colores se pintarán en la zona superior del tablero en la fila y columna correspondientes, según se a tomado como referencia el diámetro de una ficha, será el correspondiente a una séptima altura si la hubiese.

4.5.4 Pos.java

En esta clase se definen los diferentes atributos que serán necesarios para la clase posición. Así a la posición se le puede preguntar el valor de su `x` e `y`, así como definirlas entre otras opciones.

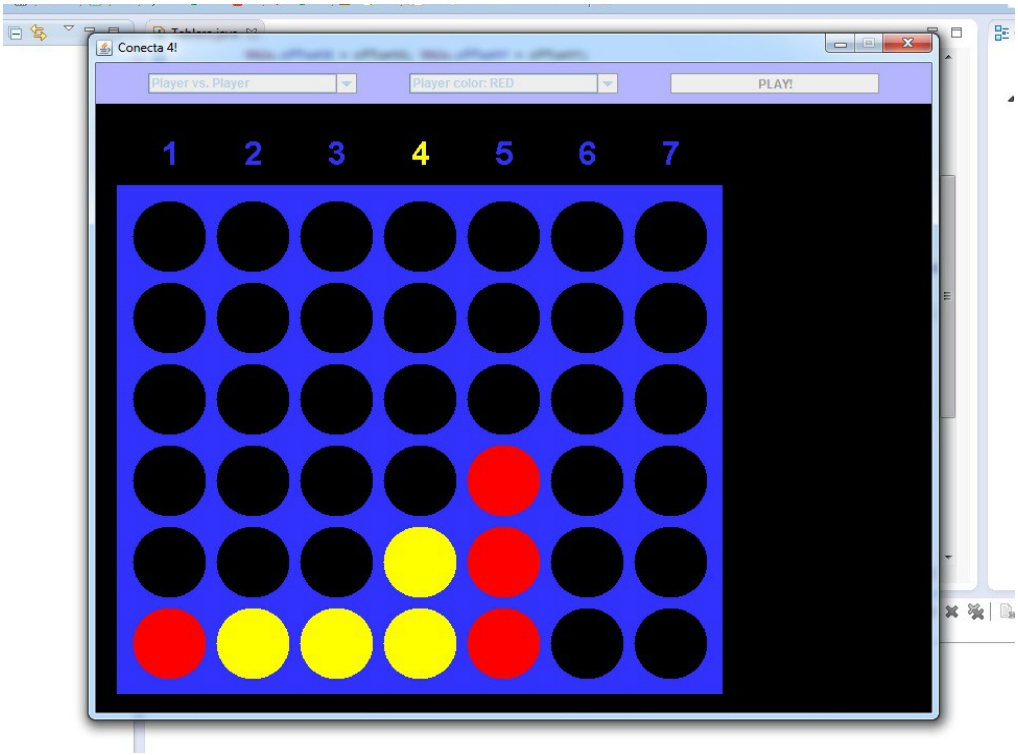
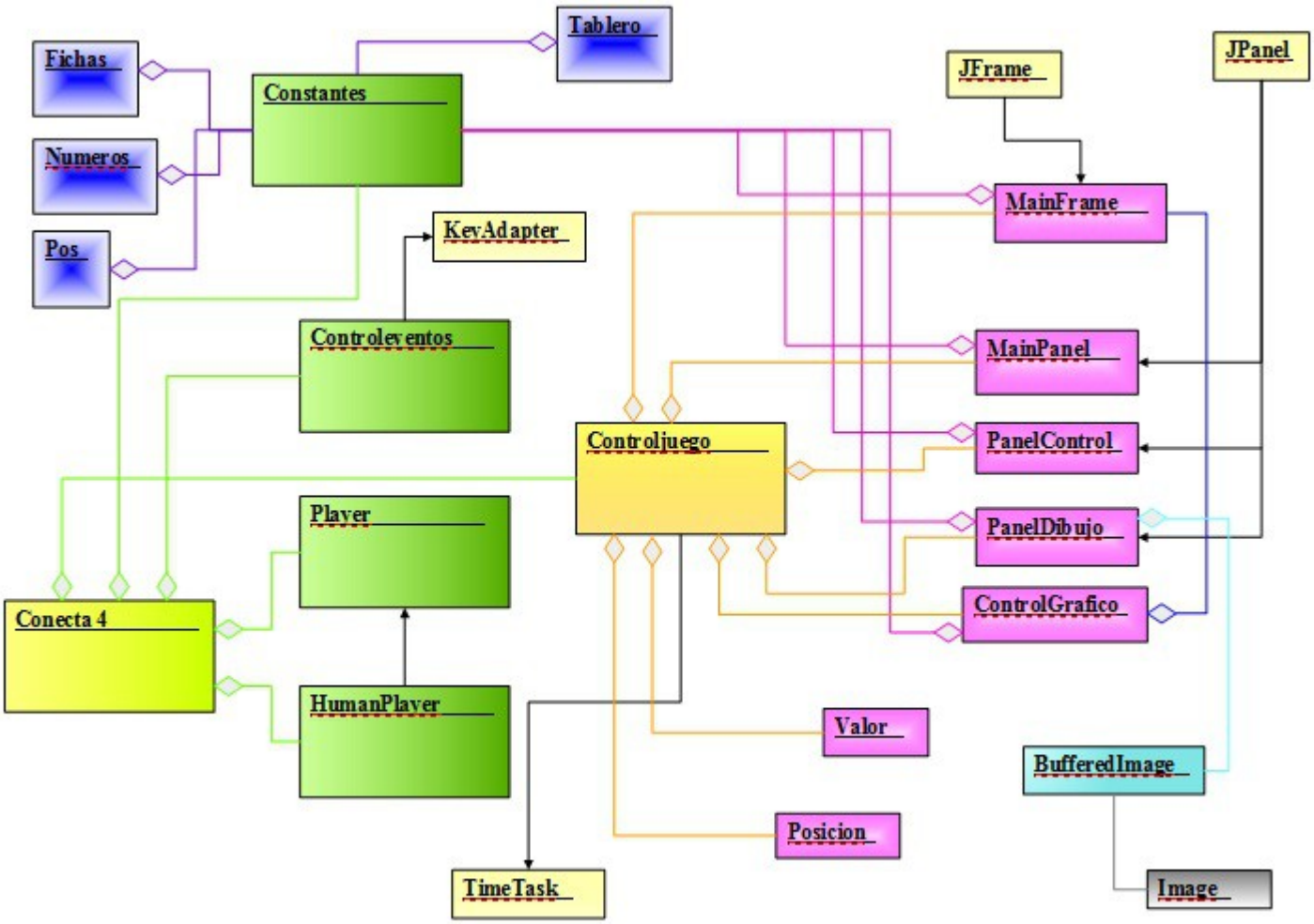


Imagen 17

4.6 Diagrama de Clases



5. PROYECTO JAVA, HUNDIR LA FLOTA

5.1 Introducción

El diseño de la interfaz gráfica en este caso se ha realizado mediante una herramienta de *Netbeans* [12], la cual permite mediante ventanas, botones, etc., crear el diseño que se le ha dado a este juego.
Este modo de diseño autogenera código en la clase de java, que corresponde a los elementos añadidos.

En la paleta de diseño aparecerán todos los *swings* utilizados, ya sean `containers`, `controls`, etc.

Esta clase se ha diseñado para ver el modo de empleo de los hilos así como de las clases de test automático, que gestionan el buen funcionamiento de la información.

Los hilos [13,14] se han utilizado como posteriormente se explicará para crear, mantener y gestionar la comunicación entre el servidor y el cliente en este juego, ya que este juego esta creado para jugarse en red. El encargado de hacer la llamada será el cliente y el jugador que ha sido retado sera el servidor.

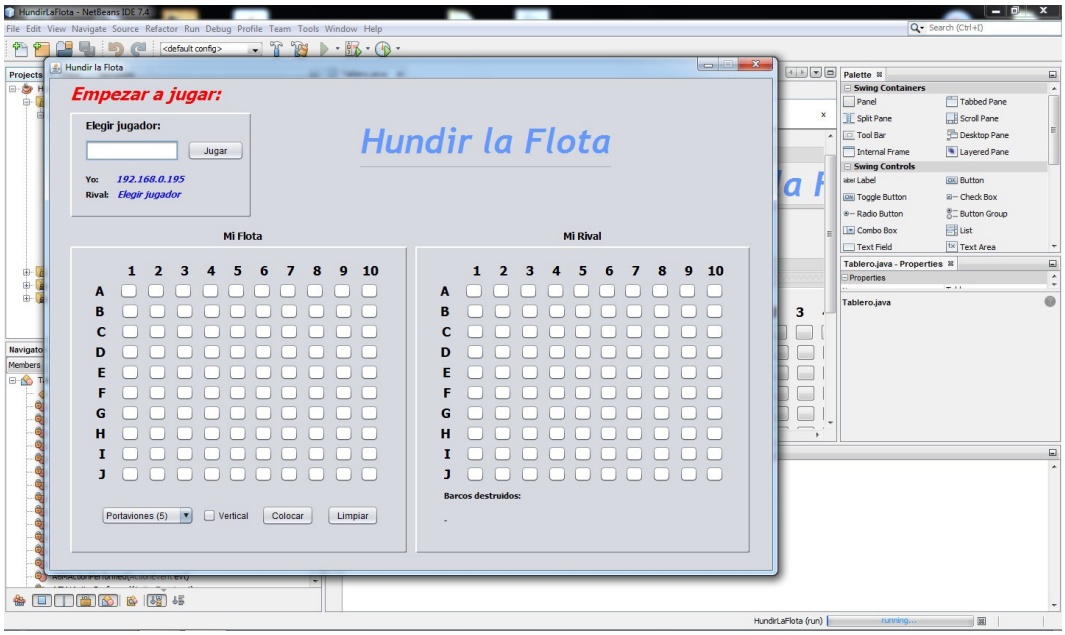


Imagen 18

5.2 Funcionamiento de la Aplicación

En primer lugar se introduce la *Ip* del usuario con el que se desea jugar. El método `ipconectActionPerformed` de la clase principal de nuestro programa cuyo nombre es `Tablero` verá si estamos desconectados, para ello comprobará el valor del botón conectar y verá si dicho valor es `jugar.` De ser así, se confirma que estamos desconectados.

```
private void ipconectActionPerformed(java.awt.event.ActionEvent evt) {
    if (botonconectar.getText().equals("Jugar")){//si esta desconectado
        if (! ipconect.getText().equals(Utilities.getIp())){
            c=new Client(ipconect.getText(),this);
            c.start();
        }
        else{
            showsms("Juego de dos personas. No puedes jugar contra ti.");
        }
    }
    else{//si esta conectado
        //cierro conexion con el otro usuario e inicializo todo el programa
        u.cerrar();
        setColorTablero();
        iniTablero();
        setSelect();
        colocar.setEnabled(true);
        botonconectar.setText("Jugar");
        ipconect.setText("");
        ipconect.setEnabled(true);
        listbarcos.setText("");
        serverlabel.setText("Elegir jugador");
        eb.prohibidos.clear();
    }
```

Imagen 19

Seguidamente comprobará que la *Ip* introducida no es la gestionada por el método `getIp` de la clase `utilities`, así confirmaremos que la *Ip* introducida es válida y no es la del *host*. Esta última se realiza como muchas otras entre un `try` y un `catch` ya que puede suceder una excepción si se rompe la conexión en este proceso, entre el cliente y el servidor.

La clase `utilities` mencionada se usará para contener todos los métodos que no se desean introducir dentro de la clases propias de la aplicación. Hará las veces de una librería de métodos.

Se iniciará una nueva clase cliente y se le darán la *Ip* introducida así como la clase tablero.

El método principal de esta clase será la que gestione el establecimiento de la conexión. Este es el método `pedirConexion`, en primer lugar iniciará el método conexión. Mediante el método `conectar,` de esta clase `cliente,` se comprueba si no esta conectado con alguien.

De ser así se crea un nuevo `soket` [15] por el que se establece la conexión entre el cliente y el servidor, por el puerto `8000`. Este puerto se encuentra en modo de escucha en el servidor y aceptara la conexión.

También se crearan los flujos de datos para el cliente hacia el servidor `ObjectOutputStream` y para el servidor hacia el cliente `ObjectInputStream` para poder intercambiar datos, una vez establecida la conexión, por parte del servidor [16] Seguidamente el método `pedirConexion` vía canal `ObjectOutputStream` mandará un mensaje `'100'` al servidor.

```
public void conectar(){
    try{

        if (! tab.estaConectado){//si no esta conectado con nadie
            //establezco conexion con el otro jugador
            //tab.s=new Socket(ip,8000);
            tab.s=new Socket();
            tab.s.connect(new InetSocketAddress(ip,8000), 2500);
            //creo los flujos por los que intercambiar datos
            tab.oo=new ObjectOutputStream(tab.s.getOutputStream());
            tab.oi=new ObjectInputStream(tab.s.getInputStream());
            //inicializo componentes del interfaz principal
            tab.estaConectado=true;
            tab.getIpConect().setEnabled(false);
            tab.getBotonConectar().setText("Parar");
        }
    }catch(Exception ex){
        showsms("Error al conectar: "+ex.getMessage());
    }
```

Imagen 20

A lo que el servidor al recibir este mensaje `'100'` cambiará el estado de conexión a `true` y pondrá en el estado de turno a `false` ya que el cliente lo ha puesto a `true`, y se desea que el jugador que establece la conexión sea el primero en jugar. Además obtendrá la

dirección de la *Ip* elegida por el cliente, deshabilitará la opción para que no se pueda cambiar una vez conectado y pasará el botón a valer `parar.`

```
/*Metodo principal que se encarga de:
    - Llamar al metodo de iniciar conexion
    - Envia la peticion de conexion (codigo 100)
    - Mientras seguir=true lee todo lo que le llegue por el socket y actua en consecuecia*/
public void pedirConexion(){
    try{
        conectar();
        tab.oo.writeObject("100");
        tab.eb.turno=true;
        while(seguir){
            buffer=leerinfo();
            if (buffer.equals("101")){//confirmacion de conexion
                showsms("Conectado con "+tab.s.getInetAddress().toString().substring(1));
                tab.estaConectado=true;
                tab.getServerLabel().setText(tab.getIpConect().getText());
```

Imagen 21

Acto seguido el servidor manda un mensaje `'101'` al cliente para verificar la conexión. A lo que el cliente pasará el valor de `estaconectado` a `true`.

```
public void atenderPeticiones(){
    try{
        while(seguir){
            bufstring=leerStringSocket();
            if (bufstring.equals("100")){//recibo una peticion de conexion para jugar
                tab.estaConectado=true;
                tab.eb.turno=false;
                tab.getIpConect().setText(tab.s.getInetAddress().toString().substring(1));
                tab.getIpConect().setEnabled(false);
                tab.getBotonConectar().setText("Parar");
                //actualizo cartel
                tab.getServerLabel().setText(tab.s.getInetAddress().toString().substring(1));
                tab.oo.writeObject("101");
                showsms("Conectado con "+tab.s.getInetAddress().toString().substring(1));
```

Imagen 22

Una vez hecho esto los `labels` tanto del cliente como el servidor tienen las *Ips* de ellos mismos y las de los contrincantes correspondientes.

Se incializan componentes del interfaz como el estado de la conexión, el cual será afirmativo ahora. La celda para introducir la *Ip* está ahora sombreada para evitar el añadir otra *Ip* una vez establecida la conexión. El valor del botón de conexión pasará ahora a valer `parar`.
Si ocurre un error en este proceso debido a una caída de la conexión por parte de uno o ambos saltará un error de excepción ya que este método se encuentra entre un `try` y un `catch`, y se mostrará un mensaje informando del tipo de error acontecido.

De no ser una *Ip* válida se mostrará `"Juego de dos personas. No puedes jugar contra ti."` al usuario, mediante una ventana emergente, ya que la *Ip* introducida será la del *host*.

Si el valor del botón no era `jugar`, lo que hace que valga `parar`, ya que solo toma estos dos valores, supondrá que el cliente ya estaba conectado. En este caso se cierra la conexión con el servidor y se inicializan todos los valores del programa al punto de inicio. Así se podrá iniciar una futura partida.

Para ello primero se llama al método `cerrar` de `Utilities`, el cual manda un mensaje `'200'` al servidor. Este restablecerá los parámetros a los valores de inicio y responderá al cliente con un mensaje `'201'`, a lo que el cliente confirmará poniendo a `false` el parámetro `seguir.`

```
else if (buffer.equals("200")){//peticion de cierre
    if (tab.estaConectado){
        tab.oo.writeObject("201");
        tab.setColorTablero();
        tab.iniTablero();
        tab.setSelect();
        tab.getColocar().setEnabled(true);
        tab.getLimpiar().setEnabled(true);
        tab.getBotonConectar().setText("Jugar");
        tab.getIpConect().setText("");
        tab.getIpConect().setEnabled(true);
        tab.getServerLabel().setText("Elegir jugador");
        tab.getListBarcos().setText("- ");
        tab.estaConectado=false;
        tab.eb.prohibidos.clear();
        tab.eb.miflotalista=false;
        tab.eb.rivalflotalista=false;
        seguir=false;
    }
}
else if (buffer.equals("201")){//confirmacion de cierre
    seguir=false;
}
else if (buffer.equals("300")){//recibo disparo
    if (u.actualizoFlota(Integer.parseInt(lanzamiento))){
        tab.oo.writeObject("301-"+lanzamiento);
```

Imagen 23

El botón de `jugar`, hace exactamente lo ya mencionado para la celda de la *Ip*, solo añadirá un par de reinicios adicionales para las flotas, ya que si no dejará colocar flotas una vez parada la partida lo cual era un error en la versión beta y el estado de conexión.
Una vez ya establecida la conexión y los flujos de datos, la partida se considera iniciada así como conectada. Es el momento de colocar los barcos en el tablero. Esto será gestionado por el método `colocarBarco` de la clase principal `Tablero`. Este método necesita como argumentos la posición inicial, el tipo de barco y el modo, ya sea horizontal o vertical, en el que se desea colocar.

Se almacenará en un vector las posiciones de los barcos para después comprobar si alguna de ellas viola alguna restricción (posición ocupada, posición perímetro prohibido o barco partido), de no ser así se colocará el barco, cambiando de color los botones correspondientes.

El método comenzará comprobando si el estado de la conexión es conectado, de ser afirmativo, se iniciarán los vectores de posiciones y variables necesarias.

En primer lugar se comprueba si el estado de la posición elegida es `false`, es decir se comprueba si esta a `!false` y por tanto esta desocupada, si además cumple que el barco no esta partido es decir entra por completo en el tablero, con lo que el valor será también `!false`, se añadirá esta posición al vector de posiciones `'casillas'` de los barcos poseídos por el jugador.

```
if (estaConectado){//si esta conectado
    Barco b=null;
    int [] bpaux=null;
    Posicion pos;
    boolean check=true;
    Vector<Posicion> casillas=new Vector <Posicion>();
    pos=(Posicion)tablero.get(inicio);
    if ((! pos.getEstado()) && (! u.barcoPartido(inicio, barco, posicion))){
        casillas.add(pos);
    }
    else{
        check=false;
    }
    if (check){
        int cont=0;
        int rep=0;
        String nombrebarco="";
        if (barco.equals("Portaviones (5)")){
            rep=4;
            b=new Barco(5);
            bpaux=new int[5];
            nombrebarco="Portaviones (5)";
```

Imagen 24

Para comprobar el estado de la posición se le pregunta a la clase `Posicion`, por el método `getEstado`, así será devuelto un `booleano 'b'`.

```java
public class Posicion {

    private JButton boton;
    private boolean b;

    public Posicion(JButton boton, boolean b){
        this.boton=boton;
        this.b=b;
    }
    //devuelve el boton
    public JButton getBoton(){
        return boton;
    }
    //devuelve el estado
    public boolean getEstado(){
        return b;
    }
    //cambia el estado del boton
    public void setEstado(boolean estado){
        b=estado;
    }
```

Imagen 25

Para comprobar si el barco esta partido se utilizará el método `barcoPartido`, de la clase `utilities`. Este método comprueba el barco que se ha seleccionado, pongamos por ejemplo que se ha seleccionado el barco `PORTAVIONES` el cual tiene 5 casillas de extensión. En este caso se comprueba que la posición inicial si ha sido colocado en estado vertical, no sea mayor o igual que 61, ya que eso implicaría que se encuentra en la fila 7, columna 1 y por tanto no entraría en el tablero, o en una posición superior por lo que tampoco entraría en el tablero.

```
public boolean barcoPartido(int inicio,String barco,boolean posicion){
    try{
        boolean res=false;
        if (posicion){//vertical
            if (barco.equals("Portaviones (5)")){
                if (inicio>=61){
                    res=true;
                }
            }
            else if (barco.equals("Acorazado (4)")){
                if (inicio>=71){
                    res=true;
                }
            }
            else if (barco.equals("Destructor 1 (3)")){
                if (inicio>=81){
                    res=true;
                }
            }
            else if (barco.equals("Destructor 2 (3)")){
                if (inicio>=81){
                    res=true;
                }
            }
            else if (barco.equals("Fragata (2)")){
                if (inicio>=91){
                    res=true;
```

Imagen 26

Este proceso se repite cambiando el valor a comparar para el resto de barcos, según su extensión.

Análogamente este proceso se repite para el estado horizontal del barco, comprobándose que en todas las filas según la extensión del barco entre de forma completa.

Si no se cumple alguna de estos compromisos, el valor `check` será `false` y el barco no será colocado.

Si satisface este compromiso, `check` seguirá valiendo `true` y se creará un nuevo barco con los parámetros correspondientes al barco seleccionado por el jugador.

```
if (check){
    int cont=0;
    int rep=0;
    String nombrebarco="";
    if (barco.equals("Portaviones (5)")){
        rep=4;
        b=new Barco(5);
        bpaux=new int[5];
        nombrebarco="Portaviones (5)";

    }
    else if (barco.equals("Acorazado (4)")){
        rep=3;
        b=new Barco(4);
        bpaux=new int[4];
        nombrebarco="Acorazado (4)";
    }
    else if (barco.equals("Destructor 1 (3)")){
        rep=2;
        b=new Barco(3);
        bpaux=new int[3];
        nombrebarco="Destructor 1 (3)";
    }
    else if (barco.equals("Destructor 2 (3)")){
        rep=2;
        b=new Barco(3);
        bpaux=new int[3];
        nombrebarco="Destructor 2 (3)";
    }
    else if (barco.equals("Fragata (2)")){
        rep=1;
        b=new Barco(2);
        bpaux=new int[2];
```

Imagen 27

Este barco se colocará de estar en vertical, la primera casilla en la posición seleccionada, con `'rep=4,de ser un portaviones(5)'` y sus cuatro casillas siguientes se colocaran añadiendo 10 casillas a un contador antes de poner a **`true`** el botón correspondiente a la siguiente casilla del barco, restando 1 al valor de `rep`, con cada interacción, consiguiendo así que un barco de 2 casillas tenga las posiciones por ejemplo 24 y 34 a `true` *(todavía no valen true).*

```
if (check){
    int cont=0;
    int rep=0;
    String nombrebarco="";
    if (barco.equals("Portaviones (5)")){
        rep=4;
        b=new Barco(5);
        bpaux=new int[5];
        nombrebarco="Portaviones (5)";

    }
    else if (barco.equals("Acorazado (4)")){
        rep=3;
        b=new Barco(4);
        bpaux=new int[4];
        nombrebarco="Acorazado (4)";
    }
    else if (barco.equals("Destructor 1 (3)")){
        rep=2;
        b=new Barco(3);
        bpaux=new int[3];
        nombrebarco="Destructor 1 (3)";
    }
    else if (barco.equals("Destructor 2 (3)")){
        rep=2;
        b=new Barco(3);
        bpaux=new int[3];
        nombrebarco="Destructor 2 (3)";
    }
```

Imagen 28

Igualmente se repite este proceso para el estado horizontal con la diferencia de que el contador sumará de 1 en 1 para cada botón a añadir al barco. Si las posiciones en las que debe estar el barco están en `false`, este barco es válido `(check=true)`, de lo contrario `(check=false)`.

```
            while (rep>0){
                cont=cont+10;
                pos=(Posicion)tablero.get(inicio+cont);
                if (! pos.getEstado()){
                    casillas.add(pos);
                    bpaux[indiceb]=inicio+cont;
                    indiceb++;
                }
                else{
                    check=false;
                }
                rep--;
            }
        }
        else{
            while (rep>0){
                cont=cont+1;
                pos=(Posicion)tablero.get(inicio+cont);
                if (! pos.getEstado()){
                    casillas.add(pos);
                    bpaux[indiceb]=inicio+cont;
                    indiceb++;
                }
                else{
                    check=false;
                }
                rep--;
```

Imagen 29

Una vez cumplidos los dos compromisos anteriores `check` vale `true.`

Por último para validar el barco y poder dibujarlo, tiene que cumplir con el compromiso de no estar en el perímetro de otro barco `(! per, debe ser true).`

Si se cumplen ambas el fondo de los botones pasará a ser elegido como negro y los estados de los botones pasarán a ser `true`.

Antes de explicar como se comprueba el perímetro, explicaré como se ha establecido este para el barco.

Mediante el método `cercarPerimetro` de la clase `EstadoBarco`, se añaden a un vector de prohibidos las posiciones que pertenecen al perímetro del barco.

Para establecer el perímetro de un barco vertical, se añaden a este vector de prohibidos las posiciones con respecto al indice (0) del barco, la posición de 10 casillas anterior (casilla superior) y con respecto a esta posición la anterior y posterior a ella, también la posición con respecto al índice (n-1) del barco, la posición de 10 casillas posterior (casilla inferior) y sus casillas siguiente y anterior. Por último se añaden las casillas siguiente (+1) y anterior (-1) desde el índice 0 hasta el n del barco.

```
public void cercarPerimetro(Barco bar, boolean posicion){
    if (posicion){//vertical
        int poscheck;
        int x;
        int izq=bar.pos[0]-10;
        prohibidos.add(izq+"");
        poscheck=izq;
        x=izq-1;
        if (!(poscheck % 10 == 1)){
            prohibidos.add(x+"");
        }
        x=izq+1;
        if (!(poscheck % 10 == 0)){
            prohibidos.add(x+"");
        }
        int der=bar.pos[bar.n-1]+10;
        prohibidos.add(der+"");
        poscheck=der;
        x=der-1;
        if (!(poscheck % 10 == 1)){
            prohibidos.add(x+"");
        }
        x=der+1;
        if (!(poscheck % 10 == 0)){
            prohibidos.add(x+"");
        }
        for (int i=0;i<bar.n;i++){
            poscheck=bar.pos[i];
            if (!(poscheck % 10 == 1)){
```

Imagen 30

De la misma manera se realiza el mismo proceso para los barcos en posición horizontal.

Para comprobar ahora el perímetro se comprueba si la posición del barco entra en conflicto con alguna de las posiciones del vector de prohibidos mediante el método `esPerimetro` de la clase `Utilities`. Una vez elegido el barco del menú de barcos, cercado su perímetro y colocado, se elimina este del menú de barcos y se obliga al jugador a elegir un nuevo barco a colocar.

Si ya se han colocado todos los barcos y por tanto el índice del menú de barcos vale -1, se cancela el uso del botón colocar y limpiar. El parámetro `miflotalista` pasa a valer `true` y se establece una conexión con el servidor para hacerle saber que se está listo para jugar.

```
    return ip;
}
//metodo con el que comunico al otro usuario que he terminado de colocar mi flota
public void estoyListo(){
    try{
        if (tab.estaConectado){//debe cumplir que este conectado a otro jugador
            tab.oo.writeObject("400");
            tab.oo.flush();
            tab.eb.miflotalista=true;
        }
    }catch(Exception ex){}
}
//metodo con el que simulo lanzar un cohete a mi rival en la coordenada (un entero) pasada como parametro
public void lanzarCohete(int id){
    if (tab.eb.miflotalista){//debe cumplir que mi flota este colocada
        if (tab.eb.rivalflotalista){//debe cumplir que la flota de mi rival este colocada
            if (tab.eb.turno){//debe ser mi turno
                try{
                    tab.oo.writeObject("300-"+id);
                    tab.oo.flush();
                    tab.eb.turno=false;
                }catch(Exception ex){

                }

            }
            else{
```

Imagen 31

Para comunicar el estado `listo`, se utiliza el método `estoyListo` de la clase `Utilities`. Se manda un mensaje `'400'` por el canal `ObjectOutputStream` al servidor, el cual mostrará en el servidor una ventana emergente donde pondrá escrito `'Tu rival esta listo para empezar'`. Además se pondrá el valor de `rivalflotalista` a `true`. Este mensaje es recíproco del servidor al cliente en caso de que él coloque los barcos primero.

```
        else{
            tab.oo.writeObject("302-"+lanzamiento);
        }
        tab.eb.turno=true;
        showsms("Tu turno");
    }
    else if (buffer.equals("301")){//Confirmacion de que le he dado
        u.actualizoFlotaRival(Integer.parseInt(lanzamiento), true);
    }
    else if (buffer.equals("302")){//confirmacion de que NO le he dado
        u.actualizoFlotaRival(Integer.parseInt(lanzamiento), false);
    }
    else if (buffer.equals("400")){//mi rival esta listo
        tab.eb.rivalflotalista=true;
        showsms("Tu rival esta listo para empezar");
    }
    else if (buffer.equals("500")){//confirmacion de que mi rival a ha perdido
        showsms("ENHORABUENA. Has ganado a tu rival. Eres el rey de los mares");
        //seguir=false;
    }
    else if (buffer.equals("501")){
        showsms("PORTAVIONES de tu rival destruido");
        tab.getListBarcos().setText(tab.getListBarcos().getText()+"PORTAVIONES  ");
    }
    else if (buffer.equals("502")){
        showsms("ACORAZADO de tu rival destruido");
        tab.getListBarcos().setText(tab.getListBarcos().getText()+"ACORAZADO  ");
    }
    else if (buffer.equals("503")){
        showsms("DESTRUCTOR 1 de tu rival destruido");
        tab.getListBarcos().setText(tab.getListBarcos().getText()+"DESTRUCTOR 1  ");
```

Imagen 32

Una vez colocados los barcos se pasa a la siguiente fase del juego, que consiste en disparar cohetes al tablero derecho para intentar tocar y hundir, los barcos del jugador rival.

Mediante el método `lanzarCohete` de la clase `Utilities` lanzaremos un cohete al botón elegido. Este método comprueba que el valor de `miflotalista` y de `rivalflotalista` sea `true` ya que toda la flota debe estar colocada, para llegar a esta fase del juego.

Si es afirmativo y además es el turno del jugador, ya que si no se mostrará un mensaje informativo diciendo que `'no es tu turno'`, entonces, se manda un mensaje `'300-+id` al servidor, siendo ID el valor del botón elegido para el cohete.

```
public void lanzarCohete(int id){
    if (tab.eb.miflotalista){//debe cumplir que mi flota este colocada
        if (tab.eb.rivalflotalista){//debe cumplir que la flota de mi rival este colocada
            if (tab.eb.turno){//debe ser mi turno
                try{
                    tab.oo.writeObject("300-"+id);
                    tab.oo.flush();
                    tab.eb.turno=false;
                }catch(Exception ex){

                }

            }
            else{
                showsms("No es tu turno");
            }
        }
        else{
            showsms("El rival no ha colocado su flota");
        }
    }
    else{
        showsms("No has colocado toda tu flota");
    }
```

Imagen33

El valor del turno del cliente pasará a ser `false`.

El servidor recibirá el mensaje y actualizará la flota. Para ello el método `actualizoFlota` de la clase `Utilities` comprueba si el valor de ID coincide con la posición de un barco en el tablero. De ser así se comprueba el valor de ID con el método `haImpactado` de esta misma clase. Se comprueba en que barco ha impactado el cohete y se añade a la lista de este barco.

El botón correspondiente a la posición del disparo del cohete cambia de color a rojo si ha impactado y de lo contrario pasa a ser azul.

El servidor mandará como respuesta un mensaje **301** mas el objetivo de su disparo, a lo que el cliente confirmará que le ha dado y actualizará la flota del rival. Con el método `acualizoFlotaRival` se comprueba si la ID corresponde a la posición de un barco y se colocará el botón rojo si le ha tocado o azul si no le ha tocado en la pantalla de la derecha (pantalla que corresponde a la del rival en el cliente).

```
                showsms("Tu "+suceso+" ha sido destruida");
            }
        }
        else{
            tab.oo.writeObject("302-"+lanzamiento);
        }
        //cambio de turno
        tab.eb.turno=true;
        showsms("Tu turno");
    }
    else if (bufstring.equals("301")){//Confirmacion de que le he dado
        u.actualizoFlotaRival(Integer.parseInt(lanzamiento), true);
    }
    else if (bufstring.equals("302")){//confirmacion de que NO le he dado
        u.actualizoFlotaRival(Integer.parseInt(lanzamiento), false);
    }

    else if (bufstring.equals("400")){//mi rival esta listo
        tab.eb.rivalflotalista=true;
        showsms("Tu rival esta listo para empezar");
    }
    else if (bufstring.equals("500")){
        showsms("ENHORABUENA. Has ganado a tu rival. Eres el rey de los mares");
        //seguir=false;
    }
    else if (bufstring.equals("501")){
        showsms("PORTAVIONES de tu rival destruido");
        tab.getListBarcos().setText(tab.getListBarcos().getText()+"PORTAVIONES  ");
    }
    else if (bufstring.equals("502")){
        showsms("ACORAZADO de tu rival destruido");
```

Imagen 34

El servidor comprueba también si ha ocurrido un suceso, debido al disparo del cliente. Para ello la clase `EstadoBarcos` tiene el método `ocurreSuceso`. Aquí se comprueba la lista de cada barco para saber si alguno de ellos ha sido hundido.

De ser así se mostrará en la parte inferior de la pantalla, el barco hundido.

Para comprobar si un barco ha sido hundido se llama al método de la clase **`Barco`**, `estaHundido`, con ella se comprueba si el numero de casillas del barco (n), coincide con el número de veces tocado, de ser así *b*, pasa a valer **`true`**, lo que significa que el barco esta hundido.

```
public void setPos(int [] pos){
    this.pos=pos;
    estado=new HashMap();
    for (int i=0;i<pos.length;i++){
        estado.put(pos[i], false);
    }
}
//metodo que me dice si un barco está hundido (todos las posiciones del barco a true)
public boolean estaHundido(){
    int cont=0;
    boolean b=false;
    for (int i=0;i<pos.length;i++){
        if ((boolean)estado.get(pos[i])){
            cont++;
        }
    }
    if (cont==n){
        b=true;
    }
    return b;
}
//comprueba si la coordenada pasada como parametro coincide con alguna de barco
public boolean tocado(int id){
    boolean res=false;
    for (int i=0;i<n;i++){
        if (id==pos[i]){
            res=true;
            estado.put(id,true);
        }
    }
    return res;
```

Imagen 35

Si un barco ha sido hundido el método `ocurreSuceso` reportará el suceso con el nombre del barco hundido (por ejemplo; `suceso="PORTAVIONES"`).

Este suceso será mandado al cliente, con un mensaje 501 o 502 o 503 o 504 o 505, según el barco hundido y se mostrará una ventana emergente con el mensaje; `'Tu PORTAVIONES ha sido destruido'`, para el caso del ejemplo 501. El cliente recibirá

este mensaje y ante él aparecerá una pantalla con el mensaje; `"PORTAVIONES de tu rival destruido".`

Si el método `ocurreSuceso` comprueba que todos los barcos están hundidos, el suceso será `perdido`, de ser así se mandará un mensaje `'500'` al cliente y en el servidor aparecerá un mensaje informando de que se ha perdido la partida. El cliente por el contrario al recibir este mensaje `'500'` habrá ganado y en su ventana aparecerá un mensaje felicitándole por la victoria.

```
public String ocurreSuceso(){
    String suceso="nada";
    Barco bar1=(Barco)flota.get("Portaviones (5)");
    Barco bar2=(Barco)flota.get("Acorazado (4)");
    Barco bar3=(Barco)flota.get("Destructor 1 (3)");
    Barco bar4=(Barco)flota.get("Destructor 2 (3)");
    Barco bar5=(Barco)flota.get("Fragata (2)");
    //compruebo a ver si esta perdida la partida
    if ((bar1.estaHundido()) && (bar2.estaHundido()) && (bar3.estaHundido()) && (bar4.estaHundido()) && (ba
        suceso="perdido";
    }
    else{//si no, compruebo uno a uno a ver si se ha undido algun barco
        if((bar1.estaHundido()) && (! dest.contains(bar1))){
            suceso="PORTAVIONES";
            dest.add(bar1);
        }
        else if((bar2.estaHundido()) && (! dest.contains(bar2))){
            suceso="ACORAZADO";
            dest.add(bar2);
        }
        else if((bar3.estaHundido()) && (! dest.contains(bar3))){
            suceso="DESTRUCTOR 1";
            dest.add(bar3);
        }
        else if((bar4.estaHundido()) && (! dest.contains(bar4))){
            suceso="DESTRUCTOR 2";
            dest.add(bar4);
        }
        else if((bar5.estaHundido()) && (! dest.contains(bar5))){
            suceso="FRAGATA";
```

Imagen 36

Una vez finalizada la partida se cierra el `soket` así como los flujos de salida y entrada.

5.3 Clases de Testeo

Se han realizado algunas clases de testeo para los métodos más significativos de las clases mas susceptibles de error. Estas clases nos permiten de forma automática detectar si ocurren errores en determinados puntos de la aplicación.

Mediante la clase `TestBarco` la cual hereda de la clase `Barco`, será comprobado su correcto funcionamiento. Para ello se crea un barco y se le asigna una posición. En el caso del ejemplo, se hará con un barco de 5 casillas `(portaviones)`. Se comprueba mediante un `assertFalse` que el `hashmap` tiene todos sus valores a `false` para este. Esta metodología se lleva a cabo dado que inicialmente se sabe que debe ser así. Se podría considerar una afirmación de algo que se sabe que debe ser así y de no serlo, confirmaría un error.

```
public class TestBarco extends TestCase{

    public void testMetodos(){
        Barco b=new Barco(5);
        int[] t=new int[5];
        t[0]=45;t[1]=46;t[2]=47;t[3]=48;t[4]=49;
        b.setPos(t);
        //compruebo que inicialmente el barco no esta undido (todos los valores del hashmap false)
        assertFalse(b.estaHundido());
        //simulo que una de las posiciones del barco ha sido tocado
        b.tocado(47);
        //compruebo el el valor de esa posicion en el hashmap está a true
        assertTrue((boolean)b.estado.get(47));
        //Simulo que el resto de posiciones del barco han sido tocadas
        b.tocado(45);
        b.tocado(46);
        b.tocado(48);
        b.tocado(49);
        //compruebo que el barco está hundido (todos los valores del hashmap true)
        assertTrue(b.estaHundido());

    }

    public static Test suite(){
        return new TestSuite(TestBarco.class);
    } // suite

    public static void main (String[] args){
        junit.textui.TestRunner.run(suite());
    } // main
```

Imagen 37

También se comprueba el método de tocar barco, asignando una casilla tocada y posteriormente verificando que el estado de ella pasa a ser **`true`**, mediante `asserttTrue`.

Además de estos métodos de testeo es obligatorio implementar dos métodos en toda clase de testeo de tipo `static` como se aprecia en la codificación (`suite` y `main`).

El resto de los testeos siguen el mismo principio de verificación. Así `TestEstadoBarcos` comprueba con la creación de dos barcos, si estos realmente se suman al estado de la *flota así* como el acceso a ellos por la clase `estadoBarcos.`

Se comprueba el correcto funcionamiento del método `cercarPerimetro,` comprobando para un barco cuyas casillas han sido asignadas si realmente el número de casillas en la lista de prohibidas es el correcto. Para ello se compara si 12 es realmente el número de casillas `prohibidas` que debería haber para un barco de tamaño 3, mediante `assertEquals`. También se comprueba si los números listados en `prohibidos` son realmente los que deberían de haber, comparándolos con los que sabemos que debe haber.

Se sigue este mismo proceso para las clases `EstadoBarcos` y `Utilities`.

6. CONCLUSIONES

Una de las principales conclusiones es la gran capacidad de Java como lenguaje, para poder funcionar en cualquier ordenador ya posea un sistema operativo de Windows, Linux, o cualquier otra marca conocida.

Esto se debe a que Java es un lenguaje independiente de la plataforma. Debido a esto, el programa funcionará correctamente, en cualquiera de ellas. Esto es posible ya que se ha creado una máquina de Java para cada sistema y esta hace de nexo entre el programa en Java y el sistema operativo, provocando que se comprendan.

Además Java es un lenguaje inmenso. Se puede hacer casi cualquier programa con él, no es tan grande como C++, pero es mucho mas robusto, para lo que se desea programar con este lenguaje, ya que limita algunas opciones, mientras que C++, es más abierto.

La forma de trabajar con herencias de otras clases, me ha parecido muy interesante, así como una herramienta muy potente aunque en ocasiones peligrosa, ya que se puede abusar de ella.

La creación del interfaz gráfico mediante un Jframe en NetBeans, me ha parecido algo bastante útil, ya que implementar el código para hacerlo me resulto algo tedioso en la primera aplicación.

La forma de crear la comunicación mediante hilos me pareció algo compleja. Por otra parte la forma de entablar una conexión en red es relativamente sencillo utilizando Java, ya que hay muchas ayudas de lenguaje.

7. LÍNEAS FUTURAS

Una posible linea de evolución de los programas desarrollados, sería la posibilidad de introducir en estos, una nueva clase IA, que contenga lo necesario para dotar a los programas de una inteligencia artificial.

Para desarrollar esta IA se debería marcar el estado inicial, desde el que se va a partir, así como el estado final, al que se desea llegar, de este modo se pueden plantear los algoritmos para llegar desde el primer estado al segundo.

Una opción posible, sería usar algoritmos genéticos [17]. Mediante estos algoritmos se puede simular una forma de reproducción de código, así como de su selección.

Para poder usar estos algoritmos se tienen que cumplir tres premisas, que en las aplicaciones desarrollas, se cumplen, que son:

1. El espacio de búsqueda debe estar delimitado en ciertos valores (casillas del tablero).
2. Debe poderse definir cuan buena o mala es una opción (por ejemplo si colocar una ficha en una casilla favorece a que el contrario gane el juego, aunque nos sea positivo a nosotros).
3. Las soluciones deben ser relativamente sencillas de programar.

Mediante la mezcla de cadenas de código estos códigos consiguen que este, esté en constante evolución. Por otra parte se desea tener el mejor resultado final, y para ello sería necesario un algoritmo o una serie de ellos para poder elegir entre las soluciones halladas, la mejor. También sería necesario introducir modificaciones cada cierto numero de cadenas de código, para favorecer la evolución del código. Una opción para favorecer la velocidad de esta evolución del código, sería poner más de un algoritmo de IA para que compitan entre si.

8. BIBLIOGRAFÍA

[1] “Java (lenguaje de programación).” 2013.*Wikipedia, la enciclopedia libre*. http://es.wikipedia.org/wiki/Java_(lenguaje_de_programaci%C3%B3n)

[2] Universidad de Burgos “Guía de Iniciación al Lenguaje JAVA” 1999 http://zarza.usal.es/~fgarcia/doc/tuto2/I_2.htm

[3] Centro de tecnología informática UNAV “Introducción Al Lenguaje JAVA”. http://www.unav.es/SI/manuales/Java/indice.html#1.1.

[4] Universidad de Burgos “Guía de Iniciación al Lenguaje JAVA” 1999 http://zarza.usal.es/~fgarcia/doc/tuto2/Index.htm

[5] “Project management for developers, marketers & agencies” 2006 - 2013, Assembla Inc. https://www.assembla.com/home

[6] “TortoiseSVN” 2004-2013 - THE TORTOISESVN TEAM http://tortoisesvn.net/downloads.html

[7] “TortoiseSVN” 2004-2013 - THE TORTOISESVN TEAM http://tortoisesvn.tigris.org/

[8] “Introducción al software libre” Jesús M. González Barahona http://curso-sobre.berlios.de/introsobre/2.0.1/sobre.html/eclipse.html

[9] Universidad de La Coruña – Departamento de Computación http://www.dc.fi.udc.es/ai/tp/practica/junit/ppasos.html

[10] “Introducción a JUnit” Universidad de Deusto http://www.slideshare.net/ikercanarias/introduccin-a-junit

[11] NetBeans – Oracle 2013
https://netbeans.org/kb/docs/java/quickstart.html

[12] “Descargas NetBeans” NetBeans – Oracle 2013
https://netbeans.org/downloads/

[13] Código de Programación, 2013
http://codigoprogramacion.com/cursos/java/133-uso-de-hilos-o-threads-en-java.html#.UtRtkvTuLrg

[14] “Hilos en Java” Blog Monillo007
http://monillo007.blogspot.com/2008/01/hilos-en-java-threads-parte-1.html

[15] Capítulo "Sockets en JAVA"
http://www.infor.uva.es/~fdiaz/sd/doc/java.net.pdf

[16] “Clases Java para comunicaciones en red” Escuela Técnica Superior Superior de Ingeniería Informática. Dpto. Sistemas Informáticos y Computación. Universidad de La Laguna, 2004-2005
http://nereida.deioc.ull.es/~cleon/doctorado/doc05/0405/pspdf/p2.pdf

[17] "Artificial Intelligence" Blog MasterEduca 2010
http://nando1-utb.blogspot.com.es/p/algoritmos-geneticos.html

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS INDUSTRIALES Y DE TELECOMUNICACIÓN

INGENIERO TÉCNICO DE TELECOMUNICACIÓN, ESPECIALIDAD EN SONIDO E IMAGEN

PROGRAMACIÓN JAVA

Xabier Martínez de Morentin
Jesús Villadangos Alonso
Pamplona, Enero 2014

INDICE

1. Introducción

1.1 Java

- Es un lenguaje de programación orientado a objetos.
- Entorno robusto.
- El API de Oracle ofrece una gran cantidad de clases.

1.2 POO

- Abstracción: Un objeto debe subdividirse en objetos de menor tamaño, para su creación.
- Encapsulación: Mediante los métodos públicos o privados el usuario solo podrá acceder a los servicios que el programador desea.
- Herencia: Eficiencia computacional.

2. Programas utilizados

2.1 Assembla

- Permite crear y gestionar un repositorio para el control de versiones.
- La base de datos, queda guardada en un servidor Apache, configurado con subversion.
- Subversion nos permitirá saber quien y que ha modificado un usuario ya que lo refleja en el historial.

  Además nos informara de posibles incompatibilidades en los cambios y nos permitirá retomar versiones anteriores.
- Hay tres carpetas principales: branches, tag y trunk.

2.2 Tortoise SVN

- Se puede gestionar el repositorio de Subversion haciendo uso del explorador de windows.
- Se usará como repositorio web, Assembla.
- El directorio creado para albergar la aplicación se podrá subir al repositorio actualizando así la versión de este y viceversa.
- Dos modos de funcionamiento: bloqueante y no bloqueante.

2. Programas utilizados

2.3 Eclipse y NetBeans

IDE Eclipse

- Ventajas
  - Ligero y rapido.
  - Es portable.
- Desventajas
  - Se necesitan diferentes versiones(visua, JEE...).

IDE NetBeans

- Ventajas
  - Con este IDE se puede trabajar múltiples aplicaciones como JEE, Android, Swing...
  - Trabajar Swing con la ventana de diseño es intuitivo.
- Desventajas
  - Pesado y lento, requiere instalación.
  - Código no manipulable al 100%

3. Aplicación Conecta4

Clase ControlJuego

```
if (state == State.THINKING) {
    if (!currentPlayer.isThinking()) {
        move = currentPlayer.getMove();
        ok = posicion.meterFicha(move);
        if (!ok) { currentPlayer.think(posicion); return; };    // Y no podemos hacer más... ya elegirá
        fila = posicion.getLastFila();
        controlGrafico.tirarFicha(move, fila, currentPlayer == player1);
        state = State.TIRAR;
    }
}

if (state == State.TIRAR) {
    controlGrafico.moveAndDraw();
    if (!controlGrafico.isEnMovimiento()) {
        if (posicion.getWinner() == Valor.PLAYER1) {
            winsPlayer1++;
            if (player2 instanceof Player);
        }
        if (posicion.getWinner() == Valor.PLAYER2) {
            winsPlayer2++;
            if (player1 instanceof Player);
        }
        if ((posicion.getWinner() != Valor.NONE) || (posicion.isDraw())) {
            System.out.println("PLAYER1    " + winsPlayer1 + " - " + winsPlay
            tick = 0;
            state = State.ENDGAME;
        } else {
            currentPlayer = (currentPlayer == player1 ? player2 : player1);
            currentPlayer.think(posicion);
            state = State.THINKING;
         }
    }
}

if (state == State.ENDGAME) {
    if (tick == 3 * Constantes.FPS) {
        controlGrafico.vaciarTablero();
        state = State.VACIAR;
    }
}

if (state == State.VACIAR) {
    controlGrafico.moveAndDraw();
    if (!controlGrafico.isEnMovimiento()) {
        controlGrafico.retirarFichas();
        frame.enableControls();
        state = State.CONTROLS;
```

```
public boolean isEnMovimiento() {
```

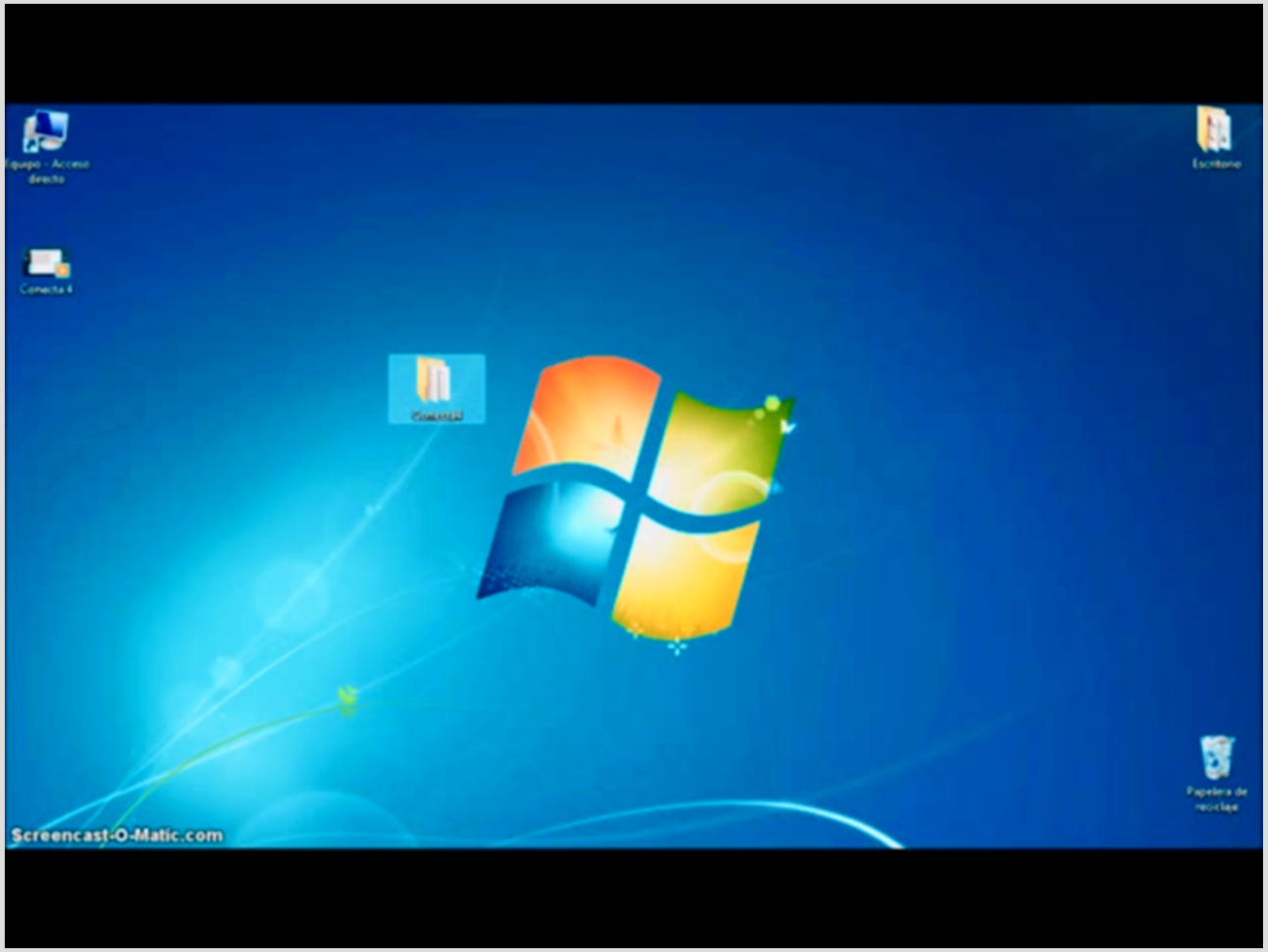
```
public Valor getWinner() {
    for (int f = 0; f < Constantes.FILAS_TABLERO; f++) {
        for (int c = 0; c < Constantes.COLUMNAS_TABLERO; c++) {
            if (hay4EnFila(f,c)) {
                return posicion.tablero[f][c];
            }
        }
    }
    return Valor.NONE;
}
```

```
private boolean hay4EnFila(int f, int c, Direccion dir) {
    int dx = dir.getDx();
    int dy = dir.getDy();
    Valor valor = posicion.tablero[f][c];
    // Si el primer valor es de "casilla vacia": no hay
    if (valor == Valor.NONE) { return false; };
    // Barrer
    for (int
        f +=
        // Si
        if (
        if (
        // Si
        if (p
    }
    // Si pas
    return tr
}
```

```
public boolean isDraw() {
    // Si no está completo, no hay tablas
    for (int c = 0; c < Constantes.COLUMNAS_TABLERO; c++) {
        if (posicion.tablero[0][c] == Valor.NONE) { return false; };
    }
    // Si hay ganador, aunque esté completo, tampoco son tablas
    if (getWinner() != Valor.NONE) { return false; };
    return true;
}
```

```
private boolean hay4EnFila(int f, int c) {
    return (hay4EnFila(f,c,Direccion.HOR) || hay4EnFila(f,c,Direccion.VER) ||
            hay4EnFila(f,c,Direccion.DIAG1) || hay4EnFila(f,c,Direccion.DIAG2));
}
```

3. Aplicación de Conecta4



4. Hundir la flota

Hilos y Clases de Testeo

- Hilos de comunicaciones en red
  - 2 clases: Cliente y Servidor
  - 2 flujos de información: ObjectOutputStream y ObjectInputStream
- Clases de Testeo
  - Permiten de forma automática detectar errores en los métodos de cada clase de la aplicación.

4. Hundir la flota

Hilos y Clases de Testeo

```
 */
package hundirlaflota;

import junit.framework.*;

public class TestBarco extends TestCase{

    public void testMetodos(){
        Barco b=new Barco(5);
        int[] t=new int[5];
        t[0]=45;t[1]=46;t[2]=47;t[3]=48;t[4]=49;
        b.setPos(t);
        //compruebo que inicialmente el barco no esta hundido (todos los valores del hashmap false)
        assertFalse(b.estaHundido());
        //simulo que una de las posiciones del barco ha sido tocado
        b.tocado(47);
        //compruebo el el valor de esa posicion en el hashmap está a true
        assertTrue((boolean)b.estado.get(47));
        //Simulo que el resto de posiciones del barco han sido tocadas
        b.tocado(45);
        b.tocado(46);
        b.tocado(48);
        b.tocado(49);
        //compruebo que el barco está hundido (todos los valores del hashmap true)
        assertTrue(b.estaHundido());

    }

    public static Test suite(){
        return new TestSuite(TestBarco.class);
    } // suite

    public static void main (String[] args){
        junit.textui.TestRunner.run(suite());
    } // main
}
```

5. Conclusiones

- Java es un lenguaje de programación orientado a objetos de una capacidad inmensa, que a su vez puede funcionar en cualquier sistema operativo gracias a la maquina virtual que hace de nexo entre Java y el sistema operativo.
- La ventana de diseño de NetBeans es una forma sencilla aunque limitada para desarrollar interfaces gráficos.
- La compilación del código en directo es una gran mejora con respecto a otros programas.

Agradecimientos

- Universidad Publica de Navarra

- Tutor de Proyecto Jesús Villadangos

- Familia y amigos