Quinta Práctica (P5)
Implementación de Interfaces de Usuario siguiendo el patrón Modelo-Vista-Controlador

Competencias específicas de la quinta práctica

- Aprender un modo de trabajo para el desarrollo de interfaces de usuario que mantiene un acoplamiento reducido entre las clases que modelan la aplicación y las que modelan la interfaz, como es el patrón de diseño Modelo-Vista-Controlador.
- Aprender a interpretar un diagrama de transición de estados para modelar el control del juego mediante la selección de las opciones disponibles al usuario (casos de uso) según el estado de la aplicación.

A) Programación y objetivos

Tiempo requerido: Dos sesiones, S1 y S2 (4 horas).

Comienzo: **semana del 26 de noviembre (grupos de prácticas de lunes a jueves) o el 7 de diciembre (grupos de prácticas de los viernes).**

Planificación y objetivos:

Sesión	**Semana**	**Objetivos**
S1	26-29 noviembre (grupos de lunes a jueves) ó 7 diciembre (grupos de los viernes)	• Interpretar una diagrama de transición de estados. • Interpretar e implementar en Java y Ruby el diagrama de clases según el patrón de diseño Modelo-Vista-Controlador (MVC).
S2	3-5 diciembre grupos de lunes a miércoles) ó 13-14 diciembre (grupos de juves y viernes)	• Revisar la implementación de Java y de Ruby realizando pruebas generales de la aplicación usando la interfaz.
La práctica se desarrollará tanto en Java como en Ruby en equipos de 2 componentes. El examen es individual.		

Nota: El **examen** de la quinta y última práctica será junto con el de la cuarta práctica, en la

semana del 17 al 21 de diciembre para todos los grupos.

Objetivos específicos:

1.	Aprender a usar diagramas de transición de estados para modelar el controlador en una interfaz de usuario.
2.	Aprender a usar el patrón de diseño modelo-vista-controlador (MVC).
3.	Saber implementar una interfaz de usuario de texto.

C) Descripción general de la práctica

En esta práctica vamos a realizar una interfaz de usuario para completar la aplicación del juego Qytetet. Para ello, vamos a usar un patrón de diseño llamado patrón Modelo-Vista-Controlador (MVC). En este patrón, todo el sistema se divide en tres partes:

- Modelo: es la parte funcional de la aplicación.
- Vista: es la parte que muestra el modelo al usuario. A veces, como en nuestro caso o en en las interfaces gráficas más comunes de Java, puede incluir también la interacción con el usuario y con el modelo.
- Controlador: es la parte que transforma las peticiones del usuario recogidas por la vista en mensajes al modelo y establece las opciones disponibles en la vista, según el estado de la aplicación. A menudo es por tanto un módulo intermedio entre el modelo y la vista pero a veces también puede interaccionar directamente la vista con el modelo.

El modelo MVC permite:

- Mantener un acoplamiento reducido entre las clases que modelan la aplicación y las que modelan la interfaz, de forma que la interfaz pueda modificarse sin tener que cambiar el modelo.
- Dentro de la interfaz de usuario, distinguir y mantener un acoplamiento reducido entre las tareas que permiten que el usuario y el sistema se puedan comunicar (vista) y las tareas que traducen esta comunicación en peticiones al modelo y respuestas del mismo (controlador).

En esta práctica hemos optado por una configuración del patrón MVC en la que la **Vista** no puede acceder directamente al modelo, tal y como aparece en la Figura 1. Por simplificar, se ha puesto un paquete para cada módulo del patrón MVC. Como puede observarse, en el caso de querer cambiar de interfaz, si el patrón MVC está bien implementado, bastaría con cambiar sólo las clases de la vista (por ejemplo usando otro paquete como **OtraVista** en la figura). También puede observarse que según la figura, el acoplamiento es más alto cuando se usa el paquete **OtraVista**, pues **OtraVista** puede acceder al modelo directamente.

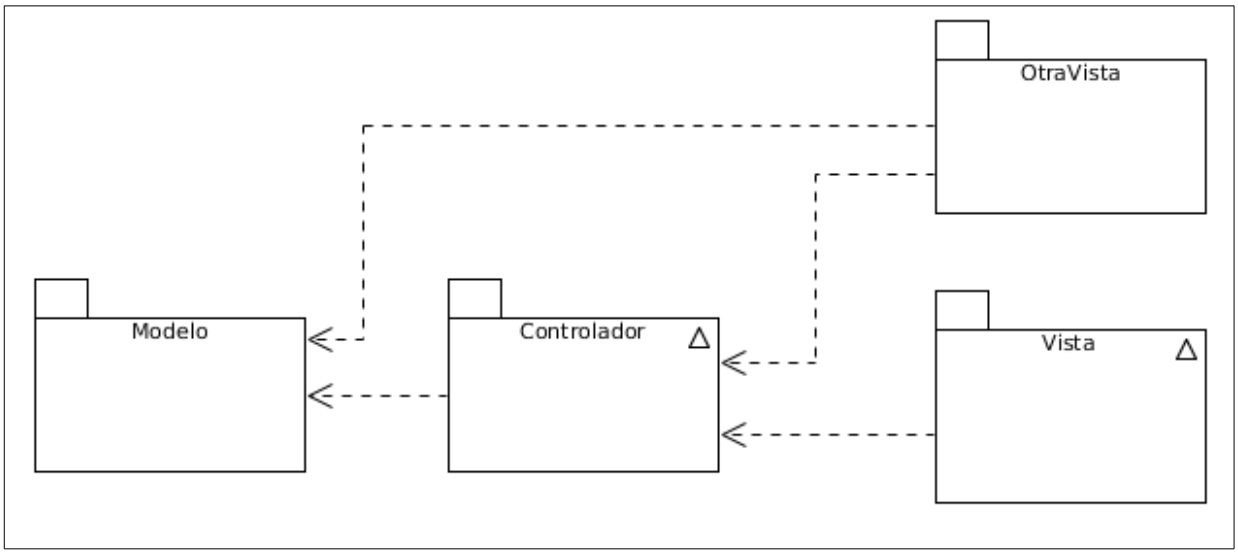

Figura 1: El patrón MVC con dos vistas alternativas.

Para implementar la interfaz de usuario es muy importante tener en cuenta el diagrama de transiciones del juego (fichero DiagramaTransicionesQytetet.pdf) y el diagrama de clases de la interfaz de usuario (fichero DCQytetetP5.pdf).

En nuestra implementación, el método principal (***main***), en la clase **VistaQytetet**, una vez que obtenga del usuario los nombres de los jugadores, entrará en un bucle infinito donde en cada iteración:
1) el usuario elegirá una operación a realizar (valores permitidos del enumerado **OpcionMenu**) y una casilla si es una operación inmobiliaria externa (gestión)
2) se llamará y mostrará al usuario el resultado (un String) del método ***realizarOperación*** del controlador para que, según la operación elegida por el usuario, realice lo siguiente:

- si es una operación relacionada con el juego: transmitir al modelo la petición correspondiente con dicha operación y devolver a la vista un mensaje para el usuario;
- si es una operación de mostrar algo al usuario: pedir al modelo el objeto serializado (método toString) y devolverlo a la vista; y
- si es la operación TERMINARJUEGO, finalizar el programa.

Los valores a mostrar al usuario (valores permitidos) del enumerado **OpcionMenu** serán devueltos por el método ***obtenerOperacionesJuegoValidas*** de la clase **ControladorQytetet** del controlador, según el diagrama de transición de estados suministrado.

Un diagrama de transición de estados es un diagrama en el que los nodos representan estados y los arcos, que deben ser dirigidos (flechas en un solo sentido), representan transiciones de un estado a otro. Este tipo de diagramas es muy útil para modelar la relación entre el estado de ejecución del modelo de la aplicación y las operaciones disponibles a usuario (vista) según dicho estado. En esta práctica este diagrama es el núcleo del método ***obtenerOperacionesJuegoValidas*** de la clase **ControladorQytetet** del controlador, de manera que el controlador permitirá a la vista saber cómo debe actualizarse (qué opciones debe mostrar al usuario) a partir de este diagrama.

Las casillas a mostrar al usuario también deben ser sólo las permitidas según la operación elegida por el usuario, y para ello se usará el método ***obtenerCasillasValidas*** del controlador.

NOTA: En el caso de haberse realizado ya una interfaz de texto siguiendo el guión opcional de la práctica 3 (fichero GuionP3UI.pdf) y el diagrama de la interfaz de usuario proporcionado entonces (fichero DCQytetetP3UI.pdf), se deben seguir los mismos pasos del guión que si no se tuviera hecha la implementación, salvo a la hora de codificar los métodos, que no habrá que implementarlos sino copiar el código desde la versión usada en la práctica 3, como se indicará en el lugar adecuado con una NOTA dentro de un marco como el actual.

D) Diseño de la estructura de clases en Java y Ruby

1) **Implementación del diagrama de clases de la interfaz** (fichero DCQytetetP5.pdf)**.**- Implementa por completo el diagrama de este fichero. Deberás así crear en Java los paquetes **controladorqytetet** y el **vistatextualqytetet**. En Ruby debes usar módulos. La clase principal (la clase con el método ***main***) se definirá en la clase **vistatextualqytetet**. Crea todas las clases de estos paquetes, con sus atributos sin olvidar los de referencia, las cabeceras de los métodos y da código al modificador básico ***setNombreJugadores*** de la clase **ControladorQytetet**.

2) **Definición de las opciones del menú** (clase enumerada **OpcionMenu** dentro del paquete **controladorqytetet**).- Define el enumerado con todos sus valores.

   En el caso de Ruby, las opciones del menú es más conveniente declararlas como una lista (Array) que como un módulo, así el contenido del fichero MenuOpcion podría ser:

```
module InterfazUsuarioQytetet
 OpcionMenu =   [:INICIARJUEGO,:JUGAR,:APLICARSORPRESA,
INTENTARSALIRCARCELPAGANDOLIBERTAD,:INTENTARSALIRCARCELTIRANDODADO,
:COMPRARTITULOPROPIEDAD,:HIPOTECARPROPIEDAD,:CANCELARHIPOTECA,
:EDIFICARCASA,:EDIFICARHOTEL,:VENDERPROPIEDAD,:PASARTURNO,
:OBTENERRANKING,:TERMINARJUEGO,:MOSTRARJUGADORACTUAL,
:MOSTRARJUGADORES,:MOSTRARTABLERO ]
end
```

   La razón de implementarlo como lista y no como constantes de un módulo es porque hace falta convertir a número cada opción del menú manteniendo el orden. El método ***constants*** aplicado a un módulo devuelve las constantes pero no en el orden de definición. Sin embargo, en una lista (Array), es muy fácil obtener la posición de un elemento con el método ***index***.

NOTA: Este enumerado ya fue creado en la sesión opcional de la práctica 3, si ya se hizo, bastará con copiarla en el paquete **controladorqytetet**.

E) Implementación de la vista en Java y Ruby

1) Definición del método ***main(String args[]): void***. A partir de ahora el método de clase ***main*** del proyecto que debe ejecutarse será el de la clase **VistaTextualQytetet** (se puede eliminar el fichero PruebaQytetet.java). El método llamará primero al método para obtener los nombres de

los jugadores y después entrará en un bucle infinito que:

1. pedirá al usuario elegir una operación,
2. si es una operación inmobiliaria sobre cualquier casilla (lo que llamaremos una gestión) deberá pedir luego la casilla sobre la que realizar dicha gestión,
3. ejecutará dicha operación

A continuación aparece un posible código para este método.

```
public static void main(String args[]) {

 VistaTextualQytetet ui = new VistaTextualQytetet();
 ui.controlador.setNombreJugadores(ui.obtenerNombreJugadores());
 int operacionElegida, casillaElegida = 0;
 boolean necesitaElegirCasilla;

 do {
  operacionElegida = ui.elegirOperacion();
  necesitaElegirCasilla = ui.controlador.necesitaElegirCasilla(operacionElegida);
  if (necesitaElegirCasilla)
   casillaElegida = ui.elegirCasilla(operacionElegida);
  if (!necesitaElegirCasilla || casillaElegida >= 0)
   System.out.println(ui.controlador.realizarOperacion(operacionElegida,
   casillaElegida));
   } while (1 == 1);
 }
```

2) Implementación del resto de los métodos:

- ***obtenerNombreJugadores.*** Debe copiarse del fichero PruebaQytetet.

- ***elegirOperacion***. Este método realiza las siguientes operaciones:
  1. Obtiener la lista de operaciones del juego permitidas llamando al método del controlador ***obtenerOperacionesJuegoValidas****.*
  2. Obtiene a partir de la lista anterior una lista con cada entero convertido a String y muestra al usuario cada valor (número y el nombre del enumerado correspondiente) para que el usuario pueda elegir introduciendo un número.
  3. Llama al método ***leerValorCorrecto***, con esa lista como argumento, devolviendo el resultado transformado a entero.

- ***elegirCasilla***. Este método realiza las siguientes operaciones:
  1. Obtiene la lista de casillas válidas llamando a ***obtenerCasillasValidas (ordinalOpcionMenu)***
  2. Si la lista devuelta está vacía, devuelve -1.
  3. En caso contrario:
     1. Obtiene a partir de la lista anterior una lista con cada entero convertido a String y muestra al usuario cada valor de la lista para que el usuario pueda elegir introduciendo un número.
     2. Llama al método ***leerValorCorrecto***, con esa lista como argumento, devolviendo el resultado transformado a entero.

- Para controlar que la información introducida por el usuario cada vez que se le pida algo no dispare una excepción por incompatibilidad de tipos (por ejemplo si se lee con ***nextInt***

para pedir un número y el usuario introduce un carácter no numérico), siempre se leerá con ***next*** o ***nextLine*** para guardar la información en un **String** y luego se comprobará que el valor elegido es correcto, usando el método:

- **leerValorCorrecto (valoresCorrectos : String[0..*]) : String**. Este método no es obligatorio pero puede ser usado por los métodos ***elegirOperacion***, ***elegirCasilla*** *y* ***obtenerNombreJugadores*** para garantizar que el usuario elige respectivamente una opción o casilla correcta de las ofrecidas. El método debe leer en un String lo que el usuario introduce por consola, comprobar si el valor pertenece a la lista de *valoresCorrectos* y dar un error si no pertenece. Se llevarán a cabo de forma iterativa estas acciones hasta que se introduzca un valor correcto que será devuelto por el método.

NOTA: Todos estos métodos ya se implementaron en la sesión opcional de la practica 3, en la clase InterfazUsuarioQytetet, se pueden copiar directamente en la clase VistaTextualQytetet. Téngase en cuenta que el tipo devuelto por el método elegirOperacion ha cambiado. Se pueden mantener el tipo antiguo copiando además el método ***obtenerOpcionMenu*** que no se define en esta práctica.

F) Implementación de los métodos del controlador en Java y Ruby

1) **Implementación del diagrama de transición de estados** (fichero diagramaTransicionesQytetet.pdf).- En el diagrama proporcionado se muestran flechas en dos sentidos, lo cual representa dos transiciones, una en cada sentido. La-s operación-es de cada sentido son las que aparecen en la punta de la flecha. Por ejemplo, entre los estados *JA_PREPARADO* y *JA_PUEDEGESTIONAR* hay dos transiciones: si se está en estado JA_PREPARADO, se puede *jugar* y llegar al estado *JA_PUEDEGESTIONAR*, mientras que si se está en estado *JA_PUEDEGESTIONAR*, se puede pasar el turno (***siguienteJugador***) y pasar al estado *JA_PREPARADO.* Se debe observar que una misma operación puede llevar a estados distintos, según lo que ocurra al realizar dicha operación. Por ejemplo, cuando se pasa el turno, el nuevo jugador actual podrá estar en estado *JA_PREPARADO*, si no está encarcelado o en estado *JA_ENCARCELADOCONOPCIONDELIBERTAD* si sí lo está. Otro ejemplo, un jugador en estado *JA_ENCARCELADOCONOPCIONDELIBERTAD* podrá conseguir salir de la cárcel (*intentarSalirCarcel*) y quedar listo para *jugar* (*JA_PREPARADO*) o pasar a estado *JA_ENCARCELADO*, un estado en el que ya no tiene opción de intentar salir de la cárcel y todo lo que puede hacer es pasar el turno.

- ***obtenerOperacionesJuegoValidas(): int[0..*]****.* Devuelve una lista de las operaciones permitidas según el estado del jugador actual y el diagrama de transiciones de estado. Para el caso de que aún no haya empezado el juego y no haya jugador actual, lo cual se cumple si la lista de jugadores en el modelo está vacía, habrá que asignar a la lista de operaciones válidas un único valor (*INICIARJUEGO*). No se debe olvidar incluir todas las opciones de visualización y la de terminar juego en la lista de operaciones que se ofrecerán al usuario, una vez que se haya iniciado el juego.

  En el caso de un enumerado:

- El método de Java ***ordinal()*** devuelve la posición de un valor enumerado dentro de la clase enumerado donde se define. Por ejemplo, para obtener el entero con la posición que ocupa el valor del enumerado *INICIARJUEGO* en el enumerado *OpcionMenu*, se escribirá:

```
OpcionMenu.INICIARJUEGO.ordinal()
```

- En Ruby, podemos usar ***index*** para obtener la posición que ocupa una opción del menú en la lista de enumerados. En Ruby se convierte un entero a string con el método to_s. Así, el equivalente Ruby al ejemplo Java anterior será:

```
OpcionMenu.index(:INICIARJUEGO)
```

NOTA: Este método ya se implementó en la sesión opcional de la practica 3, en la clase InterfazUsuarioQytetet, se puede copiar directamente en la clase ControladorQytetet. Téngase en cuenta que el tipo del valor devuelto ha cambiado en esta práctica. Se aconseja dejarlo como estaba.

2) **Traducción de las opciones del menú a envío de mensajes**.- Se realizar con el método que se explica a continuación.

- ***realizarOperacion(opcionElegida : int, casillaElegida : int) : String***. El enumerado *OpcionMenu*, contiene las operaciones a realizar sobre el juego (métodos del modelo, que se corresponden con las transiciones indicadas en el diagrama de transición de estados):

INICIARJUEGO,
JUGAR,
APLICARSORPRESA,
INTENTARSALIRCARCELPAGANDOLIBERTAD,
INTENTARSALIRCARCELTIRANDODADO,
COMPRARTITULOPROPIEDAD,
HIPOTECARPROPIEDAD,
CANCELARHIPOTECA,
EDIFICARCASA,
EDIFICARHOTEL,
VENDERPROPIEDAD,
PASARTURNO,
OBTENERRANKING

- En Java, en el caso de un enumerado, el método ***values()*** devuelve la lista de sus valores. Por ejemplo, para obtener el valor concreto de un estado del juego a partir del entero *estado* se puede escribir:

```
EstadoJuego estadoJuego=EstadoJuego.values()[estado];
```

- En el caso de *INICIARJUEGO*, debe llamar al método del modelo *inicializarJuego* pasando como argumento el atributo de instancia *nombreJugadores* con los nombres de los jugadores, que serán capturados desde la vista. Lo que hay que hacer en los demás casos se puede deducir de sus propios nombres.

- Se han puesto como dos opciones del menú las dos formas de intentar salir de la

cárcel para evitar pedir luego al usuario que elija el método para salir de la misma.

- El segundo argumento del método se usará sólo si se trata de una gestión (operación inmobiliaria externa o gestión). Por ejemplo, si se elige *CANCELARHIPOTECA*, se debe llamar al método del modelo poniendo ese valor como argumento: *cancelarHipoteca(casillaElegida).*

- El enumerado *OpcionMenu* contiene asimismo acciones informativas para el usuario o la posibilidad de terminar el juego en cualquier momento:

  TERMINARJUEGO,
  MOSTRARJUGADORACTUAL,
  MOSTRARJUGADORES,
  MOSTRARTABLERO

  en cuyo caso o bien termina el juego (caso de *TERMINARJUEGO*) o bien se llama al método toString del jugador actual (caso de *MOSTRARJUGADORACTUAL*), de jugadores (caso de *MOSTRARJUGADORES*) o de tablero (caso de *MOSTRARTABLERO*).

- Es importante devolver siempre en un String algún mensaje informativo en cada una de las opciones, por ejemplo:
  - para *APLICARSORPRESA*, se puede devolver el estado de la sorpresa antes de llamar al método *aplicarSorpresa,*
  - para JUGAR, se puede devolver el valor del dado y la casilla donde el jugador ha caído, una vez llamado al método *jugar*,
  - para una operación inmobiliaria que puede no llegar a buen término, por ejemplo por falta de saldo, se puede indicar después de llamarla que no se ha podido realizar y la razón de ello,para salir de la cárcel, en cualquier de sus dos modalidades, se puede indicar si no se ha conseguido.

NOTAS:

- Este método ya se implementó en la sesión opcional de la practica 3, en la clase InterfazUsuarioQytetet, se puede copiar directamente teniendo en cuenta que entonces no devolvía nada pero ahora debe devolver un String. Así, en vez de imprimiar directamente un texto informativo de lo que se hace en cada opción elegida por el usuario, se debe devolver dicho texto en un String, que luego será mostrado desde la vista.
- En el caso de haber tomado el método de la sesión opcional de la práctica 3, hay que eliminar la llamada al método para obtener los nombres de los jugadores.

3) **Implementación del resto de los métodos del controlador**

- ***necesitaElegirCasilla(valor : int):boolean***. Devuelve *true* sólo si el argumento es el ordinal de alguna de las *OpcionMenu* que permiten realizar operaciones inmobiliarias externas o gestiones (*HIPOTECARPROPIEDAD, CANCELARHIPOTECA, EDIFICARCASA, EDIFICARHOTEL o VENDERPROPIEDAD*).

- ***obtenerCasillaValidas(valor : int): int[0..*]***. Devuelve los *numeroCasilla* de las casillas sobre las que se pueda hacer la operación inmobiliaria externa indicada como argumento (debe tenerse en cuenta que el argumento hay que transformarlo a enumerado) Para ello, deberán usarse los métodos del *modelo* que devuelven las propiedades de *jugadorActual*,

todas o las que estén o no hipotecadas, según la operación inmobiliaria elegida (***obtenerPropiedadesJugador*** y ***obtenerPropiedadesJugadorSegunEstadoHipoteca***).

NOTA: Estos dos métodos ya se implementaron en la sesión opcional de la practica 3, en la clase InterfazUsuarioQytetet, se pueden copiar directamente en la clase ControladorQytetet.

4) Prueba del proyecto

- Revisa la implementación de Java y de Ruby, realizando pruebas generales de la aplicación usando la interfaz de usuario.