# Tercera Práctica (P3)
# Implementación completa del juego

## Parte II: Implementación de la interfaz de usuario (sesión 4)

## E) Implementación de la interfaz de usuario en Java

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 DCQytetetP3UI.pdf).

1) **Implementación del diagrama de clases de la interfaz** (fichero DCQytetetP3UI.pdf)**.**- Crea en Java el paquete *InterfazUsuarioQytetet* junto con una *Java main class* del mismo nombre, que será la nueva clase principal, y el enumerado *OpcionMenu*, siguiendo el diagrama de clases *DCQytetetP3UI*.

2) Preparación de algunos métodos básicos:

   - ***public ArrayList<String> obtenerNombreJugadores().***- Debe copiarse del fichero PruebaQytetet.

   - 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 *obtenerOpcionMenu, obtenerCasilla 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 los métodos de la interfaz de usuario utilizan como argumentos tipos de datos primitivos y *String* (o listas de ellos). Así, si se trata de un casilla, se usará su número de posición, y no el objeto casilla, si se trata de un estado del juego o de la opción elegida del menú de usuario se utilizará su número de orden en el enumerado y no el enumerado en sí.

  En el caso de un enumerado, el método *values()* devuelve la lista de sus valores y el método *ordinal()* devuelve la posición de un valor enumerado dentro de la clase enumerado donde se define. Por ejemplo, para obtener el valor concreto de un estado del

juego a partir del entero *estadoJugadorActual* se puede escribir:

```
EstadoJuego estadoJugador = EstadoJuego.values()[estadoJugadorActual];
```

y para obtener la posición que ocupa el valor de enumerado *INICIARJUEGO* en el enumerado *OpcionMenu*, se escribiría:

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

- Para convertir a *String* un entero, puede usarse el método *toString*, por ejemplo:

```
Integer.toString(OpcionMenu.INICIARJUEGO.ordinal())
```

3) **Implementación del diagrama de transición de estados** (fichero DiagramaTransicionesQytetet.pdf).- El diagrama de transición de estados se debe implementar con el método que se especifica a continuación. En un diagrama de transición de estados, los arcos deben ser dirigidos (flechas en un solo sentido). En el diagrama 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(): String[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 (recuérdese lo indicado anteriormente sobre el uso de los métodos de clases *enum, ordinal()* y *toString()* de la clase *Integer* para convertir estados de *OpcionJuego* en *String*.

**4) 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) void***. 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 el caso de *INICIARJUEGO*, debe llamarse al método de la interfaz, *obtenerNombreJugadores* y luego al método del modelo *inicializarJuego*. Los demás se pueden deducir de su propio nombre. 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. Es importante incluir antes o después de la llamada algún mensaje para el usuario, por ejemplo:

- para *APLICARSORPRESA*, se puede mostrar la sorpresa antes de llamar al método *aplicarSorpresa,*
- para JUGAR, se puede mostrar 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.

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*).

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

5) 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 *InterfazUsuarioQytetet* (se puede eliminar el fichero PruebaQytetet.java). El método consistirá 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[]) {

    InterfazUsuarioQytetet ui = new InterfazUsuarioQytetet();
    int operacionElegida, casillaElegida=0;
    boolean necesitaElegirCasilla;

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

6) Implementación del resto de los métodos

- ***obtenerOpcionMenu(menuOperaciones : String[1..*]): int***. Este método pide al usuario que elija una opción entre las contenidas en la lista de opciones válidas (el argumento) y devuelve el *ordinal* del enumerado *OptionMenu* que haya elegido el usuario, para ello deberá:
  1. Mostrar al usuario las opciones del menú que aparecen en el argumento.- Para ello se puede recorrer la lista de valores – obtenida con el método values()- de OpcionMenu y comprobar para cada valor si su traducción a *String* está en la lista del argumento del método
  2. Llamar al método *leerValorCorrecto(menuOperaciones) : String*
  3. Devolver el entero correspondiente al *String* devuelto en el paso 2.

- ***elegirOperacion(): int***. Obtiener la lista de operaciones del juego permitidas llamando al método obtenerOperacionesJuegoValidas() y usa la lista como argumento en la llamada del método *int obtenerOpcionMenu(ArrayList<String>)*, devolviendo el entero.

- ***elegirCasilla (ordinalOpcionMenu : int) : int***. 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, muestra el contenido de la lista y llama al método *leerValorCorrecto*, con esa lista como argumento, devolviendo el resultado transformado an entero.

- ***necesitaElegirCasilla(valor : int):boolean***. Devuelve *true* sólo si el argumento es el ordinal de alguna de las *OpcionMenu* que permiten realizar operaciones inmobiliarias externas (*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*).

7) Prueba y depura todo el proyecto Java.

# F) Implementación de la interfaz de usuario en Ruby

Haz lo mismo en Ruby, teniendo en cuenta las peculiaridades de Ruby. Te indicamos algunas de ellas:

- Para mostrar el contenido de una lista (Array) es mejor usar *join* que *to_s* para estar seguros de que se llamará a su vez al método *to_s* según la clase de los objetos de la lista. Por ejemplo, para mostrar la lista de jugadores, escribiremos:

```
puts @modelo.jugadores.join
```

- 
- Las opciones del menú, es más conveniente declararlas como una lista que como un módulo, así el contenido del fichero MenuOpcion debería ser:

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