Quinta Práctica (P5)

Implementación de una interfaz de usuario gráfica

Competencias específicas de la quinta práctica

- Conocer el entorno de desarrollo de Netbeans para implementar interfaces gráficas de usuario en Java.
- Aprender a aplicar el patrón Modelo-Vista-Controlador en el desarrollo de interfaces gráficas de usuario.

Objetivos específicos de la quinta práctica

- Familiarizarse con el desarrollo de interfaces gráficas de usuario en Java.
- Añadirle una interfaz gráfica de usuario a la aplicación Civitas en Java según el patrón de diseño modelo-vista-controlador.
- Observar cómo podemos cambiar la interfaz manteniendo el mismo controlador y modelo que en prácticas anteriores, de tal forma que no cambia la funcionalidad sino la interacción y presentación de la información.

Desarrollo de esta práctica

Se parte del modelo de prácticas anteriores: clases y enumerados dentro del paquete *civitas*. En esta práctica se sustituye el paquete ***juegoTexto*** por un nuevo paquete llamado ***GUI*** (Graphic User Interface) que albergará varias clases que darán soporte a la interfaz gráfica. Se hará una copia del enumerado ***Respuestas*** y de la clase ***Controlador*** de la práctica 3 y se colocarán también en el nuevo paquete. Se proporcionan dos clases nuevas para el paquete GUI, ***CapturaNombres*** para pedir los nombres de los jugadores, y un nuevo ***Dado*** con interfaz gráfica que hay que sustituir por el anterior dado.

En cuanto a las nuevas clases que darán soporte a la vista gráfica, las habrá de tres tipos: *JFrame (*ventana principal*)*, *JDialog* (ventanas secundarias, que dependen de una principal) y *JPanel* (contenedor de componentes gráficos agrupados, que puede colocarse en una ventana o en otro JPanel).

Vamos a ir describiendo cómo ir construyendo cada una de ellas.

1. CivitasView, comenzando.

Este es el nombre que le vamos a dar a la ventana principal de nuestro programa. En ella colocaremos información sobre el jugador actual y sus propiedades, la casilla actual, la siguiente operación a realizar, y los eventos que se van produciendo en el juego. Seguir los siguientes pasos para crearla y probarla:

1. Crear dentro del paquete GUI una nueva clase del tipo JFrame Form, llamada CivitasView. Por defecto, en la parte de la derecha se abre la pestaña Design/Diseño.

2. Seleccionar de la paleta de componentes un Label y colocarlo arriba de la ventana, a modo de título, modificando su contenido y nombre de variable. Entrar en propiedades (en menú desplegable con el botón derecho del ratón) y deshabilitar (*enabled*) para que no sea editable.
3. Ir a la pestaña Source/Fuente y añadir a *CivitasView* una variable de instancia ***juego*** de la clase *CivitasJuego.*
4. Crear también un método ***setCivitasJuego***, que recibe como argumento un objeto CivitasJuego y lo asigna a la variable *juego*. También, dentro de este método, poner visible la vista (*setVisible(true)*). Este método permite vincular la vista con el modelo, en nuestro caso solo para visualizar su estado. En cada clase vista asociada a una clase del modelo, se procederá igual.

2. Programa principal, probando lo creado.

Para probar la práctica, vamos a crear una clase principal con método ***main*** llamada ***TestP5***. Debemos indicar al proyecto que esta es la nueva clase principal (botón derecho sobre el nombre del proyecto y selección de Propiedades/Ejecutar). En este momento vamos a probar la captura de nombres de jugadores, el dado y la ventana anterior, que por ahora solo tiene un título. Para ello, método ***main*** debe hacer lo siguiente:

- Crear un objeto *CivitasView.*

- Crear una instancia del nuevo Dado usando ***createInstance***, el cual recibe como argumento el objeto vista anterior. Esto es así porque el ***Dado*** es un *JDialog* que depende de una ventana principal que es la vista creada. Pon el dado en modo *debug*. Asegúrate de que se cambia el paquete que usa el dado (es decir, cambia el paquete del controlador y Respuestas a GUI, y añade in import GUI en las clases que usen Dado en el paquete civitas. Borra el Dado de prácticas anteriores de civitas.

- Crear una instancia de *CapturaNombres*, usando dos argumentos, la vista, y el boolean true (porque la ventana es modal, es decir, la ventana que lanza esta ventana deja de atender eventos hasta que ésta se cierra).

- Crear un *ArrayList* de nombres, vacío y asignarle los nombres que se obtengan tras enviar el mensaje *getNombres* a la instancia de *CapturaNombres* anterior.

- Crear un objeto de *CivitasJuego*, que reciba como argumento de sus constructor los nombres anteriores.

- Crear una instancia de la clase Controlador, cuyo constructor reciba como argumento el objeto de *CivitasJuego* y el objeto de *CivitasView.*

- En la clase C*ontrolador* cambia las referencias a la clase *VistaTextual* por *CivitasView*, y de momento comenta el método *juega*.

- Enviar a la vista el mensaje *setCivitasJuego*, con el objeto *CivitasJuego* creado.

Una vez terminado el main, ejecuta el proyecto y comprueba que funciona. Debe salir una ventana para insertar los nombres de los jugadores, y al darle a comenzar debe aparecer la ventana principal, que de momento solo tiene un título.

3. JugadorPanel

Vamos a continuar ampliando la interfaz gráfica, creando ahora la clase ***JugadorPanel*** del tipo *JPanel*, como una agrupación de los atributos principales del *Jugador,* que se mostrarán en la interfaz gráfica, esto es, su nombre, saldo, si está encarcelado y si es un especulador. Después le añadiremos a su vez otro JPanel en su interior para colocar los datos de las propiedades que vaya comprando.

Ve cambiando las visibilidades de clases y métodos del módulo civitas como los vayas necesitando, así como importando las clases necesarias.

Los pasos a seguir para mostrar la información del jugador son los siguientes:

- Crear dentro del paquete GUI una nueva clase del tipo *JPanel Form*, llamada ***JugadorPanel***.

- Seleccionar de la paleta de componentes varios *label* para los nombres de los atributos y *textfield* para los valores de los atributos siguientes: nombre, saldo, si está encarcelado y si es especulador. Cambiar los textos de los *label* y deshabilitar todos los componentes (*enable*) para que no sean editables. Cambiar los nombres de las variables para que sean representativos.

- Ir a la pestaña de código fuente y añadir un atributo ***jugador*** de la clase *Jugador* y un método ***setJugador***, que recibe como argumento un objeto de la clase *Jugador* con el que se asocia. Este método debe hacer lo siguiente: asociar lo que recibe como argumento con el atributo *jugador,* dar valor (método *setText(String)*) a todas las variables *textfield* que corresponden con los atributos a mostrar. El valor *(String)* debe ser el valor del atributo correspondiente en la clase jugador. Acuérdate de añadir los import necesarios y asegúrate que Jugador tiene la visibilidad public, para que se pueda acceder desde el modulo GUI.

- Llamar al método repaint para asegurarse de que se actualizan todos los valores. En general, cuando solo se han modificado los atributos de un componente ya existente, solo hay que llamar a repaint; si se han añadido o eliminado componentes, además habría que llamar a revalidate.

- Incorporar este panel a la ventana principal *CivitasView para que* muestre los datos del jugador que acabamos de diseñar. Para ello:

1. Debes abrir la ventana *CivitasView,* con la pestaña de Diseño y una vez abierta, crear un panel en el sitio donde quieras incorporar la vista del jugador y con las dimensiones adecuadas. A este panel le puedes llamar ***contenedorVistaJugador***.
2. IMPORTANTE: A este panel hay que ponerle el *layout* de flujo. Se hace clic con el botón derecho en el panel y en el menú contextual que se abre se elige "*Set Layout*" y a continuación "*Flow Layout*".
3. Ahora, en el constructor de *CivitasView,* se escribe el código necesario para incluir en *contenedorVistaJugador,* una instancia de *JugadorPanel*, tal y como se indica a

continuación:

1. Añade un atributo a *CivitasView* de tipo *JugadorPanel* y que puede llamarse *jugadorPanel*.
2. En el constructor, después de la instrucción *initComponents*(), se incluye el código para inicializar *jugadorPanel* e incluirlo en *contenedorVistaJugador:*

   ```
   jugadorPanel = new JugadorPanel();
   contenedorVistaJugador.add (jugadorPanel);
   repaint();
   revalidate();
   ```

3. Crea el método ***actualizarVista*** en *CivitasView* y dentro de él, añade una línea de código para asociar el panel del jugador, indicando cuál es el jugador actual (con *setJugador*). Si no tienes implementado el método *getJugadorActual* en *CivitasJuego*, debes añadirlo con visibilidad pública, ya que lo vas a necesitar aquí.
4. Añade a TestP5 una línea en la que se invoque a *actualizarVista*.

Ejecuta de nuevo el proyecto para comprobar que se muestran los datos del jugador actual.

4. PropiedadPanel

Hasta ahora solo mostramos los datos básicos del jugador actual. Vamos a indicar ahora cómo mostrar sus propiedades.

Primero vamos a añadir un panel **propiedades** a *JugadorPanel*, que visualizará tantas propiedades como tenga el jugador. Para ello seleccionamos Panel de la paleta de componentes y lo colocamos donde veamos conveniente, dando un tamaño grande. A continuación seleccionaremos el panel y pulsando el botón derecho elegiremos un layout FlowLayout para este panel. Esto permitirá distribuir en el hueco del panel todas las propiedades que se crean, y pondrá una barra de desplazamiento si es necesario.

A continuación crearemos una nueva clase llamada ***PropiedadPanel*** de tipo *JPanel Form* para mostrar los datos básicos de una propiedad. Más adelante haremos que el panel *propiedades* pueda incluir en su interior muchas instancias de *PropiedadPanel* y mostrarlas todas.

1. Una vez en el diseño de *PropiedadPanel*, seleccionaremos de la paleta de componentes varios *label* para los nombres de los atributos y *textfield* para al menos los valores de los atributos siguientes: nombre del título de propiedad, número de casas y hoteles y si está hipotecada. Al igual que en clases de la vista anteriores, cambiar los textos de los *label* y deshabilitar todos los componentes (*enable*) para que no sean editables. Cambiar los nombres de las variables para que sean representativos.
2. Ir a la pestaña de código fuente de *PropiedadPanel* y añadir un atributo ***tituloPropiedad*** de la clase *TituloPropiedad* y un método ***setPropiedad***, que recibe como argumento un objeto de la clase *TituloPropiedad* con el que se asocia. Este método debe hacer lo siguiente: asociar lo que recibe como argumento con el atributo *propiedad*, dar valor (método

setText(String)) a todas las variables *textfield* que corresponden con los atributos a mostrar.

3. Ir a la clase *JugadorPanel* y añadir el método ***rellenaPropiedades*** para que complete el panel propiedades con todas las propiedades del jugador. La implementación del método es la siguiente:

```
private void rellenaPropiedades (ArrayList<TituloPropiedad> lista) {
        // Se elimina la información antigua
        propiedades.removeAll();
        // Se recorre la lista de propiedades para ir creando sus vistas individuales y
añadirlas al panel
        for (TituloPropiedad t : lista) {
           PropiedadPanel vistaPropiedad = new PropiedadPanel();
           vistaPropiedad.setPropiedad(t);

           propiedades.add(vistaPropiedad);
           vistaPropiedad.setVisible(true);
        }
        // Se fuerza la actualización visual del panel propiedades y del panel del jugador
         repaint();
        revalidate();
}
```

4. Se añade una llamada a este nuevo método, *rellenaPropiedades* al final del método *setJugador* de *JugadorPanel*, enviando como argumento las propiedades del jugador actual.

5. Completando CivitasView

Vamos ahora a añadir a la ventana principal información sobre la casilla actual ,la siguiente operación del juego y el ranking.

Información sobre la Casilla actual:

Tenemos tres posibilidades. La primera es simple, solo añadir a CivitasView un componente textfield y dentro del método *actualizarVista* asociar a ese componente (setText) el *toString*() de la casilla para que se muestre. La segunda opción es más completa y consiste en crear un JPanel para Casilla, igual que hicimos para Jugador, mostrando de forma separada y organizada cada atributo de la casilla. Otra opción más completa y compleja es mostrar en todo momento, de forma gráfica, todo el tablero y resaltar cuál es la casilla en la que está el jugador actual.

Información sobre la Siguiente operación:

Se añadiría a la vista un componente label con su textfield para mostrar el nombre de la siguiente operación del juego.

Se añadirá a CivitasView el método *mostrarSiguienteOperacion* que actualice el textfield de la siguiente operación con la operación que recibe como argumento, y que después invoque a *actualizarVista* para que se actualicen todos los componentes de la ventana principal.

En las siguientes subsecciones se describe cómo crear otras ventanas que serán abiertas desde esta ventana principal.

Ranking

Para el ranking debes crear un label y un textArea en CivitasView. Dentro del método *actualizarVista* debes poner ambos componentes como no visibles inicialmente. Después, y también dentro de este método, debes comprobar si estamos al final del juego y si es así, se pondrán visibles los dos componentes, asignando a textArea un string con toda la información relativa al ranking de jugadores. Finalmente deberás hacer repaint y revalidate.

6. DiarioDialog

Se creará ahora una nueva clase de tipo *JDialog Form* llamada *DiarioDialog* para que muestre en una nueva ventana los eventos que van ocurriendo. En la pestaña de diseño se añadirán a la interfaz un *label* con el texto "Eventos:", un *textArea* en el que quepan cinco o seis líneas, y un botón OK. Cambiaremos los nombres de las variables para que sean significativos.

En la pestaña de código fuente añadiremos a la clase un atributo de instancia que sea una instancia del diario con la que se trabaja en el juego y cuya información se muestra en esta ventana.

El constructor de esta clase y de las siguientes que sean del tipo *JDialog* tendrá un único parámetro *parent* que será el *JFrame* del que depende (será la vista principal cuando se invoque).

En el constructor contendrá el siguiente código:

- *super(parent, true);* : invoca al constructor de la clase *JFrame* de la que depende y establece el modal a true, es decir, hasta que no se cierre esta ventana no se devuelve el control a la ventana vista principal.
- initComponents(); : invoca un método que se crea por defecto y que inicializa las variables asociadas a los componentes creados en la pestaña de diseño.
- setLocationRelativeTo(null);: establece el centro de la pantalla como el lugar para colocar la ventana

El constructor, además invocará al método ***mostrarEventos*** que se describe a continuación y que hay que añadir como nuevo.

El método ***mostrarEventos*** es similar al método *mostrarEventos* que hayas implementado para la interfaz textual, generando un texto con todos los eventos pendientes, salvo que el texto debe mostrarse aquí en el componente textArea creado en el diseño (por ejemplo, *eventos*.***setText***(texto), donde *eventos* es el nombre de la variable textArea). Después tienes que poner esta ventana visible e invocar a los métodos *repaint* y *revalidate* para asegurarte de que se actualice su contenido cada vez que se abra. Comprueba al principio de este método que solo se muestren eventos y se abra esta ventana si hay eventos pendientes.

Otra cosa que debe hacerse es dar funcionalidad al **botón *OK*** creado en la pestaña de diseño. Para ello, nos situaremos en esa pestaña y pulsaremos dos veces sobre el botón. Esta acción nos lleva al código fuente, donde crea automáticamente un método con el nombre de la variable asociada al

botón, seguido de "*ActionPerformed*". En este caso solo queremos que se cierre esta ventana de diálogo, por lo que escribiremos en su interior *this.dispose().*

Se debe poner comentado el método *main* de esta clase, ya que no lo vamos a utilizar.

Para usar esta ventana, debemos añadir un método mostrarEventos() a la clase CivitasView:

```
void mostrarEventos() {
   DiarioDialog diarioD= new DiarioDialog(this); //crea la ventana del diario

 }
```

7. Comprar

Debemos añadir un método ***comprar***() a la clase *CivitasView* que muestre una ventana en la que nos pregunte si queremos comprar y devuelva el enumerado correspondiente a la respuesta que hayamos dado.

Para mostrar esa nueva ventana podemos hacer uso de la clase *JOptionPane*. Esta clase tiene varios métodos que hacen una pregunta y muestran iconos, botones o listas para seleccionar alternativas de opciones de respuesta prefijadas o que se den como parámetro.

En este caso invocaremos al método de clase *showConfirmDialog* con opciones prefijadas para sí y no, tal y como se indica a continuación:

```
int opcion= JOptionPane.showConfirmDialog(null, "¿Quieres comprar la calle actual?",
"Compra", JOptionPane.YES_NO_OPTION);
```

A continuación, solo debemos comprobar que si opcion tiene valor 0, será porque hemos elegido la SI, y si tiene valor 1, será que hemos elegido NO. Devolveremos un enumerado de Respuestas en función de esa opción.

Es el momento de hacer otra prueba de lo realizado hasta el momento. Para ello, te sugerimos que vayas al controlador y pongas como comentado el código correspondiente a la funcionalidad asociada a GESTIONAR y a SALIR_CARCEL, y que fuerces el fin del programa después de hacer una compra. En la clase TestP5, añade al final una última línea de código para enviar el mensaje ***juega*** al controlador, eliminando la que invocaba a *actualizarVista*.

8. GestionarDialog

Esta ventana también es un *JDialog* y su misión es mostrar la lista de las gestiones inmobiliarias que se pueden realizar en el juego, y los nombres de las propiedades del jugador actual, de tal forma que se pueda elegir qué se desea hacer.

En la pestaña de diseño añadiremos un *label* para el título general, otro para las gestiones y otro para las propiedades. Bajo el *label* de las gestiones y el *label* de las propiedades pondremos sendos componentes *Jlist (el de gestiones lo llamaremos listaGestiones)*, y por último un botón que llamaremos "*realizar*".

En los Jlist recogeremos el item de la lista seleccionado a través de un evento del ratón. Para ello en el modo diseño y sobre el menú desplegable de la lista (botón derecho ratón), seleccionaremos events → mouse → mouseClicked. En el método creado recogeremos sobre el atributo correspondiente el indice del *item* de la lista seleccionado (*getSelectedIndex*()).
Ejemplo:

```
private void listaGestionesMouseClicked(java.awt.event.MouseEvent evt) {
  propiedadElegida= listaGestiones.getSelectedIndex(); }
```

En la pestaña de código añadiremos a la clase dos atributos de instancia del tipo int para guardar el índice de la gestión inmobiliaria (*gestionElegida*) y la propiedad (*propiedadElegida*) seleccionadas. Añadiremos también sus consultores (*getGestion*() y *getPropiedad*() en *GestionarDialog* y en *CivitasView*). Su constructor es igual que el de los *dialog* anteriores, y además en este caso se da un valor inicial -1 a los atributos de instancia.

Crearemos un método ***gestionar*** que será invocado por la vista principal, con el código siguiente:

```
public void gestionar(Jugador jugador){
    setGestiones();

    setPropiedades(jugador);

    repaint();

}
```

El método setGestiones se encargará de rellenar la lista de gestiones inmobiliarias y el setPropiedades, de rellenar la lista de propiedades del jugador actual, que se le pasará como parámetro.

Te facilitamos la implementación de setGestiones, completa tú la de setPropiedades.

```
public void setGestiones(){
    DefaultListModel gestiones = new DefaultListModel<>(); // datos para la lista

    ArrayList<String> opciones = new ArrayList<> (Arrays.asList(

       "Vender","Hipotecar","Cancelar hipoteca",

       "Construir casa","Construir hotel",

       "Terminar"));

    for (String s: opciones){

       gestiones.addElement(s);}  //se completan los datos

    listaGestiones.setModel(gestiones); //se le dice a la lista cuáles son esos datos

  }
```

Ya solo nos queda dar funcionalidad al botón *realizar.* Este botón debe tomar el índice de las lista para la gestión y propiedad seleccionadas, actualizando los atributos de instancia (de la misma forma que se ha hecho con el evento de ratón de seleccionar item), y luego cerrar la ventana

(*dispose*()) (esto último, siempre que haya algo seleccionado en la lista de gestiones antes de pulsar el botón).

Se debe poner comentado el método *main* de esta clase GestionarDialog, ya que no lo vamos a utilizar.

Por último, se debe añadir a CivitasView un atributo de instancia ***gestionarD*** que sea una ventana de la clase *GestionarDialog*. También añadiremos un método *gestionar,* que envíe a ese atributo el mensaje *gestionar* pasándole como parámetro el jugador actual y lo haga visible (haciendo antes *pack*, *repaint* y *revalidate* para actualizar su contenido). Todos estos métodos se hacen sobre el atributo *gestionarD*.

9. Salir de la Cárcel

Queda porque el juego nos pregunte el modo en el que queremos salir de la cárcel. Para ello, también abriremos una ventana del tipo *JOptionPane*, pero en esta ocasión invocando al método de clase ***showOptionDialog***, que permite personalizar los botones con opciones alternativas que se muestran.

Para hacer esa personalización necesitamos definir una colección de Strings con los textos de los botones de las opciones y usar esa colección como argumento del método, tal y como se indica en el ejemplo:

```
String[] opciones= {“opcion A”, “opcion B”};

int respuesta= JOptionPane.showOptionDialog(null, "¿Cómo quieres salir de la cárcel?",
"Salir de la cárcel", JOptionPane.DEFAULT_OPTION,
JOptionPane.QUESTION_MESSAGE,null, opciones, opciones[0] );
```

Después se devolverá el enumerado que corresponda a SalidasCarcel, dependiendo de la opción seleccionada.