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 (MVC) 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 MVC.
- 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.

Ejercicios

Estos ejercicios se realizarán utilizando como base un ejemplo de programa proporcionado con interfaz gráfica y que además sigue el patrón de diseño MVC (archivo TutorialP5.zip). El tamaño y complejidad del modelo se ha reducido lo máximo posible para permitir al alumno centrarse en el patrón de diseño y en el uso de interfaces gráficas en Java. Para poder hacer estos ejercicios, debes tener instalada la versión Netbeans 10 o superior. En esta implementación del patrón MVC, la vista tiene acceso tanto al controlador como al modelo, pero con el segundo solo tiene una dependencia débil (es decir, el modelo no es un atributo de referencia, sino que se le pasa a cada método que necesita acceder a él).

Ejercicio 1

Después de estudiar el ejemplo proporcionado, modifica la vista usada para mostrar los elementos que aparecen al pulsar el botón para añadir. El número deberá aparecer en un campo de texto en vez de en una etiqueta, y no debe ser editable (propiedad editable desactivada). Elimina el borde del campo de texto y modifica también el grosor del borde del panel que contiene este campo de texto.

Ejercicio 2

Añade al programa una nueva funcionalidad. Ahora el modelo tendrá otra colección adicional de elementos del mismo tipo que los ya existentes que deben ser mostrados de la misma forma pero en un panel independiente. A esta segunda colección solo se podrán añadir elementos y nunca quitar.

Modifica el modelo, el controlador y la vista para recoger esta modificación. Cambia la disposición de elementos (layout) del panel de la nueva colección para que sea igual que la del panel dado (FlowLayout).

Desarrollo de esta práctica (Civitas)

Se parte del modelo de prácticas anteriores. En esta práctica se sustituye el paquete ***vistaTextualCivitas*** por un nuevo paquete llamado ***GUI*** (Graphic User Interface) que albergará varias clases que darán soporte a la interfaz gráfica. La interfaz Vista que se usó en la P3, ahora estará dentro del paquete GUI. Se proporcionan dos clases nuevas para el paquete GUI, ***CapturaNombres*** para pedir los nombres de los jugadores, y una interfaz gráfica para la clase Dado, vistaDado. Si la clase Dado y/o sus métodos *tirar* y sus consultores y modificadores básicos, tienen visibilidad de paquete, deberás cambiarlos a públicos para que sean accesibles desde la vista. Por otro lado, deberás crear el consultor público para devolver el valor del atributo *debug*.

En cuanto a las nuevas clases que darán soporte a la vista gráfica, las habrá de tres tipos: *JFrame (*ventana principal*), JDialog*[1] (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 clase para definir la ventana principal de nuestro programa. Debe implementar la interfaz Vista. 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. Seguid los siguientes pasos para crearla y probarla:

1. Crear dentro del paquete GUI una nueva clase del tipo JFrame Form, llamada CivitasView y hacer que implemente la interfaz Vista[2]. 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.
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, hay que 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, como se irá indicando a lo largo del guión.
5. Borrar el método main.

1. *La clase VistaDado que se te proporciona es de tipo JDialog.*
2. Por ahora, puede lanzarse una excepción o poner métodos vacíos en los métodos exigidos por la interfaz.

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 y la ventana anterior, que por ahora solo tiene un título. Para ello, el método ***main*** debe hacer lo siguiente:

- Crear un objeto *CivitasView*.

- 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 y el modo debug (booleano) a usar.

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

- 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 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 y si es especulador. Cambiar los textos de los *label* y deshabilitar todos los componentes TextField (propiedad *editable*) 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. 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*".

- 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, arrastrar la clase JugadorPanel de la ventana del proyecto sobre la ventana de Diseño de la clase CivitasView, de forma que se añadirá un atributo de tipo JugadorPanel.

1. Crea el método ***actualiza*** en *CivitasView* y dentro de él, añade una línea de código para asociar el panel del jugador al 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.
2. Da código al método *pausa* de CivitasView para esperar hasta que el usuario pulse una tecla o para terminar el programa:

```
public void pausa(){

      int val= JOptionPane.showConfirmDialog(null, "¿Continuar?",
"Siguiente paso", JOptionPane.YES_NO_OPTION);

      if (val==1) System.exit(0);

   }
```

3. Añade a TestP5 una línea en la que se pida al controlador que empiece el juego (método *juega*).

Ejecuta de nuevo el proyecto para comprobar que se muestran correctamente los datos del jugador actual y sigue con el guión.

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 **propiedadesPanel** 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 número de hoteles. Al igual que en clases de la vista anteriores, cambiar los textos de los *label* y deshabilitar todos los TextField (*editable*) 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 *CasillaCalle* y un método ***setPropiedad***, que recibe como argumento un objeto de la clase *CasillaCalle* 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<CasillaCalle> lista) {
        // Se elimina la información antigua
        propiedadesPanel.removeAll();
        // Se recorre la lista de propiedades para ir creando sus vistas
individuales y añadirlas al panel
        for (CasillaCalle t : lista) {
            PropiedadPanel vistaPropiedad = new PropiedadPanel();
            vistaPropiedad.setPropiedad(t);

            propiedadesPanel.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* casi al final del método *setJugador* de *JugadorPanel* (antes de repintar y revalidar), 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 (no editable) y su correspondiente label, y dentro del método *actualiza* 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 del juego:

Se añadiría a la vista un componente label con su textfield (no editable) 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. Además, si la operación actual del juego es AVANZAR, se mostrará la vista (singleton) del dado (llamando al método *createInstance* de VistaDado[3]) . Al final hay que actualizar todos los componentes de la ventana principal (repaint).

Ranking

Para el ranking debes crear un label y un textArea (no editable) en CivitasView. Dentro del método *actualiza* 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.

3. A pesar de su nombre, este método, solo crea una instancia la primera vez que es invocado.

En las siguientes subsecciones se describe cómo crear otras ventanas que serán abiertas desde la ventana principal, al igual que se hace con la ventana (JDialog) que muestra el dado y cuyo código se proporciona (VistaDado).

Puedes ejecutar de nuevo el proyecto para comprobar que se muestra correctamente el valor del dado y que puedes cambiar su modo de uso.

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 generará 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.

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 dar código también al método *mostrarEventos()* de la clase CivitasView, de modo que cree el diálogo solo si hay eventos pendientes:

```
void mostrarEventos() {
 if (!Diario.getInstance().getEventos().isEmpty())
    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= 1-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 NO, y si tiene valor 1, será que hemos elegido SI. 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 comentes la parte del código del método *juega* correspondiente a la funcionalidad asociada a GESTIONAR, forzando el fin del programa después de hacer una compra.

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, de tal forma que se pueda elegir qué se desea hacer.

En la pestaña de diseño añadiremos un *label* para para las gestiones. Bajo el *label* pondremos un componente *Jlist (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) {
       gestionElegida= listaGestiones.getSelectedIndex();
       dispose();
}
```

En la pestaña de código añadiremos a la clase un atributo de instancia del tipo int para guardar el índice de la gestión inmobiliaria (*gestionElegida*). . Añadiremos también su consultor (*getGestion*() . Su constructor es igual que el de los *JDialog* anteriores, y además en este caso se da un valor inicial -1 a los atributos de instancia.

El constructor se encargará además de rellenar la lista de gestiones inmobiliarias, te facilitamos el código para hacer eso::

```
        DefaultListModel gestiones = new DefaultListModel<>(); // datos
para la lista
        ArrayList<String> opciones = new ArrayList<> (Arrays.asList(
            "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
```

Haz visible el JDialog. Se debe borrar el método *main* de esta clase GestionarDialog, ya que no lo vamos a utilizar. Por último, se debe dar código al método *elegirOperacion* de CivitasView para que cree una instancia de este JDialog y devolver el valor de la OperacionInmobiliaria elegida.

9. PropiedadDialog

Debes hacer algo parecido a lo realizado en el paso anterior para poder obtener la propiedad sobre la que realizar la gestión inmobiliaria. El constructor del JDialog que hay que crear, deberá tener un argumento más con el jugador actual.