Quinta Práctica
Implementación de una Interfaz Gráfica de Usuario básica siguiendo el patrón Modelo-Vista-Controlador

Competencias específicas de la quinta práctica

- Conseguir una visión inicial de un entorno de desarrollo de interfaces gráficas de usuario en Java con Netbeans.
- Aprender un modo de trabajo para el desarrollo de interfaces gráficas de usuario que mantiene un acoplamiento reducido entre las clases que modelan la aplicación y las que modelan la interfaz.

A) Programación y objetivos

Tiempo requerido: Una sesión. Esta práctica se realizará exclusivamente en Java.

Planificación y objetivos:

Temporización:

- Semana del 18 al 22 de diciembre.

Objetivos:

- Familiarizarse con el desarrollo de interfaces gráficas de usuario en Java.
- Añadirle una interfaz gráfica de usuario en Java a la aplicación Qytetet según el patrón de diseño modelo-vista-controlador. En Java se suele simplificar la aplicación de este patrón uniendo el controlador con la vista, como es el caso de esta propuesta.

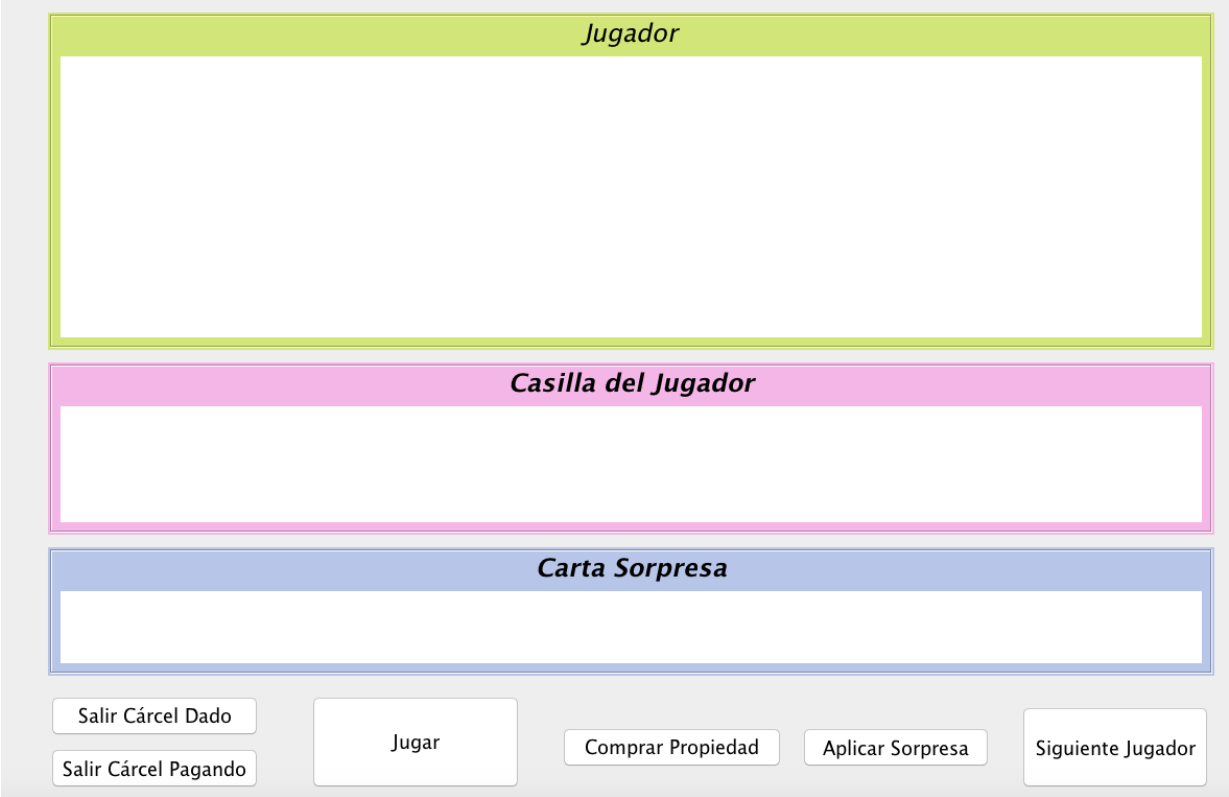
Nota: El examen de la cuarta (Java y Ruby) y quinta práctica (Java) será el día 22 de enero. Ese mismo día se entregarán los proyectos Java (quinta práctica implementada sobre la cuarta) y Ruby (cuarta práctica).

B) Descripción general de la práctica

Esta práctica indica lo mínimo que se pide para la entrega. Si lo deseas, puedes añadirle funcionalidad y mejorar la interfaz.

Importante: Se ruega no limitarse a seguir los pasos. Se debe entender lo que se está haciendo y preguntar todo lo que no se entienda.

- Se creará un conjunto de clases para añadirle una Interfaz Gráfica de Usuario (Graphics User Interface, GUI) al juego Qytetet siguiendo el patrón modelo-vista-controlador.
- Las clases del modelo ya se realizaron en prácticas anteriores. Las clases de la vista y del controlador se pondrán juntas en un paquete denominado **GUIQytetet**.
- Visualmente, la interfaz va a consistir en una ventana (JFrame) con botones para controlar el juego y distintos paneles o zonas de interés (JPanels) que mostrarán el estado del juego. Se propone hacer una versión simplificada en la que únicamente se muestre en los paneles el estado del jugador actual, la casilla actual y la carta sorpresa y haya botones únicamente para jugar, comprar propiedad, aplicar sorpresa, pasar al siguiente jugador e intentar salir de la cárcel. La figura siguiente muestra un ejemplo.



- La propuesta que hacemos en esta práctica está condicionada por la escasez de tiempo dedicado a la misma (una sesión). En ese sentido, la vista de los objetos en sí es todavía textual (el jugador, la casilla y la sorpresa se muestran simplemente como una cadena de texto). La modificación a una vista verdaderamente gráfica y la implementación del resto de la funcionalidad del juego a través de nuevos botones se propone sólo de forma voluntaria.
- No todos los botones estarán activos siempre, sino que dependerá de las acciones que el jugador pueda realizar en cada momento. La activación y desactivación de los botones la deberás programar atendiendo a las normas del juego, también puedes basarte en el diagrama de transición de estados de la práctica 3.

Enlaces interesantes

- Si no se estáis familiarizado/a con el diseño y creación de interfaces gráficas de usuario de manera interactiva con Netbeans, te recomendamos leer el siguiente enlace:
- https://netbeans.org/kb/docs/java/quickstart-gui.html

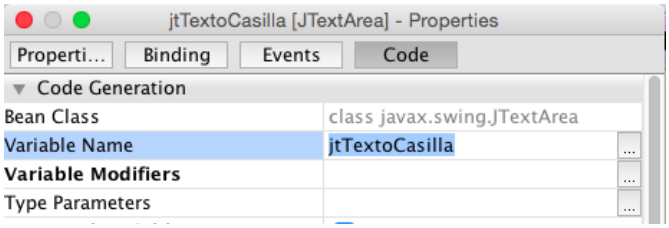
C) Implementación de la vista

- Cuando se crea en Netbeans una clase dentro de la categoría “Swing GUI Form”, se crea, además de un fichero .java con el código (“source”), otro fichero (.form) que no hay que editar y permitirá trabajar con un asistente gráfico para simplificar el diseño de las vistas. Las dos pestañas que aparecerán a la izquierda, justo encima de la ventana de edición, servirán para cambiar de un modo de trabajo a otro.
- En primer lugar vamos a crear las vistas más particulares (VistaCartasSorpresa, VistaCasilla, VistaJugador) y después el elemento que las aglutina (VistaQytetet).

- Pulsa con el botón derecho del ratón sobre el paquete GUIQytetet y crea un nuevo JPanel llamado VistaCasilla. Se habrá creado una subclase de JPanel llamada VistaCasilla. Pulsando en la pestaña “Source”, verás que ya se ha generado código correspondiente a dicha clase. Cuando se añadan elementos a través de la pestaña “Design”, se generará el código correspondiente que podremos ver pulsando en “Source”.
- Para empezar, selecciona la pestaña “Design” y añade un área de texto (JTextArea) donde durante el juego aparecerá el estado de la casilla actual. Para hacerlo, puedes arrastrarlo y soltarlo en el panel.

  Para cambiar las propiedades tanto del panel como del área de texto, basta con seleccionarlo, pulsar con el botón derecho y seleccionar “Propiedades”. Aparecerá una lista con todas las propiedades de dichos elementos que podrás modificar a tu gusto (p.ej. dimensiones, color, tipo de letra, bordes, etc.).

  Una de las propiedades más importantes es el nombre de la variable asociada a dicho elemento. Cada vez que creamos un elemento nuevo, se le asigna un nombre generado de forma automática y que es poco significativo. Cambia este nombre por otro más significativo (en la pestaña “Code” de las propiedades). Por ejemplo, al área de texto que acabamos de crear le podemos cambiar el nombre a “jtTextoCasilla” (ver la figura).



  En la pestaña “Properties” asegúrate de que el texto no es editable poniendo el valor de la propiedad *editable* a false y cierra la ventana de propiedades.

  Si vuelves al apartado “Source”, verás que aparece la declaración de una variable de tipo JTextArea y nombre jtTextoCasilla. Vamos a completar el código añadiendo el método **actualizar** para darle valor al contenido de este área de texto:

```
public void actualizar(String descripcionCasilla){
    this.jtTextoCasilla.setText(descripcionCasilla);
    this.repaint(); //Investiga para qué sirven estos métodos
    this.revalidate();
}
```

- Repite este proceso para la vista del jugador (VistaJugador) y para la vista de carta sorpresa (VistaCartaSopresa).
- Una vez definidas las tres vistas anteriores, vamos a crear la vista del juego (VistaQytetet) creando un JPanel con dicho nombre. En la pestaña “Design” aparecerá el panel vacío. Para darle contenido, arrastra cada una de las vistas creadas y colócalas donde consideres.
- Ve ahora a la pestaña “Source” y define el método actualizar que recibirá como argumento un objeto de la clase Qytetet. El cometido de este método será actualizar las tres vistas con los datos del jugador actual, la casilla actual de dicho jugador y la carta actual de qytetet. Ten en cuenta que si la carta actual es null, deberá mostrarse una cadena vacía o un mensaje indicando que no hay carta.

D) Implementación del controlador

Inicio

- En el paquete GUIQyetet crea un JFrame denominado ControladorQytetet. Como verás en la pestaña “Design”, un JFrame es una ventana que en este momento estará vacía.
- Vamos incluir la vista que hemos creado en el apartado C. Para ello, arrastra VistaQytetet sobre el JFrame.
- En la pestaña “Source” crea un atributo de instancia de tipo Qytetet denominado modeloQytetet. Define un método público denominado actualizar que recibe un objeto de tipo Qytetet para darle valor a este atributo.
- Verás que también se ha creado un método main, cambia el código por este:

```
public static void main(String[] args){
   ControladorQytetet controladorQytetet= new ControladorQytetet();
   controladorQytetet.setVisible(true);  //Esta debe ser la última
                                         //línea de código del main
}
```

Prueba 0: Si ejecutas, se mostrará una ventana con la vista, pero no podrás hacer uso de la funcionalidad del juego hasta que no completes el controlador siguiendo el resto del guión.

Lectura de jugadores

- Lo primero que vamos a hacer es capturar los nombres de los jugadores. Para ello, os proporcionamos la clase CapturaNombreJugadores.java junto con este guión. Puedes modificarla a tu gusto y debes añadirla al paquete GUIQytetet. Para usarla, añade al main:

```
CapturaNombreJugadores capturaNombres
          = new CapturaNombreJugadores(controladorQytetet, true);
ArrayList<String> nombres= capturaNombres.obtenerNombres();
```

- A continuación, también en el método main, obtén la instancia de Qytetet e inicializa el juego con los nombres leídos como has hecho en prácticas anteriores.

Nuevo dado

- Vamos a utilizar un nuevo dado, con una componente gráfica básica. Para ello se proporciona el código en la clase Dado.java que puedes personalizar y debes añadir al paquete GUIQytetet.
- En el método main, crea la instancia de dicho dado incluyendo el código:

```
Dado.createInstance(controladorQytetet);
```

- En el modelo, borra la clase Dado que tenías y cada vez que uses el dado en Qytetet pon:

```
Dado dado = GUIQytetet.Dado.getInstance();
```

Prueba 1: Añade también al main la actualización del controlador usando la instancia de Qytetet que has inicializado al final de la sección anterior y ejecuta este nuevo main. Verás que ahora aparecen los datos del jugador actual posicionado en la casilla de salida.

Definición de botones para el control del juego

- Para llevar a cabo las distintas acciones que puede realizar el jugador, se va a hacer uso del elemento gráfico JButton.
- En la pestaña “Design” añade 6 botones que se corresponderán con las acciones: intentar salir de la cárcel tirando el dado, intentar salir de la cárcel pagando, jugar, comprar, aplicar sorpresa y pasar turno. Para ello, basta con arrastrarlos a un lugar del JFrame de ControladorQytetet fuera del área asignada a la VistaQytetet.
- Puedes etiquetar los botones como prefieras, pero recuerda siempre dar nombres significativos a las variables que los referencian.

Programación de las acciones asociadas a los botones

- Para programar la acción asociada a hacer click sobre un botón, pulsa dos veces sobre el mismo y Netbeans te llevará al método asociado en la pestaña “Source” que se llama *nombrebotonActionPerformed* donde nombreboton es el nombre de la variable que referencia al Jbutton que estamos programando.
- Implementa dichos métodos siguiendo las reglas del juego y habilitando y deshabilitando los botones según el estado del juego. Por ejemplo, si la casilla actual es una calle con dueño o no es una calle, el botón de comprar debe estar deshabilitado.
- A continuación proporcionamos un ejemplo para el botón de salir de la cárcel pagando (debes modificarlo con el nombre de tus variables). Ten en cuenta que, tal y como se hace en este, en todos los métodos al final es necesario actualizar la vista de Qytetet para que se muestre el resultado de la acción.

```
private void jbSalirCarcelPagandoActionPerformed
     (java.awt.event.ActionEvent evt) {
   boolean resultado =
   modeloQytetet.intentarSalirCarcel(MetodoSalirCarcel.PagandoLibertad);
   this.jbSalirCarcelPagando.setEnabled(false);
   this.jbSalirCarcelDado.setEnabled(false);

   if(resultado){
      JOptionPane.showMessageDialog(this, "Sales de la cárcel");
      this.jbJugar.setEnabled(true);
   }else {
      JOptionPane.showMessageDialog(this, "NO sales de la carcel");
      this.jbSiguienteJugador.setEnabled(true);
    }
   this.vistaQytetet.actualizar(modeloQytetet);
}
```

- En ControladorQytetet tenemos ya una versión muy simple del método actualizar. Es necesario completarlo para establecer el estado inicial al comenzar el turno. Para ello invoca al método habilitarComenzarTurno (proporcionado a continuación) y actualiza la vista. Este método también será invocado desde el botón que pasa el turno una vez se haya actualizado el jugador actual.

```
public void habilitarComenzarTurno(){
      this.jbComprar.setEnabled(false);
      this.jbSiguienteJugador.setEnabled(false);
      this.jbAplicarSorpresa.setEnabled(false);
      if(modeloQytetet.getJugadorActual().getEncarcelado()){
              this.jbSalirCarcelPagando.setEnabled(true);
              this.jbSalirCarcelDado.setEnabled(true);
      }
      else
          this.jbJugar.setEnabled(true);
}
```

- Finalmente, ten en cuenta que cuando se llegue a la bancarrota se debe mostrar el ranking de jugadores. Para ello, puedes utilizar un JOptionPane como se ha mostrado al final de la página anterior.

Prueba 2: Prueba todo el código desarrollado y depura los posibles errores.