MVC – Interfaz de usuario

Introducción a los gráficos en computadora
Conceptos geométricos, estéticos y de usabilidad.
Componentes gráficos, jerarquía y tipos.
Administradores de disposición o diseño (Layouts)
Interacción y manejo de Eventos.
Objetos fuente y objetos oyentes
Clases oyentes y adaptadoras de eventos
Definición y uso de clases anónimas

Objetivos de la práctica

Construir interfaces gráficas
Aplicar el modelo MVC
Generar componentes gráficos en tiemo de edición y en tiempo de ejecución
Manejar eventos en los componentes gráficos.
Registro de Listeners en componentes creados en tiempo de ejecución.
Extender componentes definidos en Java.

Ejercicio:1

Modifica el juego de estrategia trabajado en las guías anteriores de manera que su interfaz gráfica aplique el patrón MVC.

Ejercicio:2

Escribe el juego del “Buscaminas” aplicando el patrón MVC

El modelo:

La clase Buscaminas tendrá un constructor con tres parámetros enteros, que indicarán el ancho y el alto del tablero, y la cantidad de minas. Dicho constructor lanzara una excepción de tipo ParametrosInvalidosException en el caso de que alguno de los tres parámetros sea negativo, o en caso de que la cantidad de minas sea mayor que la cantidad de casilleros del tablero.

La clase Buscaminas tendrá los métodos públicos

```
int getMinasCercanas (int i, int j)
```

```
boolean isMina (int i, int j)
```

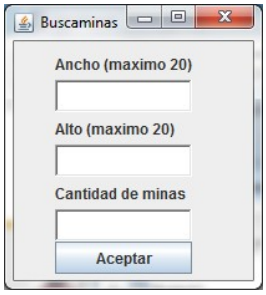
getMinasCercanas retornará la cantidad de minas que rodean la casilla ubicada en la posición [i,j].

isMina retornara true o false en caso de que haya o no una mina en la casilla ubicada en la posición [i,j].

Ambos métodos lanzaran una excepción de tipo FueraTableroException si los índices “i” o “j” hacen referencia a un casillero fuera del tablero.

La vista

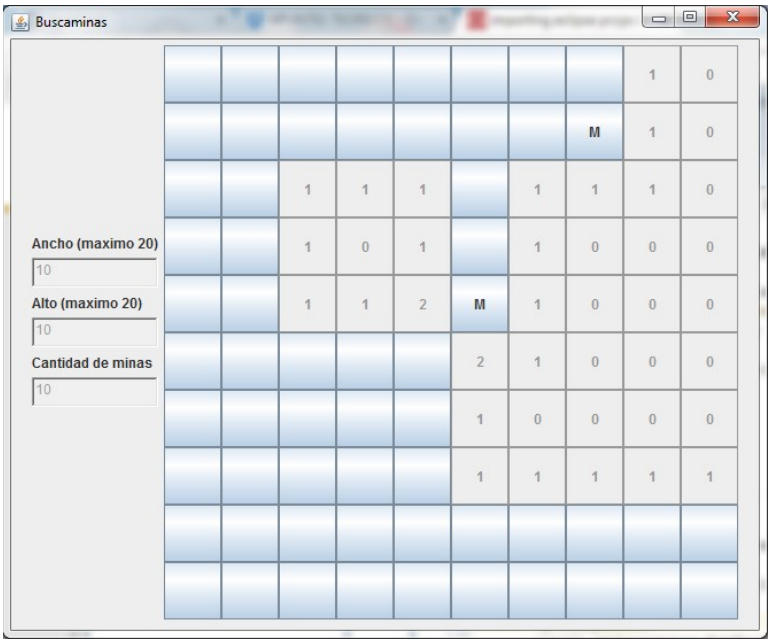
Al comenzar el juego la pantalla deberá ser similar a la de la ilustración, donde el usuario podrá seleccionar el ancho, alto y cantidad de minas del juego. Si los JTextField no contienen números enteros válidos y dentro de los límites aceptados, el botón aceptar permanecerá desactivado.



Una vez que se ingresaron los valores válidos, el juego generará los componentes necesarios.

El juego deberá comportarse de manera similar al original.

Al perder o ganar deberá mostrarse un cuadro de diálogo informativo.

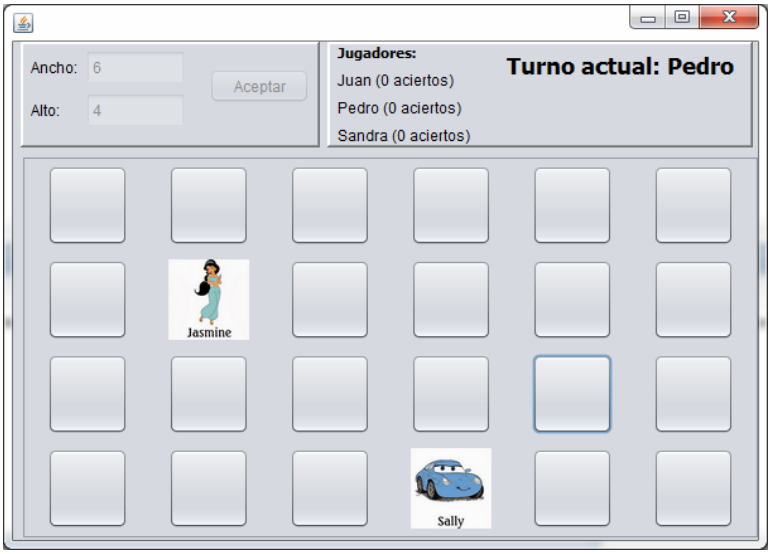


Ejercicio:3

Escribe el juego del ”Memotest” aplicando el patrón MVC.

La clase Tablero tendrá un constructor con dos parámetros enteros, que indicarán el ancho y el alto del tablero. Dicho constructor lanzara una excepción de tipo ParametrosInvalidosExpetion en el caso de que alguno de los dos parámetros sea negativo, o en caso de que la cantidad de casilleros del tablero sea impar.

La clase Memotest tendrá un constructor con dos parámetros, uno sera de tipo Tablero, y el otro será un array de String que representará los jugadores que participarán. Ambos parámetros deberán ser distintos de null, y el array de String deberá tener al menos un elemento.



Ilustracion de ejemplo

(Dentro del Memotest.zip encontrarás archivos en formato png para utilizar en tu aplicación.

En la carpeta llamada 64 encontrarás dibujos de 64x64 pixeles llamados 1_64px.PNG, 2_64_px.PNG, 3_64_px.PNG etc.

Análogamente hay una carpeta llamada 128 que contiene los mismos dibujos pero de 128x128 pixeles.)

Ejercicio:4

Una estación de servicio cuenta con surtidores capaces de proveer diferentes tipos de combustible, por ejemplo, Gasoil, Nafta Super, Nafta Premium, GNC.

Cada combustible tendrá un nombre y un precio por unidad.

Cada surtidor podrá despachar uno o más tipos de combustibles. Por cada tipo de combustible el surtidor guardará información acerca de:

- La capacidad máxima del depósito
- Cantidad de combustible y monto de la última carga realizada
- Cantidad de combustible y monto histórico del combustible en cuestión

En cada surtidor se mantiene registro de la cantidad de litros disponibles en depósito de cada tipo de combustible. En cada surtidor es posible cargar o reponer combustible.

Cuando se repone un combustible en el surtidor, se llena el depósito completo de ese combustible.

En ocasiones la cantidad de un tipo de combustible particular en un surtidor específico puede no ser suficiente para completar una carga, en ese caso se carga lo que se puede y cuando el depósito del surtidor queda vacío si no se completó la carga se lanza una excepción de tipo FaltaCombustibleException.

La excepción de tipo FaltaCombustibleException tiene un atributo privado de tipo DatoCargaInvalida que se pasará como parámetro en el constructor de la excepción, y tendrá su correspondiente getter. Al momento de lanzar la excepción, se pasarán los datos de la carga invalida.

Si se intenta cargar una cantidad negativa de combustible, entonces se lanzara una excepcion de tipo Exception, en cuyo constructor se pasará como parámetro, el mensaje “La cantidad de combustible debe ser positiva”.

Se debe crear una interfaz gráfica que permita:

- Registrar diferentes tipos de combustible, y modificar el precio por unidad de dicho combustible.
- Agregar surtidores a la estación de servicio. cada surtidor deberá poseer un panel que nos permita cargar cada tipo de combustible, indicado por cantidad de litros o por cantidad de dinero, así como también recargar el depósito de cada tipo de combustible. Además cada panel deberá mostrar la información del surtidor.