Universidad Nacional de General Sarmiento

Programación III
Comisión 2 Mañana

Trabajo Práctico

Programación III
TP1 1er Cuatrimestre 2020

Fernando Torres
Alejandro Nelis

APELLIDO Y NOMBRE	LEGAJO	EMAIL
Kairuz, Juan Pablo		juampykairuz@hotmail.com
Gauna, Nicolás		nicolas.gauna1998@gmail.com
Tula, Ignacio Mariano		ignacio.tula@gmail.com

Para el desarrollo e implementación de este trabajo práctico, se crearon 2 paquetes, 4 clases, 2 test unitarios. Y se realizó una aproximación a un modelo de Modelo-Vista-Controlador.

<table>
<tr><td>grilla.Grilla.java</td><td>-></td><td>Es una clase genérica, que permite representar un TAD Grilla, Tabla o Cuadrícula.
Es decir un conjunto de datos, organizados en filas y columnas.

<table><tr><td>(0,0)</td><td>(0,1)</td><td>(0,2)</td></tr><tr><td>(1,0)</td><td>(1,1)</td><td>(1,2)</td></tr><tr><td>(2,0)</td><td>(2,1)</td><td>(2,2)</td></tr></table>
Tad Grilla está definido como el conjunto de operaciones que permite crear una cuadrícula de fija de $n*n$ elementos con un valor por defecto (por ejemplo cero para valores numéricos), que permita modificar y obtener el valor almacenado en una posición según su fila y columna. También que obtenga todos los valores de forma serializada en una lista única, para una determinada columna, fila o la totalidad de la grilla.</td></tr>
<tr><td>grilla.GrillaTest.java</td><td>-></td><td>Son los test unitarios de la clase Grilla</td></tr>
<tr><td>juego.
TableroJuegoAritmetico.java

<pre>Grilla<Integer> grilla;
int[] resultadosColumna;
int[] resultadosFila;

boolean jugable;

int tamano;</pre></td><td>-></td><td>Es la clase que actúa como modelo en la aplicación y que permite crear tableros para el juego aritmético.

Que consiste principalmente:
Una grilla de números
Dos arreglos de entero (ambos con la misma longitud que la grilla) que representan correspondientemente la suma de los valores de las filas y las columnas.
Un valor booleano que especifica si ese tablero es jugable o no.
Un tamaño expresado en número entero.</td></tr>
</table>

public **TableroJuegoAritmetico** **Parameters:** **tamano - El tamaño del tablero** **jugable - Un tablero vacío para rellenar o sino uno con valores** private **poblarJuego** private **nuevoRandomAdecuado** public **cambiarValorCelda** public **getSumaDeLasFilas** public **getSumaDeLasColumnas** private **setSumaParcialFila** private **setSumaParcialColumna** private **ponerEnCeroResultadoSumas** private **reCalcularSumas** private **validacionInterna** private **validacionParaComparar** private **valoresEnCeroNoCuentanParaGanar** public **haGanado** static **comparaValores** static **todoCorrecto** public **getTamano** public **getTodosLosValoresSer(...)**		Esta clase genera 2 posibles tableros. Si el tablero es "jugable" se genera un tablero con ceros, que servirá para que el jugador vaya completando y averiguando cuánto va sumando, ganará cuando sus sumas de filas y columnas coincidan con un tablero no jugable. Si no es jugable, crea un tablero con valores random, que el jugador deberá descifrar valiéndose de un tablero jugable. Esta clase además es responsable de “poblar” un tablero no-jugable para obtener una solución válida asegurada. De permitir la modificación de las celdas de un tablero jugable pero no de uno no-jugable. De encargarse de obtener los valores cambiantes de la suma de filas y columnas. Y de entregar esos resultados. De realizar validaciones para que no se realicen operaciones no permitidas en tableros no-jugables, y que las comparaciones sean válidas. De realizar comparaciones, y determinar si un tablero ya ha ganado (porque llegó al objetivo) o no. De entregar la información necesaria para las operaciones fuera del modelo. Avisar si las sumas son coincidentes entre un tablero jugable y uno no-jugable, si se ha ganado, el tamaño del tablero, etc.
juego. TableroJuegoAritmeticoTest. java	->	Son los test unitarios de la clase Grilla TableroJuegoAritmetico

juego.GuiJuego.java

Es la clase que actúa como **vista** en la aplicación y que permite dar un entorno gráfico al tablero (las celdas que el usuario debe completar, los valores que debe obtener sumando filas y columnas, el menú para elegir un nuevo juego).

Esta vista nunca interactúa con el modelo directamente, sino que lanza eventos que son escuchados por un controlador.

El controlador le solicita a esta vista, que se muestre determinado tamaño de tablero, o que muestre determinados colores para determinadas celdas.

Visualmente consiste en un Jpanel principal con un borderLayout, donde se destacan:

En la posición north, la barra de menú donde puede seleccionarse un juego nuevo.

Se decidió implementar un *ArrayList de JMenuItem* para poder brindar la posibilidad al usuario de elegir la dificultad (o el tamaño del tablero) sin repetir el código para cada JMenuItem.

GuiJuego.java (líneas 80 a 89)

En la posición south, una etiqueta para expresarle mensajes al usuario.

En east y west, márgenes fijos para emprolijar.

Y finalmente, en el center, un nuevo Panel con un gridLayout en principio vacío. Donde programáticamente dicha gridLayout se irá modificando y al panel se le irán agregando los JTextField (que representan los valores que el usuario puede cambiar, y los valores de las sumas de filas y columnas) de forma programática, dependiendo del tamaño del tablero seleccionado en el menú de “nuevo juego”.

Se decidió implementar una serie de *ArrayList de JMenuItem* para distinguir entre los TextField editables que pertenecen a la parte del tablero extendido[1] que corresponde al tablero jugable y a los textField no-editables que pertenecen a las sumas que se deben obtener en filas y columnas obtenidas del tablero no-jugable.

Para el envío de eventos, se crearon funciones que reciben los objetos ActionListener desde el controlador. La vista se encarga de realizar las iteraciones necesarias para agregar a cada elemento visual, dicho ActionListener (o DocumentListener).

[1] Tablero extendido es en el apartado gráfico al tablero formado por la representación gráfica del tablero jugable más una fila y una columna adicional para las sumas de las mismas, y un espacio vacío en la última posición de fila y columna. Por ejemplo, un tablero jugable de 3x3, requiere visualmente, para ser comprendido y resuelto, otro tablero extendido de 4x4.

- **getIndiceGUIJugador**

  `public int getIndiceGUIJugador(JTextField celda)`

  Esta función permite calcular el índice que ocupa en el arreglo de la grilla del usuario una determinada celda

  Parameters:

  `celda - a la cual se quiere averiguar su posición`

  Returns:

  `Un entero con la posición que ocupa en el arreglo de la grilla del usuario`

- **inicializarEmisionEventosCeldas**

  `public void inicializarEmisionEventosCeldas(event.DocumentListener documentListener)`

  Le indica a cada celda del tablero quien va a estar escuchando las modificaciones que se realizan sobre dicha celda. Adicionalmente se carga una propiedad que contiene el índice de la celda, para conocer a quien representa en el tablero.

  Parameters:

  `documentListener - Enviado desde algún controlador, que ejecutará alguna acción al dispararse el evento`

- **inicializarEmisionEventosMenu**

  `public void inicializarEmisionEventosMenu(event.ActionListener actionEscucha)`

  Le indica a cada item del menu quién va a estar escuchando los eventos que se disparan desde cada uno.

  Parameters:

  `actionEscucha - Enviado desde algún controlador, que ejecutará alguna acción al dispararse el evento`

- **getListadoMenuItemsNuevo**

  `private ArrayList<JMenuItem> getListadoMenuItemsNuevo()`

  Returns:

  `Devuelve el ArrayList que contiene todos los items del menú de nuevo juego`

- **getIndiceMenuItemNuevo**

  `public int getIndiceMenuItemNuevo(JMenuItem menuitem)`

  Esta función permite calcular el índice que ocupa en el arreglo de items del menú nuevo juego

  Parameters:

  `menuitem - al cual se quiere averiguar su posición`

  Returns:

  `Un entero con la posición que ocupa en el arreglo del listado de items del menú nuevo`

- **setMensaje**

  `public void setMensaje(String mensaje)`

  Permite escribir un mensaje en la etiqueta inferior.

  Parameters:

  `mensaje - El mensaje que se quiere mostrar`

- **crearNuevoTablero**

  `public void crearNuevoTablero(int tamano,int[] sumaFila,int[] sumaCol)`

  Esta función se encarga de crear y dibujar los Jtextfield (editables y no editables) que representan un tablero extendido, sobre un gridLayout dinámico. Dependiendo del tamaño

  Parameters:

  `tamano - Entero que representa el tamaño en alto y/o ancho del tablero`

  `sumaFila - Listado de números enteros representando lo que deben sumar las filas ordenadamente`

  `sumaCol - Listado de números enteros representando lo que deben sumar las columnas ordenadamente`

- **actualizarColoresSumas**

  `public void actualizarColoresSumas(boolean[] resultadofilas, boolean[] resultadocolumnas)`

  Colorea las celdas no-editables para indicar si los valores expresados en el tablero del jugador suman lo indicado por dichas celdas no-editables.

  Parameters:

  `resultadofilas - Un listado contiene ordenadamente la correctitud o no de las filas que representan la suma`

  `resultadocolumnas - Un listado contiene ordenadamente la correctitud o no de las c que representan la suma`

juego.Arbitro.java

Es la clase que actúa de controlador, se le debe indicar una interfaz gráfica de que la observará diferentes eventos.
Luego el solicitará a la clase que actúa de modelo, dos tableros (uno jugable y otro no-jugable) al ser avisado de un evento informado por la GUI. También solicitará al modelo se realicen acciones sobre los tableros y las sumas y dependiendo dichos resultados, se comunicará con la interfaz gráfica para actuar en consecuencia.

- **inicializarEscuchaEventosMenu**
  `public void inicializarEscuchaEventosMenu()`
  Solicita a la interfaz gráfica que agregue los listener de este controlador a los items del menú Nuevo Juego.

- **inicializarEscuchaEventosGrilla**
  `public void inicializarEscuchaEventosGrilla()`
  Solicita a la interfaz gráfica que agregue los listener de este controlador a a cada grilla que el usuario puede modificar en ese momento y obtener así los valores que el mismo usuario está cambiando para saber si ganó o no.

- **comenzarJuegoNuevo**
  `public void comenzarJuegoNuevo(int dificultad)`
  Las acciones que realiza el controlador cuando se crea un nuevo juego. Aquí se construyen dos modelos de tablero, uno jugable y otro no-jugable. Se obtiene del modelo objetivo las sumas a las que debe llegar el usuario y se envían a la gráfica para sean mostradas de forma adecuada al usuario
  Parameters:
  `dificultad - el tamaño del tablero elegido por el usuario al indicar un juego nuevo`

  Posee además dos clases internas, que implementan interfaces de escucha de eventos. Una para recibir cambios en las celdas editables por el usuario, y otro para recibir aviso de una solicitud de juego nuevo, mediante los items del menú.
- `class` **`Arbitro.EscucharCambiosDelUsuarioEnElTablero`**
  `implements javax.swing.event.DocumentListener`
  Clase interna que recibe los eventos producidos por una modificación en el valor contenido en una grilla modificable por el usuario

  - **cambiosEnElTablero**
    `private void cambiosEnElTablero(javax.swing.event.DocumentEvent e)`
    Función en común que comparten los 3 diferentes eventos de la interface DocumentListener Aquí se obtiene el valor ingresado por el usuario en el tablero, y su posición. Se valida el dato y posteriormente se envía al modelo para que rehaga las suma y compare si el usuario ya ha ganado. Dependiendo el resultado, se le solicita a la interfaz gráfica que redibuje lo necesario. **Parameters:** `e -`

- class **Arbitro.EscucharNuevoJuego**
  implements java.awt.event.ActionListener
  Clase interna que recibe los eventos producidos por un clic en un item del menú nuevo juego y la lógica necesaria para recibir, cual es el tamaño del tablero elegido.

juego.GuiFelicitacion.java

Esta clase es una pequeña interfaz, llamada por el controlador cuando este recibe la información de que el usuario ha ganado.