Programación III

TP1 2do Cuatrimestre 2018

Estudiantes:

Figueroa, Carlos Luciano.

Legajo: 28490610/2017

Email: lumusika@gmail.com

Fernández Gómez, Carlos Alberto.

Legajo: 94053350/2016

Email: carlos070686@gmail.com

Comisión: 1 Turno Noche

18/9/2018

Docentes: Bagnes, Patricia; Marenco, Javier.

Introducción:

El presente trabajo se trata del desarrollo de una aplicación en Java para jugar al "2048". Para ello se organizó el proyecto en dos capas, una dedicada exclusivamente a la interfaz de usuario (GUI) y otra encargada de la lógica del juego. Con esto se logró garantizar la escalabilidad de la aplicación y asegurar la estabilidad ante cambios en la capa de alguna de ellas ya que se evita un alto acoplamiento. Básicamente se decidió crear dos paquetes distintos con funciones bien diferenciadas:

a) En primer lugar, un paquete llamado ***interfaz.*** Este contiene:

- ✓ La clase de tipo *Application Window* llamada **MainFrame.java**, trata sobre la representación visual del juego, para su construcción se utilizó el plugin WindowsBuilder dentro de la aplicación Eclipse para diseñar las interfaces gráficas.
- ✓ **Icon.png** Es una imagen utilizada como icono de la aplicación.
- ✓ **Informe.pdf** Es el informe en formato pdf, el cual servirá como ayuda cuando el usuario presione la tecla de ayuda *F1* o seleccione la opción *'Acerca de'* en la barra de menú en la pantalla del juego.

b) Y en segundo lugar, un paquete llamado ***negocio***. Este contiene:

- ✓ Una clase llamada **Tablero.java**, la cual se utiliza para manejar toda la representación lógica del juego, su estructura de datos.

A continuación, se detallan las partes antes descritas:

Clases:

a) MainFrame.java:

La clase **MainFrame.java** está contenida dentro de un paquete llamado *interfaz.*

Esta clase contiene las siguientes variables de instancias:

private JFrame ventana; // Es un objeto del tipo JFrame para dibujar la ventana.
private JPanel contenedorDeCuadros; // Es un objeto del tipo JPanel sobre el cual van a estar los cuadros que representan el tablero del juego.
private JTextField[][] cuadros; // Son los campos de textos utilizados para mostrar los números en los tableros.
private int cuadroPosX; // Variable auxiliar para almacenar la Coordenada X de la posición inicial de cada cuadro que contiene los números dentro del tablero de juego.
private int cuadroPosY; // Variable auxiliar para almacenar la Coordenada Y de la posición inicial de cada cuadro que contiene los números dentro del tablero de juego.
private int cuadrosTamanio; // Variable para almacenar el tamaño de cada cuadro que contiene los números dentro del tablero de juego.
private Tablero tableroDeValores; // Es un objeto del tipo Tablero en el cual voy a manejar la lógica para luego

representar gráficamente el tablero de juego.
private JTextField cuadroPuntaje; // Es el campo de texto utilizado para mostrar en la pantalla de juego el puntaje obtenido.
private String puntajeAMostrar; // Es una variable auxiliar para almacenar el puntaje entero al cual se convierte a String para poder mostrarlo en el tablero de juego.
private JTextField cuadroRecord; // Es el campo de texto utilizado para mostrar en la pantalla de juego el cuadro de records.
private String recordAMostrar; // Es una variable auxiliar para almacenar el record entero al cual se convierte a String para poder mostrarlo en el tablero de juego.
private JPanel contenedorDePuntajesHistoricos; // Es el panel sobre el cual van a estar todos los puntajes históricos.
private JTextField [] cuadrosRecordsHistoricos; // Es un Array de tipo JTextField en donde se van a almacenar los records históricos de los usuarios, sus nombres y el nivel en el cual estaban cuando alcanzaron dicho record.
private JTextArea cuadroDeMsj; // Objeto del tipo JTextArea utilizado para mostrar los mensajes dinámicos a los usuarios de acuerdo al puntaje, record alcanzado, reincios, etc.

Métodos de la Clase *MainFrame.java*:

public static void main(String[] args) // Metodo principal del juego en donde se ejecuta el juego.
public MainFrame() // Es el constructor que crea la aplicación.
private void initialize() // Metodo que inicializa los valores de ventana.
public void ventanaPrincipal () // Metodo para crear la ventana principal, sus atributos y componentes como la barra de menús, mensajes al usuario etc.
private void close() // Imprime una ventana en donde pregunta al usuario si realmente quiere salir del juego.
public void inicio() // Imprime una ventana en donde pregunta al usuario si quiere continuar con la última partida guardada, en caso afirmativo lee el juego guardado en archivo e imprime por qué usuario fue guardada.
public void tableroDeJuego() // Este método se utiliza para dibujar el tablero de juego, primero se crea el panel que contiene los cuadros, luego se escuchan los eventos de teclado del panel, se inicializan las variables del tablero gráfico y la posición de los cuadros y por último se crean los cuadros y se los ordena en el panel.
public void agregarValoresIniciales() // Agrega los valores iniciales en el tablero, dos números aleatorios (2 o 4) en lugares aleatorio.
public void puntaje() // Metodo utilizado para mostrar el puntaje en el tablero de juego.
public void record() // Metodo utilizado para mostrar el record en el tablero de juego.
private void recordsHistoricos() // Metodo utilizado para mostrar los records históricos en el tablero de juego.
public void actualizarRecordsHistoricos() // Metodo utilizado para actualizar los records históricos en el tablero de records dentro del tablero de juego dependiendo del nivel seleccionado este cambia de color.
public void escucharTeclado(JPanel panel, JButton boton) // Metodo usado para capturar los eventos del teclado para controlar el juego.
public void controlDeMovimiento(KeyEvent e) // Este método recibe un objeto del tipo *KeyEvent* y lo evalúa para determinar la dirección, luego compara el puntaje obtenido para actualizar el record y por ultimo imprime un mensaje de fin de juego y pregunta al usuario si desea volverá intentar.
public void seleccionDeNivel() // Con este método se imprime un mensaje al usuario preguntando el nivel de dificultad deseado: principiante, intermedio o experto.

public void actualizarTableroGrafico() // Método utilizado para recuperar los valores del tablero de negocio y actualizar el valor de cada cuadro gráfico y su color de fondo.
public void actualizarCuadros(JTextField cuadro, String valor) // Este método actualiza la vista gráfica del tablero (cuadros), sus valores y sus colores. De acuerdo al valor que posee el cuadro, le da un color de texto y de fondo distinto.
public void botonJuegoNuevo() // Con este método se da formato y estilo al botón de Juego Nuevo, y este queda escuchando por algún evento del tipo MouseEvent.
public void setUsuarioNuevo() // Imprime una ventana en donde pide al usuario que ingrese su nombre.
public void cuadroDeMsjAlUsuario() // Método usado para dar formato y atributos a los mensajes dinámicos que se muestran en la pantalla de juego.
public void reinicioOnuevo(String eleccion) // Este método recibe por parámetro un String en donde el usuario elige si 'reinicia' el juego; se pide al tablero de negocio que se reinicie, se borran todos los contenidos de los cuadros gráficos y se vuelve a poner el color de fondo inicial. O si el usuario elige 'nuevo' se le vuelve a pedir el usuario y el nivel de dificultad y vuelve a empezar el juego.
public void actualizarPuntaje() // Este método pasa a String el puntaje entero de la clase tablero.java.
public void actualizarRecord() // Este método pasa a String el record entero de la clase tablero.java.
public void setMsj(String msj) // Con este método selecciona todo el texto del jTextArea y luego reemplaza la selección con el texto que recibe por parámetro.
public void guardarJuego() // Método usado para guardar el juego actual en un archivo de texto llamado: 'juegoGuardado.txt'.
public void leerJuegoGuardado() // Método usado para leer el juego anteriormente guardado en un archivo de texto llamado: 'juegoGuardado.txt'.
public void msjPorEscape() // Muestra un mensaje de aliento al usuario cuando este presiona la tecla 'escape'.
public void msjPorReinicio() // Muestra un mensaje en donde pregunta al usuario si realmente quiere reiniciar el juego, y luego captura la decisión tomada.
public void msjPorSuperaRecord() // Muestra un mensaje al usuario cuando este haya conseguido un superar el record. Este será diferente de acuerdo a los puntos que se necesitaron para esto.
public void msjPorPuntaje() // Similar a los anteriores, imprime mensajes para el usuario de acuerdo al puntaje.
msjPorLograr2048EnTablero() //Imprime un mensaje de Felicitación por haber logrado 2048 en un cuadro del tablero.

b) Tablero.java:

La clase **Tablero.java** está contenida dentro de un paquete llamado *negocio* para diferenciarla del otro paquete dedicado a la interfaz gráfica antes detallada.

Esta clase contiene las siguientes variables de instancias:

private int[][] tablero; // Es una matriz de enteros el cual va representar lógicamente el tablero de juego.
private int tamanio; // Se utiliza esta variable de enteros para pasarle por parámetro al constructor al momento de inicializar la matriz del tablero de juego, de esta manera se asegura la escalabilidad del mismo, de la forma . *tablero = new int[tamanio][tamanio];*

private int fila; // Esta variable entera se utiliza para guardar temporalmente el valor de la fila en operaciones con la matriz.

private int columna; // Esta variable entera se utiliza para guardar temporalmente el valor de la columna en operaciones con la matriz.

private String usuario; // En esta variable se almacena el nombre del usuario.
private String nivel; // En esta variable se almacena el nivel seleccionado por el usuario.
private int puntaje; // En esta variable entera se almacena el puntaje obtenido por el usuario.
private String [][] records; // En esta Matriz de Strings se usa para guardar los records de los últimos mayores puntajes obtenidos con el siguiente detalle:

- *records[0]:* // Guarda el Nombre del Jugador con record.
- *records[1]:* // Guarda el Record obtenido.
- *records[2]:* // Guarda el Nivel con el que hizo ese record.

private boolean sumaDerechaEfectuada; // Esta variable booleana se usa para indicar si se realizó una suma a la derecha.
private boolean sumaIzquierdaEfectuada; // Esta variable booleana se usa para indicar si se realizó una suma a la izquierda.
private boolean sumaArribaEfectuada; // Esta variable booleana se usa para indicar si se realizó una suma hacia arriba.
private boolean sumaAbajoEfectuada; // Esta variable booleana se usa para indicar si se realizó una suma hacia abajo.
private boolean movDerechaEfectuado; // Esta variable booleana se usa para indicar si se realizó un movimiento hacia la derecha.
private boolean movIzquierdaEfectuado; // Esta variable booleana se usa para indicar si se realizó un movimiento hacia la izquierda.
private boolean movArribaEfectuado; // Esta variable booleana se usa para indicar si se realizó un movimiento hacia arriba.
private boolean movAbajoEfectuado; // Esta variable booleana se usa para indicar si se realizó un movimiento hacia abajo.

Métodos de la Clase *Tablero.java*:

- *public Tablero()* : Es el constructor de la clase, en ella se asignan los valores por defecto para cada una de sus variables de instancias antes nombradas.
- *public void agregarValoresIniciales()*: Este método llama dos veces al método *agregarValorRandom()* para agregar 2 valores distintos aleatorios en 2 posiciones distintas aleatorias.
- *boolean agregarValorRandom()* : Agrega un valor aleatorio (2 o 4) a una ubicación aleatoria del tablero.
- *boolean lugarAleatorio()*: Busca un lugar aleatorio dentro de la matriz de juego, si lo encuentra almacena la posición en las variables de instancia *int fila e int columna* y retorna **true**. Caso contrario retorna **false**.
- *int dosOCuatro()* : Este método devuelve los valores enteros 2 o 4 dependiendo del nivel de dificultad del juego seleccionado por el usuario (principiante – intermedio – experto).
- *existeLugarDisponible()* : Busca un lugar vacío disponible en la matriz del juego, si lo encuentra almacena la posición en las variables de instancia *int fila* e *int columna* y retorna **true**. Caso contrario retorna **false**.
- *public int getTamanio() :* Este método es un getter de la variable de instancia *int tamanio*.

- *public int getValor(int fila, int columna) :* Con este método se obtiene el valor del tablero de la ubicación pasada por parámetro.
- *public void setValor(int fila, int columna, int valor) :* Con este método se guarda el valor pasado por el parámetro *int valor*, al tablero en la ubicación también pasada por parámetro *int fila, int columna*.
- *public String getUsuario() :* Este método es un getter de la variable de instancia *string usuario*.
- *public void setUsuario(String usuario) :* Con este método se guarda el nombre de usuario pasado por parámetro a la variable de instancia string *usuario*.
- *public int getPuntaje() :* Este método es un getter de la variable de instancia *int puntaje*.
- *void setPuntaje(int puntaje) :* Con este método se guarda el puntaje pasado por parámetro a la variable de instancia *int puntaje*.
- *public int getRecord(int posicion) :* Este método devuelve el record de la posición pasada por parámetro *int posicion*.
- *public void setRecord(int puntaje, int posicion) :* Este método guarda el puntaje y la posición pasada por parámetro *int puntaje, int posición.*
- *String getNivel() :* Devuelve el valor de la variable de instancia *String nivel.*
- *public void setNivel(String nivel) :* Guarda el valor pasado por parámetro en la variable de instancia *String nivel.*
- *String getNivelDeLaListaDeRecords(int posicion) :* Devuelve el nivel de la posicion pasada por parámetro *int posición* de la lista de records.
- *public void setNivelDeLaListaDeRecords(String nivel, int posicion) :* Guarda el nivel en la posicion pasada por parámetro *String nivel, int posicion* en la lista de records.
- *public String getUsuarioConRecord(int posicion) :* Devuelve el usuario de la posicion pasada por parámetro *int posición.*
- *public void setUsuarioConRecord(String usuario, int posicion) :* Guarda el usuario en la posicion pasada por parámetro *String usuario, int posicion.*
- *String getRecordRealizado(int posicion)* : Devuelve el record realizado en la posicion pasada por parámetro *int posición.*
- *public void setRecordRealizado(String record, int posicion)* : Guarda el record realizado en la posicion pasada por parámetro *String record, int posicion.*
- *public String getNivelUsado(int posicion) :* Devuelve el nivel usado de la posición pasada por parámetro *int posición.*
- *void setNivelUsado(String nivelUsado, int posicion)* : Guarda el nivel usado en la posición pasada por parámetro *String nivelUsado, int posicion.*
- *public boolean estaValor2048EnTablero() : Devuelve un booleano en función si existe o no 2048 en alguna posición de la matriz tablero*
- *boolean getSumaDerechaEfectuada()* : Retorna un valor booleano de la variable de instancia *boolean sumaDerechaEfectuada.*
- *boolean getMovDerechaEfectuado()* : Retorna un valor booleano de la variable de instancia *boolean movDerechaEfectuado.*
- *boolean getSumaIzquierdaEfectuada()* : Retorna un valor booleano de la variable de instancia *boolean sumaIzquierdaEfectuada.*
- *boolean getMovIzquierdaEfectuado()* : Retorna un valor booleano de la variable de instancia *boolean movIzquierdaEfectuado.*
- *boolean getSumaArribaEfectuada()* : Retorna un valor booleano de la variable de instancia *boolean sumaArribaEfectuada.*
- *boolean getMovArribaEfectuado()* : Retorna un valor booleano de la variable de instancia *boolean movArribaEfectuado.*

- *boolean getSumaAbajoEfectuada()* : Retorna un valor booleano de la variable de instancia *boolean sumaAbajoEfectuada.*
- *boolean getMovAbajoEfectuado()* : Retorna un valor booleano de la variable de instancia *boolean movAbajoEfectuado.*
- *public void reiniciar()* : Este método se utiliza para vaciar todo el tablero, reiniciar los puntajes, los valores booleanos de la sumas y movimientos y agrega los 2 valores iniciales aleatorios en lugares aleatorios.
- *void controlDePuntajes()* : Este método va actualizando el listado de records de acuerdo al puntaje obtenido por cada usuario.
- *void vaciarTablero()* : Este método pone en cero toda la matriz del tablero de juego.
- *public boolean mover(String direccion)* : Este método se utiliza para realizar un movimiento en la dirección pasada por parámetro *String direccion* dentro del tablero de juego. Retorna **true** si pudo realizar el movimiento, caso contrario retorna **false.**
- *boolean juegoActivo()* : Este método devuelve un valor booleano **true** si se pudo realizar un movimiento o una suma, caso contrario retorna **false.**
- *void resetearSumasYMov()* : Pone en **true** todas las variable de instancias booleanas relacionadas al control de movimientos y sumas.

Estos métodos son los principales para controlar la lógica de la suma y el movimiento dentro de la matriz que representa el juego:

- *boolean sumarDerecha()* : Realiza una suma hacia la derecha si existen dos números iguales consecutivos o separados por cero (vacío), en todas las filas de la matriz del tablero de juego. Devuelve **true** si pudo realizar alguna suma, caso contrario devuelve **false.**
- *boolean sumarDerecha(int fila, int actual, int anterior)* : Realiza una suma hacia la derecha si existen los números pasados por parámetro *int actual, int anterior* y son iguales, en la fila pasada por parámetro *int fila*. Devuelve **true** si pudo realizar la suma, caso contrario devuelve **false.**
- *boolean moverDerecha()* : Realiza un movimiento hacia la derecha del contenido de todas las filas de la matriz del tablero de juego. Devuelve **true** si pudo realizar el movimiento, caso contrario devuelve **false.**
- *boolean moverDerecha(int fila, int lugarLibre, int anterior)* : Realiza un movimiento hacia la derecha si existen un lugar libre y un número pasados por parámetro *int lugarLibre, int anterior*, esto lo hace en la fila pasada por parámetro *int fila*. Devuelve **true** si pudo realizar el movimiento, caso contrario devuelve **false.**
- *boolean sumarIzquierda()* : Realiza una suma hacia la Izquierda si existen dos números iguales consecutivos o separados por cero (vacío), en todas las filas de la matriz del tablero de juego. Devuelve **true** si pudo realizar alguna suma, caso contrario devuelve **false.**
- *boolean sumarIzquierda(int fila, int actual, int siguiente)* : Realiza una suma hacia la Izquierda si existen los números pasados por parámetro *int actual, int siguiente* y son iguales, en la fila pasada por parámetro *int fila*. Devuelve **true** si pudo realizar la suma, caso contrario devuelve **false.**
- *boolean moverIzquierda()* : Realiza un movimiento hacia la Izquierda del contenido de todas las filas de la matriz del tablero de juego. Devuelve **true** si pudo realizar el movimiento, caso contrario devuelve **false.**

- *boolean moverIzquierda(int fila, int lugarLibre, int siguiente)* : Realiza un movimiento hacia la Izquierda si existen un lugar libre y un número pasados por parámetro *int lugarLibre, int siguiente*, esto lo hace en la fila pasada por parámetro *int fila*. Devuelve **true** si pudo realizar el movimiento, caso contrario devuelve **false.**
- *boolean sumarArriba()* : Realiza una suma hacia Arriba si existen dos números iguales consecutivos o separados por cero (vacío), en todas las Columnas de la matriz del tablero de juego. Devuelve **true** si pudo realizar alguna suma, caso contrario devuelve **false.**
- *boolean sumarArriba(int columna, int actual, int siguiente)* : Realiza una suma hacia Arriba si existen los números pasados por parámetro *int actual, int siguiente* y son iguales, en la Columna pasada por parámetro *int columna*. Devuelve **true** si pudo realizar la suma, caso contrario devuelve **false.**
- *boolean moverArriba()* : Realiza un movimiento hacia Arriba del contenido de todas las Columnas de la matriz del tablero de juego. Devuelve **true** si pudo realizar el movimiento, caso contrario devuelve **false.**
- *boolean moverArriba(int columna, int lugarLibre, int siguiente)* : Realiza un movimiento hacia Arriba si existen un lugar libre y un número pasados por parámetro *int lugarLibre, int siguiente*, esto lo hace en la Columna pasada por parámetro *int columna*. Devuelve **true** si pudo realizar el movimiento, caso contrario devuelve **false.**
- *boolean sumarAbajo()* : Realiza una suma hacia Abajo si existen dos números iguales consecutivos o separados por cero (vacío), en todas las Columnas de la matriz del tablero de juego. Devuelve **true** si pudo realizar alguna suma, caso contrario devuelve **false.**
- *boolean sumarAbajo(int columna, int actual, int anterior)* : Realiza una suma hacia Abajo si existen los números pasados por parámetro *int actual, int anterior* y son iguales, en la Columna pasada por parámetro *int columna*. Devuelve **true** si pudo realizar la suma, caso contrario devuelve **false.**
- *boolean moverAbajo()* : Realiza un movimiento hacia Abajo del contenido de todas las Columnas de la matriz del tablero de juego. Devuelve **true** si pudo realizar el movimiento, caso contrario devuelve **false.**
- *boolean moverAbajo(int columna, int lugarLibre, int anterior)* : Realiza un movimiento hacia Abajo si existen un lugar libre y un número pasados por parámetro *int lugarLibre, int anterior*, esto lo hace en la Columna pasada por parámetro *int columna*. Devuelve **true** si pudo realizar el movimiento, caso contrario devuelve **false.**
- *boolean existeValor(int fila, int columna)* : Devuelve un valor booleano **true** si existe algún valor distinto de cero en la posición pasada por parámetro *int fila, int columna.*
- *void sumarCuadrosDeFila(int fila, int actual, int cuadroASumar)* : Suma el contenido de las dos ubicaciones de la fila pasados por parámetro en *int fila, int actual, int cuadroASumar* y actualiza el valor de la variable de instancia *int puntaje.*
- *void sumarCuadrosDeColumna(int columna, int actual, int cuadroASumar)* : Suma el contenido de las dos ubicaciones de la Columna pasados por parámetro en *int columna, int actual, int cuadroASumar* y actualiza el valor de la variable de instancia *int puntaje.*
- *void moverCuadro(int OrigenFila, int OrigenColumna, int DestinoFila, int DestinoColumna)* : Mueve el contenido de la ubicación pasada por parámetro: *int OrigenFila, int OrigenColumna* a la ubicación *int DestinoFila, int DestinoColumna* y pone en cero el valor de la ubicación de origen.
- *void agregarValoresRandom(int cant)* : Agrega la cantidad de *int cant* pasado por parámetro de valores random.

Estos métodos se utilizan para realizar el testing con la herramienta JUnit:

- *void setFilaCompletaConValor(int fila, int valor)* : Pone en toda una fila *int fila* el valor *int valor* pasados por parámetro.
- *void setColumnaCompletaConValor(int columna, int valor)* : Pone en toda una columna *int columna* el valor *int valor* pasados por parámetro.
- *boolean verificarValoresEnColumnaCompleta(int columna, int valor)* : Verifica utilizando acumuladores booleanos que toda una columna *int columna* contenga el valor pasado por parámetro *int valor.*
- *boolean verificarValoresEnFilaCompleta(int fila, int valor)* : Verifica utilizando acumuladores booleanos que toda una fila *int fila* contenga el valor pasado por parámetro *int valor.*
- *boolean verificarValoresEnTableroCompleto(int valor)* : Verifica utilizando acumuladores booleanos que toda la matriz del juego contenga el valor pasado por parámetro *int valor.*

Conclusión:

Como resultado, podemos apreciar la excelente oportunidad brindada para asentar y poner en práctica los conocimientos adquiridos en la cátedra.
Tanto el uso de la herramienta *WindowBuilder* para la creación de la interfaz gráfica, como el uso de la *tecnología GIT* para el control de versiones y el trabajo colaborativo en equipo, amplían mucho nuestra gama de recursos y herramientas, tan necesarios para dar soluciones a problemas con mayor eficiencia.
Si bien esos son los temas centrales de este trabajo, pudimos hacer uso de otros instrumentos vistos en materias anteriores como el uso de acumuladores booleanos al recorrer elementos para hacer el testing, el uso de captura de las excepciones generadas con Try Catch, el uso en mayor profundidad del plugin *Junit* para las pruebas mediante Test unitarios, etc. Todas ellas muy importantes a la hora de planificar e implementar soluciones ante problemas dados, y nos da una base mucho más firme para empezar con los trabajos prácticos siguientes.
Sin duda, este recorrido no podría haber sido posible sin el acompañamiento de nuestros excelentes docentes que, con mucha paciencia y claridad, brindaron su amplio conocimiento y nos guiaron en la adquisición de los contenidos antes mencionados.