

PROGRAMACIÓN ORIENTADA A OBJETOS

INTERFAZ 2024-2

Laboratorio 5 / 6

Ciclo 0: Ventana vacía – Salir

[En *.java y lab05.doc]

El objetivo es implementar la ventana principal de **Clustering** con un final adecuado desde el icono de cerrar. Utilizar el esquema de prepareElements-prepareActions.

1. Construyan el primer esquema de la ventana de Clustering únicamente con el título "Clustering". Para esto cree la clase ClusteringGUI como un JFrame con su creador (que sólo coloca el título) y el método main que crea un objeto ClusteringGUI y lo hace visible. Ejecútenlo. Capturen la pantalla. (Si la ventana principal no es la inicial en su diseño, después deberán mover el main al componente visual correspondiente) Para esto creamos la clase ClusteringGUI y creamos el método main en donde se crea un objeto JFrame con el título Clustering, lo cual significa que la ventana tendrá ese nombre en la barra superior.

```
import javax.swing.*;
import java.awt.*;

public class ClusteringGUI {

    Run | Debug
    public static void main(String[] args) {
        JFrame frame = new JFrame(title:"Clustering");
        frame.setSize(width:854, height:480);
        frame.setVisible(b:true);
    }
}
```



2. Modifiquen el tamaño de la ventana para que ocupe un cuarto de la pantalla y ubíquela en el centro. Para eso inicien la codificación del método `prepareElements`. Capturen esa pantalla.

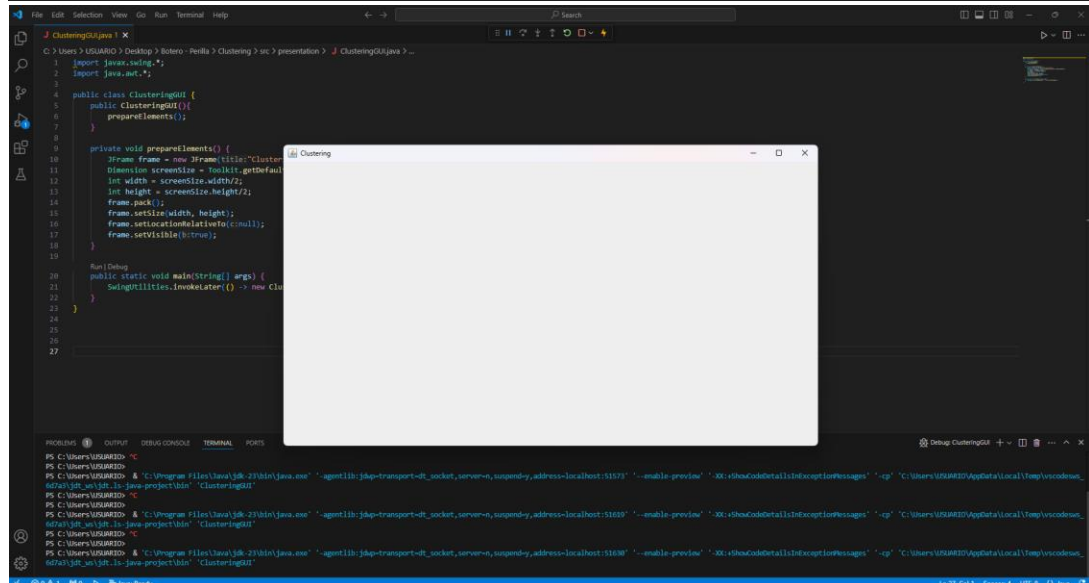
Se inició la codificación del método `prepareElements`, por lo que se movieron las líneas de código que se encontraban en el método `main` al método `prepareElements`, además, se codificaron las líneas que permiten que la ventana se encuentre en el centro y esta ocupe un cuarto de la pantalla

```
import javax.swing.*;
import java.awt.*;

public class ClusteringGUI {
    public ClusteringGUI(){
        prepareElements();
    }

    private void prepareElements() {
        JFrame frame = new JFrame(title:"Clustering");
        Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
        int width = screenSize.width/2;
        int height = screenSize.height/2;
        frame.pack();
        frame.setSize(width, height);
        frame.setLocationRelativeTo(c:null);
        frame.setVisible(b:true);
    }

    Run | Debug
    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> new ClusteringGUI());
    }
}
```



The screenshot shows an IDE with the `ClusteringGUI.java` file open. The code defines a `ClusteringGUI` class with a `prepareElements` method that creates a `JFrame` window titled "Clustering". The window is centered and its size is set to half the screen width and height. The `main` method uses `SwingUtilities.invokeLater` to create a new `ClusteringGUI` instance. A running window titled "Clustering" is visible in the foreground, showing a white rectangular area.

3. Traten de cerrar la ventana. ¿Termina la ejecución? ¿Qué deben hacer en consola para terminar la ejecución?

Se cerró la ventana, pero la ejecución no termina, para solucionar este problema, se debe poner en el método `prepareElements`, la siguiente línea de código:

```
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

Esto asegura que cuando el usuario cierre la ventana, la aplicación se cierre completamente. Esto es útil para finalizar la aplicación de forma ordenada.

4. Estudien en `JFrame` el método `setDefaultCloseOperation`. ¿Para qué sirve? ¿Cómo lo usarían si queremos confirmar el cierre de la aplicación? ¿Cómo lo usarían si queremos simplemente cerrar la aplicación?

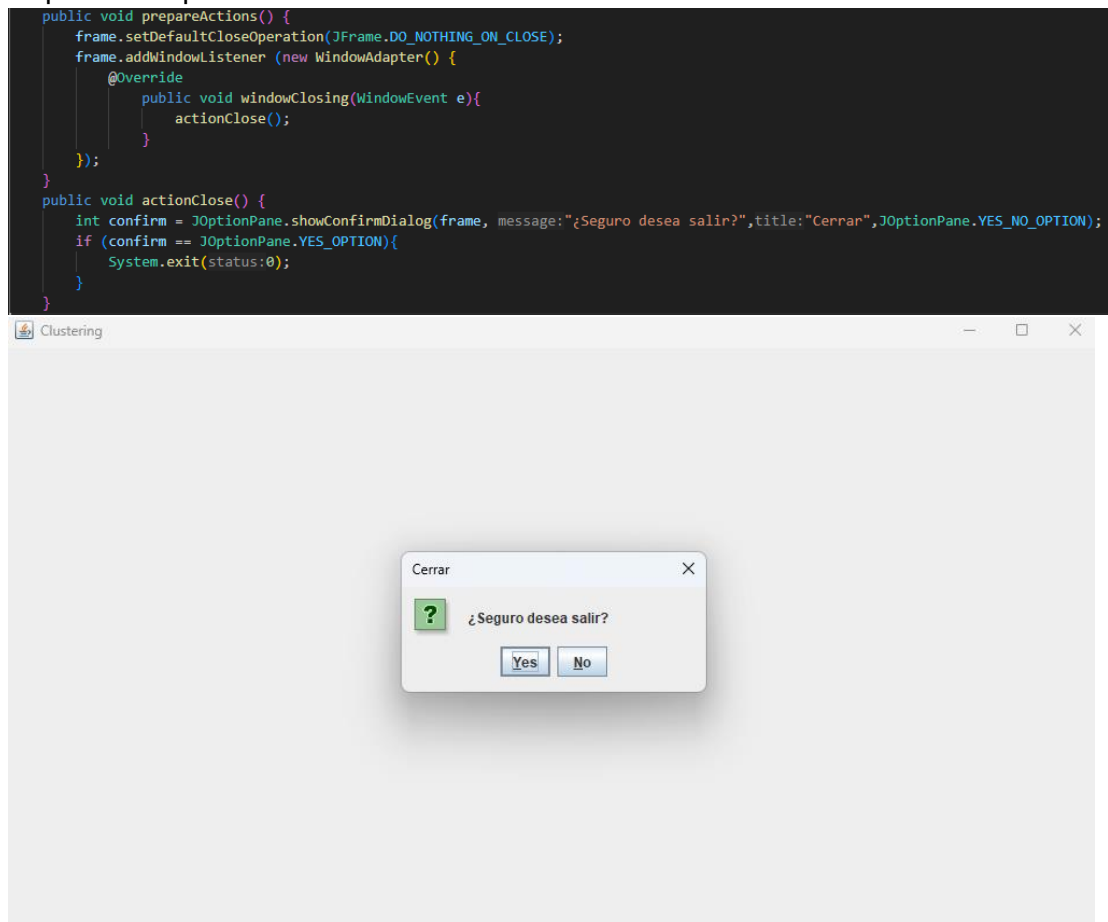
El método `setDefaultCloseOperation` en la clase `JFrame` de Java se utiliza para definir el comportamiento de la aplicación cuando el usuario cierra la ventana. Este método especifica qué debe hacer el programa al intentar cerrar la ventana, y se puede configurar con diferentes opciones para manejar este evento, estas opciones son:

- `JFrame.EXIT_ON_CLOSE`: Cierra la aplicación y termina el proceso.
- `JFrame.DISPOSE_ON_CLOSE`: Cierra la ventana pero no termina el proceso de la aplicación.
- `JFrame.HIDE_ON_CLOSE`: Oculta la ventana, pero no la destruye.
- `JFrame.DO_NOTHING_ON_CLOSE`: No hace nada, permite manejar el cierre de la ventana manualmente.

Para confirmar el cierre de la ventana antes de que la aplicación se cierre, se puede usar `JFrame.DO_NOTHING_ON_CLOSE` en conjunto con un `WindowListener` o un `WindowAdapter`. Esto permite interceptar el evento de cierre y mostrar un cuadro de confirmación.

Se usaría `JFrame.EXIT_ON_CLOSE` para cerrar la ventana y terminar el programa cuando el usuario cierre la ventana.

5. Preparen el “oyente” correspondiente al icono cerrar que le pida al usuario que confirme su selección. Para eso inicien la codificación del método `prepareActions` y el método asociado a la acción (`exit`). Ejecuten el programa y cierren el programa. Capturen las pantallas.



Ciclo 1: Ventana con menú – Salir

[En *.java y lab05.doc]

El objetivo es implementar un menú clásico para la aplicación con un final adecuado desde la opción del menú para salir. El menú debe ofrecer mínimo las siguientes opciones :Nuevo, Abrir – Salvar y Salir . Incluyan los separadores de opciones.

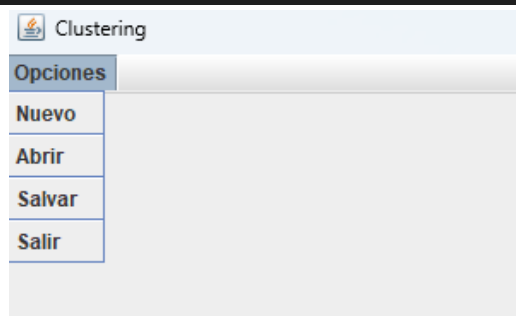
1. Expliquen los componentes visuales necesarios para este menú. ¿Cuáles serían atributos y cuáles podrían ser variables del método prepareElements? Justifique.
Los atributos necesarios, serían frame, menuBar este de tipo JMenuBar ya que es la que referencia la barra de menú de la aplicación, los atributos newItem, openItem, saveItem, exitItem, estos de tipo JMenuItem, por otro lado, habrá una variable llamada fileMenu de tipo JMenu que almacenará los JMenuItem.
2. Construya la forma del menú propuesto (prepareElements - prepareElementsMenu) . Ejecuten. Capturen la pantalla.

```
public void prepareElements() {
    frame = new JFrame(title:"Clustering");
    Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
    int width = screenSize.width/2;
    int height = screenSize.height/2;
    frame.setSize(width, height);
    frame.setLocationRelativeTo(c:null);
    frame.setVisible(b:true);
    menuBar = new JMenuBar();
    frame.setJMenuBar(menuBar);
    prepareElementsMenu();
}

public void prepareElementsMenu() {
    JMenu fileMenu = new JMenu(s:"Opciones");
    newItem = new JMenuItem(text:"Nuevo");
    openItem = new JMenuItem(text:"Abrir");
    saveItem = new JMenuItem(text:"Salvar");
    exitItem = new JMenuItem(text:"Salir");

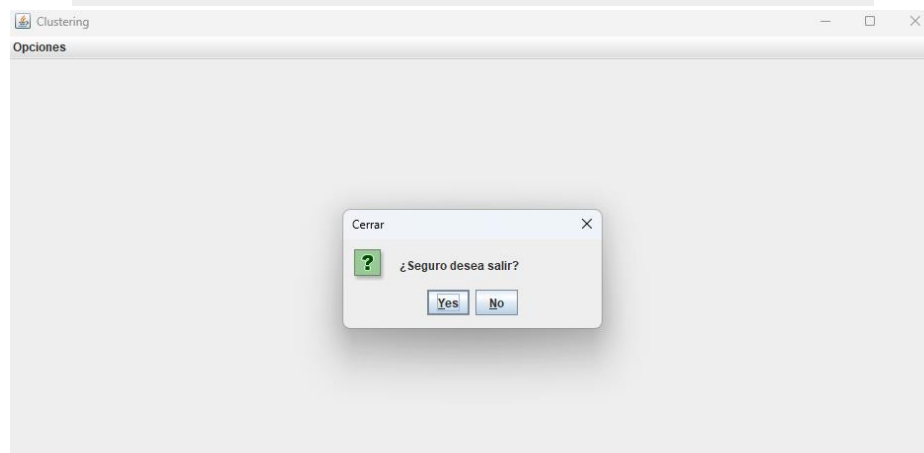
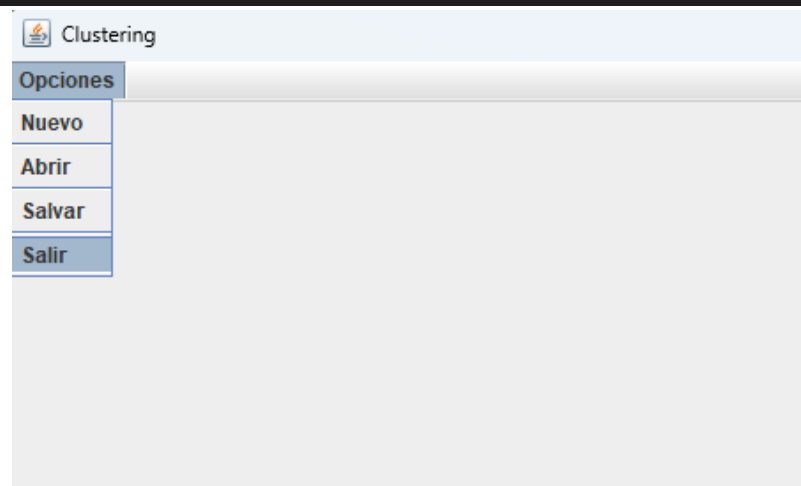
    fileMenu.add(newItem);
    fileMenu.addSeparator();
    fileMenu.add(openItem);
    fileMenu.addSeparator();
    fileMenu.add(saveItem);
    fileMenu.addSeparator();
    fileMenu.add(exitItem);

    menuBar.add(fileMenu);
}
```



3. Preparen el “oyente” correspondiente al icono cerrar con confirmación (prepareActions - prepareActionsMenu). Ejecuten el programa y salgan del programa. Capturen las pantallas.

```
public void prepareActions() {  
    frame.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);  
    frame.addWindowListener( new WindowAdapter() {  
        @Override  
        public void windowClosing(WindowEvent e){  
            actionClose();  
        }  
    });  
    prepareActionsMenu();  
}  
  
public void prepareActionsMenu() {  
    exitItem.addActionListener( new ActionListener(){  
        public void actionPerformed( ActionEvent e){  
            actionClose();  
        }  
    });  
}  
  
public void actionClose() {  
    int confirm = JOptionPane.showConfirmDialog(parentComponent:null, message:"¿Seguro desea salir?",title:"Cerrar",JOptionPane.YES_NO_OPTION);  
    if (confirm == JOptionPane.YES_OPTION){  
        System.exit(status:0);  
    }  
}
```



Ciclo 2: Salvar y abrir

[En *.java y lab05.doc]

El objetivo es preparar la interfaz para las funciones de persistencia

1. Detalle el componente JFileChooser especialmente los métodos : JFileChooser, showOpenDialog, showSaveDialog, getSelectedFile.

El JFileChooser es un componente en Java que permite a los usuarios seleccionar archivos o directorios de su sistema de archivos mediante una interfaz gráfica.

El constructor JFileChooser se utiliza para crear una nueva instancia de JFileChooser.

Existen diferentes formas de inicializar un JFileChooser:

- **JFileChooser()**: Crea un JFileChooser sin especificar un directorio inicial. El directorio inicial será el directorio de trabajo predeterminado del sistema, es decir, la carpeta donde se ejecuta el programa, o la carpeta predeterminada del sistema operativo.
- **JFileChooser(File currentDirectory)**: Crea un JFileChooser inicializado en el directorio proporcionado por el parámetro currentDirectory.
- **JFileChooser(String currentDirectoryPath)**: Similar al anterior, pero se pasa una ruta de directorio como cadena de texto.

El método showOpenDialog se utiliza para mostrar un cuadro de diálogo para abrir un archivo. Este cuadro de diálogo permite al usuario navegar en su sistema de archivos y seleccionar un archivo para abrir.

El método showSaveDialog se utiliza para mostrar un cuadro de diálogo para guardar un archivo. A diferencia de showOpenDialog, el cuadro de diálogo permite seleccionar un archivo para guardarlo, y si el archivo no existe, el usuario puede crear uno nuevo.

El método getSelectedFile obtiene el archivo seleccionado en el cuadro de diálogo después de que el usuario haya realizado una acción (abrir o guardar). Este método devuelve un objeto de tipo File que representa el archivo que el usuario ha seleccionado.

2. Implementen parcialmente los elementos necesarios para salvar y abrir. Al seleccionar los archivos indique que las funcionalidades están en construcción detallando la acción y el nombre del archivo seleccionado.

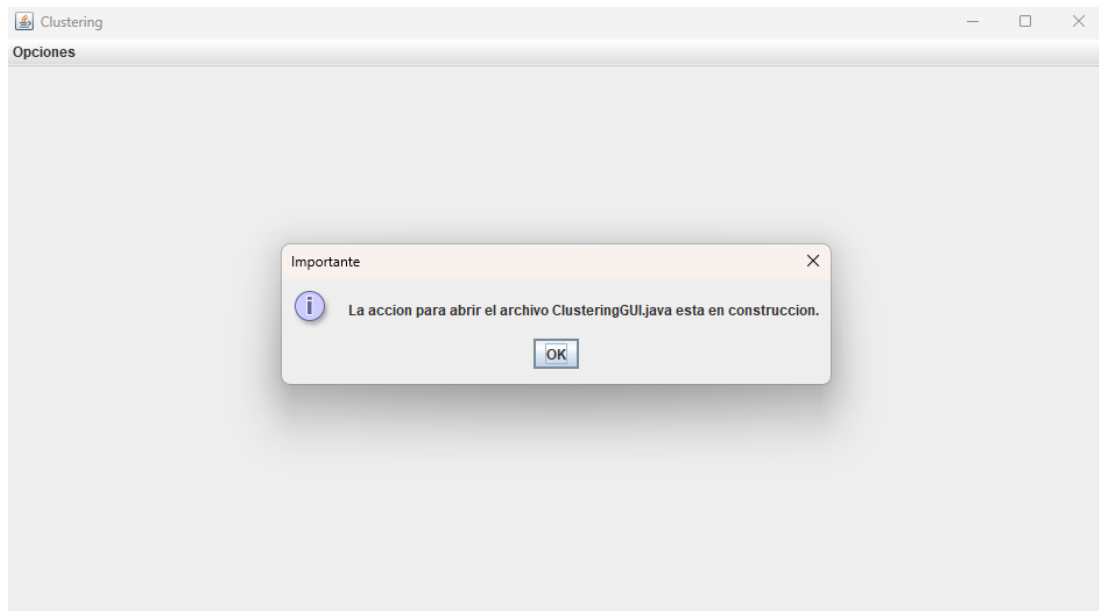
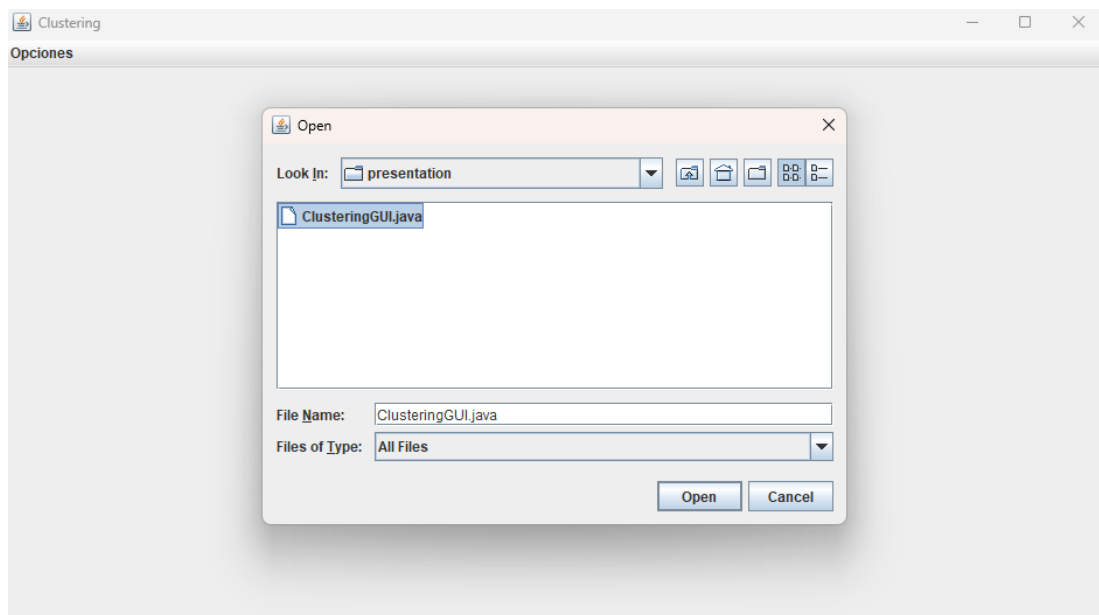
```
public void prepareActionsMenu() {
    exitItem.addActionListener( new ActionListener(){
        public void actionPerformed( ActionEvent e){
            actionClose();
        }
    });
    openItem.addActionListener( new ActionListener(){
        public void actionPerformed( ActionEvent e ){
            actionOpen();
        }
    });
    saveItem.addActionListener( new ActionListener(){
        public void actionPerformed( ActionEvent e ){
            actionSave();
        }
    });
}

public void actionOpen() {
    File file = null;
    int confirmation = fileSelector.showOpenDialog(openItem);
    if (confirmation == JFileChooser.APPROVE_OPTION) {
        file = fileSelector.getSelectedFile();
    }
    JOptionPane.showMessageDialog(openItem, "La accion para abrir el archivo " + file.getName() + " esta en construccion.",title:"Importante",messageType:1);
}

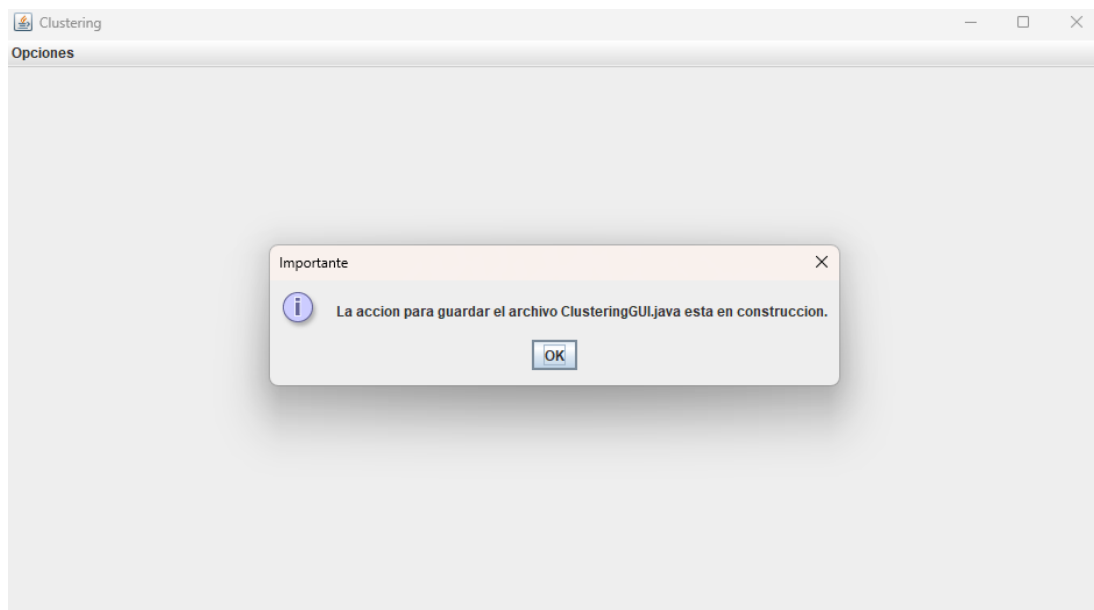
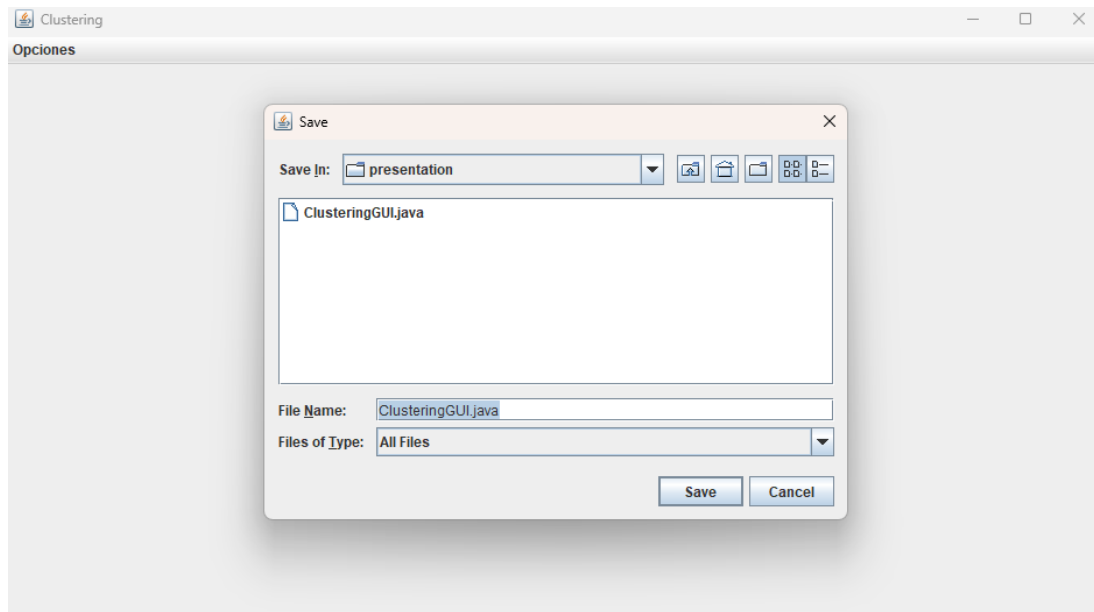
public void actionSave() {
    File file = null;
    int confirmation = fileSelector.showSaveDialog(saveItem);
    if (confirmation == JFileChooser.APPROVE_OPTION) {
        file = fileSelector.getSelectedFile();
    }
    JOptionPane.showMessageDialog(saveItem, "La accion para guardar el archivo " + file.getName() + " esta en construccion.",title:"Importante",messageType:1);
}
```


3. Ejecuten las dos opciones y capturen las pantallas más significativas.

Para la opción de abrir



Para la opción de guardar

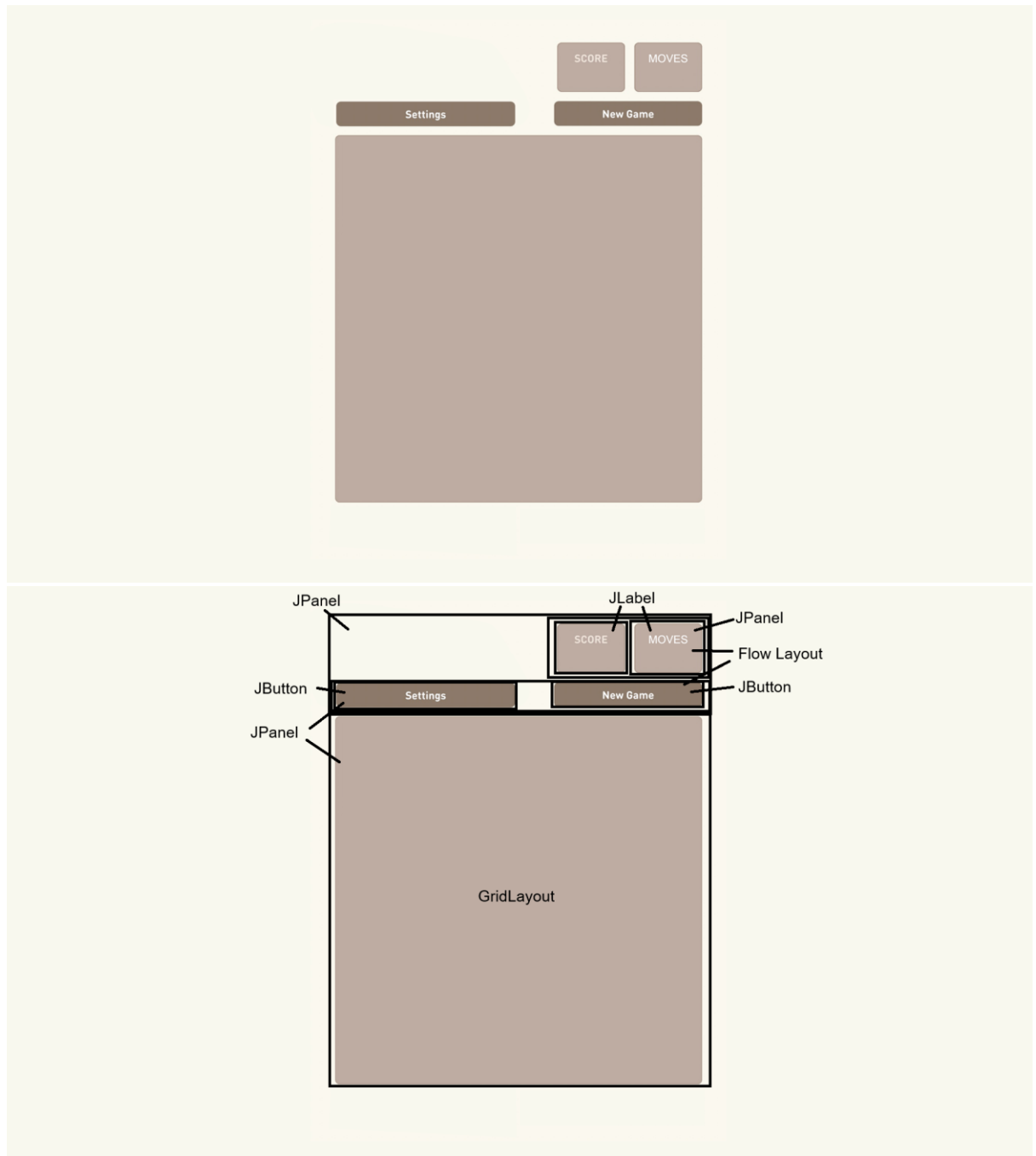


Ciclo 3: Forma de la ventana principal

[En *.java y lab05.doc]

El objetivo es codificar el diseño de la ventana principal (todos los elementos de primer nivel)

1. Presenten el bosquejo del diseño de interfaz con todos los componentes necesarios.

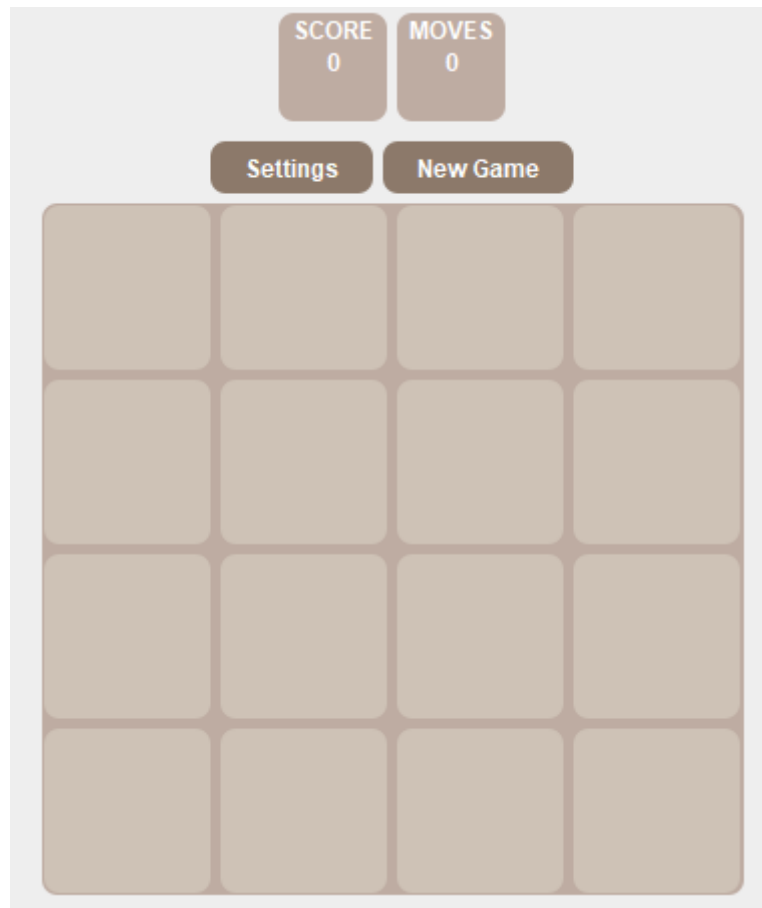


- Continúe con la implementación definiendo los atributos necesarios y extendiendo el método `prepareElements()`. Para la zona del tablero defina un método `prepareElementsBoard()` y un método `refresh()` que actualiza la vista del tablero considerando, por ahora, el tablero inicial por omisión. Este método lo vamos a implementar realmente en otros ciclos.

```
public void prepareElements() { 1 usage
    frame = new JFrame( title: "Clustering");
    frame.setLayout(new BorderLayout());
    frame.setBackground(new Color( r: 248, g: 248, b: 238));
    Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
    int width = screenSize.width/2;
    int height = screenSize.height/2;
    frame.setSize(width, height);
    frame.setLocationRelativeTo(null);
    frame.setVisible(true);
    frame.setResizable(false);
    menuBar = new JMenuBar();
    frame.setJMenuBar(menuBar);
    prepareElementsMenu();
    prepareElementsBoard();
    JPanel eastPanel = new JPanel();
    JPanel westPanel = new JPanel();
    int panelWidth = (int) (frame.getWidth() * 0.31);
    eastPanel.setPreferredSize(new Dimension(panelWidth, frame.getHeight()));
    westPanel.setPreferredSize(new Dimension(panelWidth, frame.getHeight()));
    frame.add(BorderLayout.EAST, eastPanel);
    frame.add(BorderLayout.WEST, westPanel);
    JPanel southPanel = new JPanel();
    int panelHeight = frame.getHeight() / 16;
    southPanel.setPreferredSize(new Dimension(frame.getWidth(), panelHeight));
    frame.add(BorderLayout.SOUTH, southPanel);
    fileSelector = new JFileChooser();
}
```

```
public void prepareElementsBoard() { 2 usages
    board = new JPanel() {
        @Override
        protected void paintComponent(Graphics g) {
            super.paintComponent(g);
            Graphics2D g2d = (Graphics2D) g;
            g.setColor(new Color( r: 190, g: 172, b: 162));
            g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_ON);
            g2d.fill(new RoundRectangle2D.Float( x: 0, y: 0, getWidth(), getHeight(), arcw: 15, arch: 15));
        }
    };
    Color[][] tiles = clustering.getTilesColors();
    int row = tiles.length, column = tiles[0].length;
    board.setLayout(new GridLayout(row, column, hgap: 5, vgap: 5));
    for (int i = 0; i < row; i++) {
        for (int j = 0; j < column; j++) {
            TileButton tile = getFileButton(tiles, i, j);
            board.add(tile);
        }
    }
    frame.add(BorderLayout.CENTER, board);
}
```

3. Ejecuten y capturen esta pantalla.



Ciclo 4: Cambiar colores

[En *.java y lab05.doc]

El objetivo es implementar este caso de uso.

1. Expliquen los elementos (vista – controlador) necesarios para implementar este caso de uso.

Cada baldosa del tablero es un JButton, el cual cuando se oprima llame al metodo changeColor() de clustering y se haga un cambio de color a la baldosa correspondiente.

2. Detalle el comportamiento de JColorChooser especialmente el método estático showDialog

Se presenta un cuadro de diálogo similar a JOptionPane que permite al usuario seleccionar un color de una paleta de colores. El color seleccionado puede ser almacenado en una variable para su uso en el programa.

3. Implementen los componentes necesarios para cambiar el color de las fichas.

```
TileButton tile = new TileButton(initialColor);
if (Objects.equals(initialColor, new Color( r: 206, g: 194, b: 182))) {
    tile.setEnabled(false);
}
tile.addActionListener( new ActionListener(){
    public void actionPerformed( ActionEvent e){
        Color color = JColorChooser.showDialog(frame, title: "Escoge un color:", initialColor);
        if (color != null) {
            tile.changeColor(color);
        }
        frame.requestFocusInWindow();
    }
});
```

```
public class TileButton extends JButton { 5 usages
    private Color color; 3 usages

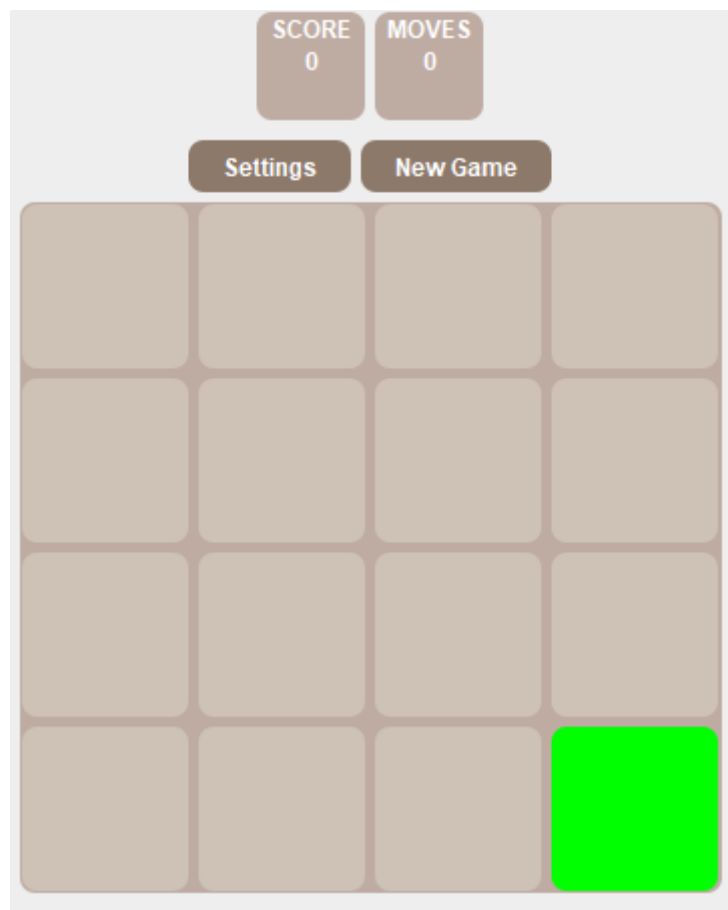
    public TileButton(Color color) { 1 usage
        this.color = color;
        this.setOpaque(false);
        this.setContentAreaFilled(false);
        this.setBorderPainted(false);
    }

    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        Graphics2D g2d = (Graphics2D) g;
        g2d.setColor(color);
        g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_ON);
        g2d.fill(new RoundRectangle2D.Float(x: 0, y: 0, getWidth(), getHeight(), arcw: 15, arch: 15));
    }

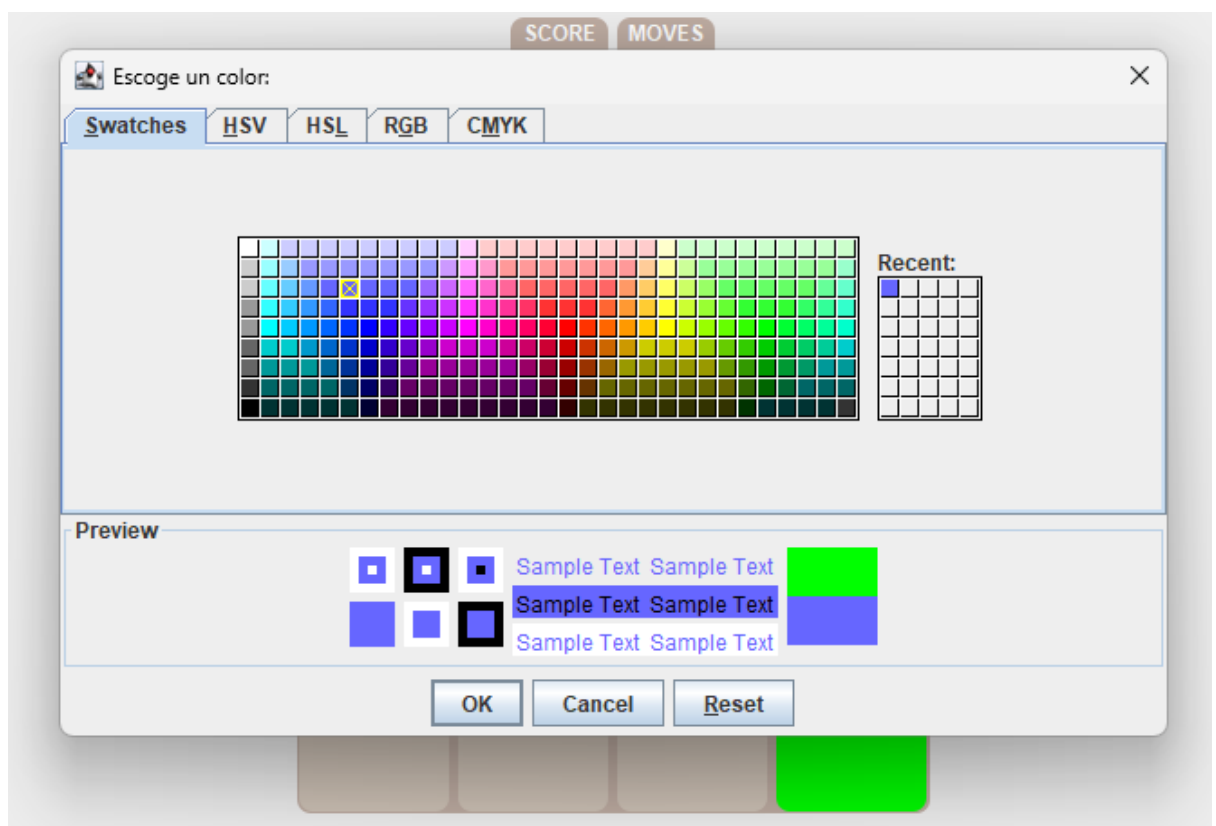
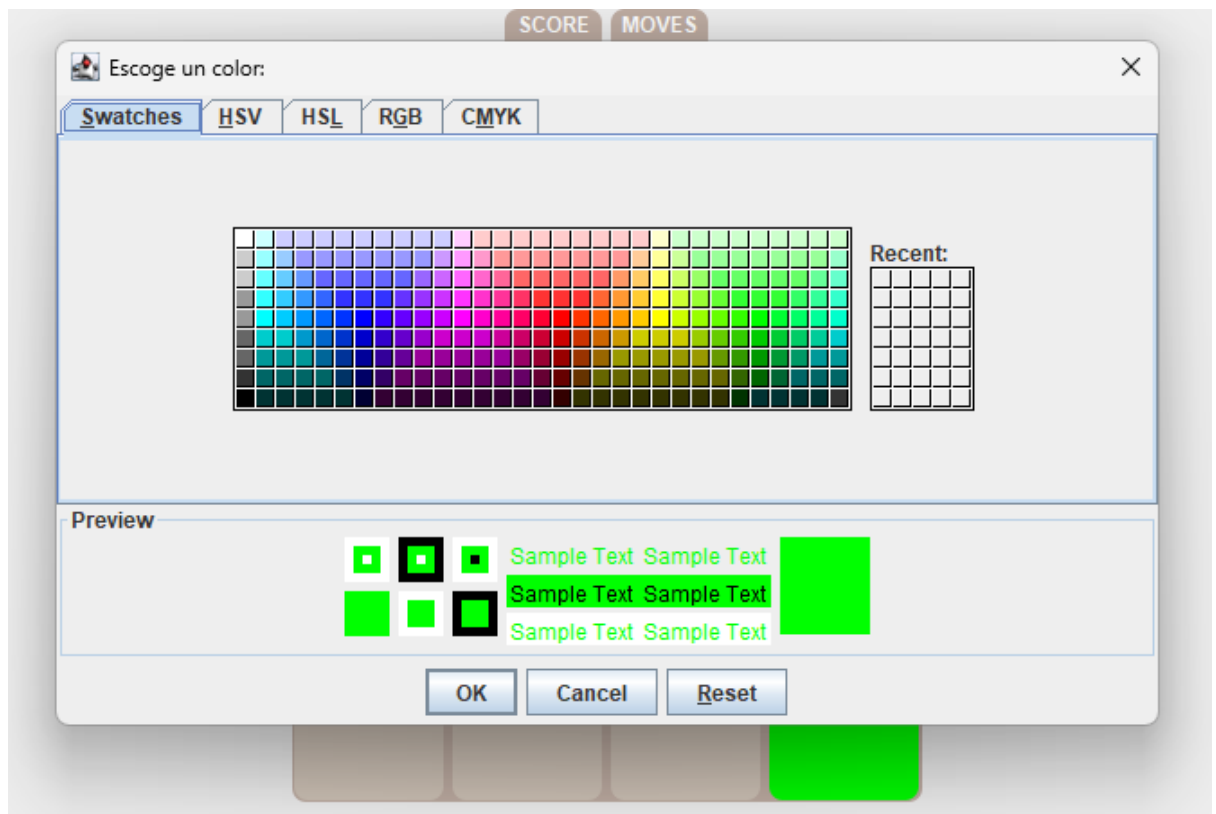
    public void changeColor(Color newColor) { 2 usages
        this.color = newColor;
        repaint();
    }
}
```

4. Ejecuten el caso de uso y capturen las pantallas más significativas.

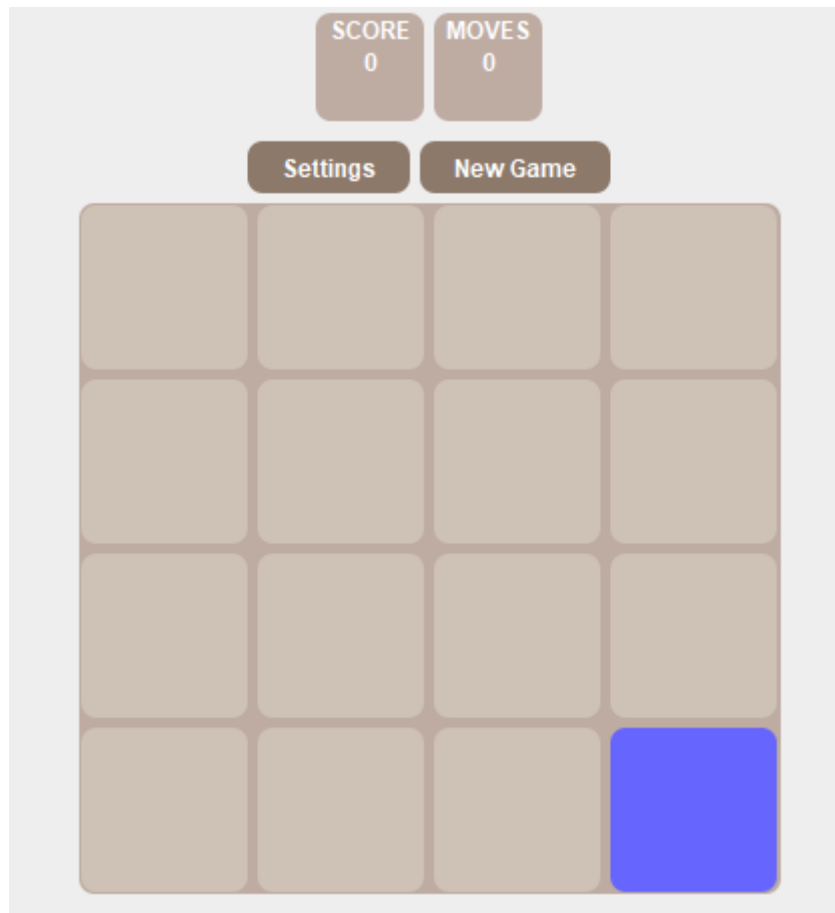
Se asume que en la fila 4 con columna 4 del tablero existe una baldosa de color verde



En el caso de oprimir el botón se abre la ventana de JColorChooser



En el caso de seleccionar un color y presionar el botón de Ok, este color se guardará en la variable color, y se repinta el rectángulo en la subclase TileButton de JButton que se implementó







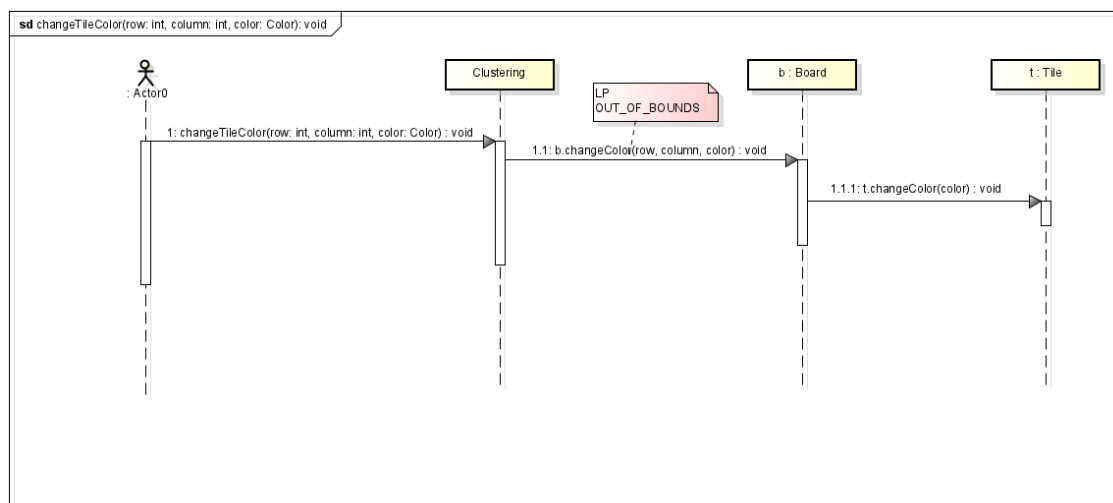
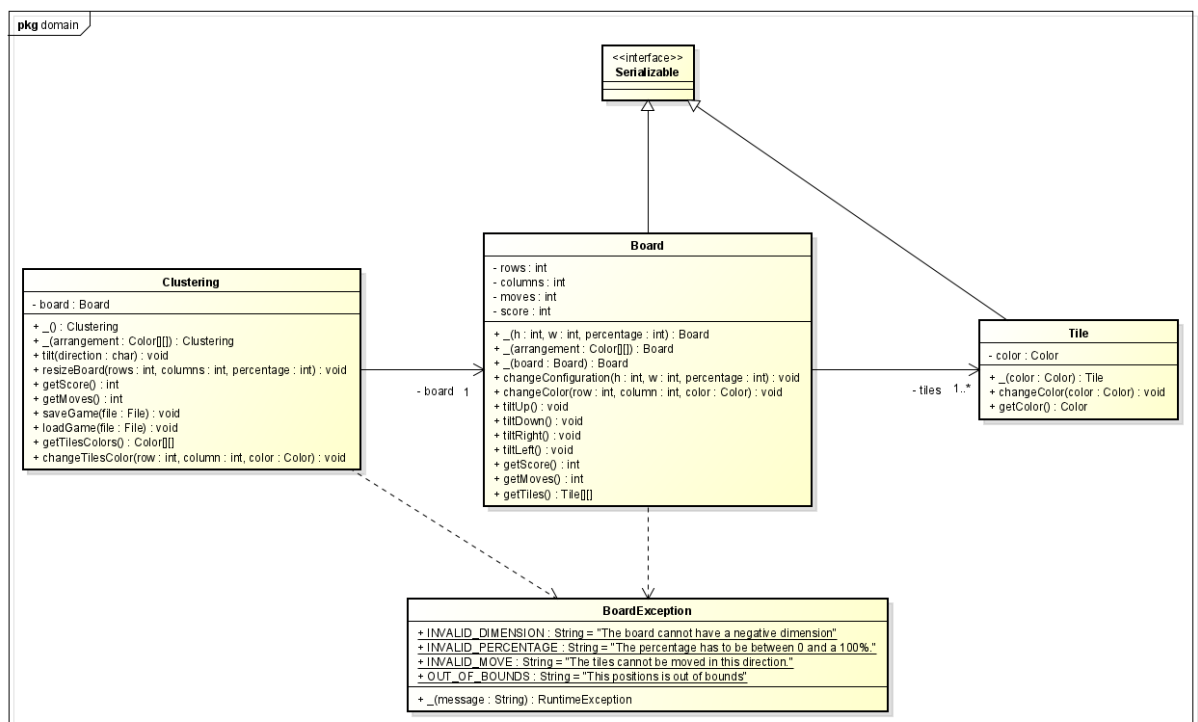
Ciclo 5: Modelo Clustering

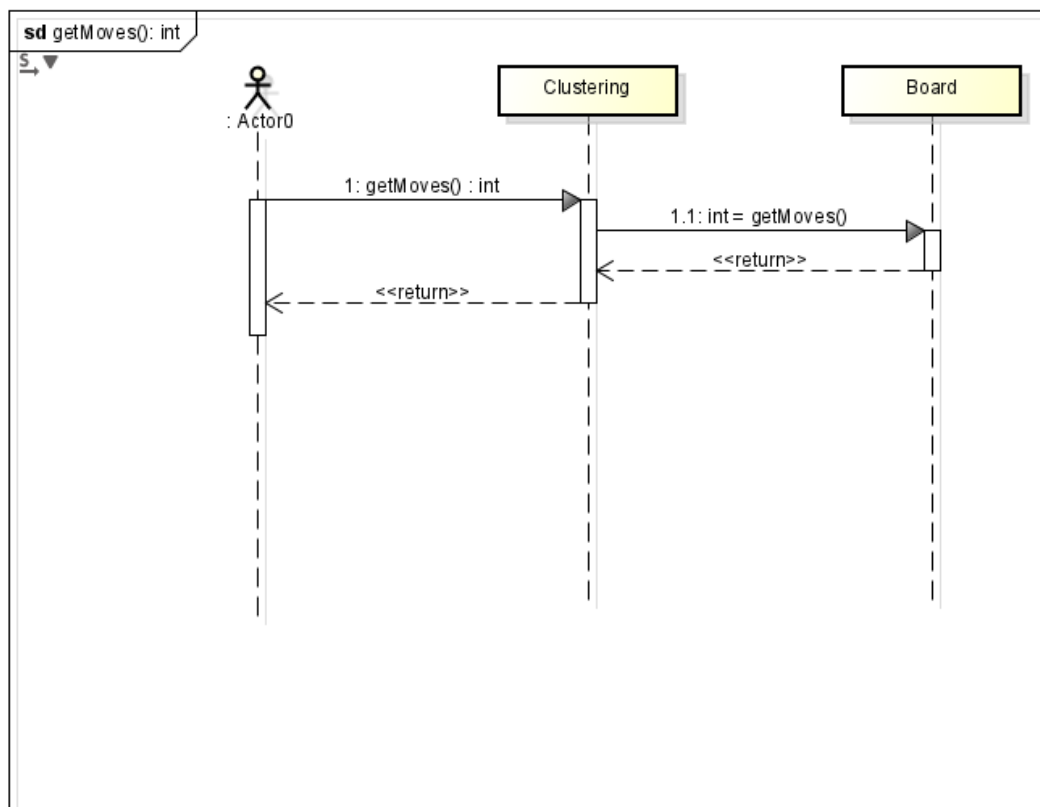
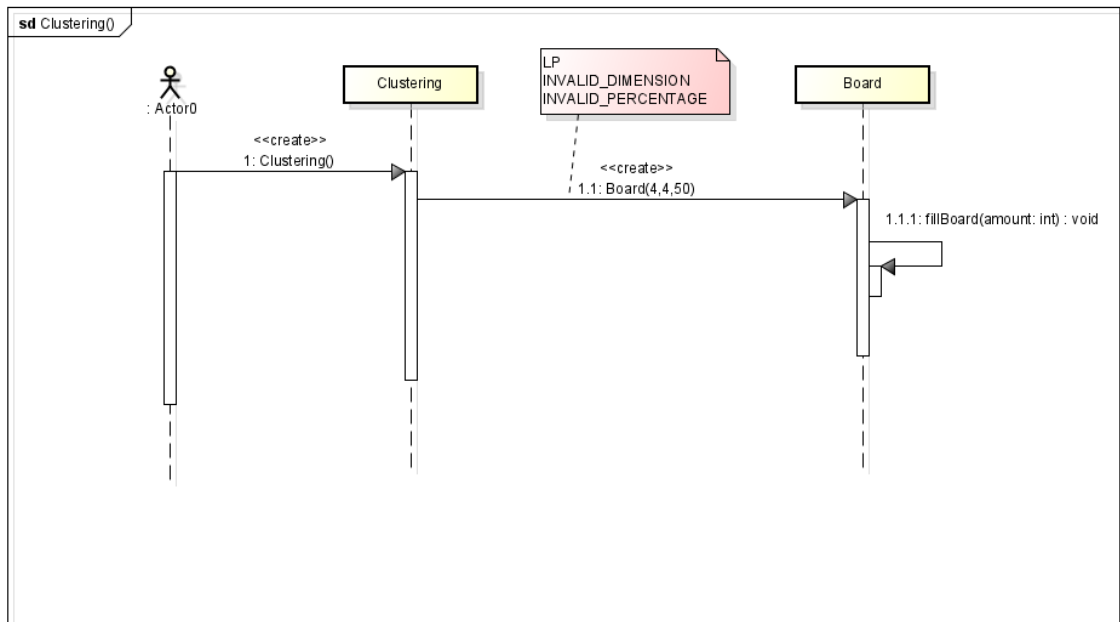
[En *.java y lab05.doc]

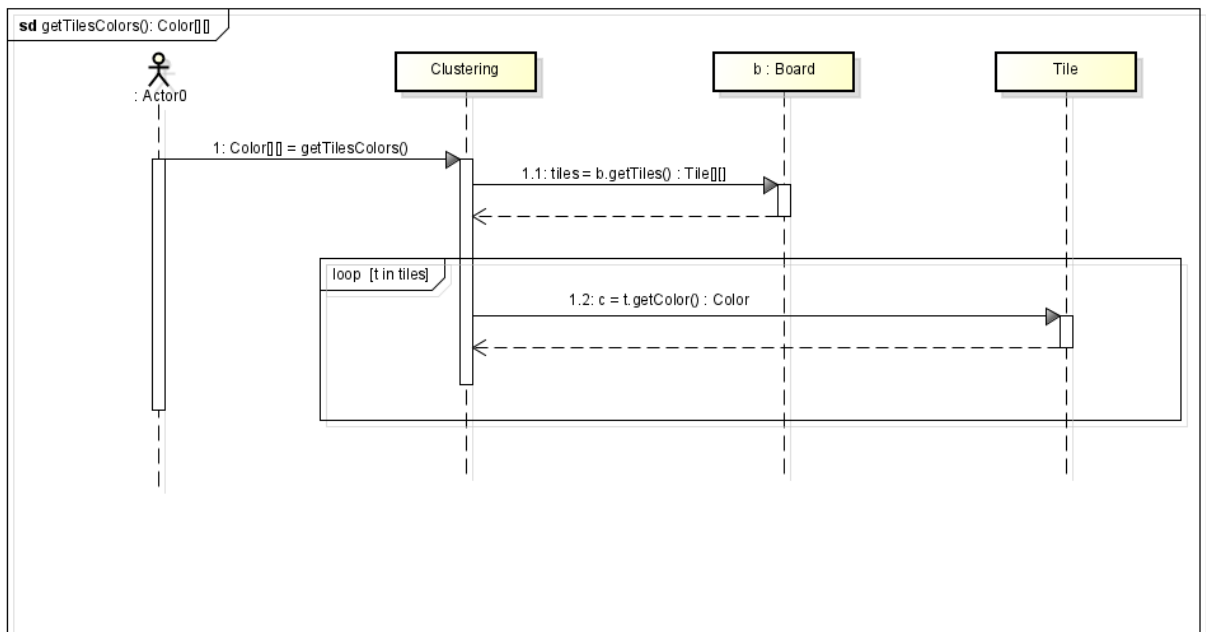
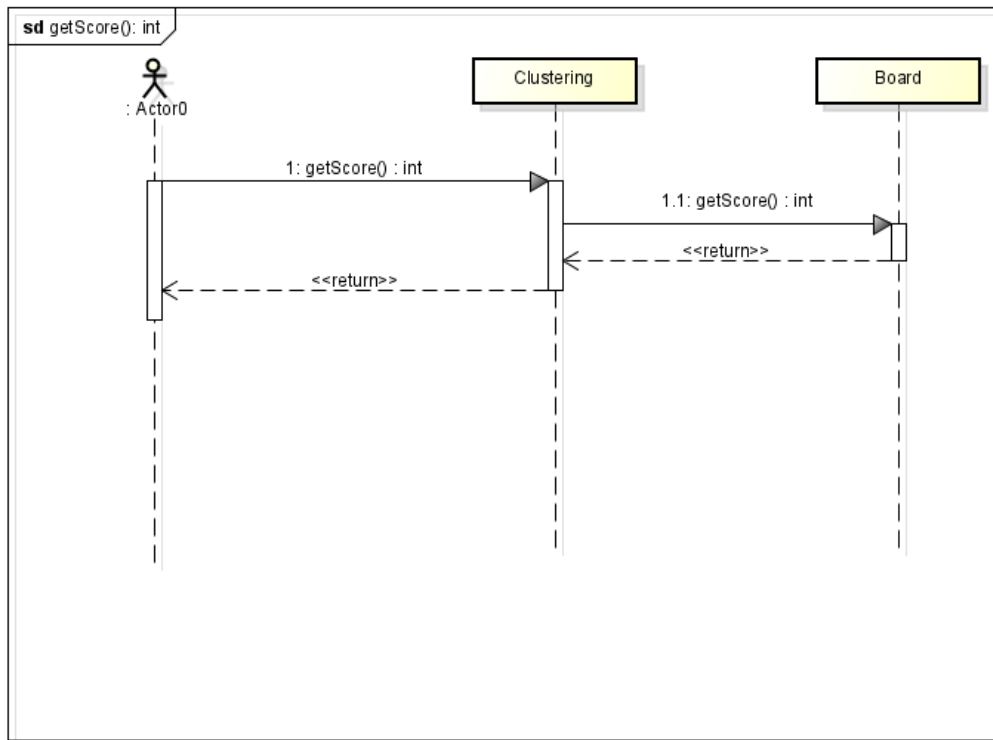
El objetivo es implementar la capa de dominio para Clustering

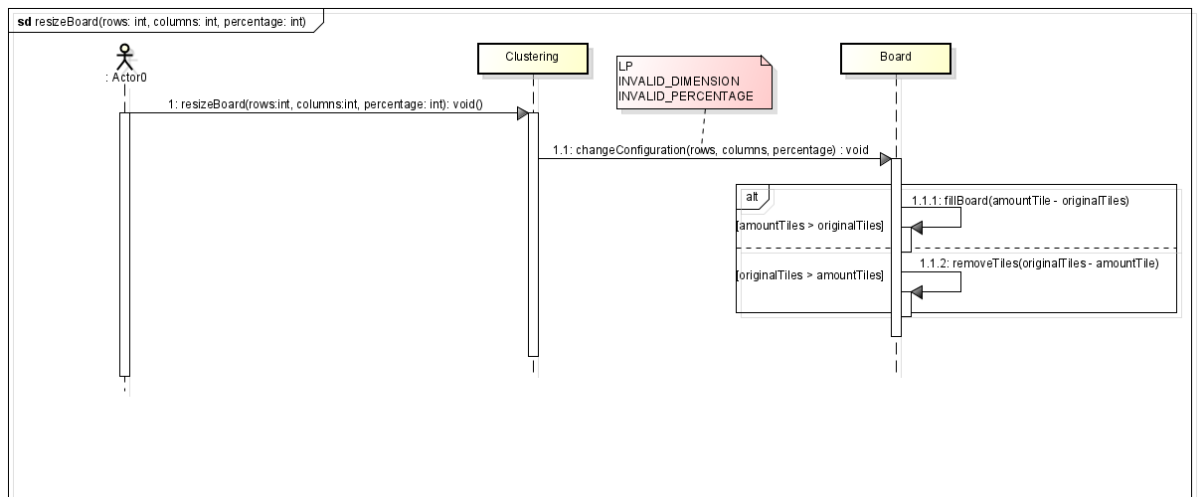
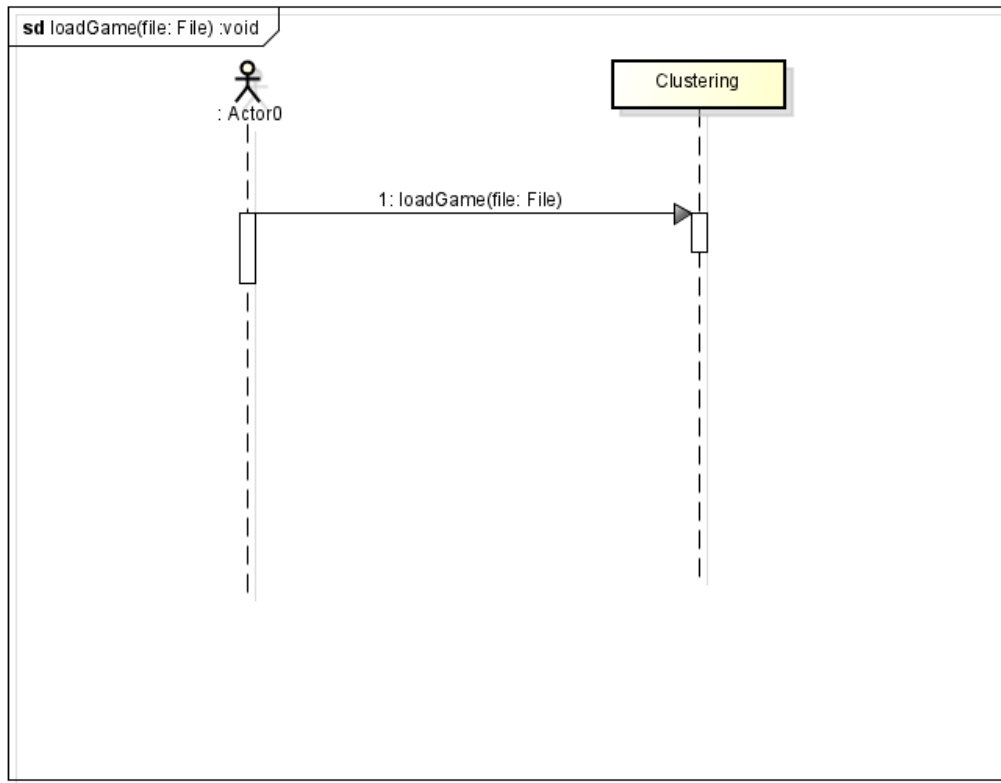
1. Construya los métodos básicos del juego (No olvide MDD y TDD)

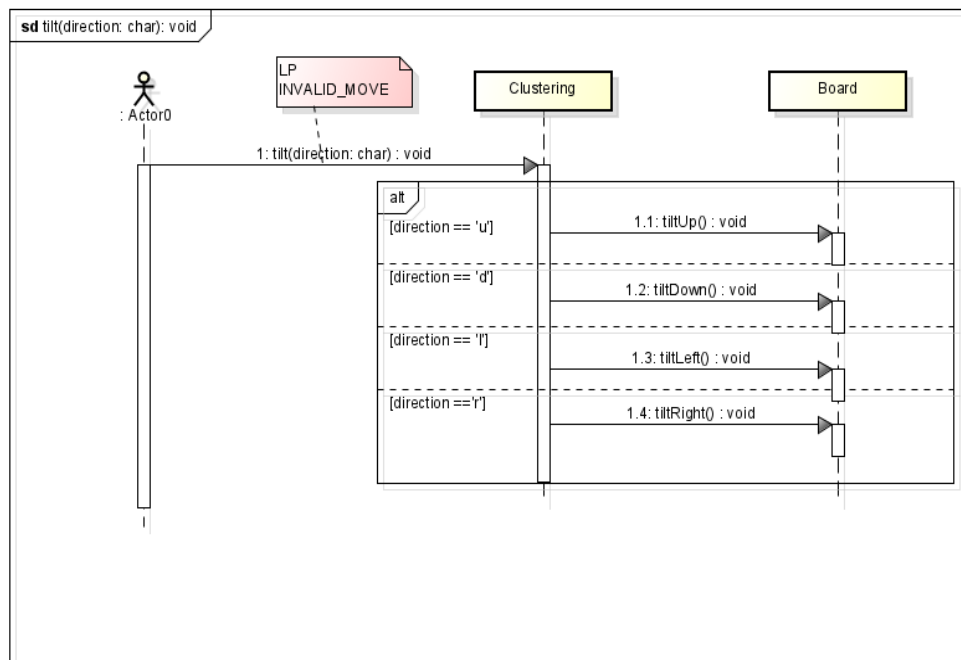
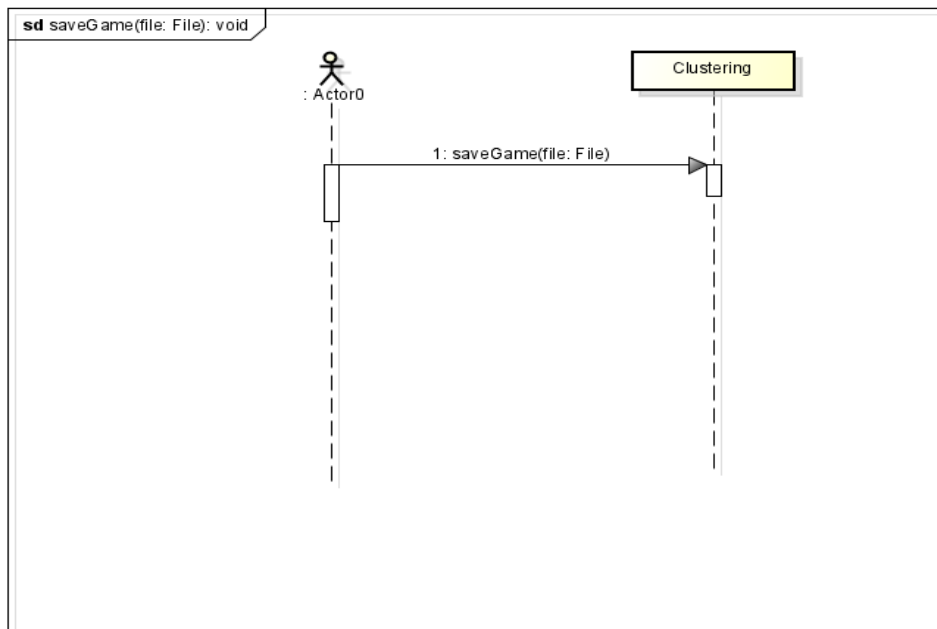
 Board	11/15/2024 8:36 AM	Java Source File	8 KB
 BoardException	11/14/2024 9:59 PM	Java Source File	1 KB
 Clustering	11/15/2024 9:08 AM	Java Source File	3 KB
 Tile	11/15/2024 8:38 AM	Java Source File	1 KB











2. Ejecuten las pruebas y capturen el resultado.

Test Name	Duration
ClusteringTest	61 ms
shouldReturnTheRightMoves	6 ms
shouldReturnTheRightScore	1 ms
shouldTiltRight	1 ms
shouldTiltUp	1 ms
shouldLoadSavedGame	49 ms
shouldTiltDown	0 ms
shouldTiltLeft	1 ms
shouldChangeTileColor	0 ms
shouldIncreaseBoardSize	2 ms

✓ Tests passed: 9 of 9 tests - 61 ms

C:\Users\santi\.jdk\openjdk-23.0.1\bin\java.exe ...

Process finished with exit code 0

Ciclo 6: Jugar

[En *.java y lab05.doc]

El objetivo es implementar el caso de uso jugar.

1. Adicione a la capa de presentación el atributo correspondiente al modelo.

```
public class ClusteringGUI {  
    private static Clustering clustering; 16 usages  
    private JFrame frame; 34 usages  
    private JMenuBar menuBar; 3 usages  
    private JMenuItem newItem; 3 usages  
    private JMenuItem openItem; 4 usages  
    private JMenuItem saveItem; 4 usages  
    private JMenuItem exitItem; 3 usages  
    private JFileChooser fileSelector; 6 usages  
    private JButton settings; 7 usages  
    private JButton newGame; 7 usages  
    private JLabel scoreNumber; 5 usages  
    private JLabel movesNumber; 5 usages  
    private JPanel board; 6 usages
```

2. Perfeccionen el método refresh() considerando la información del modelo de dominio.

```
public void refresh() { 7 usages  
    Component[] components = board.getComponents();  
    Color[][] tiles = clustering.getTilesColors();  
    int column = tiles[0].length;  
    int componentIndex = 0;  
    for (Color[] tile : tiles) {  
        for (int j = 0; j < column; j++) {  
            Color color = tile[j] != null ? tile[j] : new Color(r: 206, g: 194, b: 182);  
            if (components[componentIndex] instanceof TileButton button) {  
                button.setColor(color);  
                button.setEnabled(!Objects.equals(color, new Color(r: 206, g: 194, b: 182)));  
            }  
            componentIndex++;  
        }  
    }  
    scoreNumber.setText(String.valueOf(clustering.getScore()));  
    movesNumber.setText(String.valueOf(clustering.getMoves()));  
}
```

```
private void refreshBoard() { 3 usages  
    frame.remove(board);  
    prepareElementsBoard();  
    frame.revalidate();  
    frame.repaint();  
    refresh();  
}
```

3. Expliquen los elementos necesarios para implementar este caso de uso.

Para el desarrollo del juego es necesario que se actualice la vista y se repite todas las baldosas en cada lado, por ende, se debe iterar sobre cada TileButton (La subclase de JButton que de implemento) donde cambie el color y la actividad del botón dependiendo los tiles del tablero. Adicionalmente se implementó el método refreshBoard(), éste permite actualizar el tablero una vez se haga el cambio de configuración o se cargue una partida desde abrir ya que permite cambiar las filas y columnas del tablero en la vista.

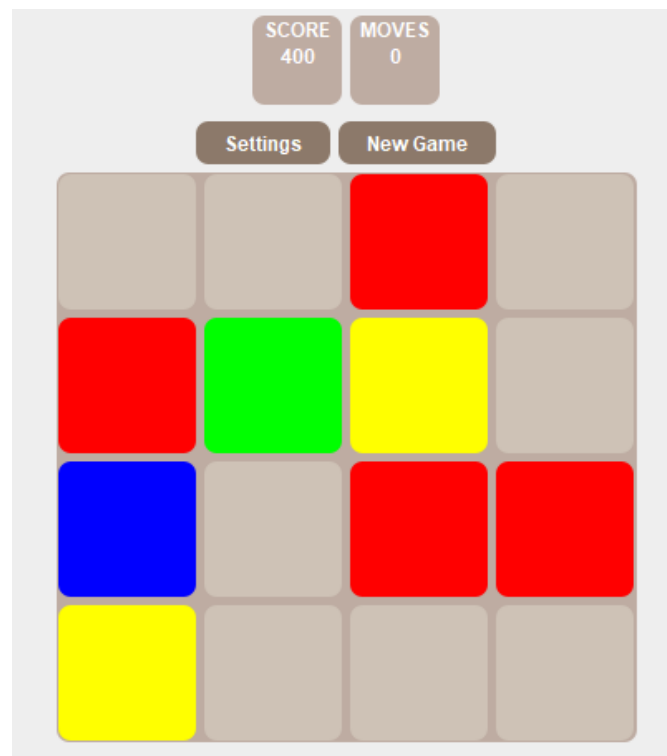
4. Implementen los componentes necesarios para jugar. ¿Cuántos oyentes necesitan? ¿Por qué?

```
frame.addKeyListener(new KeyListener() {  
    @Override  
    public void keyPressed(KeyEvent e) {  
        int keyCode = e.getKeyCode();  
        switch (keyCode) {  
            case KeyEvent.VK_UP:  
                clustering.tilt(direction: 'u');  
                refresh();  
                break;  
            case KeyEvent.VK_DOWN:  
                clustering.tilt(direction: 'd');  
                refresh();  
                break;  
            case KeyEvent.VK_LEFT:  
                clustering.tilt(direction: 'l');  
                refresh();  
                break;  
            case KeyEvent.VK_RIGHT:  
                clustering.tilt(direction: 'r');  
                refresh();  
                break;  
        }  
    }  
});  
  
@Override  
public void keyTyped(KeyEvent e) {  
}  
  
@Override  
public void keyReleased(KeyEvent e) {  
}  
});  
frame.setFocusable(true);  
frame.setFocusTraversalKeysEnabled(false);  
}
```

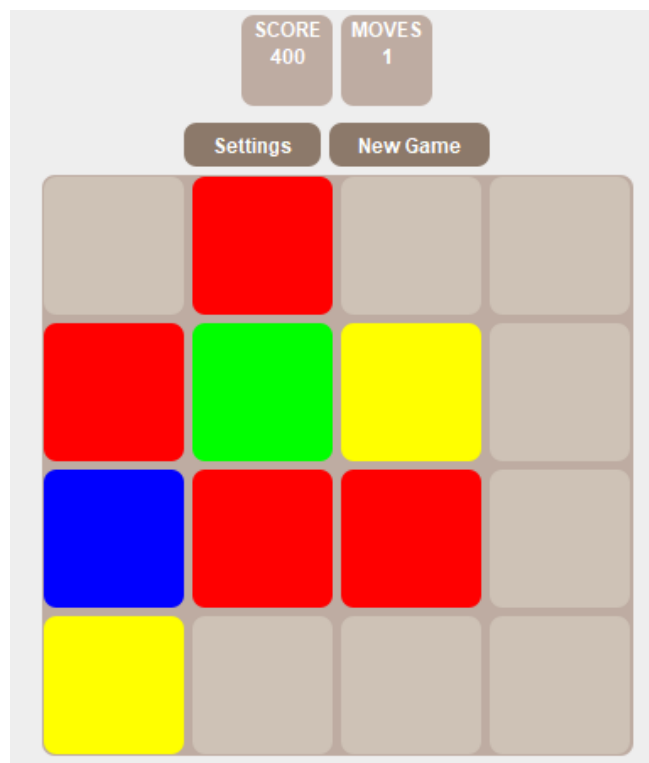
Únicamente se hizo uso de un oyente sobre el frame, debido a que la manera como el jugador interactúa con las fichas del tablero es mediante las flechas del teclado, cada vez que se presione una tecla va a llamar al método tilt en clustering con su dirección correspondiente.

5. Ejecuten el caso de uso y capturen las pantallas más significativas.

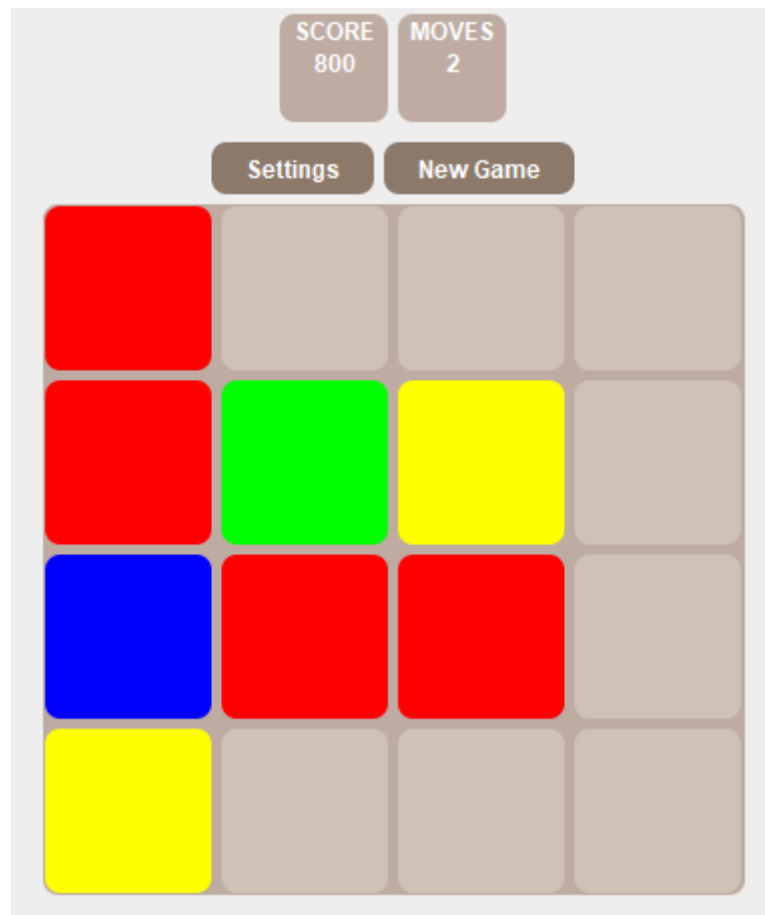
Se va a ejecutar el main de ClusteringGUI, creando un tablero arbitrario:



Esta pantalla muestra el puntaje correcto y los movimientos, entonces se desea realizar un ladoeo a la izquierda presionando la flecha izquierda:



Se evidencia que se actualiza los movimientos realizados, por ende, se desea realizar otro lado hacia la izquierda:



Se puede evidenciar que tanto el puntaje como los movimientos se actualizan correctamente.

Ciclo 7: Reiniciar

[En *.java y lab05.doc]

El objetivo es implementar este caso de uso.

1. Expliquen los elementos a usar para implementar este caso de uso.

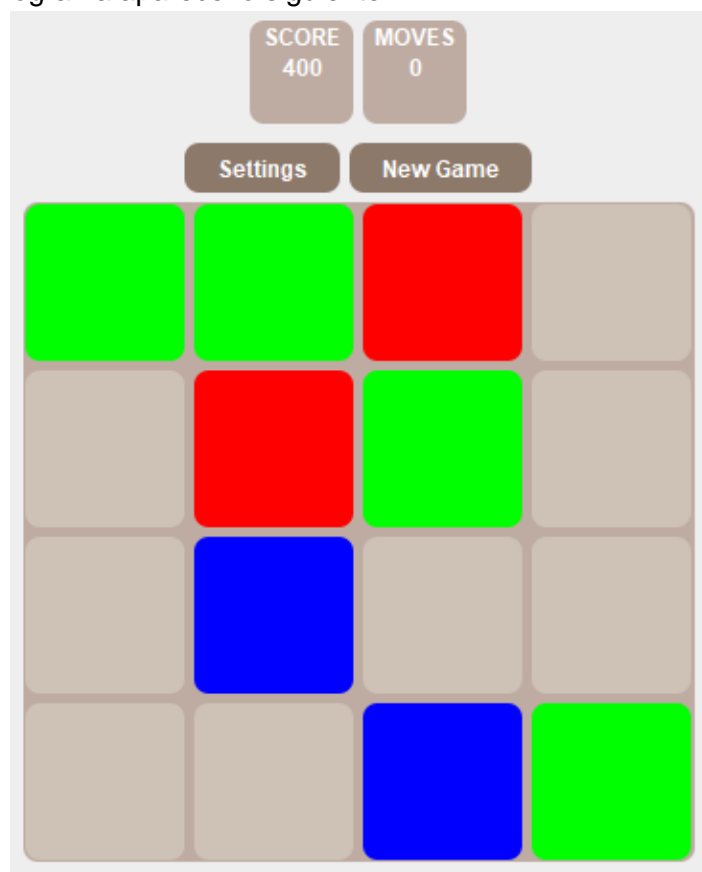
Para implementar este caso de uso se debe crear un nuevo JButton llamado newGame, el cual tendrá un ActionListener con el fin de que al presionarlo se le pregunte al usuario si desea reiniciar el juego, si oprime ok se modifica todo el tablero, en caso de que oprima cancel, no se realiza ningún cambio.

2. Implementen los elementos necesarios para reiniciar

```
newGame.addActionListener( new ActionListener(){
    public void actionPerformed( ActionEvent e ){
        actionRestart();
    }
});
}
public void actionRestart(){
    int ans = JOptionPane.showConfirmDialog(parentComponent:null, message:"¿Desea reiniciar el tablero?", title:"Reiniciar tablero", JOptionPane.OK_CANCEL_OPTION);
    if (ans == JOptionPane.OK_OPTION){
        restart();
    }
}
public void restart(){
    clustering = new Clustering(4,4,50);
    refresh();
}
```

3. Ejecuten el caso de uso y capture las pantallas más significativas.

Al ejecutar el programa aparece lo siguiente



Oprimimos New Game



Al oprimir ok, sucede lo siguiente



Se reinicio el tablero, como se esperaba, debido a lo que se implementó anteriormente.

Ciclo 8: Cambiar el tamaño

[En *.java y lab05.doc]

El objetivo es implementar este caso de uso.

1. Expliquen los elementos a usar para implementar este caso de uso.

Para implementar este caso de uso, se creó un JButton llamado settings, el cual tendrá un ActionListener con el fin de que al presionarlo, se abra otra ventana, en donde uno de los botones es cambiar las dimensiones del tablero, el usuario presiona en este botón y aparecerá otra ventana, la cual le permite ingresar la cantidad de filas y columnas que desee, finalmente, este presiona en aplicar y el tablero se debe modificar.

2. Implementen los elementos necesarios para cambiar el tamaño del juego.

```
private void openSettingsDialog() { 1 usage
    JDialog settingsDialog = new JDialog(frame, title: "Configuración", modal: true);
    settingsDialog.setLayout(new FlowLayout());
    settingsDialog.setSize( width: 300, height: 150);
    settingsDialog.setLocationRelativeTo(frame);

    JButton changeDimensionsButton = new JButton( text: "Cambiar Dimensiones");
    changeDimensionsButton.addActionListener(e -> {
        openChangeDimensionsDialog();
        settingsDialog.dispose();
    });

    settingsDialog.add(changeDimensionsButton);

    settingsDialog.setVisible(true);
}

private void openChangeDimensionsDialog() { 1 usage
    JDialog dimensionsDialog = new JDialog(frame, title: "Cambiar Dimensiones del Tablero", modal: true);
    dimensionsDialog.setLayout(new GridLayout( rows: 3, cols: 2, hgap: 10, vgap: 10));
    dimensionsDialog.setSize( width: 300, height: 150);
    dimensionsDialog.setLocationRelativeTo(frame);

    JLabel rowsLabel = new JLabel( text: "Número de filas:");
    JTextField rowsField = new JTextField("");
    JLabel colsLabel = new JLabel( text: "Número de columnas:");
    JTextField colsField = new JTextField("");

    JButton applyButton = new JButton( text: "Aplicar");
    applyButton.addActionListener(e -> {
        try {
            int rows = Integer.parseInt(rowsField.getText());
            int columns = Integer.parseInt(colsField.getText());
            clustering.resizeBoard(rows, columns);
            refreshBoard();
            dimensionsDialog.dispose();
        } catch (NumberFormatException ex) {
            JOptionPane.showMessageDialog(dimensionsDialog, message: "Ingrese números válidos para las dimensiones.");
        }
    });
}
```

```
    } catch (NumberFormatException ex) {  
        JOptionPane.showMessageDialog(dimensionsDialog, message: "Ingrese números válidos para las dimensiones.");  
    }  
});  
  
dimensionsDialog.add(rowsLabel);  
dimensionsDialog.add(rowsField);  
dimensionsDialog.add(colsLabel);  
dimensionsDialog.add(colsField);  
dimensionsDialog.add(new JLabel());  
dimensionsDialog.add(applyButton);  
  
dimensionsDialog.setVisible(true);  
}  
private void refreshBoard() { 1 usage  
    frame.remove(board);  
    prepareElementsBoard();  
    frame.revalidate();  
    frame.repaint();  
    refresh();  
}
```

3. Ejecuten el caso de uso y capture las pantallas más significativas.

Se oprime el botón settings, el cual se encuentra en la parte superior del tablero



Al presionar el botón aplicar, el tablero se modifica de la siguiente manera



Efectivamente, cambiaron las dimensiones de un tablero 4x4 a un tablero 8x8.

RETROSPECTIVA

1. ¿Cuál fue el tiempo total invertido en el laboratorio por cada uno de ustedes?
(Horas/Hombre)

13 horas

2. ¿Cuál es el estado actual del laboratorio? ¿Por qué?

El laboratorio está completo.

3. Considerando la práctica XP del laboratorio ¿por qué consideran que es importante?

La práctica de Refactorización Continua fue la más útil en el laboratorio, ya que nos permitió mantener el código limpio y bien estructurado a lo largo del desarrollo. Al hacer mejoras y ajustes continuamente, pudimos facilitar la comprensión del código para cada miembro del equipo.

4. ¿Cuál consideran fue su mayor logro? ¿Por qué? ¿Cuál consideran que fue su mayor problema? ¿Qué hicieron para resolverlo?

El diseño de la interfaz para que se refrescara la vista con cada lado o interacción que ocurre en el tablero.

5. ¿Qué hicieron bien como equipo? ¿Qué se comprometen a hacer para mejorar los resultados?

Trabajar juntos como equipo, distribuyendo el trabajo equitativamente y apoyándonos con las dudas que pudieran surgir. Nos comprometemos a administrar mejor el tiempo.