Trabajo Práctico Especial - POO - Diciembre 2022

Tabla de contenidos

Autores

- Pérez de Gracia, Mateo - Legajo: **63401**
- Quian Blanco, Francisco - Legajo: **63006**
- Stanfield, Theo - Legajo: **63403**

Objetivo

El objetivo del proyecto es la implementación de una aplicación de dibujo de figuras, desarollado en el lenguaje Java con interfaz visual en JavaFX. Se aplicaran los conceptos sobre la programación orientada a objetos adquiridos en la materia.

Descripción

El programa consta de una aplicación de dibujo, estilo Paint de Microsoft, el cual permite dibujar distintas figuras movibles. Dentro de estas figuras se permiten dibujar cuadrados, rectángulos, elipses y círculos. También se permite seleccionar las figuras, las cuales al estar seleccionadas cambian el color de su borde al color rojo. Una vez seleccionada una figura esta se puede eliminar.

Funcionamiento

Para la fácil separación del front-end y el back-end, estos dos se encuentran totalmente separados. Dentro del front-end se encuentran diferentes archivos, los cuales se encargarán de todo tipo de acción o experiencia con el usuario, y dentro del back-end se podrá encontrar todo el procesamiento interno que realiza el programa.

Front-end Recorriendo el front-end nos encontramos con todos los archivos tipo java encargados de la composición del "Frame". Tambien tenemos la clase "CanvasState" cuya funcionalidad es todo procesamiento que ocurra en el canvas.

Se tiene una lista de las figuras presentes en la pantalla, tal que se vuelven sencillas las actividades como eliminar y agregar distintas figuras, como también copiar el formato de las mismas.

```
private final List<FigureRender<? extends Figure>> list = new ArrayList<>();
```

Como se puede ver, se utilizó una separación tipo "VBox" para separar el "Frame" en dos, siendo el menú, el más elevado de los dos. Ahora sí, pasando a la segunda división, se utilizó una división de "BorderPane" ya que nos resultó la más fácil para situar barras de herramientas al lado izquierdo de la pantalla como también a la parte superior de la misma (el canvas se situa en el lugar restante, situado a la derecha de la pantalla). Esta asignación es muy sencilla debido a las funciones "setTop" y "setLeft" que contiene "BorderPane".

```
setTop(topBar);
setLeft(sideBar);
setRight(canvas);
```

La barra de herramientas situada a lado izquierdo de la pantalla es distribuida utilizando una "VBox" para que los elementos quedan alineados uno encima del otro. Por último la barra de herramientas situada en la parte superior de la pantalla será distribuida utilizando nuevamente una "VBox" para poder separarla en dos, la parte superior de esta conteniendo los botones de copiar, cortar y pegar, y la parte inferior de esta conteniendo los botones de rehacer y deshacer. Una vez conseguida esta separación se la distribuye con una "HBox" para que los elementos queden asignados uno al lado del otro.

Pasando a la carpeta User Interface (ui), podemos ver la separación entre los botones, los renders y la interfaz funcional "RedrawCanvas". Junto con los botones vamos a poder observar, los botones tipo toggle, que se pueden desactivar y activar, las barras de herramienta, como también la interfaz "MouseActions" que contiene todo tipo de acción realizable con el mouse. Siguiendo con los renders, estos serán los encargados de dibujar nuevamente el canvas cada vez que se realiza una acción. Dentro de "operations" se encuentra la lógica detrás del deshacer y rehacer, utilizando dos stacks de operaciones.

```
private final Stack<Operation> redoStack = new Stack<>();
private final Stack<Operation> undoStack = new Stack<>();
```

Dentro de render, podemos ver los distintos renders para cada figura, los cuales se encargan de crear la nueva figura y asignarle un estilo, a través de la clase "FigureStyle". Por ultimo tenemos "RedrawCanvas", interfaz con la funcionalidad de dibujar nuevamente el canvas con cada cambio efectuado en el mismo.

Back-end En el back-end encontramos todas las clases de las figuras las cuales implementan una interfaz "MovableSketch", encargada de permitir a la figura desplazarse por del canvas.

Tarea 1 - Personalización de Figuras

Para agregar las nuevas secciones de borde, relleno y copiar formato, simplemente se tuvo que investigar el uso de las clases Label, Slider, Color y ColorPicker y luego simplemente añadirlas a "SideBar" previamente mencionada.

```
Slider borderWidthSlider = new Slider(1, 50, currentStyle.getBorderWidth());
```

```
ColorPicker borderColorPicker = new ColorPicker(currentStyle.getBorderColor());
```

Luego de añadir estas opciones al menú lateral del canvas, se tuvo que crear la funciones para agregar las funcionalidades a estas opciones. Con el Slider se utiliza el setBorderWidth para cambiar el borde de la figura, con el ColorPicker se utiliza el setBorderColor/setFillColor para cambiar el color del borde o del relleno.

En el caso de copiar formato se le asigna un setOnAction el cual permite que cada vez que sea activado se ejecute la funcion de onActionCopyFormatButton. Esta funcion se guarda el formato de la figura seleccionada y luego si otra figura es seleccionada y hay un existe un estilo el cual fue copiado anteriormente, se le aplica este estilo a la figura seleccionada, de no existir este estilo, la figura simplemente se selecciona.

```
if (getCanvasState().existsStyleToCopy()) {
    aux.setStyle(getCanvasState().getStyleToCopy());
} else {
    aux.select();
    getCanvasState().selectFigure(aux);
}
```

Tarea 2 - Copiar, Cortar y Pegar

Para realizar esta tarea se crean los botones de cortar, copiar y pegar con sus respectivos iconos y se posicionan en la barra superior de la pantalla. Luego a cada accion se le aplica un evento, al hacer click sobre el boton, el cual llama a la funcion que realiza la accion de cortado, copiado o pegado. En el caso de cortar, este copia la figura, la guarda y luego elimina la restante. Al contrario, en copiado, esta simplemente guarda la figura.

Para pegar una nueva figura, primero se crea esta figura y se la aplica el mismo tipo de estilo y por ultimo se mueve al centro de la pantalla.

```
public FigureRender<? extends Figure> paste() {
    FigureRender<? extends Figure> ret = copiedFigure.copy();
    Figure retFigure = ret.getFigure();
    retFigure.move((canvasWidth / 2) - (retFigure.getStartPoint().getX() + retFigure.getEndPoint().getX
    return ret;
}
```

Para poder hacer los "shortcuts", por ejemplo CTRL + C para copiar, se tuvo que crear una funcion para ver cuando se apretan estas dos teclas al mismo tiempo. Entonces se configuran callbacks cada vez que se presionan las combianciones de teclas. Una vez que se presiona alguna combiancion de estas teclas se llama a la accion ya creada anteriormente para poder cortar, copiar y pegar utilzando los botones.

```
if (event.isControlDown()) {
    if (event.getCode() == KeyCode.X) {
        onActionCutButton();
    } else if (event.getCode() == KeyCode.C) {
        onActionCopyButton();
    } else if (event.getCode() == KeyCode.V) {
        onActionPasteButton();
    }
}
```

Tarea 3 - Deshacer y Rehacer

Para agregar los botones, se los agrego directamente en TopBar, pero en la parte inferior a esta, debajo de los botones de cortar, copiar y pegar. Debido al Stack creado anteriormente, cada vez que se realicen ciertas actividades que modifique el canvas también se agrega esta actividad al Stack. Cada vez que se presione alguno de estos dos botones, se llamara a la acción correspondiente de rehacer la actividad o deshacer la actividad. Si el Stack no está vacío, se llama a la función redo o undo, encargada de hacer el pop para quitar la operación del Stack para luego retornarla y aplicarla nuevamente, hacer el push de esta operación al Stack contrario (si estaba en undoStack ahora debe estar en redoStack) y también cambiar los dos labels de ambas acciones.

```
public Operation undo(List<FigureRender<? extends Figure>> list, FigureRender<? extends Figure> copied)
    Operation ret = undoStack.pop();
    redoStack.push(new Operation(list, ret.toString(), copied));
    setUndoRedoLabelText();
    return ret;
}
```

Para poder setear el label de cada uno, se utiliza el toString de cada operación para saber qué tipo de operación es y sobre qué tipo de figura se realiza esta operación, y para el numero simplemente se utiliza el tamaño del Stack.

```
private String getUndoStackMessage() {
    if (undoStackEmpty()) {
        return "0";
    }
    return String.format("%s %d", peekUndo().toString(), undoSize());
}
```

Diagrama UML