TP FINAL

Gonzalo Candisano 62.616

Emilio Neme 62.601

Theo Shlamovitz 62.087

gcandisano@itba.edu.ar; emneme@itba.edu.ar;
tshlamovitz@itba.edu.ar;

Programación Orientada a Objetos
Instituto Tecnológico de Buenos Aires

Antes de comenzar a realizar el trabajo, al momento de revisar el código fuente dado, encontramos varias malas prácticas en fragmentos de código, los cuales no cumplían con el paradigma orientado a objetos, tanto en el backend como en el frontend. Por ejemplo en el backend, el “Square” es un “Rectangle” pero con la característica de que sus lados miden lo mismo pero este, no extendía a la clase “Rectangle” a pesar de ser una variación del mismo. Esto mismo pasaba con las clases de “Ellipse” y “Circle” (donde “Circle” es una variación de “Ellipse”). También modificamos la visibilidad de las variables tanto de Point como de las figuras, de public/protected a private. También convertimos a la interfaz Figure en una clase abstracta, ya que necesitábamos guardar el formato de la figura en la misma, para de este modo no tener que repetirlo en cada figura. En una primera instancia guardamos tanto el color de relleno, como el del borde y el grosor del mismo, pero más adelante lo reemplazamos por una instancia de una nueva clase *FigureStyle*, la cual contenía dichos valores. Para esto convertimos los métodos de la misma en abstractos y que las demás clases ahora en vez de implementar Figure, la extiendan.

Al ver las clases del Front-End nos dimos cuenta que en la clase “PaintPane.java” habia implementaciones provistas por la catedra que tampoco cumplian con el paradigma visto durante el cuatrimestre y por esta razón agregamos los métodos *void move(double diffX, double diffY)*, boolean *belongs(Point eventPoint)* y *void draw(GraphicsContext gc)* a la clase Figure, realizando la correcta implementación de los mismos en cada clase que extendía a esta. Luego utilizamos estos métodos para remover las cadenas de if instanceof else. Para algunas de estas cadenas, también tuvimos que realizar clases nuevas como ControlButton(botón con KeyCode) y FunctionButton(botón con método para crear figura deseada). El método redraw() de PaintPane.java estaba declarado como público pero al solo utilizarlo dentro de su clase, lo cambiamos a que sea un método privado.

Una vez hechos estos cambios, comenzamos a agregar las nuevas funcionalidades requeridas y algunas extras las cuales son:

1. Cambio de color del relleno de una figura.
    - Para implementar dicha funcionalidad lo que hicimos fue, en el caso de crear una figura nueva con un nuevo color de fondo, obtener el valor del “ColorPicker” correspondiente (colorPicker.getValue()) y en el momento de crear la figura, le pasamos dicho valor dentro del FigureStyle del constructor.
2. Cambio de color del borde de una figura.
    - Para cambiar el color del borde de la figura creamos un color picker llamado “lineColorPicker” que si hay una figurada seleccionada y se aprieta este botón, se modifica el color del borde de la misma utilizando selectedFigure.setLineColor. Para estas dos funcionalidades, utilizamos setOnAction en cada colorPicker.
3. Cambio de grosor del borde de una figura.
    - Para poder realizar esta acción creamos un Slider llamado “BorderSlider” cuya finalidad es que una vez finalizado de arrastrarlo, mediante el metodo setOnMouseReleased, llamar a selectedFigure.setBorderSize con el value del slider.
4. Botón de Copiar el formato de una figura.
    - Para realizar la funcionalidad de este botón, utilizamos la clase FigureStyle para guardar el formato en CanvasState. Luego, chequeamos que si se encuentra seleccionado el botón de copiar formato y previamente se seleccionó una figura, si se hace click dentro de otra figura, se le asigna a la

figura en cuestión, el formato de la seleccionada previamente mediante el método figure.copyFormat y se vuelve a dibujar el canvas.

5. Botón de Cortar
    - Para implementar el funcionamiento de este botón, creamos una variable de figura auxiliar llamada “clipBoardFigure” en canvasState, luego nos fijamos que haya una figura seleccionada, igualamos nuestra figura auxiliar a la seleccionada y borramos la figura original.
6. Botón de Copiar
    - Para implementar el funcionamiento de este botón, nos fijamos que haya una figura seleccionada y luego guardamos dicha figura en la figura auxiliar mencionada anteriormente y hacemos que la figura seleccionada sea null así no pudiendo copiar más de una figura a la vez.
7. Botón de Pegar
    - Si clipBoardFigure del canvas no está vacío entonces creamos una nueva figura en el centro del canvas (utilizando el método getCenteredCopy)y vaciamos el clipBoardFigure.
8. Shortcuts en el teclado para los botones 5, 6, 7 (CTRL + X / C / V)
    - Creamos una clase interna (private class controlButton) para que cada botòn tenga un keyCode (tecla para activar su función), así pudiendo agregarlos en un array y recorrerlo según la tecla apretada para accionarlos. Esto lo realizamos para en el posible caso de que en un futuro se quiera agregar un botón nuevo con un shortcut, no tener que agregar un “if else”.
    - Mediante el uso de setOnKeyPressed, primero chequeamos que la tecla control esté apretada (.isControlDown()) y si esta tecla está apretada, si se apreta la “X” (.getCode() == KeyCode.X) hacemos que se dispare la implementación del botón de cortar (cutButton.fire()). Misma implementación para los botones de copiar (CTRL + C) y pegar (CTRL + V).
    - Agregamos tres shortcuts extras para hacer del uso del paint más dinámico (CTRL + Z / Y, DELETE)
        - Deshacer (CTRL + Z), Rehacer (CTRL + Y) y Suprimir (DELETE)
9. Botón de Deshacer
    - Para realizar la funcionalidad de este botón creamos un enum con las diferentes operaciones que el usuario puede realizar. Dentro de cada constante existen dos métodos de instancias que son “undo” y “redo”. La razón por la que hicimos un enum es porque estos métodos son distintos según la acción que se haya realizado anteriormente.
    - Luego creamos la clase “OperationsHistory” que tiene como variable de instancia a dos Deque de “Operations”. UndoDeque permite llevar un registro de las operaciones que se realizaron anteriormente para poder deshacerlas.
10. Botón de Rehacer
    - Dentro de la clase “OperationsHistory” el otro deque es redoDeque cuyo fin es llevar el historial de las operaciones que fueron deshechas, que se vacía cuando se realiza una nueva operación.
    - Por último, creamos los botones “undoButton” y “redoButton”, y los labels “undoNext”,”redoNext”,”undoAmount” y “redoAmount” y los metimos en un HBox que alineamos al centro.

Para estos últimos dos puntos, tuvimos que agregar en cada operación un llamado al método addOperation de OperationsHistory, el cual los agregaba al Stack correspondiente.

El primer problema encontrado durante el desarrollo del proyecto fue que antes de darnos cuenta de que Figure debía ser una clase abstracta, a la hora de cambiar tanto el color de relleno como el color de borde y el grosor del borde, al no tener las variables en particular para cada figura, cada vez que cambiamos el color se cambiaba para todas las figuras. Luego al intentar de crear los shortcuts para los botones de copiar, pegar y cortar no sabíamos cómo comprobar al mismo tiempo que esté el CTRL apretado y otra tecla, pero lo logramos resolver fácilmente buscando en la documentación. Otra dificultad que nos llevó bastante tiempo, fue poder lograr centrar correctamente los botones de deshacer/rehacer y sus respectivos Labels, sin que se desplacen cuando el texto de la próxima acción se modificaba. La última dificultad encontrada fue pensar la correcta manera de implementar los botones de deshacer y rehacer, y luego la correcta implementación de los métodos, para poder lograr distintas combinaciones de operaciones junto con deshacer y rehacer, y que estas funcionen de la manera esperada.

Apéndice:

Documentación del proyecto:

- https://tpe-poo-website.vercel.app/

UML del proyecto:

- De todas formas se encuentra en el .zip para una mejor visualización)

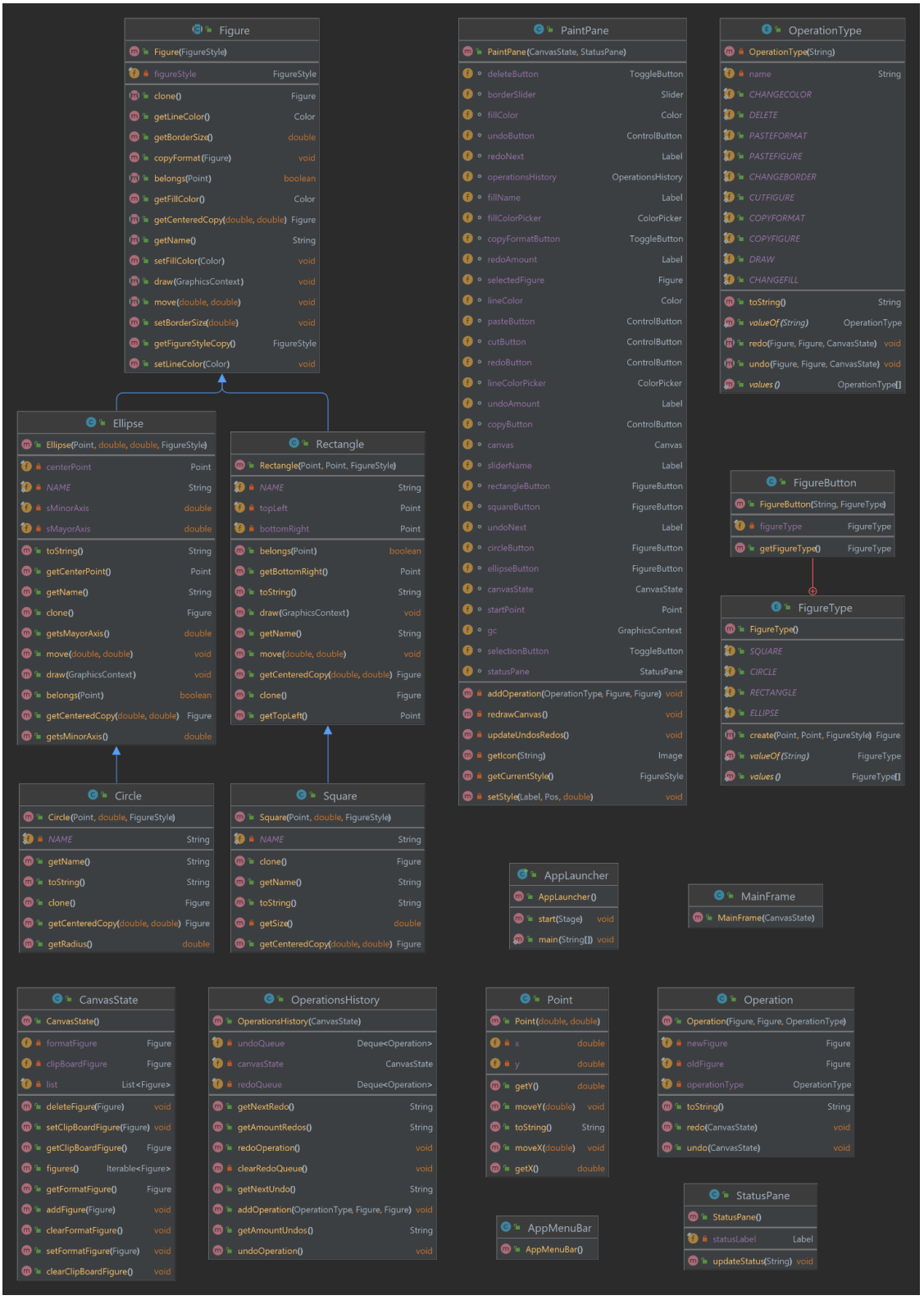
Figure
Figure(FigureStyle)
figureStyle FigureStyle
clone() Figure
getLineColor() Color
getBorderSize() double
copyFormat(Figure) void
belongs(Point) boolean
getFillColor() Color
getCenteredCopy(double, double) Figure
getName() String
setFillColor(Color) void
draw(GraphicsContext) void
move(double, double) void
setBorderSize(double) void
getFigureStyleCopy() FigureStyle
setLineColor(Color) void
PaintPane
PaintPane(CanvasState, StatusPane)
deleteButton ToggleButton
borderSlider Slider
fillColor Color
undoButton ControlButton
redoNext Label
operationsHistory OperationsHistory
fillName Label
fillColorPicker ColorPicker
copyFormatButton ToggleButton
redoAmount Label
selectedFigure Figure
lineColor Color
pasteButton ControlButton
cutButton ControlButton
redoButton ControlButton
lineColorPicker ColorPicker
undoAmount Label
copyButton ControlButton
canvas Canvas
sliderName Label
rectangleButton FigureButton
squareButton FigureButton
undoNext Label
circleButton FigureButton
ellipseButton FigureButton
canvasState CanvasState
startPoint Point
gc GraphicsContext
selectionButton ToggleButton
statusPane StatusPane
addOperation(OperationType, Figure, Figure) void
redrawCanvas() void
updateUndosRedos() void
getIcon(String) Image
getCurrentStyle() FigureStyle
setStyle(Label, Pos, double) void
OperationType
OperationType(String)
name String
CHANGECOLOR
DELETE
PASTEFORMAT
PASTEFIGURE
CHANGEBORDER
CUTFIGURE
COPYFORMAT
COPYFIGURE
DRAW
CHANGEFILL
toString() String
valueOf(String) OperationType
redo(Figure, Figure, CanvasState) void
undo(Figure, Figure, CanvasState) void
values() OperationType[]
Ellipse
Ellipse(Point, double, double, FigureStyle)
centerPoint Point
NAME String
sMinorAxis double
sMayorAxis double
toString() String
getCenterPoint() Point
getName() String
clone() Figure
getsMayorAxis() double
move(double, double) void
draw(GraphicsContext) void
belongs(Point) boolean
getCenteredCopy(double, double) Figure
getsMinorAxis() double
Rectangle
Rectangle(Point, Point, FigureStyle)
NAME String
topLeft Point
bottomRight Point
belongs(Point) boolean
getBottomRight() Point
toString() String
draw(GraphicsContext) void
getName() String
move(double, double) void
getCenteredCopy(double, double) Figure
clone() Figure
getTopLeft() Point
FigureButton
FigureButton(String, FigureType)
figureType FigureType
getFigureType() FigureType
FigureType
FigureType()
SQUARE
CIRCLE
RECTANGLE
ELLIPSE
create(Point, Point, FigureStyle) Figure
valueOf(String) FigureType
values() FigureType[]
Circle
Circle(Point, double, FigureStyle)
NAME String
getName() String
toString() String
clone() Figure
getCenteredCopy(double, double) Figure
getRadius() double
Square
Square(Point, double, FigureStyle)
NAME String
clone() Figure
getName() String
toString() String
getSize() double
getCenteredCopy(double, double) Figure
AppLauncher
AppLauncher()
start(Stage) void
main(String[]) void
MainFrame
MainFrame(CanvasState)
CanvasState
CanvasState()
formatFigure Figure
clipBoardFigure Figure
list List<Figure>
deleteFigure(Figure) void
setClipBoardFigure(Figure) void
getClipBoardFigure() Figure
figures() Iterable<Figure>
getFormatFigure() Figure
addFigure(Figure) void
clearFormatFigure() void
setFormatFigure(Figure) void
clearClipBoardFigure() void
OperationsHistory
OperationsHistory(CanvasState)
undoQueue Deque<Operation>
canvasState CanvasState
redoQueue Deque<Operation>
getNextRedo() String
getAmountRedos() String
redoOperation() void
clearRedoQueue() void
getNextUndo() String
addOperation(OperationType, Figure, Figure) void
getAmountUndos() String
undoOperation() void
Point
Point(double, double)
x double
y double
getY() double
moveY(double) void
toString() String
moveX(double) void
getX() double
Operation
Operation(Figure, Figure, OperationType)
newFigure Figure
oldFigure Figure
operationType OperationType
toString() String
redo(CanvasState) void
undo(CanvasState) void
AppMenuBar
AppMenuBar()
StatusPane
StatusPane()
statusLabel Label
updateStatus(String) void