Vector Based Drawing Application Report

Team Members:

- Bassent Mostafa Saad Zaghloul Ahmed (21)
- Mira Samir Ragheb (78)

Description:

Drawing and painting applications are very popular and have a huge user base. "Paint" Application is a user-friendly painting application that offers number of features that includes:

- Drawing shapes like Line Segment, Elliptical shapes (Ellipses and Circles) and Polygons (Rectangles and Squares).
- Coloring shapes: either their borders or their inside body.
- Resizing.
- Undo and Redo any instruction so as to make the application more usable.
- Moving and deleting any shape.
- One of the main features is saving user's drawings in a file and modifying it later. User can choose whether to save their drawings in an XML or a JSON file.

The concept of dynamic class loading is widely spread in computer applications. It is an option that allows the user to extend application features at runtime.

- "Paint" Application provides an option that allows for selecting the class library file to load, on selecting and loading the desired file, the isolated shape is appended to the available list of shapes in the application.

Paint Design and Implementation:

Design Patterns used:

1 – MVC pattern:

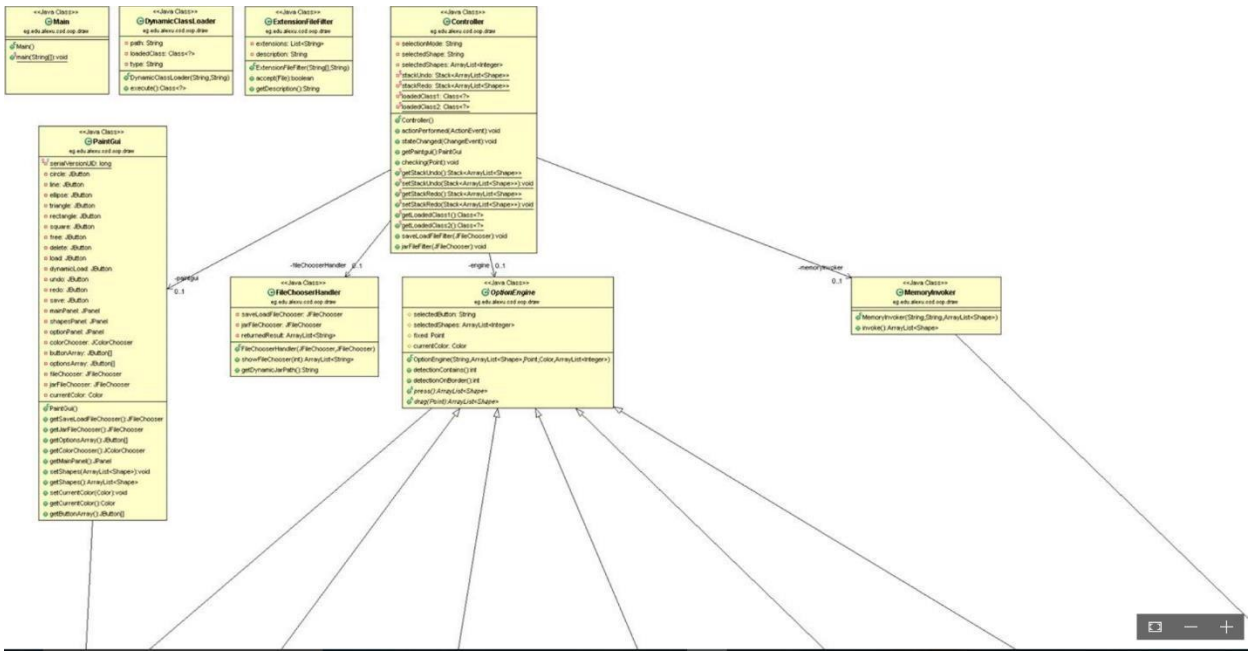
"MVC Pattern stands for Model-View-Controller Pattern. Model–view–controller (MVC) is a software design pattern for implementing user interfaces on computers. It divides a given software application into three

interconnected parts, so as to separate internal representations of information from the ways that information is presented to or accepted from the user."

-Wikipedia

MVC design pattern in "Paint" application:

- **Model** - Model represents Classes which interact with Controller Class and update shapes and history on Paint GUI.
- **View** - View represents the visualization of the data that model contains. The View is the Paint GUI Class.
- **Controller** – Controller Class acts on both model (Implemented Classes for painting options) and view (Paint GUI). It controls the data flow into model object and updates the view whenever data changes. It keeps view and model separate.



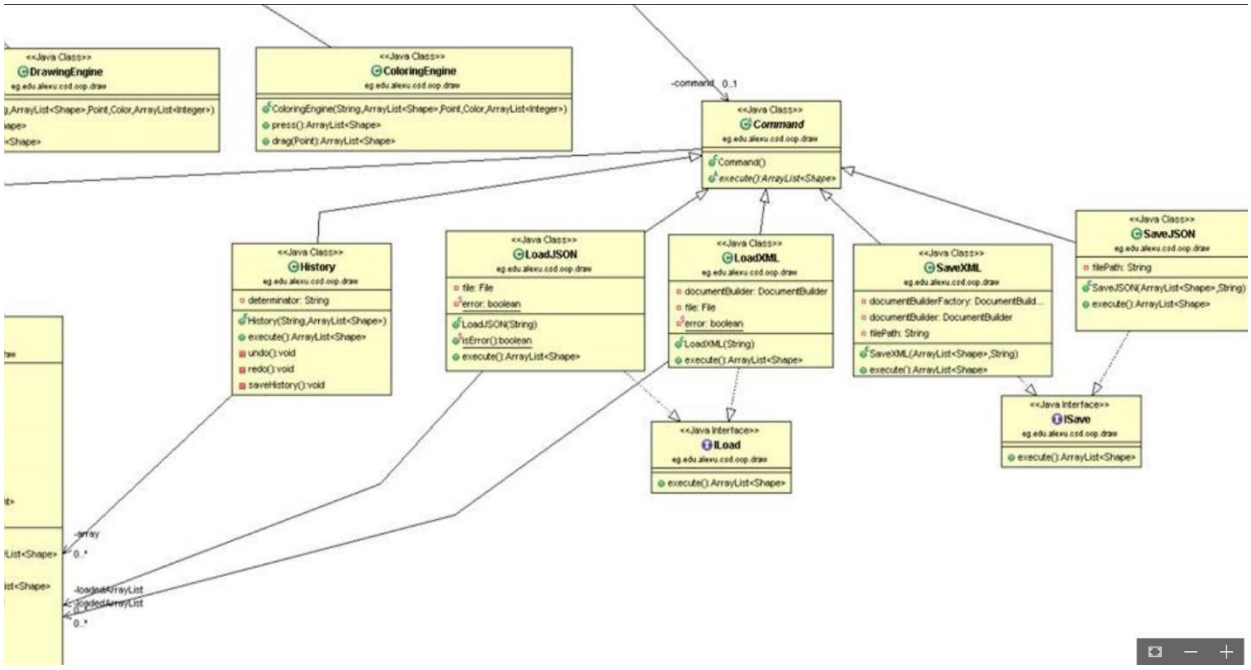
2 – Strategy Design Pattern:

"The strategy pattern (also known as the policy pattern) is a software design pattern that enables an algorithm's behavior to be selected at runtime. The strategy pattern defines a family of algorithms,

encapsulates each algorithm, and makes the algorithms interchangeable within that family.
Strategy lets the algorithm vary independently from clients that use it."

-Wikipedia

Strategy Design pattern is used in implementing both Save and Load techniques, since "Paint" Application enables users to save/load both xml and json files. Strategy Pattern helps in having two different implementations depending on user's desire for one option.



3- Command Design Pattern:

"In object-oriented programming, the command pattern is a behavioral design pattern in which an object is used to encapsulate all information needed to perform an action or trigger an event at a later time. This information includes the method name, the object that owns the method and values for the method parameters."

-Wikipedia

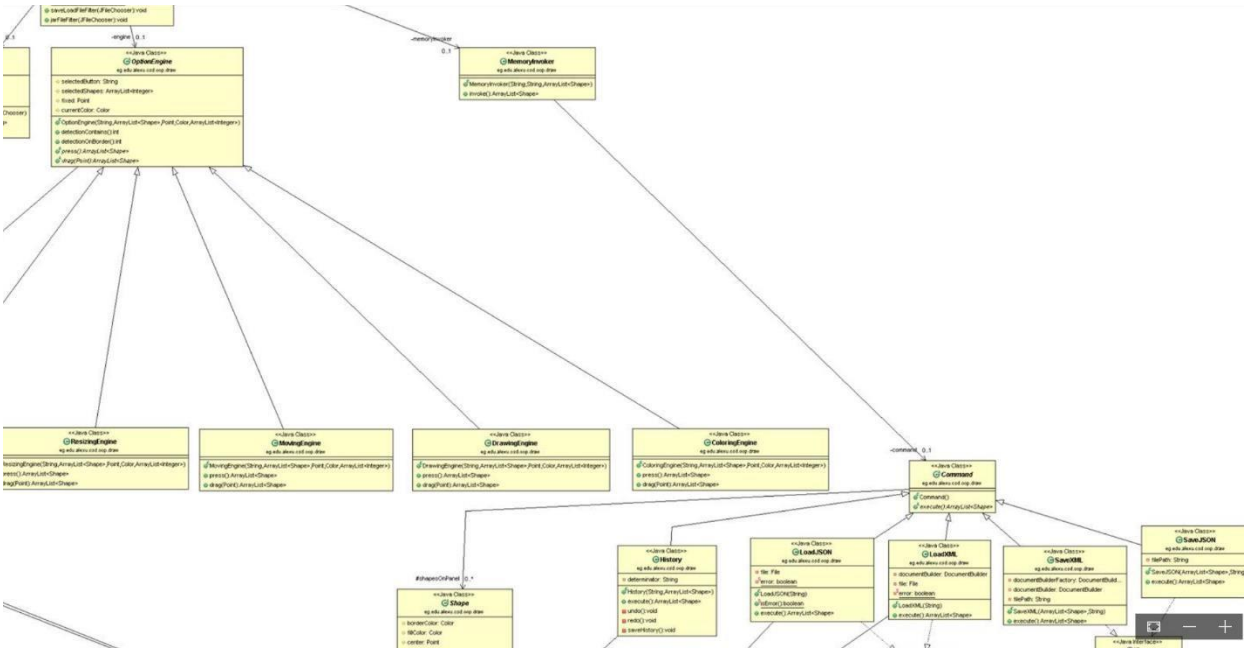
Command Design Pattern is implemented as follows:

1 – Abstract Command Class

2 – Save, Load and History Classes extends Command Class with a common execute method.

3 – Invoker Class instantiating Command Class object.

4 – The Controller invokes Command by an Invoker (Invoker calls execution methods)



==========

Object Oriented Principles:

1- Interfaces:

- IShape , ILoad , ISave interfaces defines the contract/ methods used by classes which implements them.

2- Abstraction:

- Abstract Shape Class implements IShape, defines the common attributes and methods shared by all shapes which extends Shape Class.
- Abstract Command Class which is extended by History, Save and Load Classes to be Invoked through Memory Invoker.
- Abstract Polygon Class defines common attributes and methods shared by Polygon shapes.
- Abstract OptionEngine Class defines common attributes and methods shared by the core painting classes: Drawing, Coloring, Moving and Resizing Classes.

3- **Polymorphism:**

- Controller Class deals with drawing paint tools through an OptionEngine.
- Controller Class deals with Memory tools through an Invoker instance which Invokes a Command to be executed.

4- **Inheritance:**

- LineSegment, Ellipse, Rectangle, FreeShape Classes inherit their common attributes and methods from Abstract Shape Class.
- Circle class inherits its attributes and methods from Ellipse Class.

- Square class inherits its attributes and methods from Rectangle Class.
- Drawing, Moving, Resizing, Coloring engine Classes inherit their common attributes and methods from an Abstract OptionEngine Class.
- Memory classes e.g. Save, Load , History classes inherits their common execute method from an abstract Command Class.

5- Encapsulation:

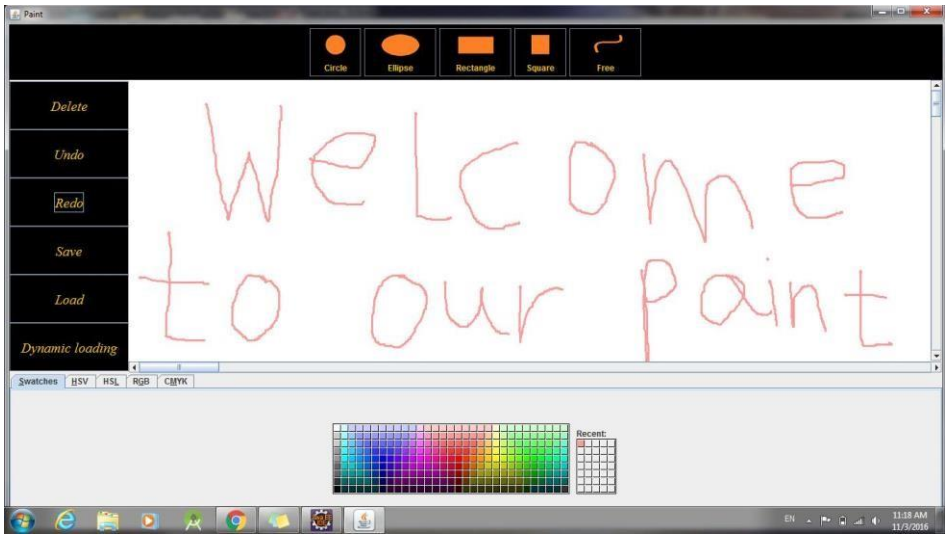
Encapsulation principle is preserved in "Paint" application with each Class having private attributes can only be accessed through getters and setters.

==========

More added features:

- **Free hand Button:**

  Allows the user to draw a free hand line.
  User can change the free-hand line color either before drawing or after.
  User can delete a selected free-hand line.
  User can save and load free-hand line together with other implemented shapes.



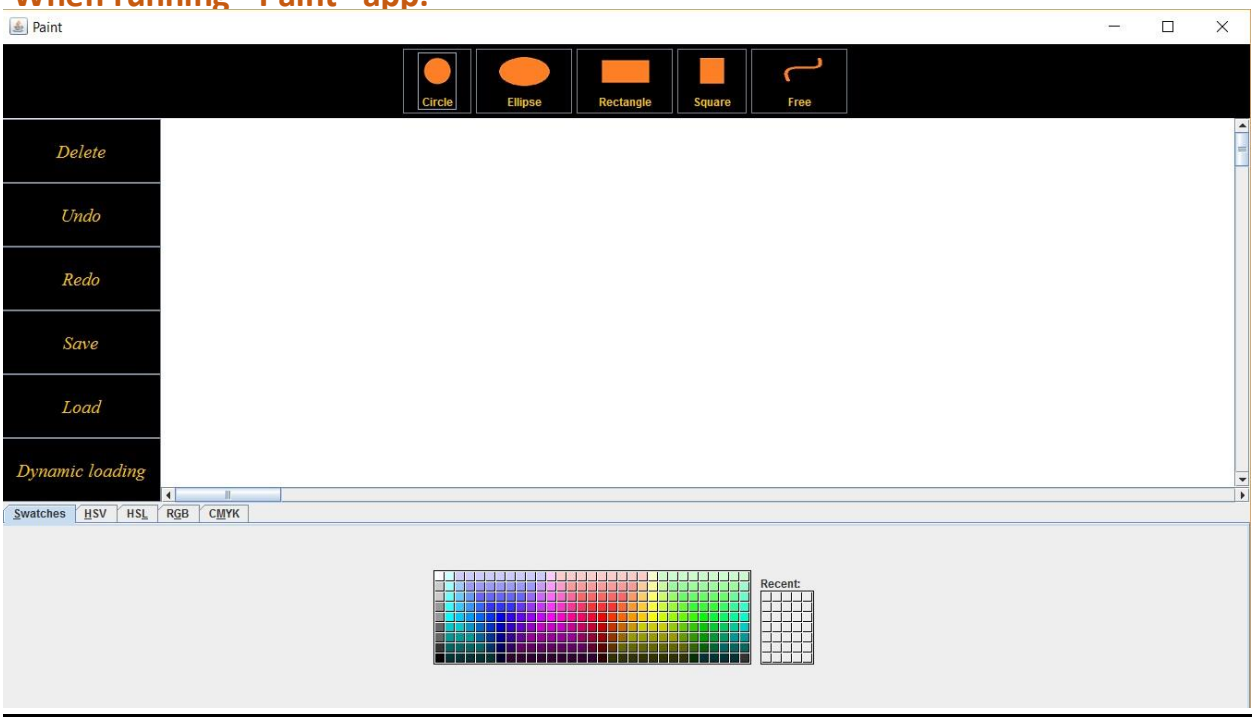
- **Selecting Multiple Shapes:**

  An added feature is that user can select multiple drawn shapes at a time and they can either:

  - Move selected shapes as one object.
  - Delete selected shapes.

“Paint” User Guide:

When running “Paint” app:



“Paint” GUI Description:

1- Shapes panel at the top:

Panel containing buttons of Shapes that can be drawn.

⇨ *Triangle and Line Segment Shapes Buttons are disables by default, to be active User must dynamically load external jars.*

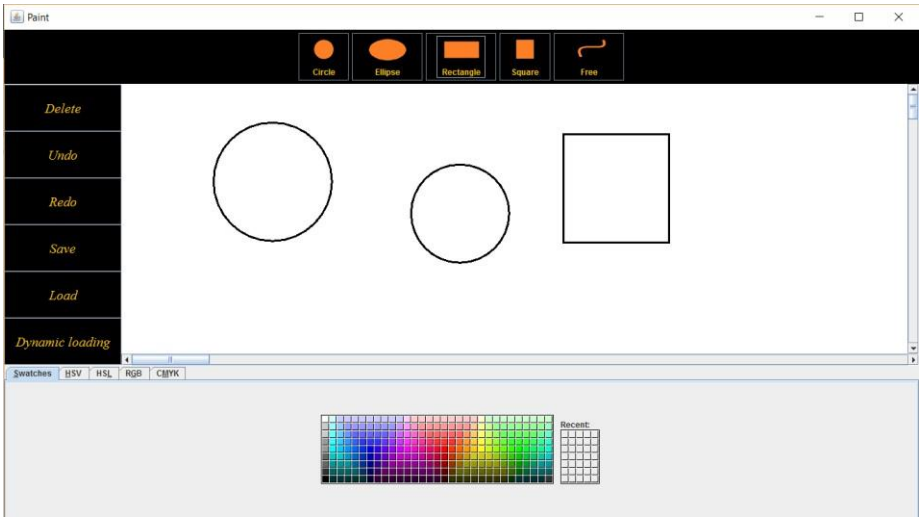
2- Options panel at the left:

Panel containing all the options that user can use on shapes drawn.

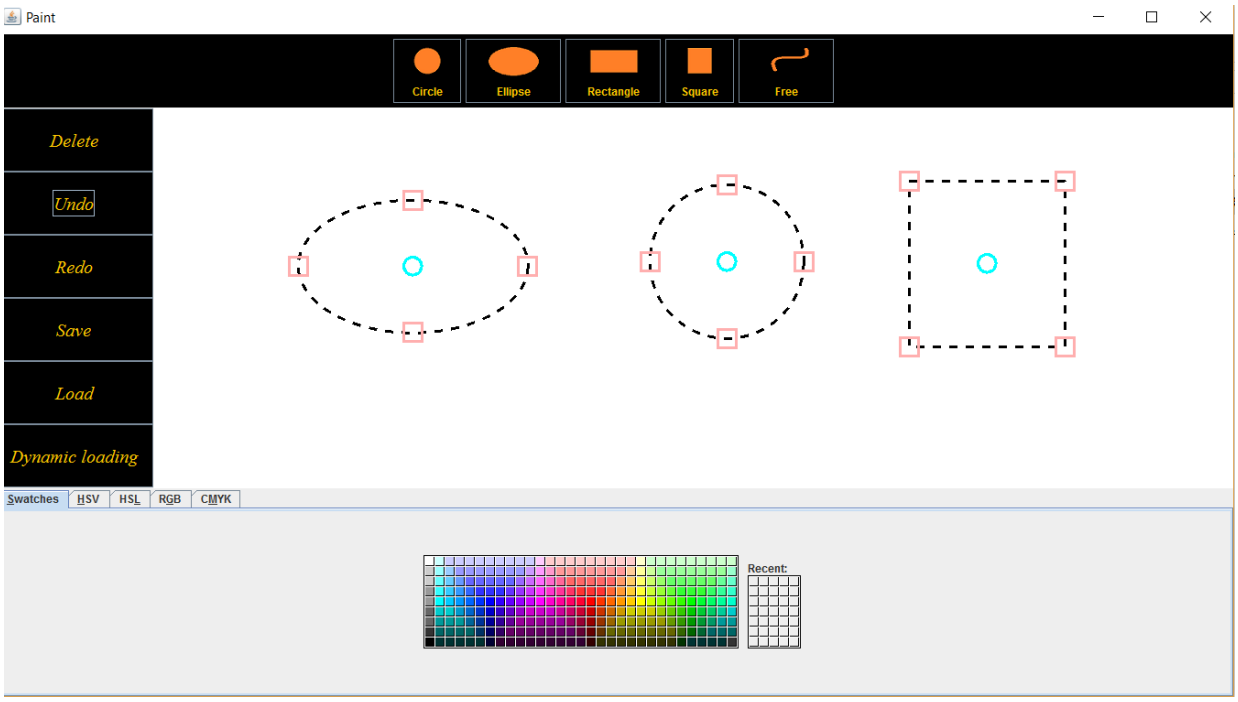
2- Color chooser Panel at the bottom.

⇨ Cursor changes its shape when resizing, moving and drawing.

**Drawing Shapes:



a. User can move objects by selecting the desired shape and drag its center.
b. User can resize shapes by selecting them and dragging the squares positioned at the corners, once the mouse is released the shape is released.



⇨ *It's worth noticing that free shape can be selected to be deleted only or to change its color. But not to be moved or resized.*

Resizing:

1- *Ellipse and circles are resized with the same center point. Circles preserve their definition when resizing.*

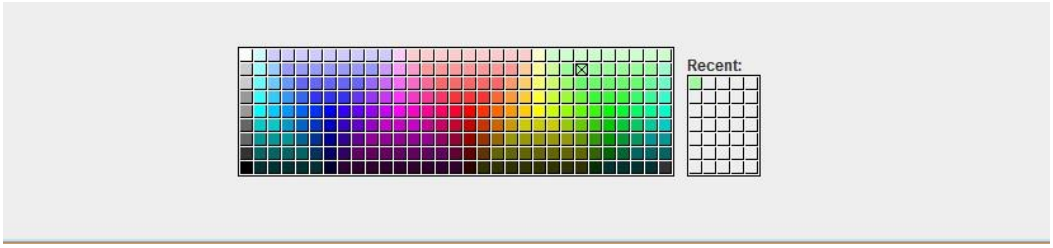
2- *Rectangles and squares are resized around a fixed corner. (Squares preserve their definition when resizing).*

**Coloring Shapes

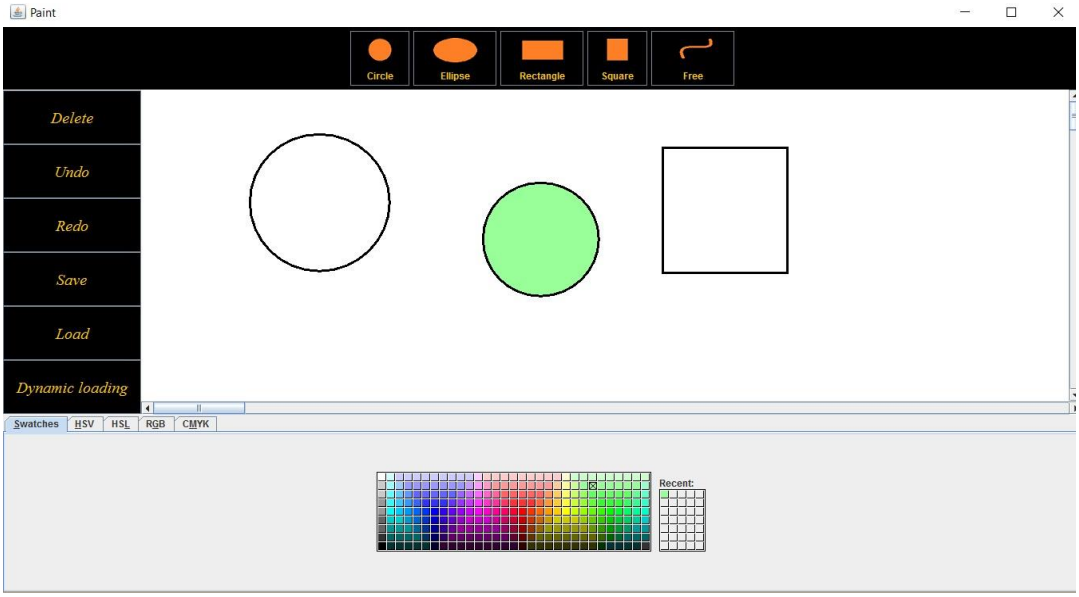
using Coloring Chooser at the bottom:

1 – User chooses the desired color from the coloring chooser at the bottom.

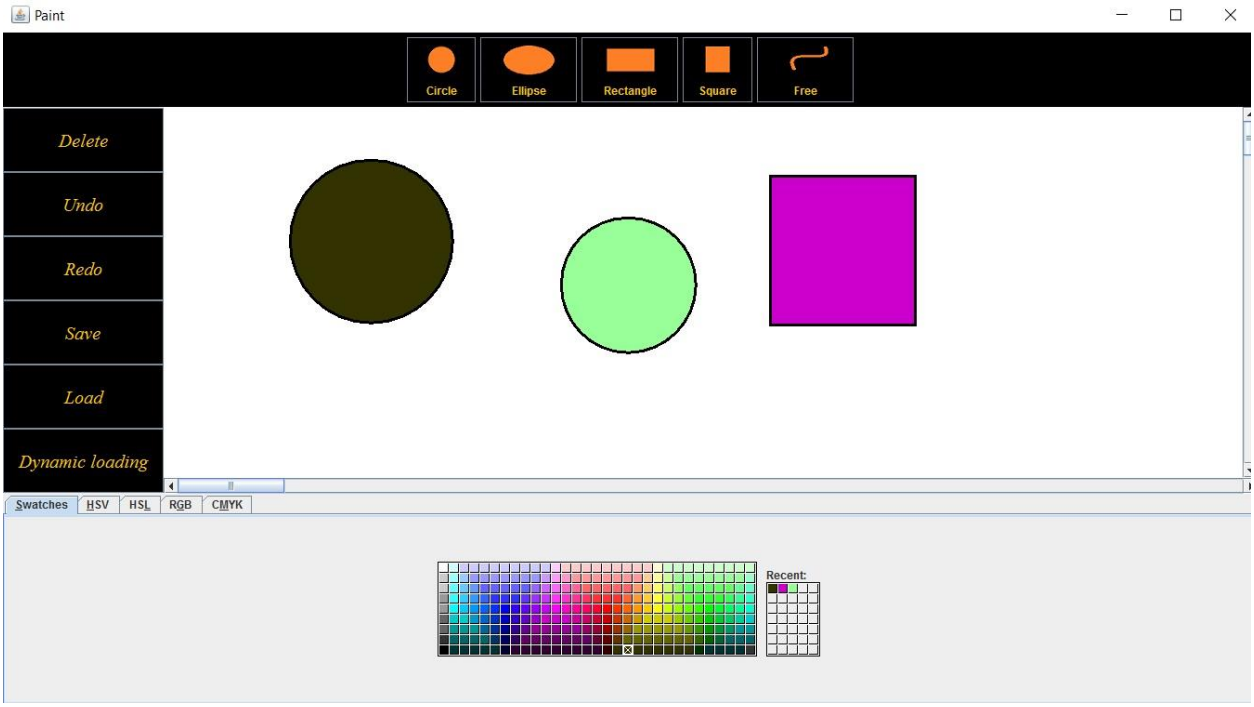
The chosen Color appears at the right of the color chooser.



2 – User then clicks within the desired shape to color.



3 – Coloring other shapes by the same way.

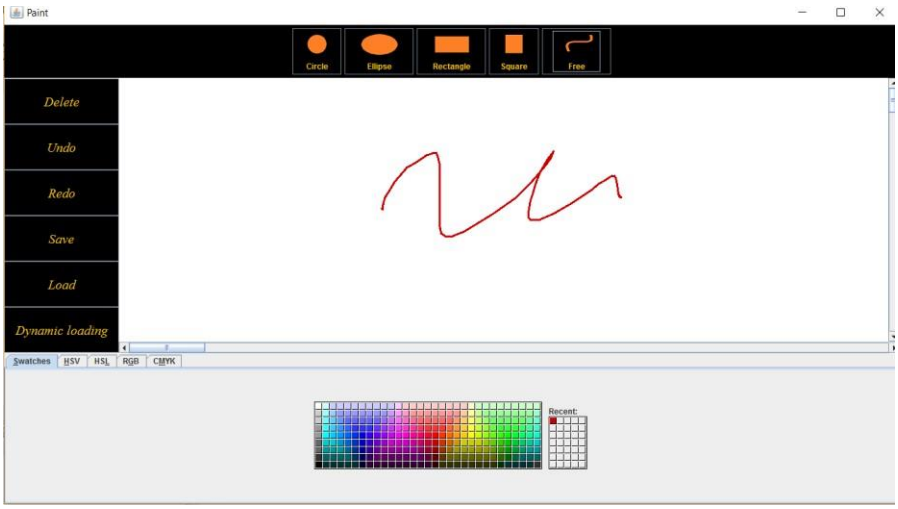


⇨ *User can draw shapes of other border colors than black, this can simply be done by:*

1- User chooses the desired color first from the color chooser pane:

2- User then clicks on the shape they desire to draw:

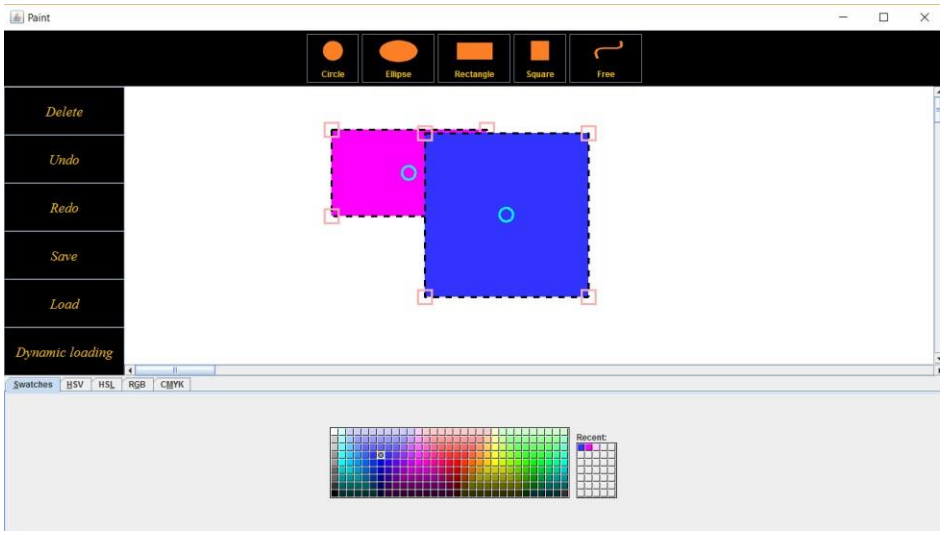
3- Then draws:



**Deleting shapes:

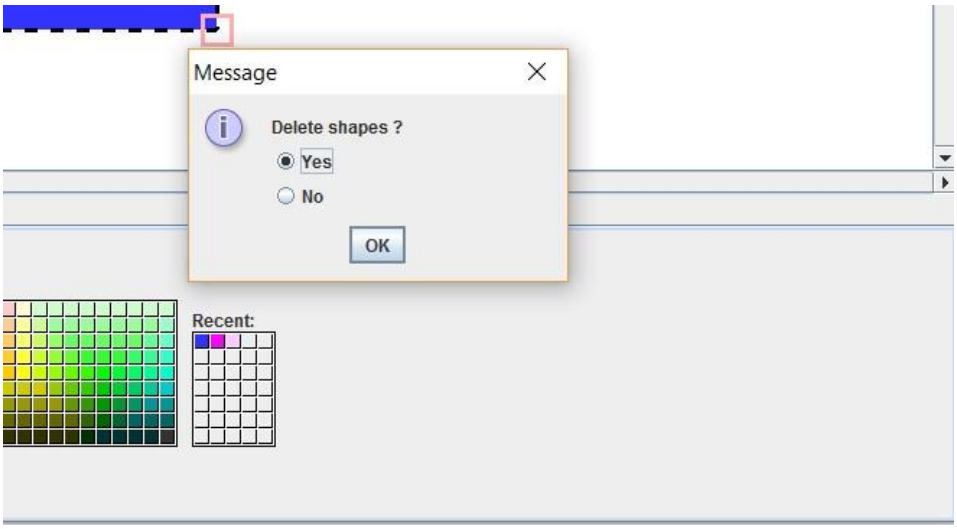
User can delete one or more selected shapes.

1- Select shapes to be deleted

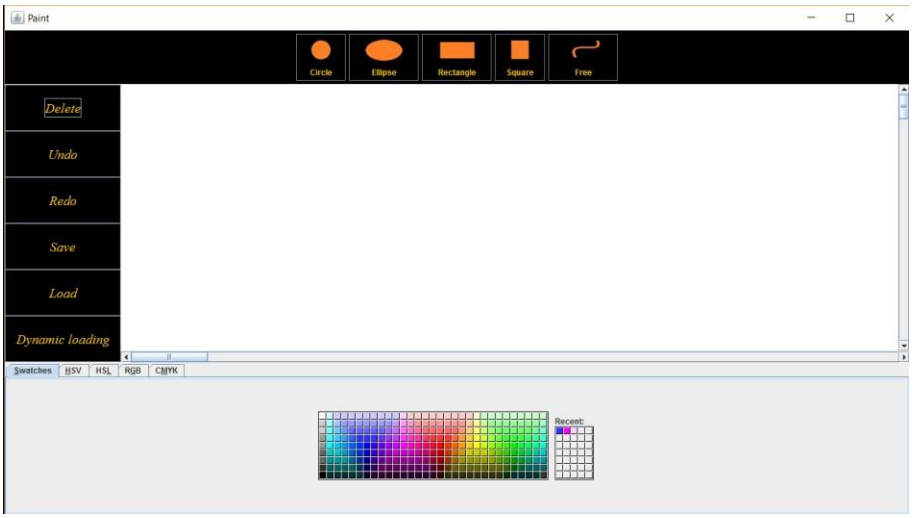


2- Press delete button on the left.

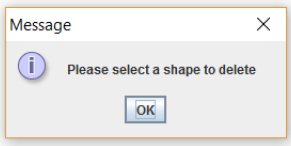
3- A Confirmation message appears, user can decline their deleting request or continue deleting the selected shapes.



When confirming the deletion request:



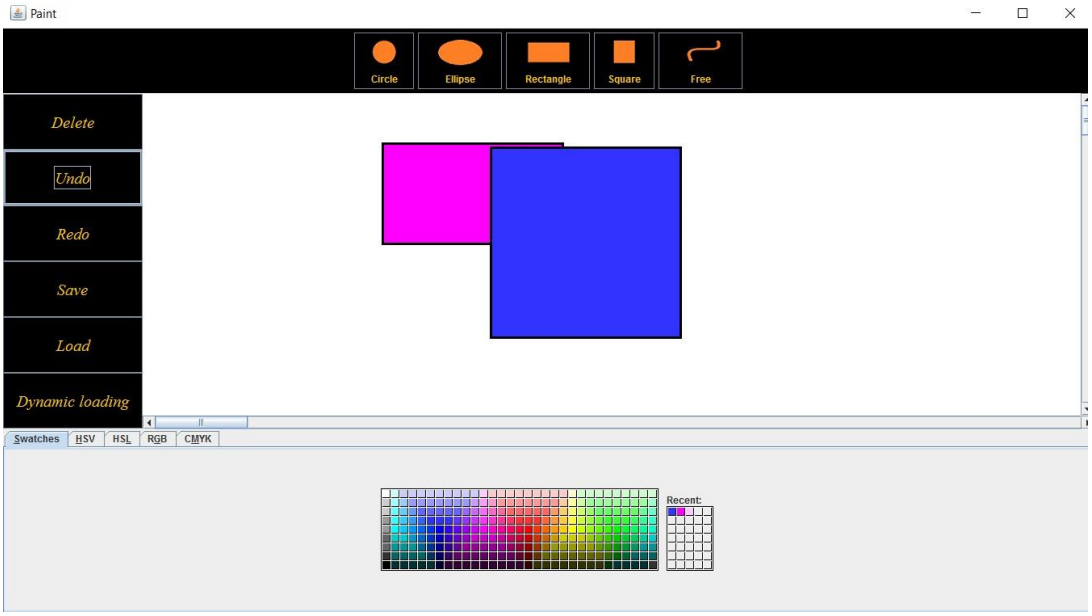
Deleting without selecting a shape or with a white canvas with no shape drawn triggers a message that asks for selecting a shape before requesting a delete.



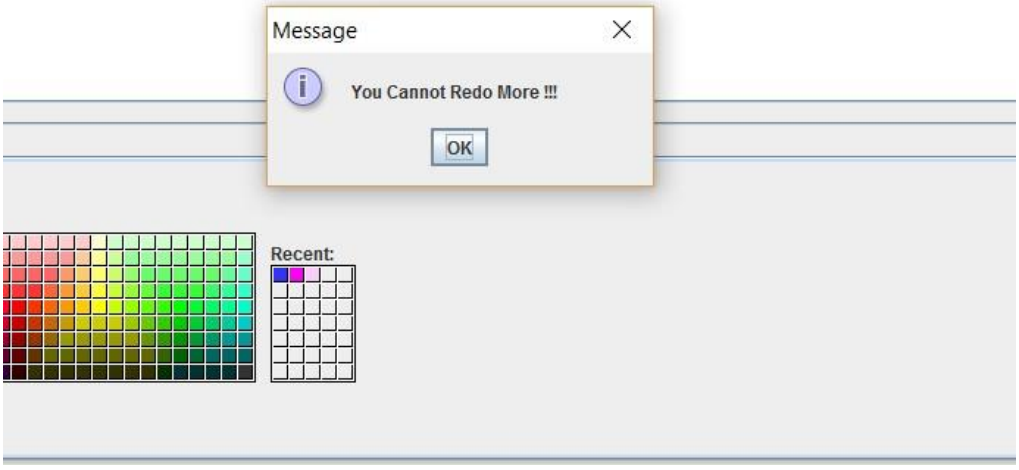
**Undo and redo:

User is allowed to undo and redo their moves.

⇨ *(Undoing the last delete action)*

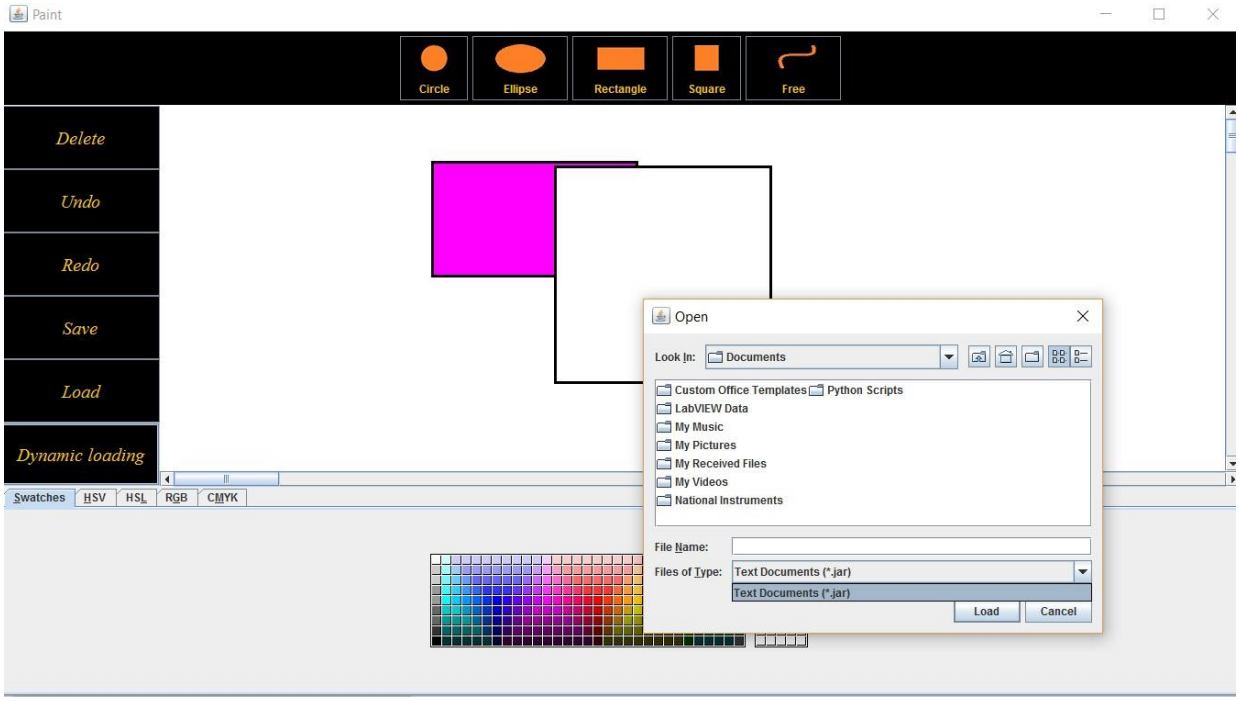


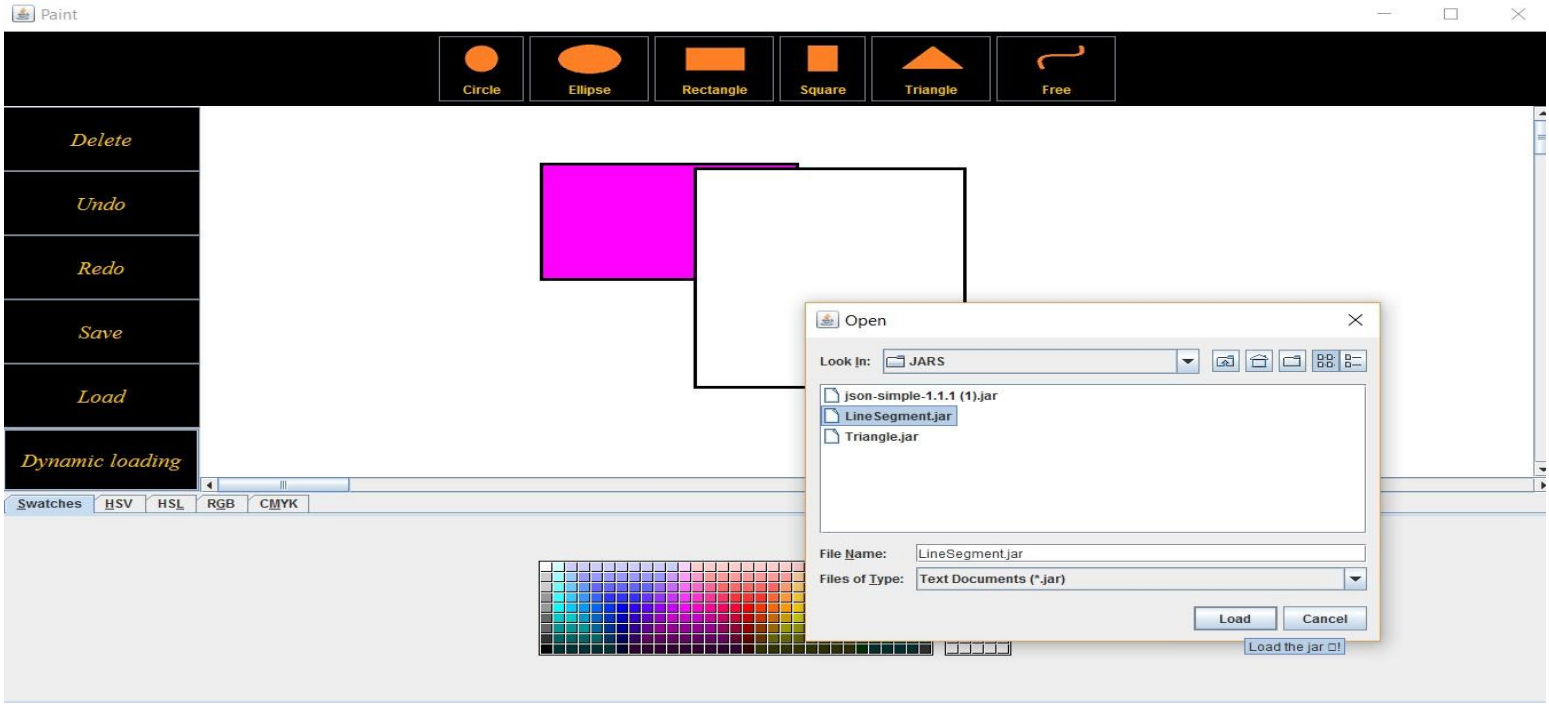
⇨ *When reaching their last move, pressing redo again pops out a warning message that there are no more redo's. Same goes for the undo option.*



**Dynamic loading for external classes implementing the same IShape interface:

- Users can dynamically load an external plug-in /shape
- "Paint" app gives an example by dynamically loading either line segment or triangle shape.
- When pressing the "Dynamic loading" button: file chooser panel pops out asking for the path of the external jar.

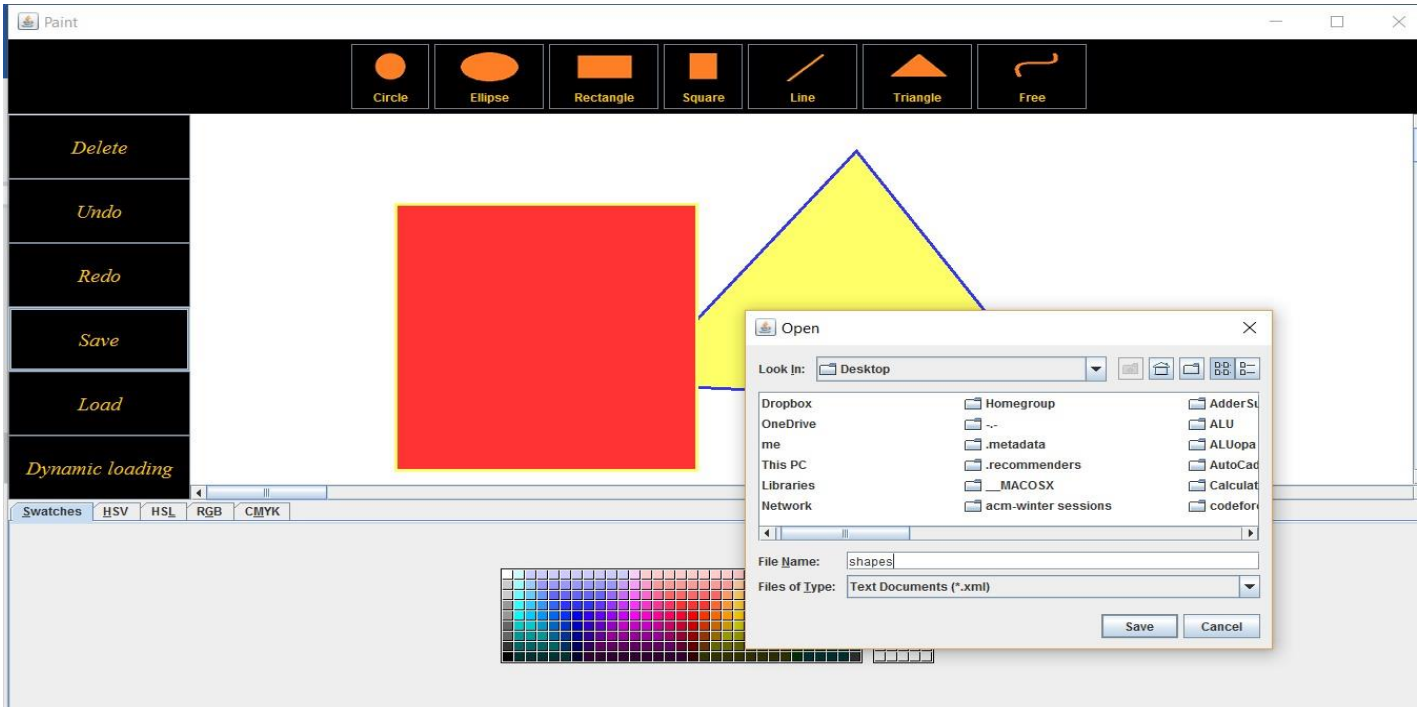




Line Segment Button is enabled and user can draw line segments

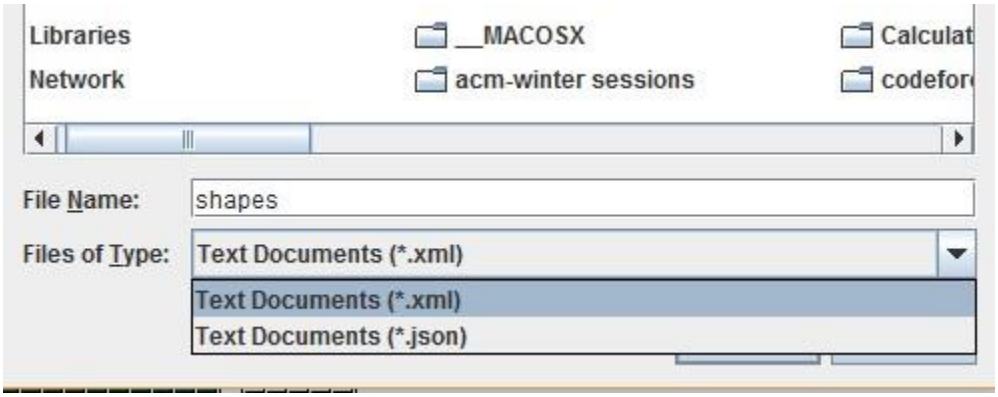
⇨ Loading an invalid jar file triggers an error message.

**Saving and loading masterpieces:

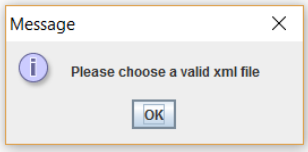


- User can save their masterpiece either as an .xml or a .json file and load them back again whenever they like.
- A file chooser pane pops out asking for the file path either to save or load it depending on the user's desire.

The file filter filters the types of files to show/ is acceptable to .xml and .json files only.

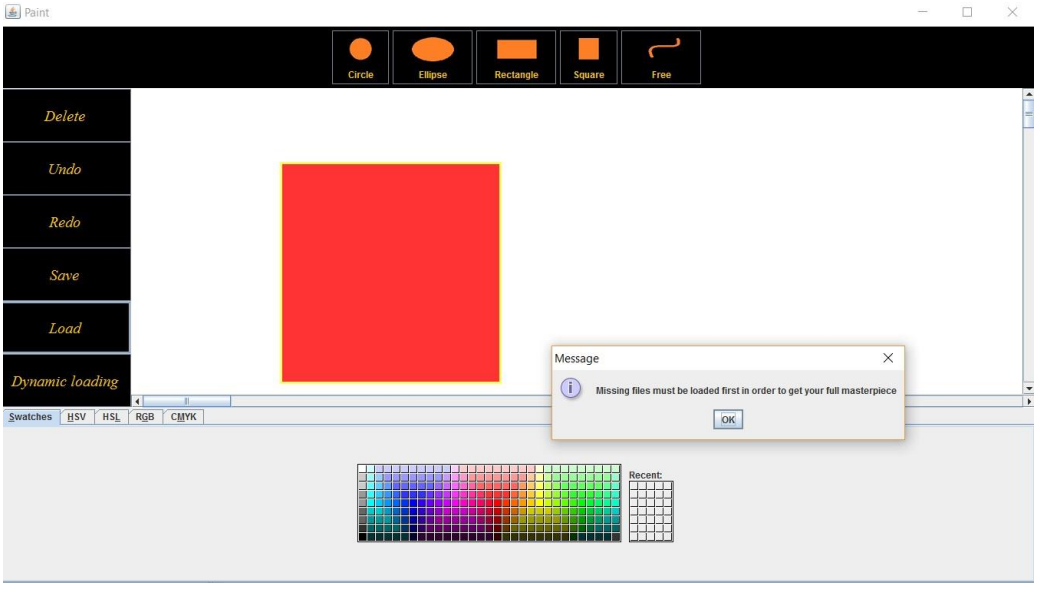


⇨ If user chooses an invalid file with an xml/json extension, an error message appears



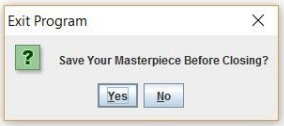
⇨ *It's worth mentioning that the default whenever the "Paint" app reruns is that both line segment and triangle shapes are disabled and needs to be dynamically loaded, so in case of loading a saved file containing either of them on rerunning the application, the default shapes appear but an error message explains that some shapes are missing and needs to be dynamically loaded first in order to get the full masterpiece.*

When rerunning the program and loading the saved "shapes.xml " file.

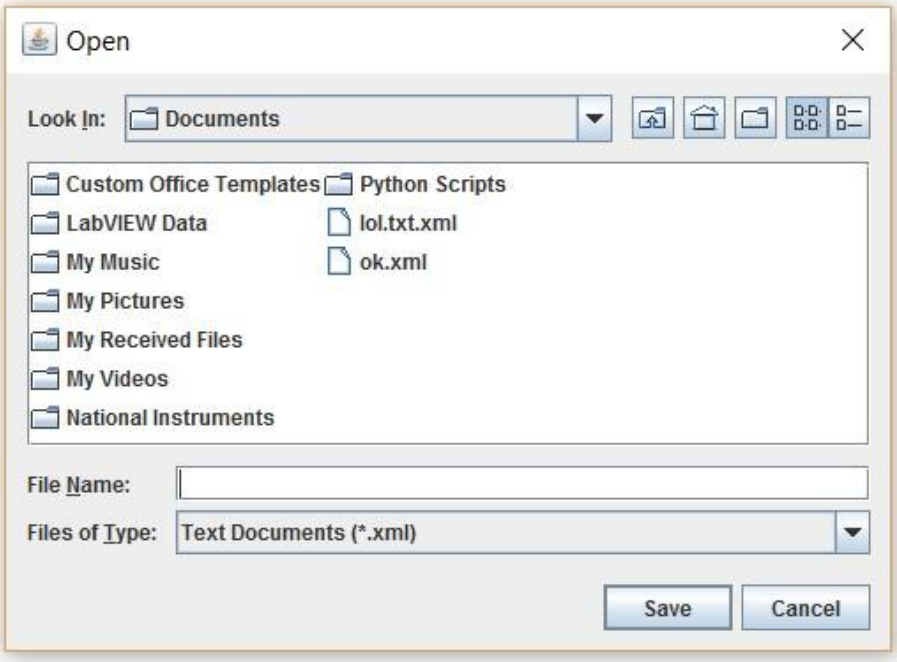


**Exit program:

On clicking the exit icon, a message pops out asking the user whether to save their masterpiece or not before exiting.



When clicking yes, file chooser pane appears.



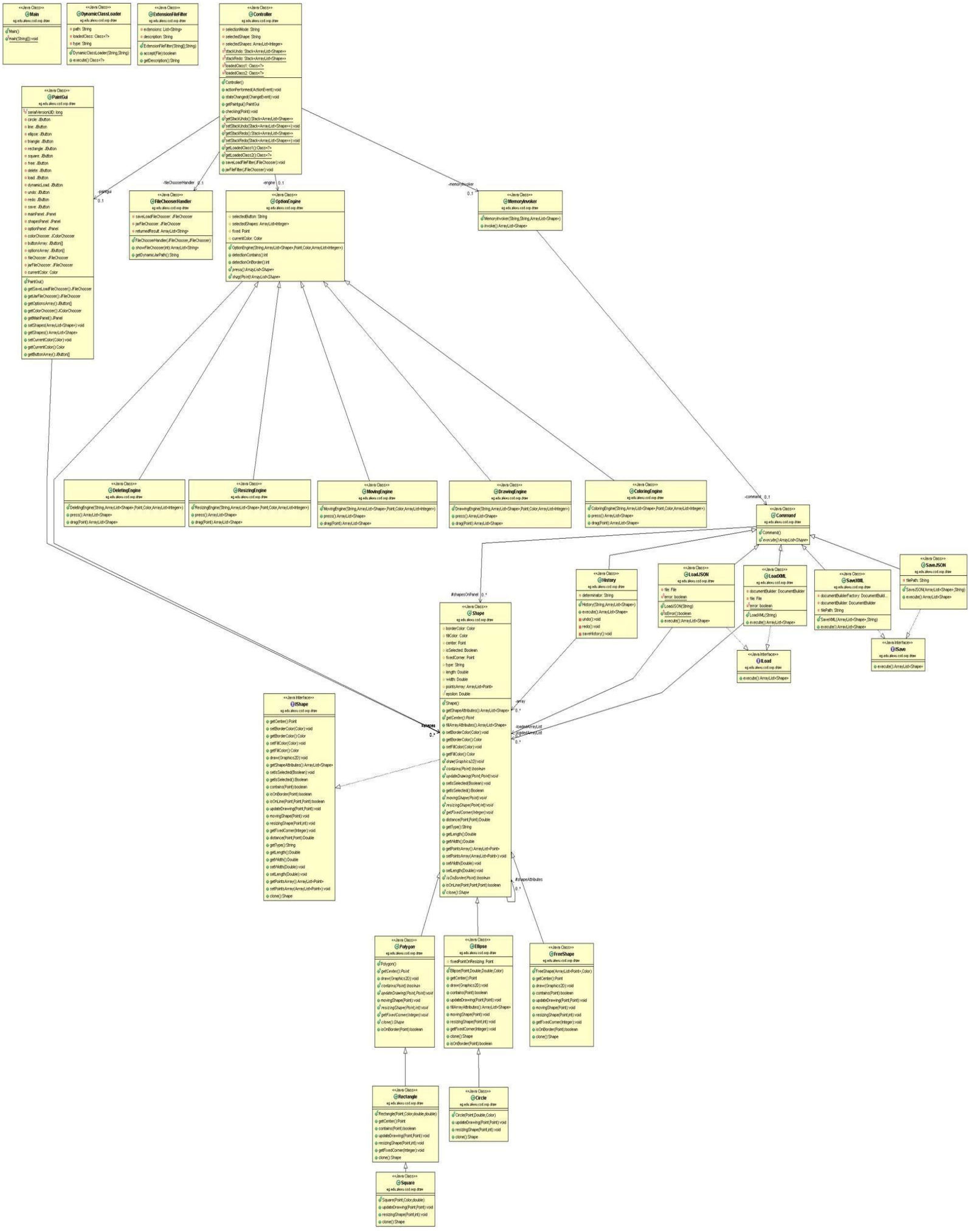
On both cases, whether user chooses to save their work or not, program is terminated.

**UML Diagram:

Description:

- *IShape interface holds the contract of the common methods that must be common amongst the implemented shapes.*
- *Abstract Shape Class implements the IShape methods.*
- *LineSegment, Ellipse, Polygon and FreeShape classes extends the abstract Shape Class.*
- *Triangle and Rectangle classes extends Polygon Class. - Circle extends Ellipse Class - Square extends Rectangle Class.*
- *Controller Class deals with Paintgui class, Invoker and OptionsEngine Classes only. (MVC)*
- *Invoker initiates an instance of Command Class and executes this command.*
- *Absract Command Class is extended by SaveXML, SaveJSON, LoadXML, LoadJSON and History classes.*
- *ISave interface holds the contract of the methods present in both SaveXML and SaveJSON classes. (Strategy Design)*
- *ILoad interface holds the contract of the methods present in both LoadXML and LoadJSON classes. (Strategy Design)*
- *Drawing, Coloring, Moving, Resizing and Deleting engines extends from Absract OptionEngine Class.*

⇨ **For a better view, visit UMLdiagram.png in the project Package**.


Main
Main()
main(String[]):void
DynamicClassLoader
path: String
loadedClass: Class<?>
type: String
DynamicClassLoader(String,String)
execute():Class<?>
ExtensionFileFilter
extensions: List<String>
description: String
ExtensionFileFilter(String[],String)
accept(File):boolean
getDescription():String
Controller
selectionMode: String
selectedShape: String
selectedShapes: ArrayList<Integer>
stackUndo: Stack<ArrayList<Shape>>
stackRedo: Stack<ArrayList<Shape>>
loadedClass1: Class<?>
loadedClass2: Class<?>
Controller()
actionPerformed(ActionEvent):void
stateChanged(ChangeEvent):void
getPaintgui():PaintGui
checking(Point):void
getStackUndo():Stack<ArrayList<Shape>>
setStackUndo(Stack<ArrayList<Shape>>):void
getStackRedo():Stack<ArrayList<Shape>>
setStackRedo(Stack<ArrayList<Shape>>):void
getLoadedClass1():Class<?>
getLoadedClass2():Class<?>
saveLoadFileFilter(JFileChooser):void
jwFileFilter(JFileChooser):void
PaintGui
serialVersionUID: long
circle: JButton
line: JButton
ellipse: JButton
triangle: JButton
rectangle: JButton
square: JButton
free: JButton
delete: JButton
load: JButton
dynamicLoad: JButton
undo: JButton
redo: JButton
save: JButton
mainPanel: JPanel
shapesPanel: JPanel
optionPanel: JPanel
colorChooser: JColorChooser
buttonArray: JButton[]
optionsArray: JButton[]
fileChooser: JFileChooser
jwFileChooser: JFileChooser
currentColor: Color
PaintGui()
getSaveLoadFileChooser():JFileChooser
getJwFileChooser():JFileChooser
getOptionsArray():JButton[]
getColorChooser():JColorChooser
getMainPanel():JPanel
setShapes(ArrayList<Shape>):void
getShapes():ArrayList<Shape>
setCurrentColor(Color):void
getCurrentColor():Color
getButtonArray():JButton[]
FileChooserHandler
saveLoadFileChooser: JFileChooser
jwFileChooser: JFileChooser
returnedResult: ArrayList<String>
FileChooserHandler(JFileChooser,JFileChooser)
showFileChooser(int):ArrayList<String>
getDynamicJwPath():String
OptionEngine
selectedButton: String
selectedShapes: ArrayList<Integer>
fixed: Point
currentColor: Color
OptionEngine(String,ArrayList<Shape>,Point,Color,ArrayList<Integer>)
detectionContains():int
detectionOnBorder():int
press():ArrayList<Shape>
drag(Point):ArrayList<Shape>
MemoryInvoker
MemoryInvoker(String,String,ArrayList<Shape>)
invoke():ArrayList<Shape>
DeletingEngine
DeletingEngine(String,ArrayList<Shape>,Point,Color,ArrayList<Integer>)
press():ArrayList<Shape>
drag(Point):ArrayList<Shape>
ResizingEngine
ResizingEngine(String,ArrayList<Shape>,Point,Color,ArrayList<Integer>)
press():ArrayList<Shape>
drag(Point):ArrayList<Shape>
MovingEngine
MovingEngine(String,ArrayList<Shape>,Point,Color,ArrayList<Integer>)
press():ArrayList<Shape>
drag(Point):ArrayList<Shape>
DrawingEngine
DrawingEngine(String,ArrayList<Shape>,Point,Color,ArrayList<Integer>)
press():ArrayList<Shape>
drag(Point):ArrayList<Shape>
ColoringEngine
ColoringEngine(String,ArrayList<Shape>,Point,Color,ArrayList<Integer>)
press():ArrayList<Shape>
drag(Point):ArrayList<Shape>
Command
Command()
execute():ArrayList<Shape>
SaveJSON
filePath: String
SaveJSON(ArrayList<Shape>,String)
execute():ArrayList<Shape>
History
determinator: String
History(String,ArrayList<Shape>)
execute():ArrayList<Shape>
undo():void
redo():void
saveHistory():void
LoadJSON
file: File
error: boolean
LoadJSON(String)
isError():boolean
execute():ArrayList<Shape>
LoadXML
documentBuilder: DocumentBuilder
file: File
error: boolean
LoadXML(String)
execute():ArrayList<Shape>
SaveXML
documentBuilderFactory: DocumentBuild...
documentBuilder: DocumentBuilder
filePath: String
SaveXML(ArrayList<Shape>,String)
execute():ArrayList<Shape>
Load
execute():ArrayList<Shape>
Save
execute():ArrayList<Shape>
IShape
getCenter():Point
setBorderColor(Color):void
getBorderColor():Color
setFillColor(Color):void
getFillColor():Color
draw(Graphics2D):void
getShapeAttributes():ArrayList<Shape>
setIsSelected(Boolean):void
getIsSelected():Boolean
contains(Point):boolean
isOnBorder(Point):boolean
isOnLine(Point,Point,Point):boolean
updateDrawing(Point,Point):void
movingShape(Point):void
resizingShape(Point,int):void
getFixedCorner(Integer):void
distance(Point,Point):Double
getType():String
getLength():Double
getWidth():Double
setWidth(Double):void
setLength(Double):void
getPointsArray():ArrayList<Point>
setPointsArray(ArrayList<Point>):void
clone():Shape
Shape
borderColor: Color
fillColor: Color
center: Point
isSelected: Boolean
fixedCorner: Point
type: String
length: Double
width: Double
pointsArray: ArrayList<Point>
epsilon: Double
Shape()
getShapeAttributes():ArrayList<Shape>
getCenter():Point
fillArrayAttributes():ArrayList<Shape>
setBorderColor(Color):void
getBorderColor():Color
setFillColor(Color):void
getFillColor():Color
draw(Graphics2D):void
contains(Point):boolean
updateDrawing(Point,Point):void
setIsSelected(Boolean):void
getIsSelected():Boolean
movingShape(Point):void
resizingShape(Point,int):void
getFixedCorner(Integer):void
distance(Point,Point):Double
getType():String
getLength():Double
getWidth():Double
getPointsArray():ArrayList<Point>
setPointsArray(ArrayList<Point>):void
setWidth(Double):void
setLength(Double):void
isOnBorder(Point):boolean
isOnLine(Point,Point,Point):boolean
clone():Shape
Polygon
Polygon()
getCenter():Point
draw(Graphics2D):void
contains(Point):boolean
updateDrawing(Point,Point):void
movingShape(Point):void
resizingShape(Point,int):void
getFixedCorner(Integer):void
clone():Shape
isOnBorder(Point):boolean
Ellipse
fixedPointOnResizing: Point
Ellipse(Point,Double,Double,Color)
getCenter():Point
draw(Graphics2D):void
contains(Point):boolean
updateDrawing(Point,Point):void
fillArrayAttributes():ArrayList<Shape>
movingShape(Point):void
resizingShape(Point,int):void
getFixedCorner(Integer):void
clone():Shape
isOnBorder(Point):boolean
FreeShape
FreeShape(ArrayList<Point>,Color)
getCenter():Point
draw(Graphics2D):void
contains(Point):boolean
updateDrawing(Point,Point):void
movingShape(Point):void
resizingShape(Point,int):void
getFixedCorner(Integer):void
isOnBorder(Point):boolean
clone():Shape
Rectangle
Rectangle(Point,Color,double,double)
getCenter():Point
contains(Point):boolean
updateDrawing(Point,Point):void
resizingShape(Point,int):void
getFixedCorner(Integer):void
clone():Shape
Circle
Circle(Point,Double,Color)
updateDrawing(Point,Point):void
resizingShape(Point,int):void
clone():Shape
Square
Square(Point,Color,double)
updateDrawing(Point,Point):void
resizingShape(Point,int):void
clone():Shape
-fileChooserHandler 0..1
-engine 0..1
-memoryInvoker 0..1
-command 0..1
#shapesOnPanel 0..*
-array 0..*
-loadedArrayList 0..*
-savedArrayList 0..*
#shapes 0..*
#shapeAttributes 0..*