SSP- Term 4

Programming II

Paint Project

Report

Done By:

Mohamad Hisham Gaballah 4683

Aya Kandil 4699

Monica Charles 4705

Hana Hesham Nazmy 4952

Paint Program Using Java

Our paint project aims to facilitate the drawing operation and make it fun to all users even to the ones who don't know how to draw by choosing from a variety of shapes. Applying OOP principles using JAVA, we designed our painting project to let the users free and create their awesome designs easily. To save shapes drawn by the user, he can save them to XML and JSON file to load them from at his next entry.

Features

Available shapes: circle, square, ellipse, line segment, rectangle and triangle.

Available operations: draw a shape, draw multiple shapes, change border color, change fill color, copy, move, resize, delete, select or deselect a shape, select or deselect multiple shapes, undo and redo.

Available type of files: XML and JSON.

Design Overview:

To preserve OOP concepts, we separated our classes into 3 packages: paint.model, paint.controller, paint.view to apply MVC.

Model consists of: Abstract class Myshape which implements Shape interface, classes of required shapes which extend MyShape. Each shape has its unique properties put in its map.

Controller consists of: singleton Engine class implementing DrawingEngine interface, Memento class and CareTaker contains list of shapes, Command interface which is implemented by Undocommand and RedoCommand. We also created State interface to handle important operations as editing, drawing, saveLoad and directState by creating EditingState, DrawingState, SaveLoadState and DirectState.

View consists of GUI which contains panel (where user will draw) and buttons with icons to have a user-friendly interface.

Assumptions:

- The user must choose a specific shape then drag and drop it.
- The user must select one or multi shapes before editing.
- When a shape is selected, its border line becomes dashed and its fill color turns into white.

Teamwork:

Mohamed: draw, resize, select method, deselect method, Strategy DP.
Aya: Strategy DP, State DP, Command DP, Memento DP.
Hana: implementation of paint.model , move, copy method, select line, Memento DP, use case, class diagram, package diagram.
Monica: implementation of paint.model, factory DP, singleton DP and GUI.

Data Structure:

- Linked List: (3) for all shapes and for selected ones and another one is used in JSON.
- Map where key is String and value is Double for every Shape.
- Stack already implemented in java : (2) used in redo and undo.

Description of the important functions/modules:

(A)Create a Shape

Factory DP:

This type of design pattern comes under creational pattern as this pattern provides one of the best ways to create an object. By defining an abstract class, MyShape, its subclasses decide which class to instantiate. In other words, subclasses are responsible to create the instance of the class. ShapeFactory Class is used to get a Shape object. Information of the Shape (Point begin, Point last, String type) will be passed to ShapeFactory. In ShapeFactory Class, according to the information passed, a shape will be created.

(B)Engine Class

Singleton DP:

Singleton pattern comes under creational pattern as this pattern provides one of the best ways to create an object.

This pattern involves a single class, Engine, which is responsible to create an object while making sure that only single object gets created. Engine Class provides a way to access its only object which can be accessed directly without need to instantiate the object of the class. Engine class has its constructor as private and has a static instance of itself. Engine Class provides a static method to get its static instance to outside world. All state classes will use Engine class to get an Engine object but it will remain pointing on only one instance.

Engine Class includes many important methods:

1) refresh: draws all shapes in selectedShapes linked list and shapeList linked list.
2) addShape: adds the new drawn shape to the shapeList and adds a memento in the caretaker.
3) draw: passes the begin point, last point and shape type to the factory and receives a shape, draws it and refreshes the canvas.
4) select: takes a point as a parameter, loops on all shapes in the shapeList, if a shape contains this point then add it to the selectedShapes linked list and remove it from the shapeList. Also add memento to the caretaker.
5) deselect: takes a point as a parameter, loops on all shapes in the selectedShapes linked list, if a shape contains this point then add it to the shapeList linked list and remove it from the selectedShapes. Also add memento to the caretaker.
6) deselectAll: loops on all shapes in the selectedShapes linked list, then add all the shapes to the shapeList linked list and clear the selectedShapes. Also add memento to the caretaker.
7) removeShape: removes the shape from the selectedShapes linked list.
8) updateShape: removes the old shape from the selectedShapes linked list and adds the new shape.
9) removeSelected: clears the selectedShapes list, deletes these shapes.

10) moveShape: loops on all shapes in the selectedShapes linked list, know the instance of the selectedShape(s) and set its position according to the difference between the 2 points,the begin and end, then let the begin point equals the end point.
11) resize: loops on all shapes in the selectedShapes linked list, know the instance of the selectedShape(s) and set the dimensions/properties of the shape according to the difference between the 2 points, then let the begin point equals the end point.
12) cloneShape: loops on all shapes in the selectedShapes linked list, know the instance of the selectedShape(s), let cloned shape equals the selected shape, set the position of the cloned shape to (0,0) and add the cloned shape to the shapeList.
13) colorInside: loops on all shapes in the selectedShapes linked list and set the fill color to the color chosen. Also add memento to the caretaker.
14) colorBorder: loops on all shapes in the selectedShapes linked list and set the border color to the color chosen . Also add memento to the caretaker.
15) saveNewListM: takes the shapeList and selectedShapes and returns a new memento.
16) getShapesFromMemento: clears the shapeList, gets a state of the last shapeList put in memento and add all shapes to the shapeList, clears the selectedShapes, gets another state of the last selectedShapes put in memento and add all shapes to the selectedShapes.
17) undo: creates a an instance of Undocommand, calls the command execute, returns the last memento by calling getShapesFromMemento method.
18) redo: creates a an instance of Redocommand, calls the command execute, returns the last memento by calling getShapesFromMemento method.
19) save: checks the saving file type that the user chose and creates a new instance of the file type class, either xml or json, and calls another save method, exists in the json/xml Class, that takes the shapeList as a parameter.

20)load: checks the saving file type that the user saved it, creates a new instance of the file type class, either xml or json, and calls another save method, exists in the json/xml Class, that returns a shapeList.

(C)Get The ActionCommand from GUI

State DP:

The State pattern allows an object to change its behavior when its internal state changes. The state changes according to the ActionCommand implemented in MyActionListener Class .

There is a State Interface that includes methods and defines actions that change the current state. There are 4 Concrete Classes that implement the interface. The first Concrete Class is the DrawingState which is responsible for changing the state of shapes by drawing a shape once the mouse is released. The second Concrete Class is the ColorState which is responsible for changing the state of the border color and the fill color. The third Concrete Class is the DirectState which is responsible for changing the state of shapes by doing an undo,redo,delete and copy. The fourth Concrete Class is the EditingState which is responsible for changing the state of shapes by making a resize, move, select and deselect.
There is a Context Class, GUIState , that carries, gets and sets the current state according to the buttons pressed in the GUI.

(D)Undo/Redo

Memento DP:

Memento DP is mainly used in order to restore state of drawings to the previous state. In our paint program, we used the memento by applying it to let the user undo and redo up to 20 times.

We created 3 classes for this DP. First, the Memento Class, which saves the state of the drawn shapes in a linked list and the state of the selected shapes in another linked list. Then, we can get (restore) the state of any linked list whenever we want. Second, the CareTaker Class, which is responsible of restoring states from Memento Class. It includes 2 stacks: one for undo and the other for redo. It is responsible of adding a Memento (new state), undoing and redoing. The Undo/Redo operation is limited to 20 steps, so we check the undo stack before adding a memento if it has 20 elements then we remove the first one and we check the redo stack before undoing if it has 20 elements then we remove the first one. Third, the Originator Class, which is the Engine Class, that creates and saves states in Memento objects.

Command DP:

There is a “Commander” Interface for executing Undo/Redo operation. There are 2 Concrete Command classes that implement the Commander Interface and implements the execute method. These classes create a binding between the action and the receiver. They also take an instance of CareTaker object from the Engine to perform execute function (either undo or redo). The Invoker is the Engine Class, which decides when to execute. One of the important aspects of Command pattern is creating an interface to isolate the Invoker from the Receiver.

(E)Save and Load

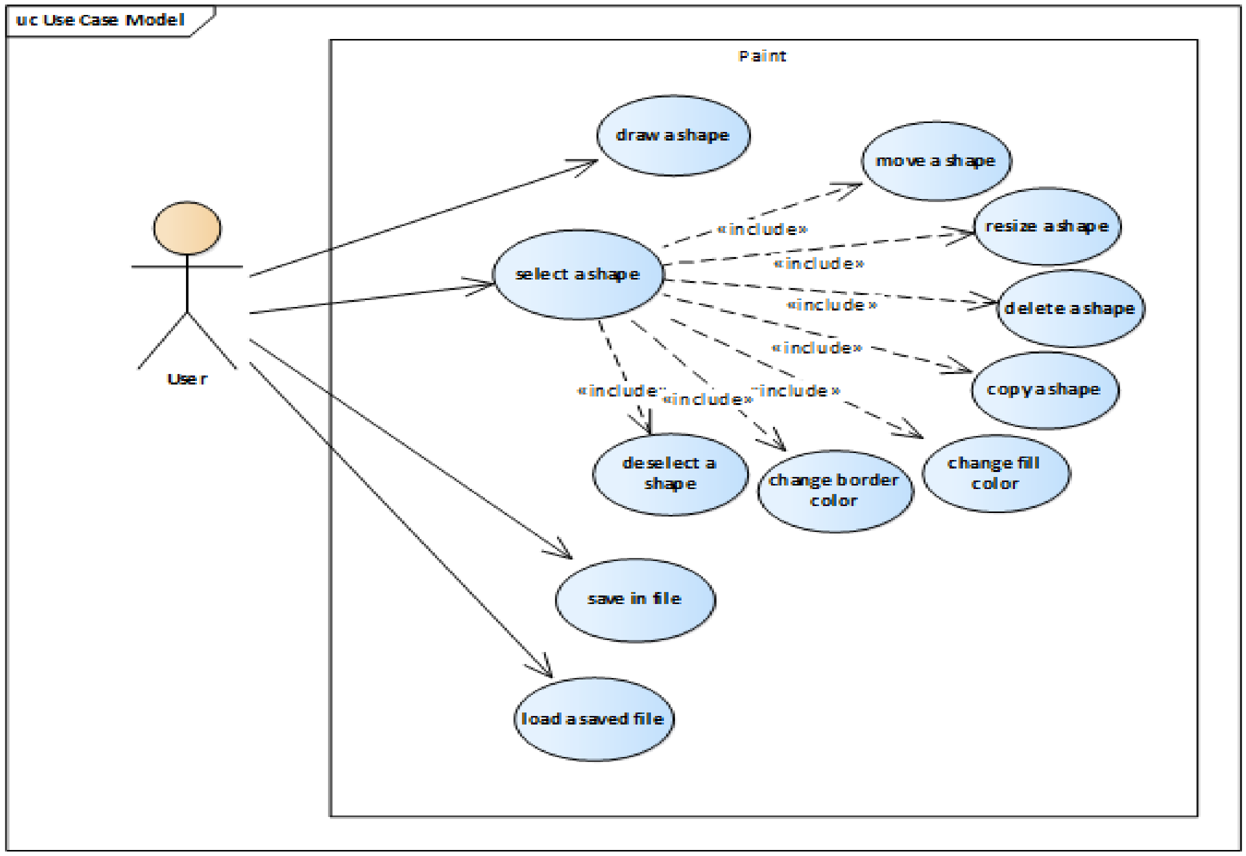
Strategy DP:

This pattern provides a choice of implementations of algorithms for save and load, eliminates large conditional statements that could be written.

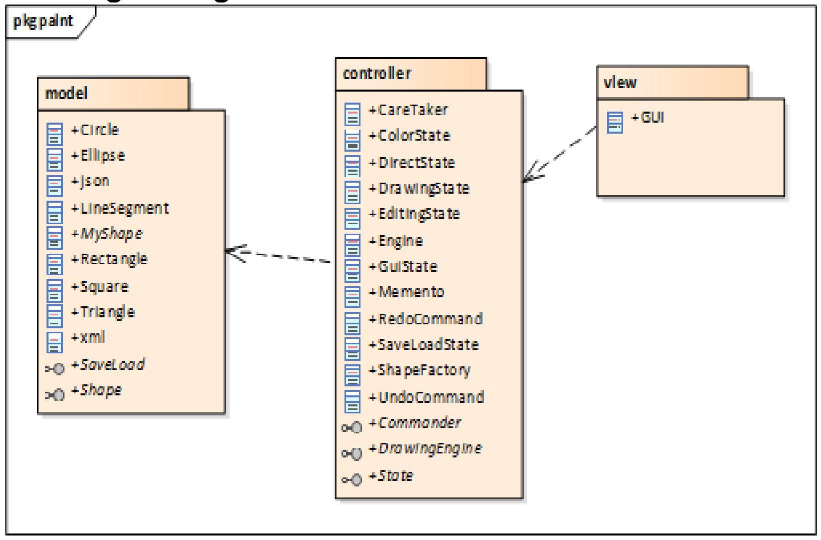
There is an interface, SaveLoad. There are 2 ConcreteStrategy Classes, json and XML, that implement SaveLoad interface. Each ConcreteStrategy Class holds how a file can be saved in a specific algorithm. There is a Context Class, SaveLoadState , that carries, gets and sets the current algorithm for saving according to the buttons pressed in the GUI.

UML Diagrams:

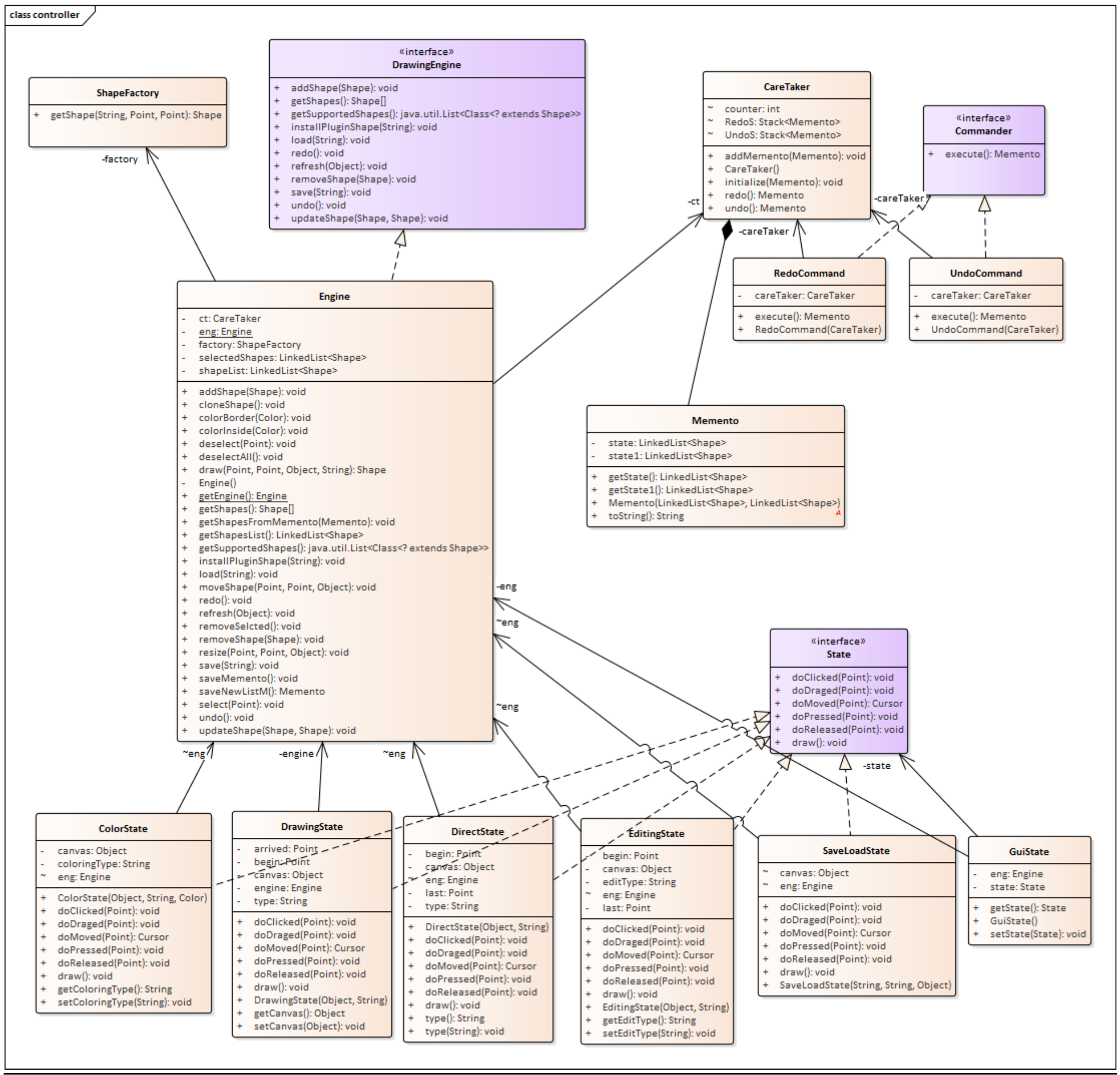
Use Case:

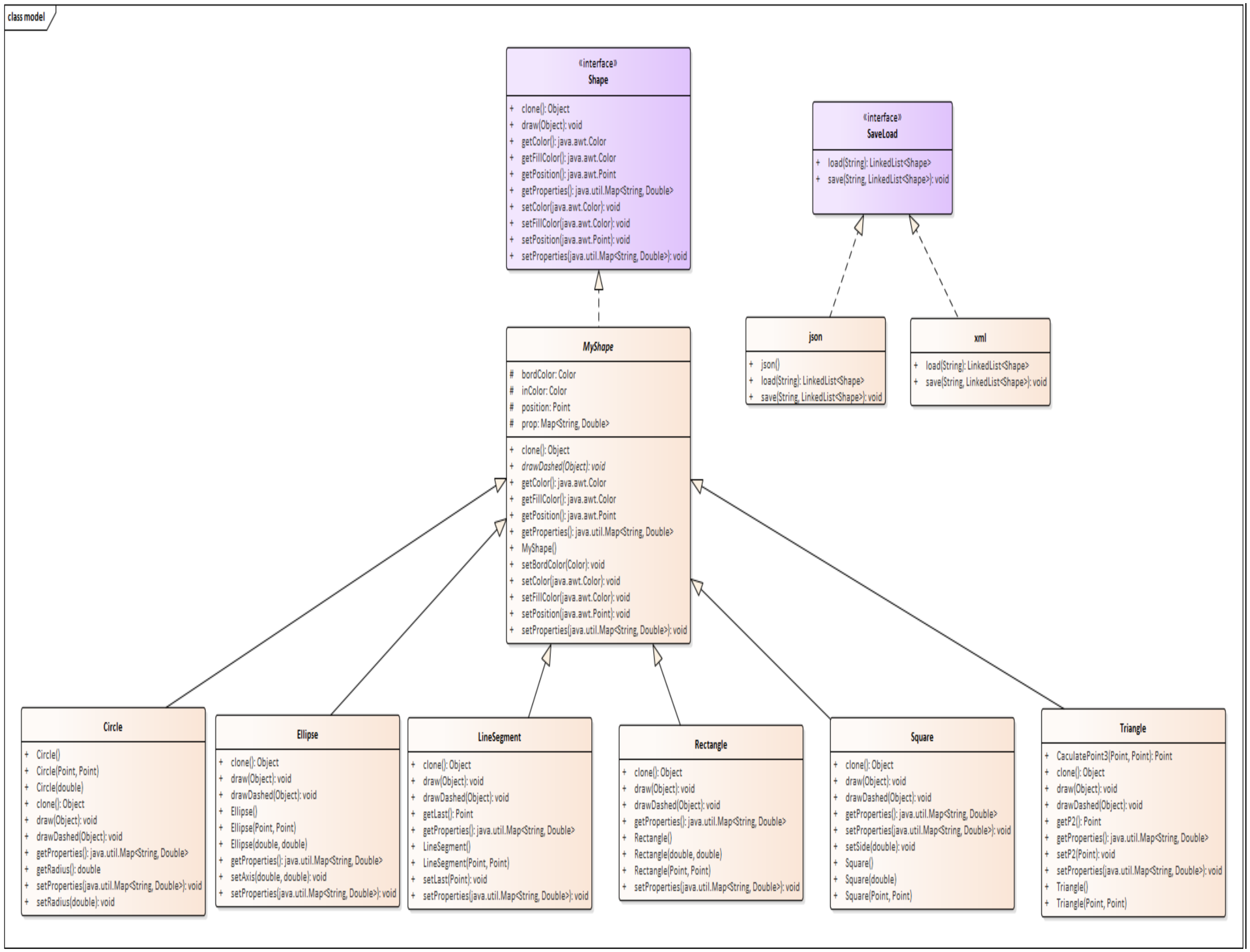


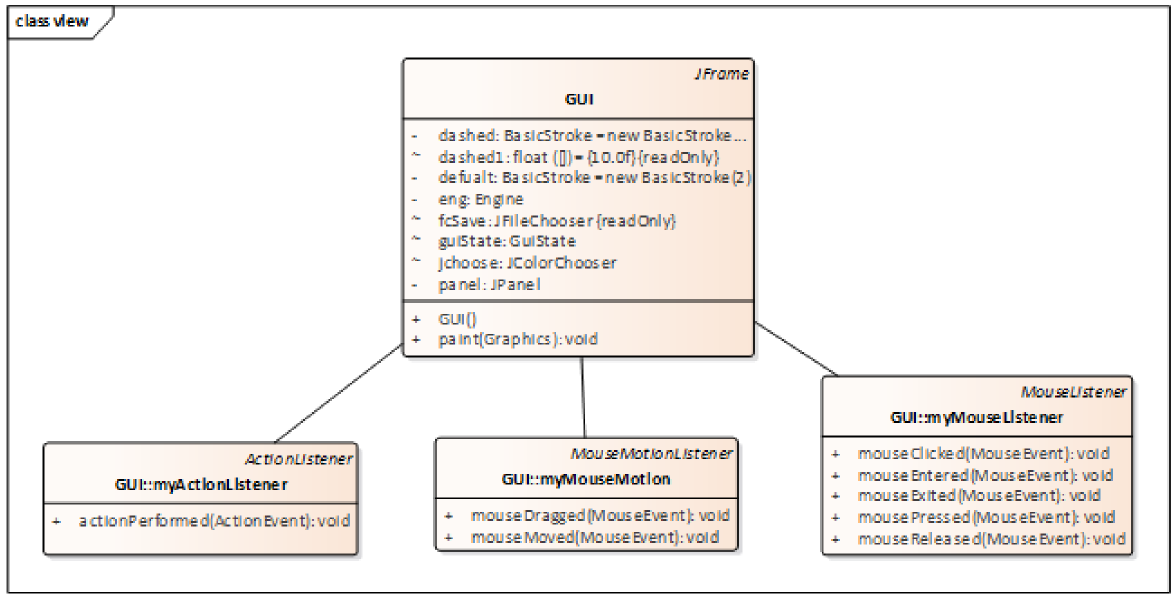
Package Diagram:



Class Diagrams:

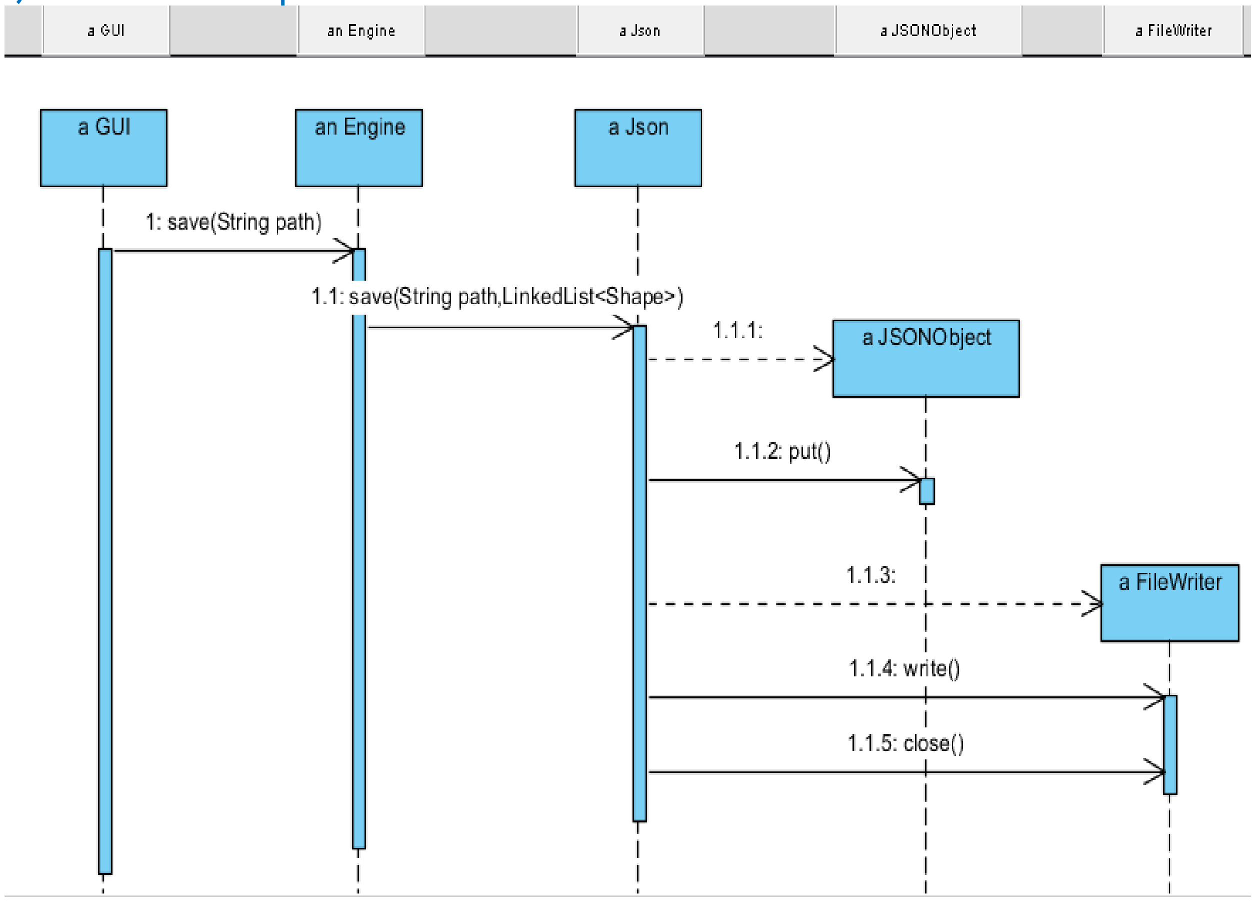



class model
«interface»
Shape
+ clone(): Object
+ draw(Object): void
+ getColor(): java.awt.Color
+ getFillColor(): java.awt.Color
+ getPosition(): java.awt.Point
+ getProperties(): java.util.Map<String, Double>
+ setColor(java.awt.Color): void
+ setFillColor(java.awt.Color): void
+ setPosition(java.awt.Point): void
+ setProperties(java.util.Map<String, Double>): void
«interface»
SaveLoad
+ load(String): LinkedList<Shape>
+ save(String, LinkedList<Shape>): void
MyShape
# bordColor: Color
# inColor: Color
# position: Point
# prop: Map<String, Double>
+ clone(): Object
+ drawDashed(Object): void
+ getColor(): java.awt.Color
+ getFillColor(): java.awt.Color
+ getPosition(): java.awt.Point
+ getProperties(): java.util.Map<String, Double>
+ MyShape()
+ setBordColor(Color): void
+ setColor(java.awt.Color): void
+ setFillColor(java.awt.Color): void
+ setPosition(java.awt.Point): void
+ setProperties(java.util.Map<String, Double>): void
json
+ json()
+ load(String): LinkedList<Shape>
+ save(String, LinkedList<Shape>): void
xml
+ load(String): LinkedList<Shape>
+ save(String, LinkedList<Shape>): void
Circle
+ Circle()
+ Circle(Point, Point)
+ Circle(double)
+ clone(): Object
+ draw(Object): void
+ drawDashed(Object): void
+ getProperties(): java.util.Map<String, Double>
+ getRadius(): double
+ setProperties(java.util.Map<String, Double>): void
+ setRadius(double): void
Ellipse
+ clone(): Object
+ draw(Object): void
+ drawDashed(Object): void
+ Ellipse()
+ Ellipse(Point, Point)
+ Ellipse(double, double)
+ getProperties(): java.util.Map<String, Double>
+ setAxis(double, double): void
+ setProperties(java.util.Map<String, Double>): void
LineSegment
+ clone(): Object
+ draw(Object): void
+ drawDashed(Object): void
+ getLast(): Point
+ getProperties(): java.util.Map<String, Double>
+ LineSegment()
+ LineSegment(Point, Point)
+ setLast(Point): void
+ setProperties(java.util.Map<String, Double>): void
Rectangle
+ clone(): Object
+ draw(Object): void
+ drawDashed(Object): void
+ getProperties(): java.util.Map<String, Double>
+ Rectangle()
+ Rectangle(double, double)
+ Rectangle(Point, Point)
+ setProperties(java.util.Map<String, Double>): void
Square
+ clone(): Object
+ draw(Object): void
+ drawDashed(Object): void
+ getProperties(): java.util.Map<String, Double>
+ setProperties(java.util.Map<String, Double>): void
+ setSide(double): void
+ Square()
+ Square(double)
+ Square(Point, Point)
Triangle
+ CaculatePoint3(Point, Point): Point
+ clone(): Object
+ draw(Object): void
+ drawDashed(Object): void
+ getP2(): Point
+ getProperties(): java.util.Map<String, Double>
+ setP2(Point): void
+ setProperties(java.util.Map<String, Double>): void
+ Triangle()
+ Triangle(Point, Point)

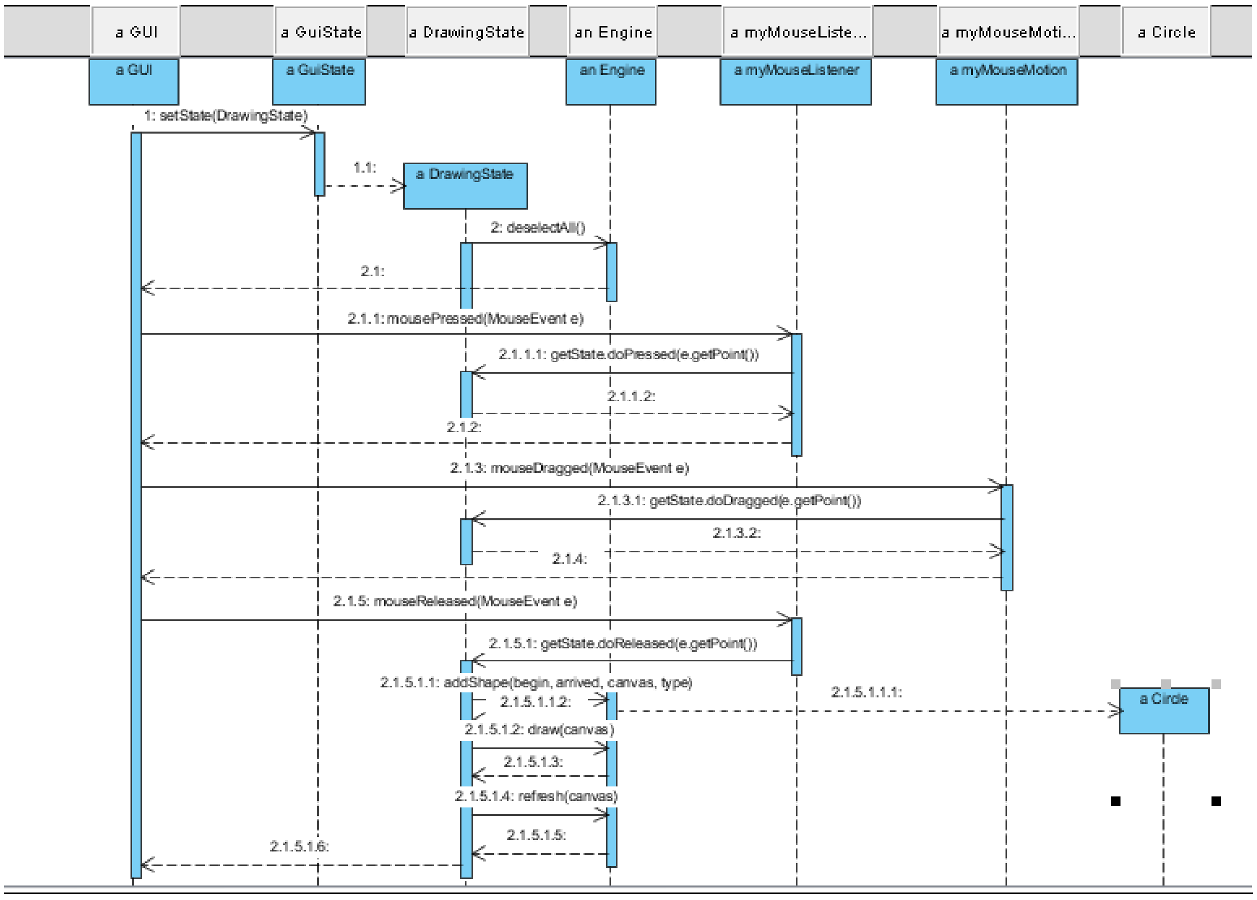

class view
JFrame
GUI
- dashed: BasicStroke = new BasicStroke...
~ dashed1 : float ([]) = {10.0f} {readOnly}
- defualt: BasicStroke = new BasicStroke(2)
- eng: Engine
~ fcSave : JFileChooser {readOnly}
~ guiState: GuiState
~ jchoose: JColorChooser
- panel: JPanel
+ GUI()
+ paint(Graphics): void
ActionListener
GUI::myActionListener
+ actionPerformed(ActionEvent): void
MouseMotionListener
GUI::myMouseMotion
+ mouseDragged(MouseEvent): void
+ mouseMoved(MouseEvent): void
MouseListener
GUI::myMouseListener
+ mouseClicked(MouseEvent): void
+ mouseEntered(MouseEvent): void
+ mouseExited(MouseEvent): void
+ mousePressed(MouseEvent): void
+ mouseReleased(MouseEvent): void

Sequence Diagram:

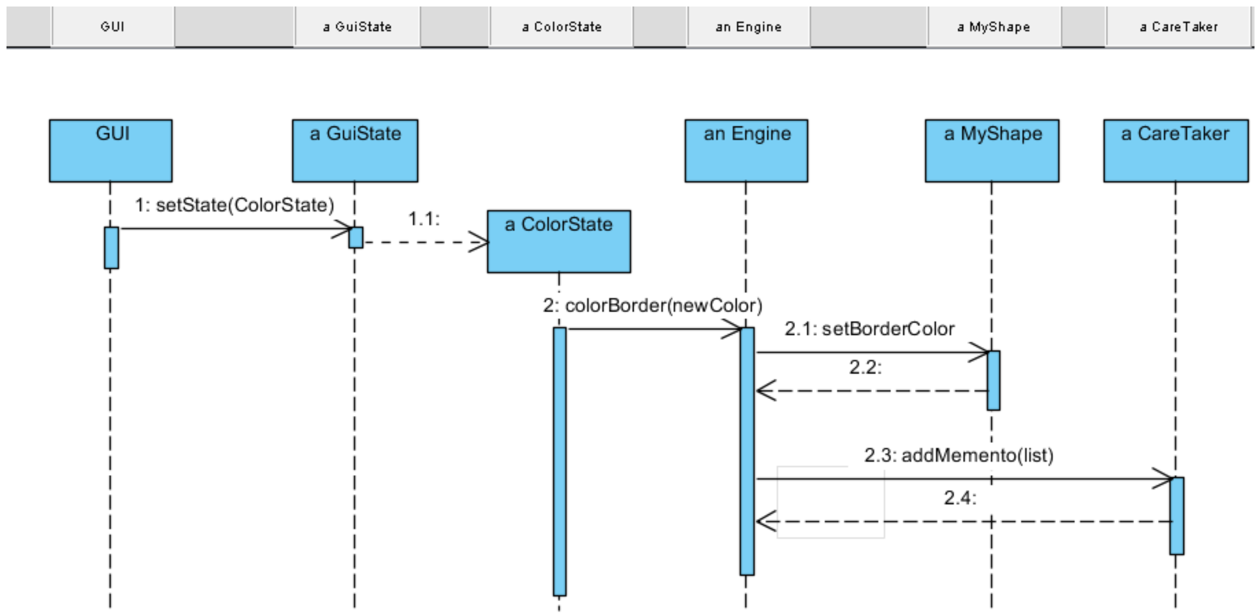
1) Save some shapes to a JSON file:



2) Drawing a Circle:


a GUI
a GuiState
a DrawingState
an Engine
a myMouseListe...
a myMouseMoti...
a Circle
a GUI
a GuiState
an Engine
a myMouseListener
a myMouseMotion
1: setState(DrawingState)
1.1:
a DrawingState
2: deselectAll()
2.1:
2.1.1: mousePressed(MouseEvent e)
2.1.1.1: getState.doPressed(e.getPoint())
2.1.1.2:
2.1.2:
2.1.3: mouseDragged(MouseEvent e)
2.1.3.1: getState.doDragged(e.getPoint())
2.1.3.2:
2.1.4:
2.1.5: mouseReleased(MouseEvent e)
2.1.5.1: getState.doReleased(e.getPoint())
2.1.5.1.1: addShape(begin, arrived, canvas, type)
2.1.5.1.2:
2.1.5.1.1.1:
a Circle
2.1.5.1.2: draw(canvas)
2.1.5.1.3:
2.1.5.1.4: refresh(canvas)
2.1.5.1.5:
2.1.5.1.6:

3) Editing a ellipse:

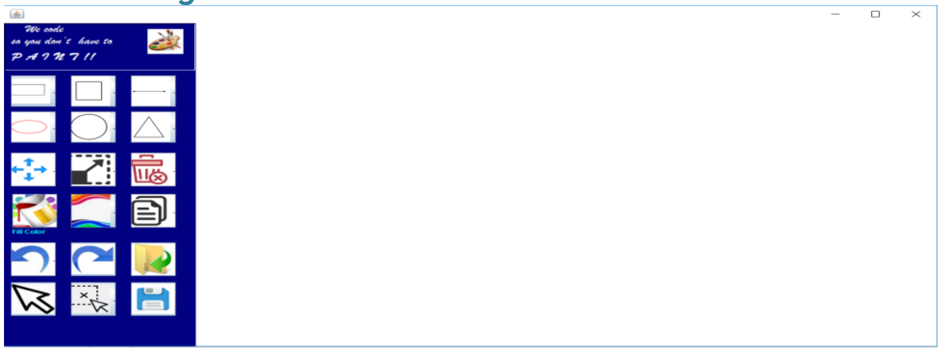


User Manual:

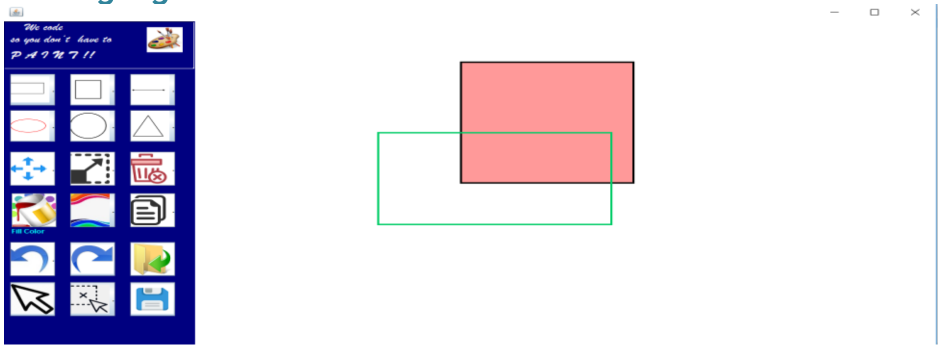
-You can draw a circle, square, ellipse, line segment, rectangle or triangle by drag and drop.
-You must select a shape in order to move, copy, delete, resize or change its color.
-You can select multiple shapes.
-You can save your drawings in a file.
-You can load a saved file.

Sample Runs:

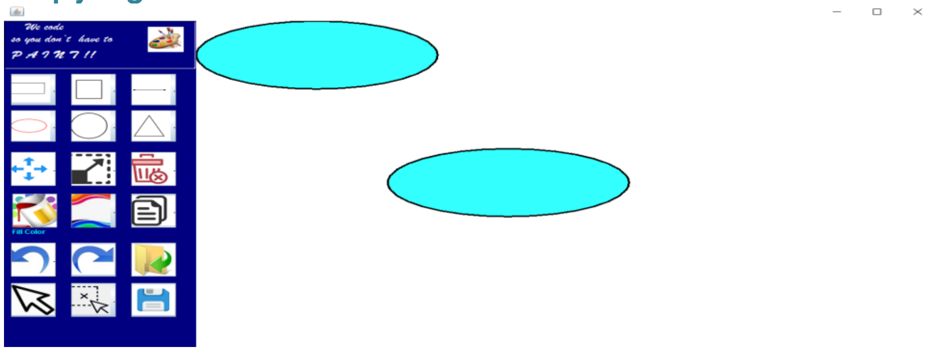
On running:



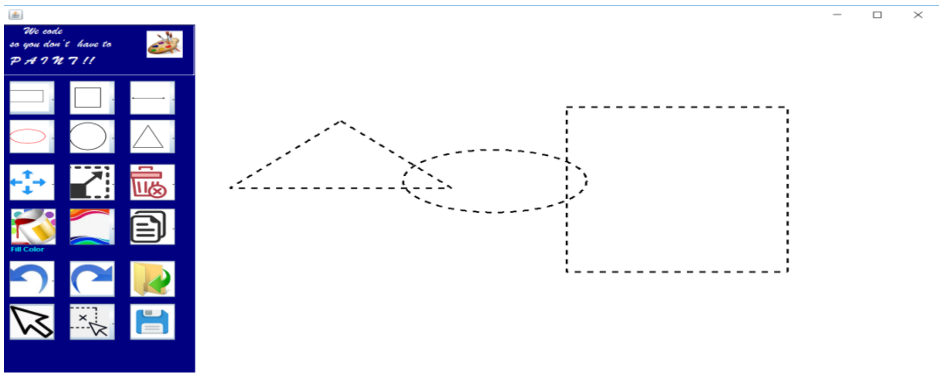
Changing border and fill color:

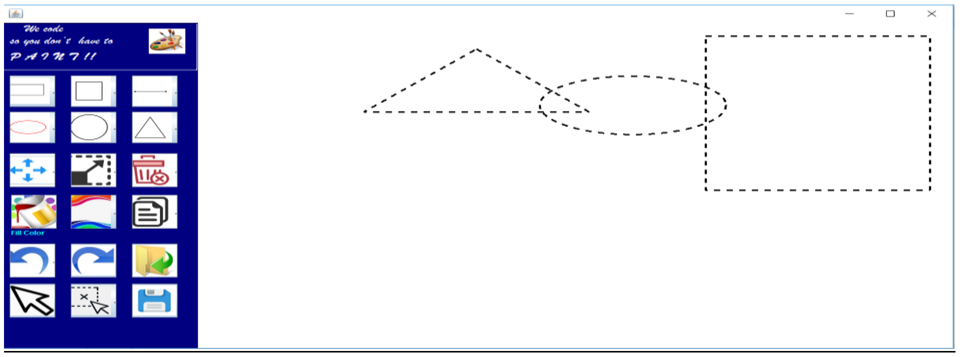


Copying:

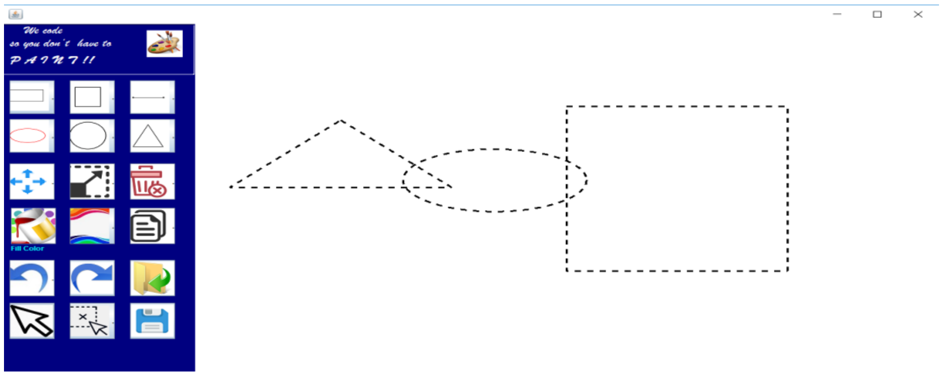


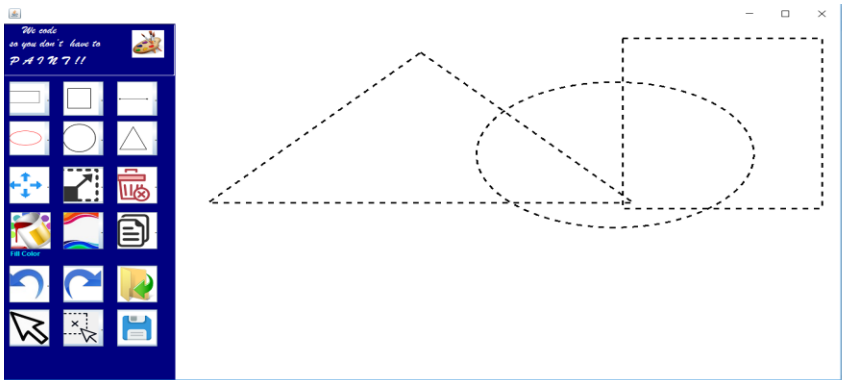
Moving:



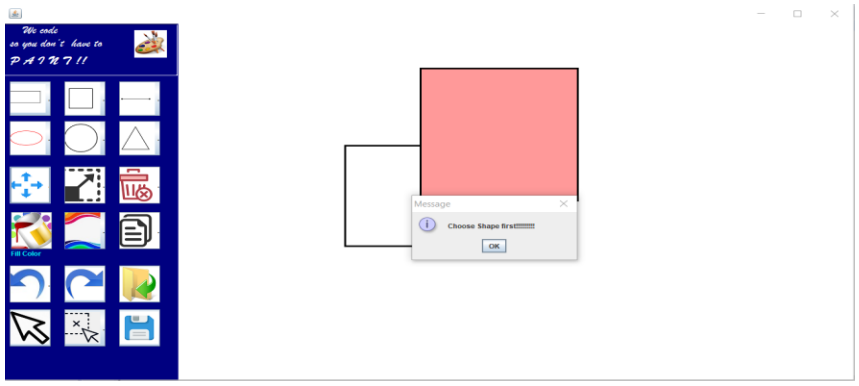


Multi-Select and Resizing:



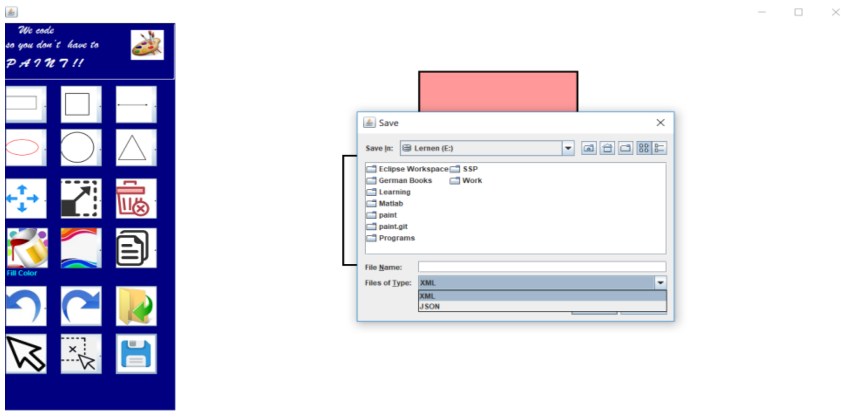


Handling not selecting error:



Choose Shape first!!!!!!!!!!

On saving:



Save In: Lernen (E:)

Eclipse Workspace
German Books
Learning
Matlab
paint
paint.git
Programs
SSP
Work

File Name:

Files of Type: XML

XML
JSON

References:

(1) Design Patterns Explained simply.
(2) https://sourcemaking.com/design_patterns