Programming-2

Assignment 2:
Paint

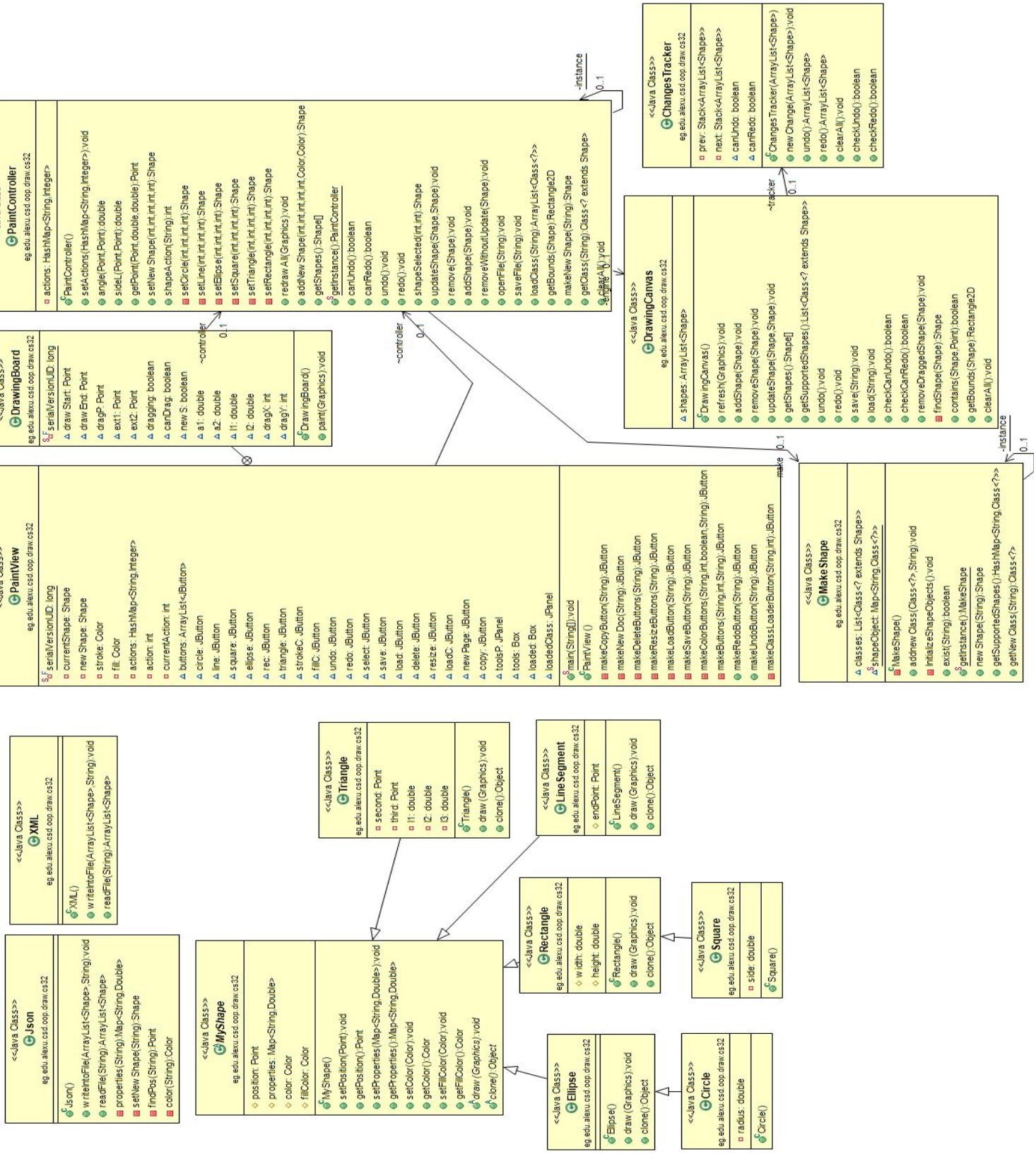
Description of project

Geometric shapes belong to different groups (ex: Elliptical Shapes, Polygons, Sectors...etc.); members of these different groups are related to each other in the sense that they share common properties. To be able to implement an efficient and object-oriented drawing application, it is essential to design a model that takes these relations into consideration.

Problem Statement

Designing a painting application and applying different design patterns suitable for the application. The application should support following geometric shapes: Line Segment, Circle, Ellipse, Triangle, Rectangle and Square. Applying the concepts of inheritance and polymorphism to the design.

UML diagram


PaintController
DrawingBoard
PaintView
DrawingCanvas
ChangesTracker
MakeShape
XML
Json
MyShape
Triangle
LineSegment
Rectangle
Square
Ellipse
Circle

Classes

Json

A Json parser to save and load paints as json file.

Methods:

- public void writeIntoFile(ArrayList<Shape> shapes, String path);

  ->Save the paint in json file format.

- public ArrayList<Shape> readFile(String path);

  ->Read from json file

- private Map<String, Double> properties(String s);

  ->Gets properties of a shape from the file

- private Shape setNewShape(String s);

  ->Makes object from shape saved in the file

- private Point findPos(String s);

  ->Find position of shape from file

- private Color color(String s);

  ->Gets stroke\fill color from file

XML

Save and load shapes in xml files using java.beans

Methods :

- public void writeIntoFile(ArrayList<Shape> shapes, String path);
- public void writeIntoFile(ArrayList<Shape> shapes, String path);

MakeShape

Factory to make shapes.

Methods:

public void addnewClass(Class<?> clasS, String s) ;

->Add new shape class to the supported classes

public boolean exist(String str);

->Check if shape exists in the supported shapes

public Shape newShape(String name) ;

->Create new shape

public HashMap<String, Class<?>> getSupportedShapes() ;

->Return supported shapes' classes

public Class<?> getNewClass(String text);

->Return class of shape

ChangesTracker

public void newChange(ArrayList<Shape> shapes);

->New change in the shapes

It clears the next stack and pushes the shapes to the prev stack then checks if the size of the prev stack exceeded the full capacity or not, (21), if yes it removes the first element

public ArrayList<Shape> undo();

->undo the changes. Pushes the last element in the prev stack to the next stack unless the prev stack's size == 1 then it just return prev.peek.

public ArrayList<Shape> redo();

->redo the changes. Pushes the last element in the next stack to the prev stack unless the next stack's size == 0 , then it just return prev.peek.

public void clearAll() {

->clear all history (after load / new)

public boolean checkUndo() \ checkRedo();

-> checks if the user can redo or undo the changes

DrawingCanvas

public void refresh(Graphics canvas) {

-> Re-draw all the shapes drawn

public void addShape(Shape shape) {

-> add new shape to the canvas

public void removeShape(Shape shape);

-> remove shape from canvas

public void updateShape(Shape oldShape, Shape newShape);

-> update old shape with a new change

public Shape[] getShapes();

-> return the drawn shapes

public List<Class<? extends Shape>> getSupportedShapes();

-> Gets the classes of the shapes currently supported in the runtime

public void undo();

-> undo the changes

public void redo();

public void save(String path);

-> save xml, json files

public void load(String path);

-> Load xml, jason files

public boolean checkCanUndo()\ checkCanRedo();

-> Check if can undo or redo

public void removeDraggedShape(Shape currentShape);

-> removing without updating the history

private java.awt.Shape findShape(Shape sh);

->Gets the graphics 2D shape

public boolean contains(Shape sh, Point p);

->check if shape contains point

public void clearAll() ;

->Reset everything

PaintView (GUI)

The gui view (UI).The drawing is done by dragging except for the dynamically uploaded shapes.

PaintController

A link between PaintView and the implemented shapes\engine

public void setActions(HashMap<String, Integer> a) ;

->Set every shape type with the action ID

public double angle(Point p1, Point p2);

->The slope

public double sideL(Point p1, Point p2);

->distance between two points

public Point getPoint(Point p1, double a1, double l1);

->Gets point on a line knowing the distance from another point in the line and the slope

public Shape setNewShape(int n, int x1, int y1, int x2, int y2);

public int shapeAction(String string);

->gets shape action ID

private Shape setCircle(int x1, int y1, int x2, int y2) ;

private Shape setLine(int x1, int y1, int x2, int y2) ;

```
private Shape setEllipse(int x1, int y1, int x2, int y2) ;

private Shape setSquare(int x1, int y1, int x2, int y2) ;

private Shape setTriangle(int x1, int y1, int x2, int y2);

 private Shape setRectangle(int x1, int y1, int x2, int y2);

 public void redrawAll(Graphics g);

 public Shape addNewShape(int n, int x1, int y1, int x2, int y2, Color fill,
Color stroke);

 public Shape[] getShapes();

 public boolean canUndo()\canRedo();

 public void undo()/redo();

 public Shape shapeSelected(int x, int y);

        ->gets the shape that conatins that point

 public void updateShape(Shape currentShape, Shape newShape);

        ->update shape onthe canvas

 public void remove(Shape currentShape);

        ->remove  new shape from the canvas

 public void addShape(Shape newShape);

        ->add new shape to the canvas

 public void removeWithoutUpdate(Shape currentShape) {

        ->remove without updating history

 public void openFile(String path);

        -> load file

 public void saveFile(String path);

        ->save file

 public ArrayList<Class<?>> loadClass(String path) ;

        ->dynamic load class
```

public Shape makeNewShape(String simpleName) ;

->Make shape.

public Class<? extends Shape> getClass(String text) ;

public void clearAll() ;

->Re-initialize everything.

MyShape

Implements Shape interface

Sub-Classes

-LineSegment

- Circle

- Ellipse

- Rectangle

- Square

- Triangle

Data-structures

ArrayLists<> { shapes, supportedShapes, dynamically loaded classes}.

HashMaps<>{shape properties, gui actions, classes}

Stacks {Undo \ Redo}

UNDO \ REDO mechanism

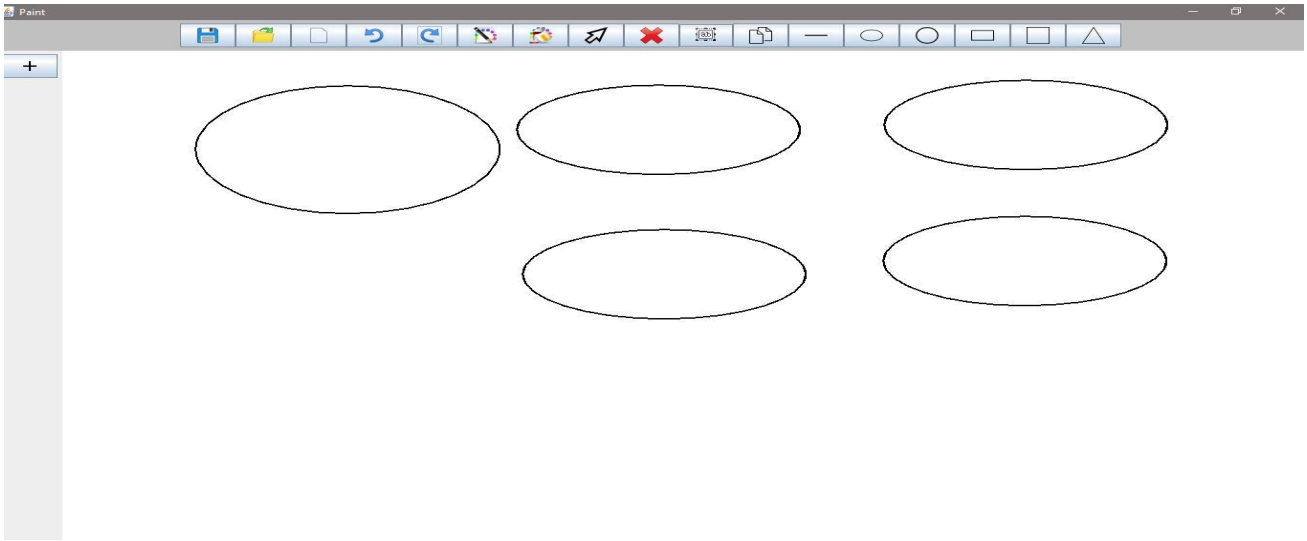
I used stacks to keep track of any changes that happen to any of the shapes during drawing.

Two stacks :

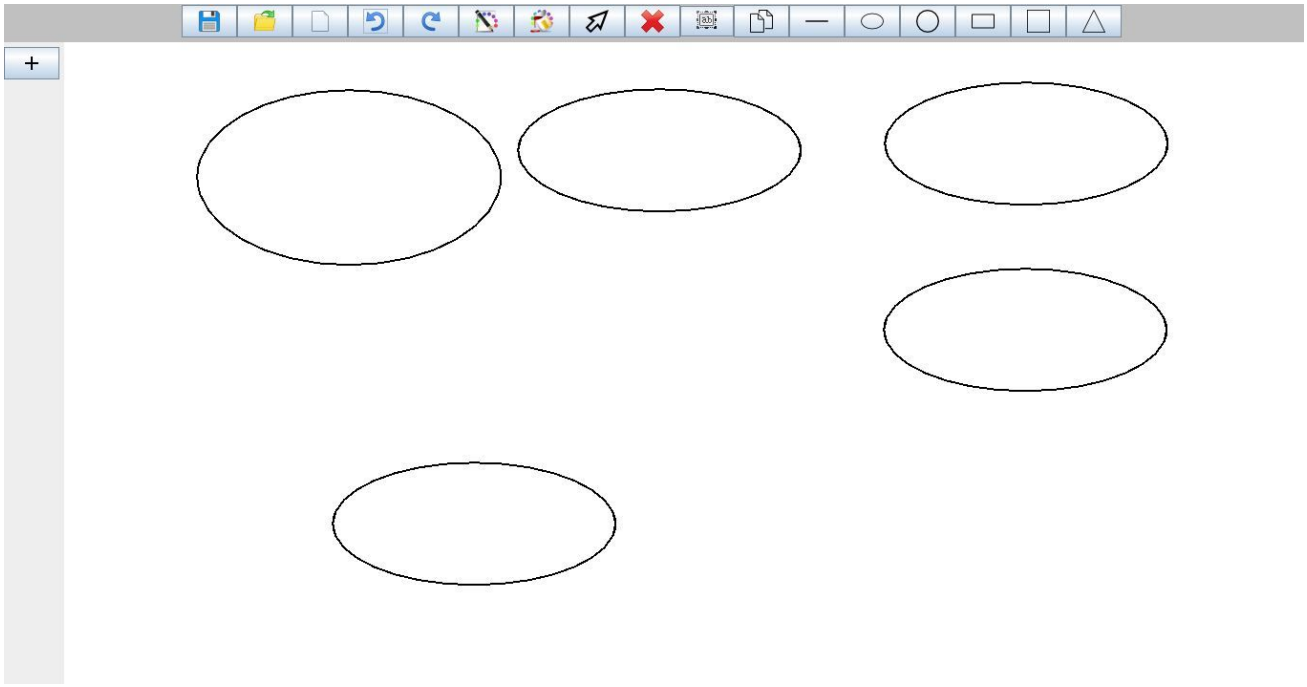
Prev -> The undo history and its last index is the current shapes drawn

Next -> The redo changes.

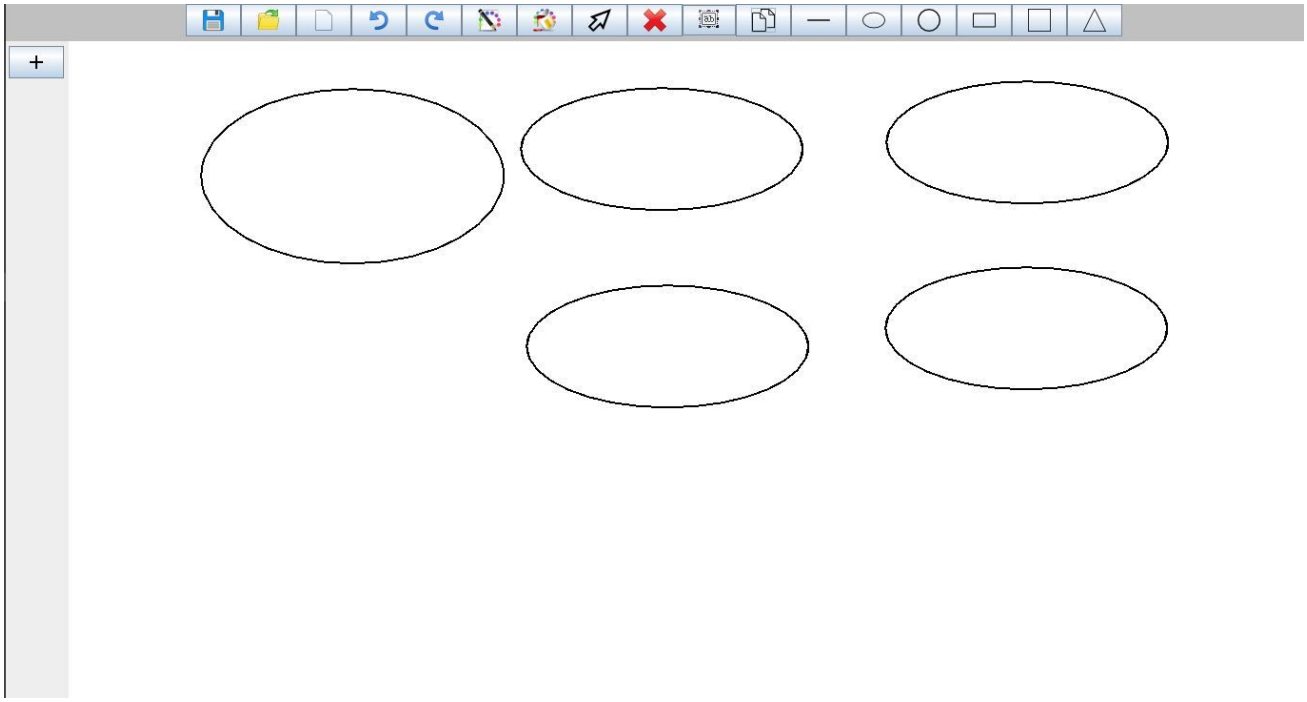
Shapes



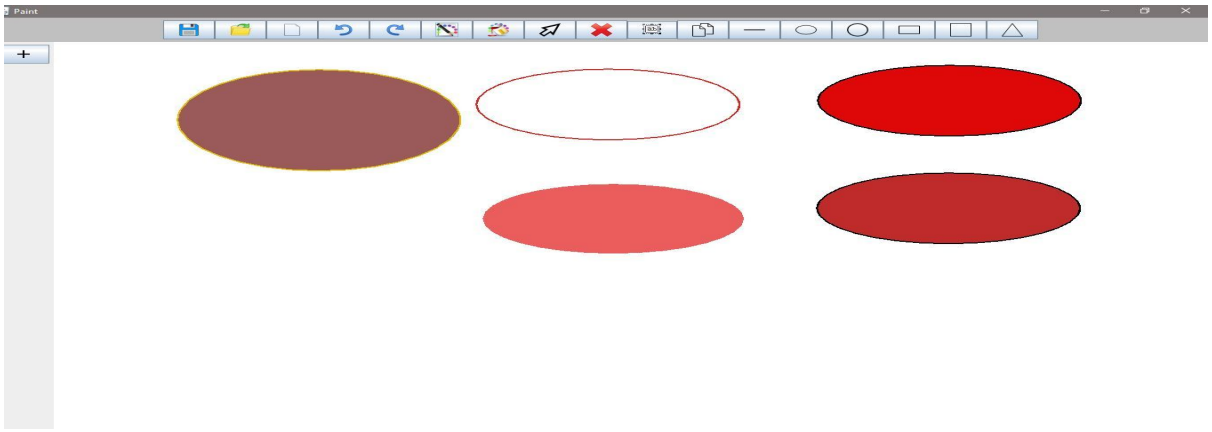
After undo

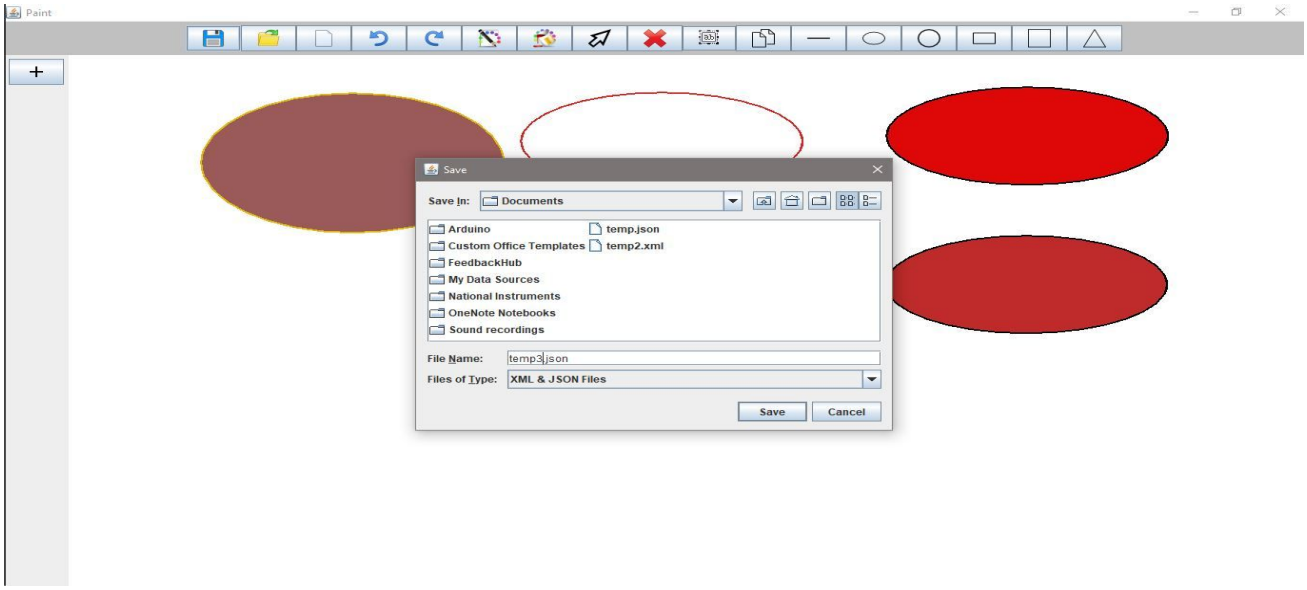


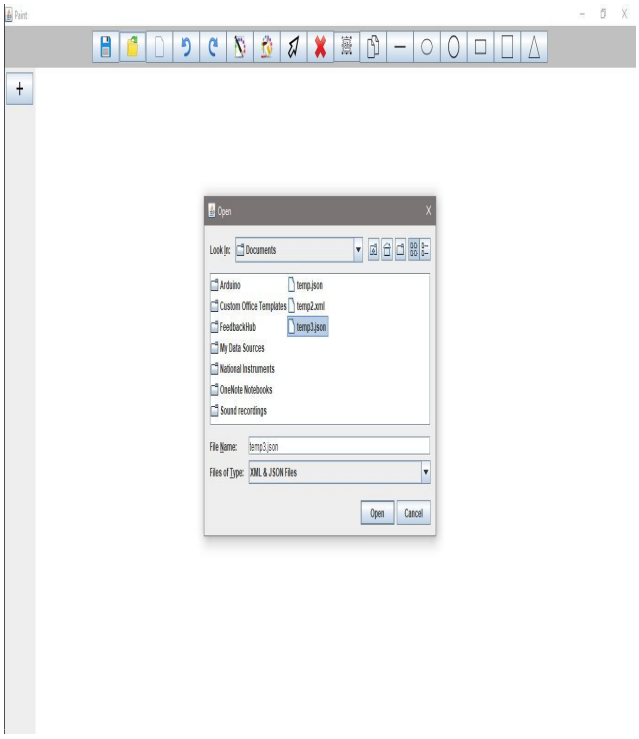
After pressing redo again

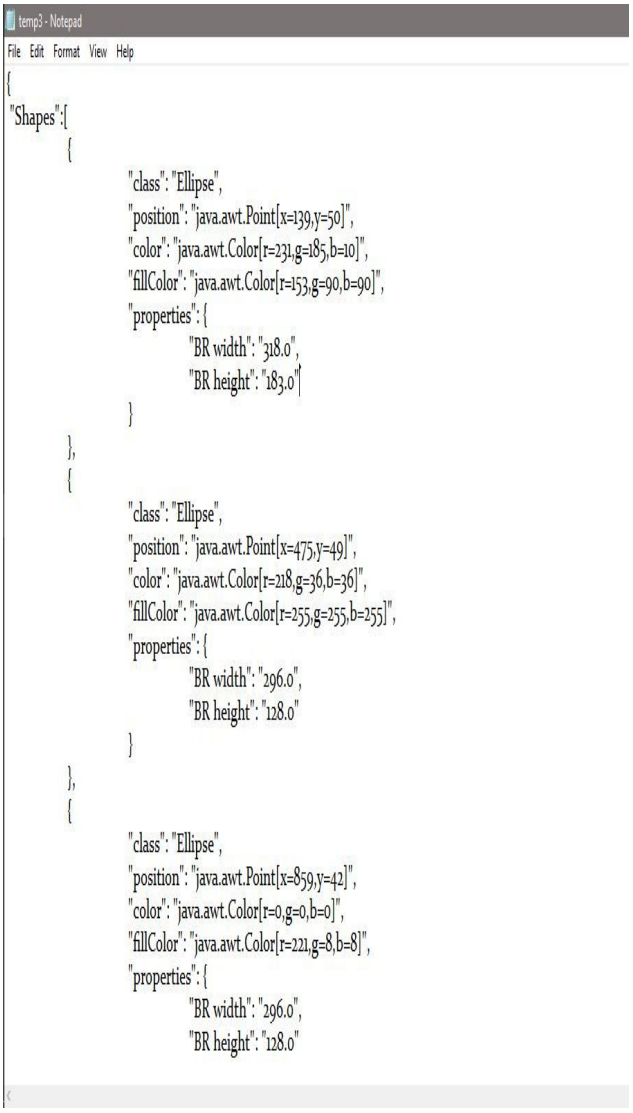


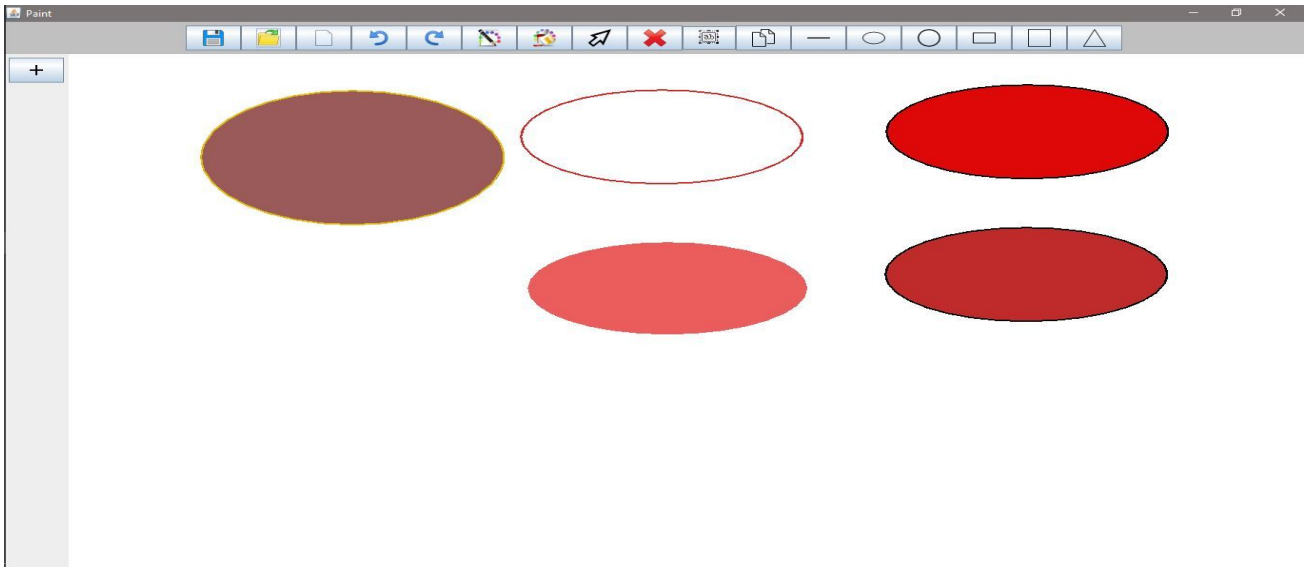
Save JSON :



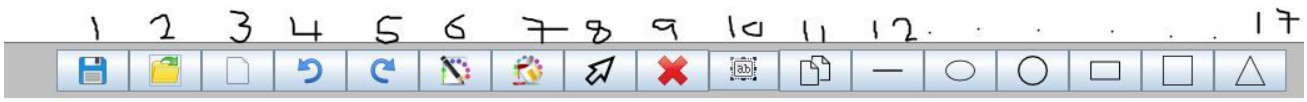

Paint
Save
Save In:
Documents
Arduino
Custom Office Templates
FeedbackHub
My Data Sources
National Instruments
OneNote Notebooks
Sound recordings
temp.json
temp2.xml
File Name:
temp3.json
Files of Type:
XML & JSON Files
Save
Cancel


Paint
Open
Look In: Documents
Arduino
Custom Office Templates
FeedbackHub
My Data Sources
National Instruments
OneNote Notebooks
Sound recordings
temp.json
temp2.xml
temp3.json
File Name: temp3.json
Files of Type: XML & JSON Files
Open
Cancel


temp3 - Notepad
File Edit Format View Help
{
"Shapes":[
{
"class": "Ellipse",
"position": "java.awt.Point[x=139,y=50]",
"color": "java.awt.Color[r=231,g=185,b=10]",
"fillColor": "java.awt.Color[r=153,g=90,b=90]",
"properties": {
"BR width": "318.0",
"BR height": "183.0"
}
},
{
"class": "Ellipse",
"position": "java.awt.Point[x=475,y=49]",
"color": "java.awt.Color[r=218,g=36,b=36]",
"fillColor": "java.awt.Color[r=255,g=255,b=255]",
"properties": {
"BR width": "296.0",
"BR height": "128.0"
}
},
{
"class": "Ellipse",
"position": "java.awt.Point[x=859,y=42]",
"color": "java.awt.Color[r=0,g=0,b=0]",
"fillColor": "java.awt.Color[r=221,g=8,b=8]",
"properties": {
"BR width": "296.0",
"BR height": "128.0"


Paint

User guide:



Button 1 :

Save json \ xml file

Button 2 :

Open json\xml files

Button 3 :

New blank page

Button 4:

Undo

Button 5 :

Redo

Button 6:

Select stroke color

To change shape outline color select the color first then select the shape

Button 7 :

Select fill color

To change shape fill color select the color first then select the shape

Button 8:

Select any shape on the canvas

Hold and drag to move the shape

Button 9:

Delete any shape

First you must select the shape then click the button to delete it

Button 10 :

Resize shape

First you must select the shape then click the button to delete it

Button 11 :

Copy and paste shape
The shape must be selected first

Button 12 to 17 :

Different shapes to be drawn

 --> To dynamic load any jar file.