

**Alexandria University**

**Faculty of Engineering**

**Computer and Systems Engineering Department**

**Mathematics for computer science**

Object Oriented Programming

Paint

**Professor:**

**Professor Khaled Nagy**

**TA:**

**Shehab**

**Poula**

**Team Members:**

**Febronia Ashraf(29)**

**Neveen Samir(58)**

****

**-Problem statement:**

Design an object oriented model for geometric shapes :

1- Design an object oriented model that covers the following geometric shapes: Line Segment, Circle, Ellipse, Triangle, Rectangle and Square.

2- The application would allow the user to undo or redo any action performed.

3- The cursor should be used to select the location of a shape while drawing it, or moving it to another location, for more accurate control on the shape parameters (ex: size), dialog boxes could be used, or you are free to implement it in a more user friendly way of your choice.

4- The application allows for selecting the class library file.

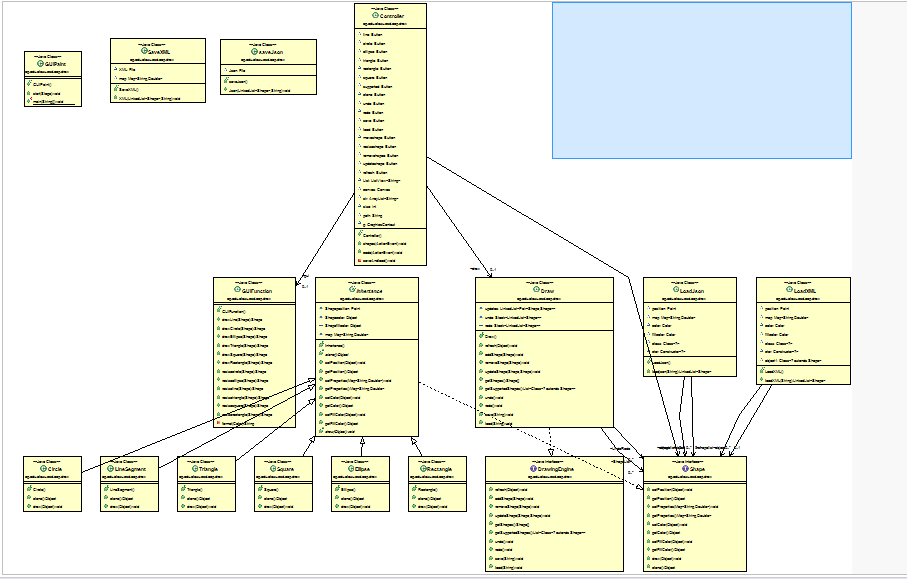
5- On selecting and loading the file, the isolated shape should be appended to the available list of shapes in the application.

6- The User can save the drawing in XML and JSON file.

7- The Application Provide an option to load previously saved drawings and modify the shapes.

8- The Application allows the User to choose where to save the file.

**-UML Digram:**

****

**- Algorithms used documented using pseudo code:**

Function For refresh:

**public** **void** refresh(Object canvas) {

**for** (**int** i = 0🡪 LinkedList.size()) {

LinkedList.get(i).draw;

}

}

Function For add Shapes :

**public** **void** addShape(Shape shape) {

**if** (StackRedo.size() == 20) {

StackUndo.clear();

}

**if** (LinkedList.size() != 0) {

StackUndo.push(**new** LinkedList<Shape>(LinkedList));

}

LinkedList.add(shape);

}

Function For remove Shapes :

**public** **void** removeShape(Shape shape) {

**if** (StackRedo.size() == 20) {

StackUndo.clear();

}

**for** (**int** i = 0🡪 LinkedList.size()) {

**if** (LinkedList.get(i).equals(shape)) {

StackUndo.push(**new** LinkedList<Shape>(LinkedList));

LinkedList.remove(i);

**return**;

}

}

**throw** **new** UnsupportedOperationException("Error");

}

Function For Uptade Shapes :

**public** **void** updateShape(Shape oldShape, Shape newShape) {

**if** (StackRedo.size() 🡪 20) {

StackUndo.clear();

}

**for** (**int** i = 0🡪 ShapeList.size()) {

**if** (LinkedList.get(i).equals(oldShape)) {

StackUndo.push(**new** LinkedList<Shape>(LinkedList));

LinkedList.remove(i);

LinkedList.add(newShape);

**return**;

}

}

**throw** **new** UnsupportedOperationException("Error");

}

Function For getting Shapes :

**public** Shape[] getShapes() {

Shape[] shape;

shape = **new** Shape[LinkedList.size()];

**for** (**int** i = 0🡪 ShapeList.size()) {

shape[i] = LinkedList.get(i);

}

**return** shape;

}

Function For getting the supported shapes :

**public** List<Class<? **extends** Shape>> getSupportedShapes() {

List<Class<? **extends** Shape>> shapes = **new** ArrayList<Class<? **extends** Shape>>();

File pathToJar = **new** File("C:\\Users\\EL-hamd\\Downloads\\Compressed\\OOP\_TAs-oop-course-projects-319f39a1fbc5\\RoundRectangle.jar");

JarFile jarFile;

**try** {

jarFile = **new** JarFile(pathToJar);

Enumeration<JarEntry> e = jarFile.entries();

URL[] urls = { **new** URL("jar:file:" + pathToJar+"!/") };

URLClassLoader cl = URLClassLoader.*newInstance*(urls);

**while** (e.hasMoreElements()) {

JarEntry je = e.nextElement();

**if**(je.isDirectory() || !je.getName().endsWith(".class")){

**continue**;

}

// -6 because of .class

String className = je.getName().substring(0,je.getName().length()-6);

className = className.replace('/', '.');

**try** {

Class c = cl.loadClass(className);

shapes.add(c);

} **catch** (ClassNotFoundException e1) {

e1.printStackTrace();

}

}

}**catch** (IOException e1) {

e1.printStackTrace();

}

**return** shapes;

}

Function For undo :

**public** **void** undo() {

**if** (StackRedo.size() == 20) {

StackRedo.remove(0);

**if** (StackUndo.size() != 0 & LinkedList.size()!=1) {

StackRedo.push(**new** LinkedList<Shape>(LinkedList));

LinkedList = **new** LinkedList<Shape>(StackUndo.pop());

}**else** **if**(LinkedList.size()==1) {

StackRedo.push(**new** LinkedList<Shape>(LinkedList));

}

} **else** {

**if** (StackUndo.size() != 0) {

StackRedo.push(**new** LinkedList<Shape>(LinkedList));

LinkedList = **new** LinkedList<Shape>(StackUndo.pop());

} **else** **if** (StackUndo.size() == 0) {

StackRedo.push(**new** LinkedList<Shape>(LinkedList));

LinkedList = **new** LinkedList<Shape>();

}

}

}

Function For Redo:

**public** **void** redo() {

{

StackUndo.push(**new** LinkedList<Shape>(LinkedList));

LinkedList = **new** LinkedList<Shape>(StackRedo.pop());

}

}

Function For Save shapes:

**public** **void** save(String path) {

**if** (path.contains("XmL")) {

SaveXML xml = **new** SaveXML();

xml.XML(ShapeList, path);

} **else** {

saveJson json = **new** saveJson();

json.Json(ShapeList, path);

}

}

Function For Load shapes:

**public** **void** load(String path) {

**if** (path.contains("XmL")) {

LoadXML xml = **new** LoadXML();

LinkedList = **new** LinkedList<Shape>(xml.loadXML(path));

} **else** {

LoadJson json = **new** LoadJson();

LinkedList = **new** LinkedList<Shape>(json.loadjson(path));

}

}

Class Inheritance For all the shapes:

**public** **abstract** **class** Inhertance **implements** Shape, Cloneable{

Point Shapeposition = **new** Point(**null**);

Color Shapecolor = **null**;

Color Shapefillcolor = **null**;

Map<String, Double> map = **null**;

**public** **abstract** Object clone() **throws** CloneNotSupportedException ;

**public** **void** setPosition(Object position) {

Shapeposition = **new** Point((Point)position);

}

**public** Object getPosition() {

**return** (Object)Shapeposition;

}

**public** **void** setProperties(Map<String, Double> properties) {

map = **new** HashMap<String,Double>(properties);

}

**public** Map<String, Double> getProperties() {

**return** map;

}

@Override

**public** **void** setColor(Object color) {

Shapecolor = (Color) color;

}

@Override

**public** Object getColor() {

**return** (Object)Shapecolor;

}

**public** **void** setFillColor(Object color) {

Shapefillcolor = (Color) color;

}

**public** Object getFillColor() {

**return** (Object)Shapefillcolor;

}

**public** **abstract** **void** draw(Object canvas);

}

Class For Save shapes as XmL:

**public** **class** SaveXML {

File XML;

Map<String, Double> map;

**public** **void** XML(LinkedList<Shape> shape, String path) {

**try** {

**if** (path.equals(**null**) ) {

**throw** **new** UnsupportedOperationException("Error");

}

XML = **new** File(path);

XML.createNewFile();

} **catch** (IOException e1) {

e1.printStackTrace();

}

FileWriter fileWriter;

**try** {

fileWriter = **new** FileWriter(XML);

PrintWriter printWriter = **new** PrintWriter(fileWriter);

printWriter.println("<Paint>");

**for** (**int** i = 0; i < shape.size(); i++) {

Class<? **extends** Shape> c = (Class<? **extends** Shape>) shape.get(i).getClass();

**if**(shape.get(i).getProperties()!=**null**) {

map = **new** HashMap<String, Double>(shape.get(i).getProperties());

}

printWriter.println("<Shape>");

printWriter.println("ID : " + c);

**if**((shape.get(i).getPosition()) != **null**) {

printWriter.println("<Position>");

printWriter.println("<x = " + ((Point) shape.get(i).getPosition()).getX()+">");

printWriter.println("<y = " + ((Point) shape.get(i).getPosition()).getY()+">");

printWriter.println("</Position>");

}

**if**(shape.get(i).getProperties()!=**null**) {

printWriter.println("<Properties>");

Set<Map.Entry<String, Double>> st = map.entrySet();

**for** (Map.Entry<String, Double> me : st) {

printWriter.println("<"+me.getKey()+" : "+me.getValue()+">");

}

printWriter.println("</Properties>");

}

**if** (shape.get(i).getColor() != **null**) {

printWriter.println("<Color>");

printWriter.println("<Color : " + shape.get(i).getColor()+">");

printWriter.println("</Color>");

}

**if** (shape.get(i).getFillColor() != **null**) {

printWriter.println("<FillColor>");

printWriter.println("<Color : " + shape.get(i).getFillColor()+">");

printWriter.println("</FillColor>");

}

printWriter.println("</Shape>");

}

printWriter.println("</Paint>");

printWriter.close();

} **catch** (IOException e) {

// **TODO** Auto-generated catch block

e.printStackTrace();

}

}

calss For Load shapes as XmL:

**public** **class** LoadXML {

LinkedList<Shape> shapelist = **new** LinkedList<Shape>();

Point position ;

Map<String, Double> map=**null**;

Color color = **null**;

Color fillcolor = **null**;

Shape object ;

**public** LinkedList<Shape> loadXML(String path) {

File XML = **new** File(path);

**if** (XML.length() == 0) {

**throw** **new** UnsupportedOperationException("Error");

}

**try** {

BufferedReader br = **new** BufferedReader(**new** FileReader(XML));

String st = "";

String get = "";

**while** ((st = br.readLine()) != **null**) {

**if**(st.equals("{") || st.equals("}")|| st.equals("},")) {

**continue**;

}**else** **if**(st.contains("ID")) {

get=**new** String(st.substring(8, st.length()-3));

**if**(get.getClass().getName().contains("Circle")) {

object = **new** Circle();

}**else** **if**(get.getClass().getName().contains("Square")) {

object = **new** Square();

}**else** **if**(get.getClass().getName().contains("Rectangle")) {

object = **new** Rectangle();

}**else** **if**(get.getClass().getName().contains("Ellipse")) {

object = **new** Ellipse();

}**else** **if**(get.getClass().getName().contains("Triangle")) {

object = **new** Triangle();

}**else** **if**(get.getClass().getName().contains("LineSegment")) {

object = **new** LineSegment();

}**else** {

object = **new** DummyShape();

}

}**else** **if**(st.contains("/Shape")) {

shapelist.add(object);

}**else** **if**(st.contains("/Paint")) {

**break**;

}**else** **if**(st.contains("Color")&&!st.contains("/Color")) {

st=br.readLine();

get=**new** String(st.substring(9, st.length()-2));

object.setColor(Color.*getColor*(get));

}**else** **if**(st.contains("Position")&&!st.contains("/Position")) {

st=br.readLine();

get = **new** String(st.substring(5, st.length()-2));

Double x = Double.*parseDouble*(get);

st=br.readLine();

get = **new** String(st.substring(5, st.length()-2));

Double y = Double.*parseDouble*(get);

object.setPosition(**new** Point(x.intValue(),y.intValue()));

}**else** **if**(st.contains("Properties")&&!st.contains("/Properties")) {

**while**(!st.contains("/Properties")) {

st=br.readLine();

**if**(st.contains("/Properties")) {

**break**;

}

String key ="";

**int** index = st.indexOf(':');

**if**(index>1) {

key = **new** String(st.substring(1, index-2));

}**else** {

key=**null**;

}

get = **new** String(st.substring(index+2, st.length()-2));

map = **new** HashMap<String,Double>();

map.put(key, Double.*parseDouble*(get));

object.setProperties(map);

}

}**else** **if**(st.contains("FillColor")&&!st.contains("/FillColor")) {

st=br.readLine();

get=**new** String(st.substring(9, st.length()-2));

object.setFillColor(Color.*getColor*(get));

}**else** {

**continue**;

}

}

} **catch** (IOException e) {

e.printStackTrace();

}

**return** shapelist;

}

Class For Save shapes as Json:

**public** **class** saveJson {

File Json;

**public** **void** Json(LinkedList<Shape> shape, String path) {

**try** {

**if** (path.equals(**null**) || shape.size() == 0) {

**throw** **new** UnsupportedOperationException("Error");

}

Json = **new** File(path);

Json.createNewFile();

} **catch** (IOException e1) {

e1.printStackTrace();

}

FileWriter fileWriter;

**try** {

fileWriter = **new** FileWriter(Json);

PrintWriter printWriter = **new** PrintWriter(fileWriter);

printWriter.println("{");

**for** (**int** i = 0; i < shape.size(); i++) {

printWriter.println("\"shape\": {");

Class<? **extends** Shape> c = (Class<? **extends** Shape>) shape.get(i).getClass();

printWriter.println("\"ID\" : \"" + c + "\",");

**if**((Point)(shape.get(i).getPosition())!=**null**) {

printWriter.println("\"position\" : \"" + ((Point) shape.get(i).getPosition()).getX()+" "+((Point) shape.get(i).getPosition()).getX() + "\",");

}

**if**(shape.get(i).getColor()!=**null**) {

printWriter.println("\"color\" : \"" + shape.get(i).getColor() + "\",");

}

**if**(shape.get(i).getFillColor()!=**null**) {

printWriter.println("\"fill color\" : \"" + shape.get(i).getFillColor() + "\",");

}

**if**(shape.get(i).getProperties()!=**null**) {

printWriter.println("\"Properties\": {");

Map<String, Double> map = **new** HashMap<String, Double>(shape.get(i).getProperties());

Set<Map.Entry<String, Double>> st = map.entrySet();

**int** temp = 0;

**for** (Map.Entry<String, Double> me : st) {

temp++;

**if** (temp == map.size() - 1) {

printWriter.println("\"" + me.getKey() + "\" : \"" + me.getValue() + "\"");

} **else** {

printWriter.println("\"" + me.getKey() + "\" : \"" + me.getValue() + "\",");

}

}

printWriter.println("},");

}

printWriter.println("},");

}

printWriter.println("}");

printWriter.close();

} **catch** (IOException e) {

e.printStackTrace();

}

}

Class For Load shapes as Json:

**public** **class** LoadJson {

LinkedList<Shape> shapelist = **new** LinkedList<Shape>();

Point position ;

Map<String, Double> map;

Color color = **null**;

Color fillcolor = **null**;

Shape object ;

**public** LinkedList<Shape> loadjson(String path) {

File json = **new** File(path);

**if**(json.length()=0) {

**throw** **new** UnsupportedOperationException("Error");

}

**try** {

BufferedReader br = **new** BufferedReader(**new** FileReader(json));

String st = "";

String get = "";

**while** ((st = br.readLine()) != **null**) {

**if**(st.equals("{") || st.equals("}")|| st.equals("},")) {

**continue**;

}**else** **if**(st.contains("ID")) {

get=**new** String(st.substring(8, st.length()-3));

**if**(get.getClass().getName().contains("Circle")) {

object = **new** Circle();

}**else** **if**(get.getClass().getName().contains("Square")) {

object = **new** Square();

}**else** **if**(get.getClass().getName().contains("Rectangle")) {

object = **new** Rectangle();

}**else** **if**(get.getClass().getName().contains("Ellipse")) {

object = **new** Ellipse();

}**else** **if**(get.getClass().getName().contains("Triangle")) {

object = **new** Triangle();

}**else** **if**(get.getClass().getName().contains("LineSegment")) {

object = **new** LineSegment();

}**else** {

object = **new** DummyShape();

}

}**else** **if**(st.contains("position")) {

get=**new** String();

**int** j=14;

**for**(j=14; !(st.charAt(j)==' ');j++) {

get+=st.charAt(j);

}

Double x = Double.*parseDouble*(get);

get = **new** String();

**for**(j=j+1;!(st.charAt(j)=='"');j++) {

get+=st.charAt(j);

}

Double y = Double.*parseDouble*(get);

object.setPosition(**new** Point(x.intValue(),y.intValue()));

}**else** **if**(st.contains("color")) {

get = **new** String(st.substring(11, st.length()-3));

object.setColor(Color.*getColor*(get));

}**else** **if**(st.contains("fill color")) {

get = **new** String(st.substring(16, st.length()-3));

object.setFillColor(Color.*getColor*(get));

}**else** **if**(st.contains("Properties")) {

**while**(!st.equals("},")) {

{

st=br.readLine();

**if**(st.equals("},")) {

**break**;

}

**int** index = st.indexOf(':');

String key="";

**if**(index>2) {

key = **new** String(st.substring(1, index-3));

}**else** {

key=**null**;

}

**if**(st.charAt(st.length()-1)==',') {

**if**(st.length()>3) {

get = **new** String(st.substring(index+3, st.length()-2));

}

}**else** {

**if**(st.length()>2) {

get = **new** String(st.substring(index+3, st.length()-1));

}

}

{

map = **new** HashMap<String,Double>();

map.put(key, Double.*parseDouble*(get));

}

}

}

}

**if**(st.contains("shape")) {

shapelist.add(object);

}

}

} **catch** (IOException e) {

e.printStackTrace();

}

**return** shapelist;

}

The class of inheritance which the interface of shape is implemented in it is used for all shapes as the all shapes have same properties like position, properties, color and fill color but there is some different properties of shapes like drawing circle is different than drawing square so every class of shapes has the implementing of draw and implementing of clone the shape.

So every class of shapes extends the class of inheritance which implements the interface of shape.

**- Design Decisions:**

**The paint application helps the user to draw different shapes like Circle, Ellipse, Square,**

**Rectangle, Triangle and Line Segment.**

**The user can draw supported shapes he want just he should call the class that draw that shape.**

**The application runs as the following:**

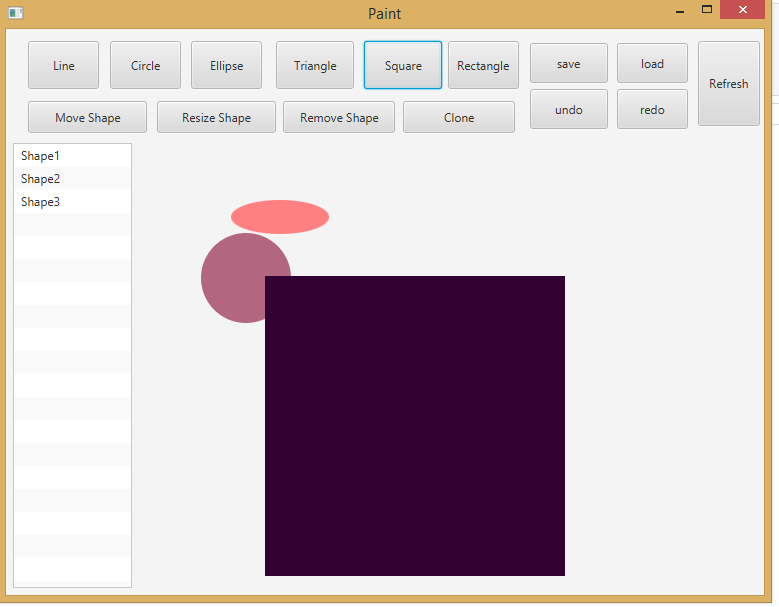
**The user selects the shape that he wants to draw, he selects the position which is the center of the shape then he determines the properties of the shape like that:**

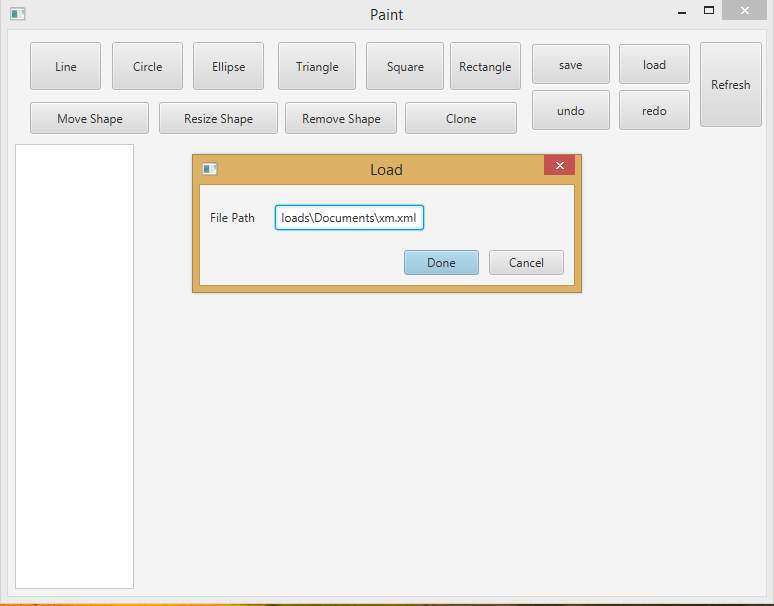
**If he want to draw circle he enters the center (x, y) then he enters the radius of the circle (R) so the circle will be drawn, the user can enter the color of circle he also can color all the circle as he should enter the fill color of the shape, the user can clone the shape that he want.**

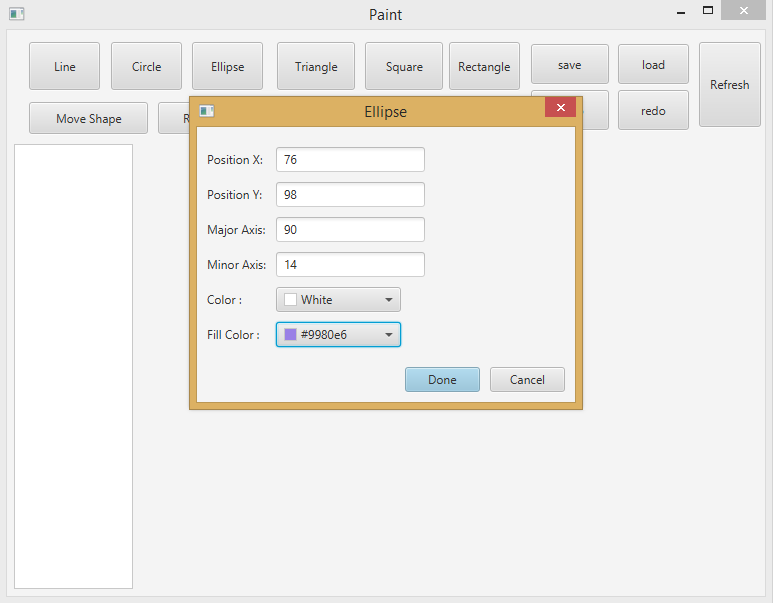
**After finishing the properties of the shape the user adds it, he user can remove, update and move the shape, if he want to undo the old shapes he should select the button of undo or redo as he want, if he undo times more than 20 he could redo only 2on times.**

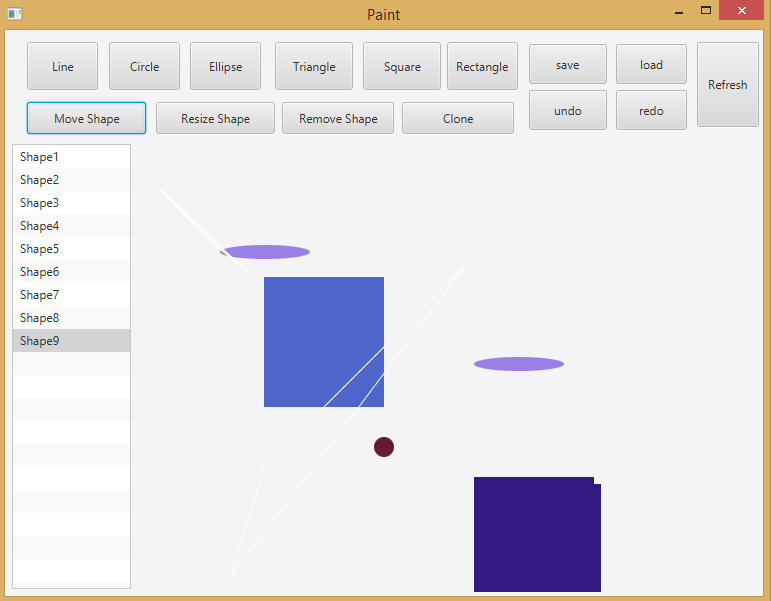
**After finishing all of that the user can save the shapes that he wants to save and he should select where he want to save the shapes, and he can load these shapes again as he should enter where he saves the shapes then the application will draw the shapes that the user saves.**

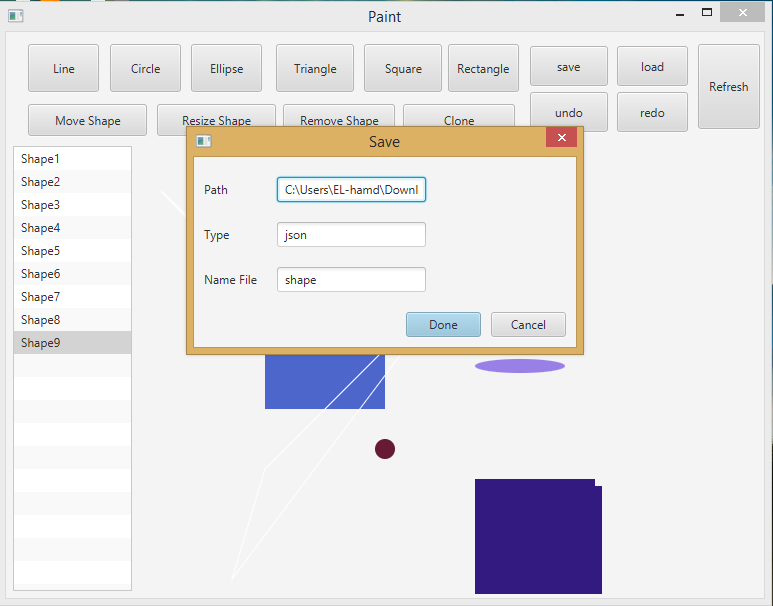
**- Sample runs :**











-References :

<https://stackoverflow.com/questions/11016092/how-to-load-classes-at-runtime-from-a-folder-or-jar>

{\displaystyle {\begin{aligned}x&\equiv a\_{1}{\pmod {n\_{1}}}\\x&\equiv a\_{2}{\pmod {n\_{2}}},\end{aligned}}}