Alexandria University
Faculty of Engineering
Computer and Systems Engineering Dept.
Second Year
Fall 2015

CS221: Object Oriented Programming
Programming Assignment 1

Version 3.0, updated 23th October

Vector Based Drawing Application

Objectives

Upon completion of this assignment, you will be able to:

- Design an object oriented model for geometric shapes
- Draw a UML class diagram that represents your model
- Apply the OOP concepts of inheritance and polymorphism to your design
- Create an advanced GUI with 2D Graphics capabilities
- Enable dynamic extensions to your applications

Part 1: Geometric Shapes Data Model

Description:

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. In order 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.

Tasks:

1- Design an object oriented model that covers the following geometric shapes: Line Segment, Circle, Ellipse, Triangle, Rectangle and Square.
2- Draw a UML Class diagram that represents your model, showing all the classes, attributes and methods.
3- Apply the concepts of inheritance and polymorphism to your design.

```java
package eg.edu.alexu.csd.oop.draw;

public interface Shape{
	public void setPosition(java.awt.Point position);
	public java.awt.Point getPosition();

	/* update shape specific properties (e.g., radius) */
	public void setProperties(java.util.Map<String, Double> properties);
	public java.util.Map<String, Double> getProperties();

	public void setColor(java.awt.Color color);
	public java.awt.Color getColor();

	public void setFillColor(java.awt.Color color);
	public java.awt.Color getFillColor();

	/* redraw the shape on the canvas */
	public void draw(java.awt.Graphics canvas);

	/* create a deep clone of the shape */
	public Object clone() throws CloneNotSupportedException;
}
```

```java
package eg.edu.alexu.csd.oop.draw;

public interface DrawingEngine {
	/* redraw all shapes on the canvas */
	public void refresh(java.awt.Graphics canvas);

	public void addShape(Shape shape);
	public void removeShape(Shape shape);
	public void updateShape(Shape oldShape, Shape newShape);

	/* return the created shapes objects */
	public Shape[] getShapes();

	/* return the classes (types) of supported shapes that can
	 * be dynamically loaded at runtime (see Part 3) */
	public java.util.List<Class<? extends Shape>> getSupportedShapes();

	/* limited to 20 steps. Only consider in undo & redo
	 * these actions: addShape, removeShape, updateShape */
	public void undo();
	public void redo();

	/* use the file extension to determine the type,
	 * or throw runtime exception when unexpected extension */
	public void save(String path);
	public void load(String path);
}
```

Part 2: Drawing and Painting Application

Description:

Drawing and painting applications are very popular and have a huge user base. They generally offer a big number of features that includes but is not limited to: Drawing, Coloring, and Resizing. They also include a number of built in, and possibly extensible set of geometric shapes, and classically, they allow the user to undo or redo any instructions so as to make the application more usable

Tasks:

1- Implement your design from part 1 in an OOP language of your choice.
2- Design and implement a GUI that allows the following functionalities for the user on all the shapes defined in part 1: Draw Color, Resize, Move, and Delete.
3- Implement your application such that it would allow the user to undo or redo any action performed.
4- 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.

Part 3: Dynamic Application Extensions and plug-ins

Description:

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. This can be easily done by the dynamic class loading capabilities that OOP languages offer.

Tasks:

1- Pick one of the Shape Classes listed in part 1, and compile it as a class library.
2- Provide an option in the GUI of your application that allows for selecting the class library file.
3- On selecting and loading the file, the isolated shape should be appended to the available list of shapes in the application.

Part 4: Save and load

Description:

One of the main features in any paint application is saving user's drawings in a file and modifying it later.

Tasks:

1- Provide an option in GUI to save the drawing in XML and JSON file (You should implement both).
2- Provide an option to load previously saved drawings and modify the shapes.
3- User must choose where to save the file.

Deliverables

- You should work in groups of two.
- For exceptional reasons, groups can be of a three members. However, they must support Java Network Launching Protocol (JNLP) in their deliverables
- The implementation for the given interfaces.
- Develop this assignment in Java.
- A self executable Jar: The program should be executable by simply double clicking the icon provided that you have a running JRE.
- You should deliver your source code using your git repository.
- Do not use any external libraries, and for XML & JSON parsing & formatting you can use JDK classes or write your own ones.
- You are allowed to use any graphics libraries.
- You should deliver a report that contains the required UML diagram, describes your design thoroughly, and contains snapshots of your GUI and a user guide that explains how to use your application. Any design decisions that you have made should be listed clearly.
- Delivering a copy will be severely penalized for both parties, so delivering nothing is so much better than delivering a copy.