Design Activity - Drawing Editor

Drawing Object Types
The design must accommodate an indefinite number of drawing object types, including (but not limited to) lines, polylines (i.e., a sequence of line segments treated as a unit), curves, rectangles, ellipses, text, and images. The user must be able to choose an object type (from *either* a menu or a toolbar of buttons) and then draw the object using a mouse. For this prototype you need only support lines and rectangles, but it must be obvious from the design, its documentation, and your presentation, that the other object types mentioned could be easily integrated.

Manipulating Objects in the Drawing
The program must allow the user to also manipulate objects already in the drawing by selecting them with the mouse. The program will provide some visual indication of the selection. Once selected, the design must support the operations such as:

1. Delete the object
2. Copy the object
3. Move the object
4. Edit the object

Other operations (expand, shrink, rotate, etc.) may be added in the future, but for this prototype only these four operations are required; as in the case of drawing object types, your design must make it obvious how such extended operations could be incorporated. As above, the user has the option of selecting the operation from *either* a menu or a toolbar of buttons.

Editing Objects
Editing objects is accomplished by having the object produce a dialog box with elements that define the object's properties. The system will display the dialog, and when the dialog is closed it is passed back to the object so that the updated properties can be extracted and applied. For the prototype, the only line property is color, and the only rectangle properties are line color and corner style. Color choice in the prototype will be limited to black, red, green, and blue. The rectangle corners can be either square or rounded.

Grouping Objects
Objects may also be grouped into larger units. You are free to choose a reasonable means of selecting the objects to be grouped and to combine them into a single group object. All operations *except edit* can be used with group objects, and the effect is to delete, copy, or move all the objects in the group as a unit. Of course group objects can contain other group objects to an arbitrary depth, so be careful in defining the operations! If a group is selected the visual indication should identify all objects in the group. Note that it must be possible to place any combination of primitive objects or groups into a group. When a group is moved or copied all elements within the group will maintain the same relative position and orientation to the other group objects.
Groups can be selected and "ungrouped", which simply removes one layer of grouping. The "ungroup all" operation removes the top most group and all the groups it contains (recursively), so that only the primitive objects (lines and rectangles) remain in the drawing.

Saving and Restoring Drawings
The program will use a file for persistent storage of the drawing. The program should be able to generate this file from a drawing and also display a drawing read from a file. The program will have an Open file and Save file operation. These operations can be selected from either a menu or

a toolbar of buttons. In addition, if a command line argument is provided when the program is started the program will interpret this as the name of a drawing file to display after startup of the program. The program should provide appropriate warnings to the user to ensure that no unsaved work is lost when a file is opened or the user exits the program.
The drawing file is an ASCII text file. Each object is described on a single line as follows:

line *X_1 Y_1 X_2 Y_2 color*
rect *X_1 Y_1 X_2 Y_2 color style*

The first letter identifies the type of object, followed by two pairs of coordinates. For a line the coordinate values are the beginning and end points. For a rectangle, (X_1,Y_1) is the upper-left corner of the rectangle and (X_2,Y_2) is the lower right corner. Color is coded as: **k**, **r**, **g**, **b** for black, red, green and blue, respectively. For rectangles there is also a corner style: **s**, **r** for square and rounded, respectively. Coordinates should be integer values representing pixel locations on the drawing canvas. The origin (0,0) is the top left corner. The X axis is horizontal increasing to the right and the Y axis is vertical increasing in a downward direction.

Groups are defined by two additional file entries:
begin
end

The first entry **begins** the definition of a group. All the following drawing elements up to the matching **end** file entry are considered part of the group. The file format, like the drawing program, supports grouping to an arbitrary depth.

Export to XML
Customers plan to use this program to create drawings used with other applications. To that end, they want the ability to export a drawing to XML as shown below:

Example for lines

```
<line>
   <begin>
      <x>10</x>
      <y>50</y>
   </begin>
   <end>
      <x>20</x>
      <y>25</y>
   </end>
   <color>black</color>
</line>
```

Other possible colors are **red**, **green**, and **blue**.

Example for rectangles

```
<rectangle>
   <upper-left>
      <x>10</x>
      <y>20</y>
   </upper-left>
   <lower-right>
      <x>20</x>
```

```
      <y>35</y>
   </lower-right>
   <color>black</color>
   <corner>rounded</corner>
</rectangle>
```

The colors are the same as lines, and corners can be either **rounded** or **square**.

For groups

```
<group>
   ... one or more lines, rectangles, and nested groups ...
</group>
```

Key Points of Flexibility: The Design Must Allow for the Following Extensions

1. New primitive drawing object types (e.g., ellipses).
2. Adding new operations on drawing objects (e.g., rotate).
3. Adding or replacing editors for new and existing object types (e.g., line style).
4. New save or export file formats (e.g., JPEG)

Miscellaneous

1. The program must support a canvas of at least 400 by 400 pixels. All testing will be with primitives and groups that fit within a bounding box of 400 by 400. Optionally, your program can handle a larger canvas area or support scrolling within a larger canvas. This, however, is not a requirement and will not be tested.
2. Primitives within a drawing can be rendered in an arbitrary order. This is a 2D editor with no concept of a Z dimension or depth ordering of the 2D primitives. When two or more lines overlap at a particular pixel location the pixel will be rendered in the line color of the last one drawn by whatever arbitrary ordering the program has made.

Assessment

Component	Percent	Comments
Submission	5	Degree to which the project is submitted in accordance with the specification given in the section **Submission Instructions**
Design and Design Document	25	Refer to the **Design Evaluation Rubric**
Implementation	70	Refer to the **Implementation Rubric**.

The Design Evaluation Rubric and Implementation Rubric will be updated on Moodle.

Submission Instructions

One of the team members will submit a zip file named **Unit1-N.zip**, where N is your group number, to the **U1 Project** drop-box in *myCourses*. If there are several submissions for a given team, the last submission will be the one that is graded. The contents of the zip file will include:

1. All the source files required to compile, assemble, and run your project. You may either choose **JAVA or C# or Python** for implementation.
2. All files must be zipped file with the executable class files

3. There should be a **single design document** containing all the documentation. The final design document, in **pdf** format, including *at a minimum*:

    a. Title information, including the name of your team, the name of your product, the date, and a list of all the team members.

    b. UML class diagram that you have done in class. You may choose to refine the diagram and resubmit.

    c. A table summarizing the responsibilities of each major class.

    d. Sequence diagrams that provide insight into the key characteristics of your design.

    e. A narrative outlining how the design reflects a balance among competing criteria such as low coupling, high cohesion, separation of concerns, information hiding, the Law of Demeter, extensibility, reusability, etc. This should include a discussion of the design patterns used to achieve this balance in the context of expected product evolution.

4. A file **README.txt** that tells how to compile the code and run your code, and includes any other information the team wishes the instructor to know.

Documentation Instructions

For this design exercise, you must provide detailed sequence diagrams to describe the interactions between objects in your system for at least ***two*** of the following operations:

1. Mouse selection of an element that is in a group, drag to a new position, update of the model, refresh of the drawing.
2. Menu choice to edit the current object, user change of attributes, user confirmation of change, update of element in the model, refresh of the drawing.
3. User indicates to save an XML file, contents of drawing being written to the file, file closed.
4. Mouse selection of an element that is in a group, ungrouping of the elements in the group.