SCHOOL OF ENGINEERING AND COMPUTER SCIENCE

COMP102 2013 Tri 1:

Assignment 10:

- Due 9 Jun 11:55pm

Goals

This assignment is intended primarily to give you experience in writing larger programs
- with multiple classes
- using collections of objects stored in ArrayLists
- that store data to files and read data from files
- with multiple classes that implement an interface class (a specification of a type, not a user interface)

It builds on many elements from previous assignments and all the topics of the course so far:
- Defining objects that have state that can change (`Bouncer` from A6)
- Loading data from files and saving data to files (eg `ExamTimes` from A5, `Plotter` and `ScrabbleBoard` from A7)
- Drawing and selecting objects with the mouse (`MiniPaint` from A6, `BalloonGame` from A8, `MotelBooker` from A9)
- Dealing with Arrays or ArrayLists of objects and changing their state (`ScrabbleBoard` from A7, `BalloonGame` and `Genealogy` from A8, `ImageProcessor` and `MotelBooker` from A9)
- Searching for objects and adding/removing things in an array or ArrayList (`BalloonGame` and `Genealogy` from A8, `MotelBooker` from A9)

Resources and links

- Download Zip file of necessary code and data.
- Source Code and data Files (for viewing online)
- Videos of the demo MiniDraw Exercise (Exercise has sound synching problem)
- Submit your answers.
- Ask for online assistance from the staff:
- Marking sheet
- Model Solution Code

Overview

For this assignment, you will complete a simple drawing program (`MiniDraw`) which lets a user create and edit simple diagrams and save and load the diagrams to/from files. The key difference between `MiniDraw` and the `MiniPaint` program of assignment 6 is that the `MiniDraw` program must remember all the shapes that the user has drawn so that they can be moved, deleted, or modified.

This assignment involves an ArrayList containing Objects of different types - the different kinds of shapes. The list is an ArrayList of `Shape` objects. `Shape` is an interface class and all the different shape classes `implement` the `Shape` interface.

The assignment builds on elements from many of the other assignments. You should refer to your solutions and/or the model solutions of the other assignments to get ideas for how to do parts of this assignment. You will find `MiniPaint` and `BalloonGame` particularly helpful.

The exercise is a very limited version of the MiniDraw; completing it first should be helpful.

Preparation

The `MiniDraw` program is larger than the programs in previous assignments. Understanding the structure of the program and the role of each class will take some time. You should spend some time reading the template files, to see what the classes are for. We will also talk about the program in the lectures.

Read this handout carefully and run the demo program, making sure you understand what you are required to do.

Download the zip file for assignment 10 and extract it to your home folder. It should contain templates for the Java program you are to complete. Read through the whole assignment to see what you need to do. You can run the demo programs on the ECS lab computers.

Make sure you have looked through the model answers to recent assignments to make sure you understand how they work.

To Hand In:

You should submit your best version of the `MiniDraw` program and your `Reflection.txt` file. Please submit all the java files you used (even if you didn't modify them). Don't forget to complete the final step of the submission process after you have uploaded the files.

Note that the deadline is midnight at the end of Sunday (9 Jun); but we certainly won't be starting the marking or posting model solutions until the middle of the next day at the earliest!

You may work in pairs for the Core and Completion parts, but if you do you must include a comment at the top of your program saying who you worked with. You must do the Challenge parts on your own.

You are reminded of the School's policy on plagiarism - see the COMP102 Web site for details.

Exercise

The `Exercise` program lets the user draw circles and move them around. `Circle` objects store the position and color of a circle. It is a miniature version of `MiniDraw`, which will help you to understand how `MiniDraw` works. Most of the code you write will be directly useful for `MiniDraw`.

The `Exercise` program has fields to store

- an ArrayList of Circle objects
- whether it is currently drawing new circles or moving them around
- the Circle the mouse was last pressed on.

Pressing the buttons simply records whether the user is now drawing circles ("Circle") or moving circles ("Move") in the `currentAction` field.

Pressing the mouse will record the Circle that the mouse was pressed on (or null if there was no circle under the mouse) in the `currentCircle` field.

Releasing the mouse will either draw a new circle or move a circle:

- If the program is currently drawing new circles, then the program will create a new Circle object at the released point, add it to the collection, and redraw the collection.
- If the program is currently moving circles around, then the Circle in `currentCircle` (if it isn't null) will be moved to the position the mouse is released.

The `Circle` class is written for you. You must complete the `Exercise` class:

- the constructor (set up mouse listener and buttons)
- the `mousePerformed` method (respond to "pressed": find circle; and "released": either make new Circle and add to array, or move the currently selected Circle)
- the `findCircle` method (return the circle that (x,y) is on)
- the `redrawCircles` method (redraw each Circle in the array)

MiniDraw

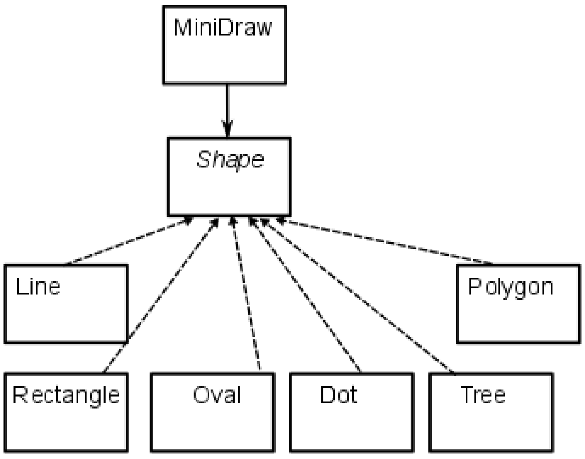
The `MiniDraw` program is intended to be a very simple drawing editor. (Note that a drawing editor is different from a paint program --- a paint program edits the pixels in an image; a drawing editor edits a set of shape objects.)

MiniDraw allows the user to create drawings consisting of shapes including rectangles, ovals, lines, dots, and polygons, of different colours. The user can add a new shape to the drawing, delete a shape from the drawing, move a shape to a different position, or change the size of a shape. The user may also save the current drawing to a file and may load a previous drawing from a file and edit it further.

The central data structure in the program is the list of shapes in the current drawing. MiniDraw uses an ArrayList of `Shapes` to store the list of shapes. The design of `MiniDraw` therefore involves several classes and an interface class: a class for the main program, a class for each kind of shape, and an interface class for the `Shape` type. We have provided a framework of the design for you, partly to provide an example of good design, and partly because designing it from scratch would be too large an assignment for most students in the course at this point. (You may modify the design if you wish, as long as it still uses an ArrayList of `Shape` and the `Shape` interface class.)

Classes

When you open the project in BlueJ, you will see a simple class diagram of the classes, similar to the one below. (The diagram here shows just the major relationships between the classes; BlueJ sometimes adds all the minor relationships, which makes the diagram more confusing.)



`MiniDraw`
sets up the user interface and contains methods to respond to all the buttons and the mouse. It also contains an ArrayList of `Shape` objects that has the shapes in the current diagram. Note that `Shape` is an interface class, so you cannot make a `Shape` object. Rather, the ArrayList contains `Line`, `Rectangle` *etc* objects, which all implement the `Shape` interface, and therefore are of the `Shape` type. The methods in MiniDraw call methods and constructors of the shape classes, but leave as many details as possible to those classes.

`Shape`
An interface defining the `Shape` type. Specifies methods for

- determining whether a point (x, y) is on top of the shape
- moving the shape
- redrawing the shape on the graphics pane
- resizing the shape
- returning a String description of the shape

Shape classes
`Rectangle`, `Oval`, `Line`, `Dot`, `Polygon` These classes all implement the `Shape` type (ie, define all the methods specified in `Shape`) and have two constructors. The first constructor is passed the position, size, and color for the shape; the second constructor is passed a `Scanner` and reads the information from the `Scanner`.

User Interface

The user interface contains a set of buttons: buttons for dealing with the whole drawing ("New", "Open", "Save"), buttons for setting the colour and moving, modifying, and deleting shapes ("Color", "Move", "Resize", "Delete"), and buttons for specifying what kind of shape will be added next. Some of the buttons (eg "New", "Open", "Save", "Colour") perform an action immediately. The other buttons simply remember the name of the button so that when the user presses or releases the mouse, the program can take the appropriate action. For example, if the user clicks the "Oval" button, the next mouse release should create a new Oval shape; if the user clicks the "Move" button, the next mouse release should change the position of the shape the mouse was pressed on.

When drawing a shape, if the user presses the mouse at point *p1* and releases the mouse at *p2*, then

- A Line should have *p1* and *p2* as its two ends
- A Rectangle should have *p1* and *p2* as diagonally opposite corners
- An Oval should be placed so that its enclosing rectangle would have diagonally opposite corners at *p1* and *p2*
- A Dot should be centered at *p2* (and ignore *p1*).
- A Polgon should either be created with just one point (*p2*), or the point *p2* should be added to the current Polygon.

File Format

The program can read and save drawings from and to files. The format of the file is that there is one line of the file for each shape. Each line must start with the name of the type of shape, followed by a specification of the parameters of the shape, the colour first (three integers for red, blue, and green) and then the position, size, etc (the other values will depend on the shape). Each shape class has a `toString()` method which will return a string describing the shape in a form suitable for writing to one of these files. Each shape class also has a Constructor which will read the parameters from a Scanner in the same format. The `toString()` method should therefore match the constructor.

What is provided in the Template code:

- `MiniDraw` : a field for storing the shapes, some suggested fields for storing other information; method headers. Note that you don't have to use all these method headers and you may want different or addtional fields. but they will guide you to a reasonable design for your program.
- The `Shape` interface. This is complete, and you do not need to change it at all. Read it so that you know what methods a `Shape` must have.
- The `Line` class. This is complete, so you will be able to add lines to the drawing even when you haven't completed the other shapes.
- Templates of the `Rectangle` and `Oval` classes. You will need to define fields, constructors, and methods. They each contain a test method for testing the class independently.
- An empty `Dot` class - you are to complete this class from scratch
- An empty `Polygon` class - you are to complete this class from scratch
- An empty `Tree` class - you are to complete this class from scratch

Core

For the core, you should produce a version of `MiniDraw` that draws, moves, and deletes Lines, Rectangles, Ovals, and Dots. It does not need to resize shapes, or load or save drawings from files.

We strongly advise you to build your program in stages, testing each stage and getting it working before progressing to the next stage!

The following steps are one way of constructing your program in stages. You don't need to follow them, but it may be helpful to you. They give one way of breaking the program into stages, each of which results in a working (though limited) program.

- Stage 1. Lets the user make a new (empty) drawing, and add lines to it:
  - Define the constructor for setting up the interface.
  - Define the `buttonPerformed` method.
  - Define the `mousePerformed` method so that it, at least, stores the position when "pressed" and calls `addShape` when "released".
  - Define the `newDrawing()` methods for responding to the `New` button.
  - Define the `addShape` method (which is called by `mousePerformed` when mouseAction equals "released") to create a new shape and add it to the collection of shapes. Although the kind of shape should depend on the value in the `currentAction` field, at this point, it will only be a `Line`.
  - Complete the `drawDrawing()` method.

- Stage 2. Lets the user make a drawing with coloured rectangles and lines:
  - Complete the `selectColor()` method so that the user can set the color for new shapes.
  - Complete the methods and first constructor of the `Rectangle` class. (For the core, you do not need to complete the second constructor which has a Scanner as a parameter).
  - Extend the `addShape` method to add `Rectangle` shapes as well as `Line` shapes, depending on the value in the `currentAction` field. Note: you can test your `Rectangle` class using the provided `TestRectangle` class (using BlueJ, call `main` on the `TestRectangle` class).

- Stage 3. Lets the user move shapes around in the drawing:
  - Extend the `mousePerformed` method and define the `findShape()` method so that when the mouse is pressed, the shape the mouse is on (if any) is recorded in the `currentShape` field.
  - Extend the `mousePerformed` method so that if the `currentAction` field contains "Move", it calls the `moveShape` method (passing the amounts that the mouse moved since the last "pressed" event), otherwise calls the `addShape` method.
  - Define the `moveShape` method, so that it moves the current shape by the appropriate amount. You will need to look carefully at the relevant methods in the `Shape` interface.

- Stage 4. Lets the user add Ovals and Dots to the drawing:
  - Extend the `addShape` method so that it can add `Oval` and `Dot` shapes.
  - Complete the `Oval` class. You can use the `TestOval` class to test it.
  - Construct a `Dot` class for adding small circular dots to the drawing. You can use the `TestDot` class to test it.

- Stage 5. Lets the user delete shapes from the drawing:
    - Extend the `mousePerformed` method so that if the `currentAction` field contains "Delete", it calls the `deleteShape` method (passing the x and y where the mouse was released).
    - Complete the `deleteShape` method so that it finds the shape that the mouse was released on, and removes it from the drawing. `deleteShape` should use `findShape` to find the shape. (Why should `deleteShape` find the shape the mouse was released on instead of using the current shape - where the mouse was pressed?)

Completion

For the completion, you should complete the program so that it can save and load drawing files, draw polygons, and change the size of shapes.

- Define the methods for responding to `Save` button so that you can save your drawing to a file. You will have to look at the file itself to see how saving should work, since you haven't made the program load a drawing file yet. Note that saving a file requires that `MiniDraw` can open the file and write a list of shapes to the file, and that each kind of Shape can generate a string specifying the shape (position, size, colour, etc).
- Complete the second constructors of the `Rectangle`, `Line`, `Oval` and `Dot` classes. These constructors each take a Scanner as an argument. The constructors assume that the Scanner is already attached to a file (they don't need to open the file), and the first word on the line (the type of shape) has already been read. The constructor should read the rest of the values on the line to construct the shape. The test methods can test this functionality.
- Complete the methods in `MiniDraw` for the `Open` button so that it reads a whole list of shapes from a Scanner. It needs to read the first word on each line to determine what kind of shape to construct, and must then call the appropriate constructor for that shape, passing the Scanner to the constructor. Check that you can save a drawing, make a new drawing, and then load the original drawing back in.
- Add a `Polygon` button, complete the `Polygon` class, and extend the program to allow the user to draw polygons. Polygons are more complicated because they have to have a list of x positions and y positions of all the vertices. Each vertex will require a separate mouse event, so the program will have to construct a Polygon with the first mouse event, and then keep adding points to the current Polygon with further mouse events, until the user clicks on a button to indicate the polygon is complete. Run or watch the demo carefully to see one way of doing polygons.
- Complete the methods in `MiniDraw` and in the shape classes for resizing a shape in the drawing. Note that resizing should never make a shape disappear - there should be a minimum size that still leaves the shape visible. The demo has a very simple resizing function that only works well when you drag the mouse from the top right corner of a shape. It is just fine to do it the same way. (The challenge asks you to implement a better interaction mechanism.)

Challenge

This program can be extended in lots of ways.

- Add a Tree class that draws a random branching tree. It would be nice if it looked better than the ones in the demo program. Note that if you use random numbers to make the tree, you will need to store the "seed" and restart the random number generator with the same seed every time you draw the tree, otherwise it will change itself everytime the drawing is modified in any way! You will need to use an instance of the `Random` class for this.

The user interface is not really very nice.

- Add "rubberbanding" so that you can see the outline of the shape as you are drawing (or moving or resizing) it. (Note that this may be easier than in MiniPaint since the drawing will be redisplayed once the shape is drawn (or moved or resized).
- The resizing in the demo works OK when you click on the top right corner of a shape, but it isn't nice otherwise. Fix it to work better.
- Change the user interface design to the more common pattern that allows the user to first select a shape (which should then be highlighted in some way) and then act on it (deleting it, changing its size, changing its colour).
- Make the "New" and "Load" buttons check whether the current drawing has been saved and offer to save it first.

The program is also very limited.

- Allow the user to modify other aspects of the shapes, including the border (colour and thickness).
- Allow the user to move the shapes forward and backward in the drawing (changing which shapes are in front of other shapes).
- Allow the user to select a group of shapes and align them (by their tops or left sides, at least).

Reflection.

This was the last assignment for COMP102. Think back to the beginning of the course and what you knew at that point, and then list some of the new skills and understanding that you believe that you have gained during this course.

- For some of you who knew a lot already, there may only have been a few new things, but try to identify them anyway.
- For some of you who are still finding the course difficult, you could try doing some of the early assignments again; you may find them a lot easier the second time round, demonstrating that you have learned stuff!