CS5001 Object-Oriented Modelling, Design and Programming
Practical 4 – Vector Drawing

School of Computer Science
University of St Andrews

Due Friday week 10, weighting 30%
MMS is the definitive source for deadlines and weightings.

In this practical you are required to write a simple vector graphics drawing program using Java Swing. Your program should use either the Model–View–Controller (MVC) or the Model–Delegate (MD) design pattern. You should use JUnit to test your Model, and you should include these tests in your submission.

At a minimum, your program should support the following feature requirements:

Basic requirements

- Drawing straight lines
- Drawing rectangles
- Drawing ellipses
- Drawing diagonal crosses (see the figure below)
- Undo/redo
- Different colours

A very good implementation of these basic requirements could award up to a 17. For higher grades, you should implement one or more of the following:

Advanced requirements

- Support for drawing squares and circles. One way of implementing this feature would be using a key (say the Shift key) to lock aspect ratio during the drawing of rectangles and ellipses.
- Add several more shapes: triangles, parallelograms, hexagons, etc.
- Load and save vector drawings in a format that permits them to be manipulated as vector drawings after loading.
- Select a previously drawn object and change its location, colour or size.

- Add Murray polygons (look on the floor next time you walk into the Jack Cole building!)
- Use networking to allow two people to work on a single drawing collaboratively.

A very good implementation of three of these advanced requirements could award a grade of up to 20.

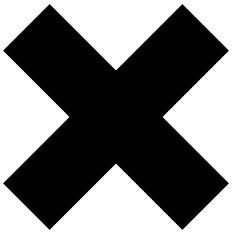

Figure 1: Example of a diagonal cross

Implementation

For this practical you will have to think very carefully about the classes you will require for your program. You should design a suitable set of classes to represent the shapes which can be created with a specified colour, fill, etc. and whose dimensions may be defined by a start and end position. Consider that you will probably want to be able to deal with a collection of abstract shapes in your program code in a polymorphic manner rather than dealing with different collections for each shape. You should think carefully about where to place the code which specifies how to draw each shape in the GUI.

Whether you choose MD or MVC, you should aim to provide a Model component of your program comprising classes that model the shapes that have been drawn. The Model should provide operations to select the current drawing shape, create and manipulate these shapes. The model should also provide methods to undo and redo various operations (including shape creation) as well as methods to save or load a shape collection from file, etc. depending on how many features you implement. The classes comprising the Delegate or View/Controller should translate button presses to the appropriate calls to methods defined in the Model. You should use an appropriate notification structure such as `PropertyChangeListener`, so that the drawing canvas is redrawn when the model has changed.

See the figure below for an example of what your GUI might look like.

Testing

Along with your program, you should create a test suite that tests the functionality of your Model. This test suite should be written in JUnit, and should test the broadest possible range of inputs to your Model. You should aim to execute every line of code in your Model somewhere in this test suite, and it should also include edge cases and invalid inputs. Your JUnit tests do not need to test the view/controller or delegate parts of your program, but you should test these thoroughly by hand to make sure that they work.

As well as running on your own computer, the tests should be runnable on the lab computers or host servers. Make sure to run your JUnit suite on the School infrastructure and ensure that

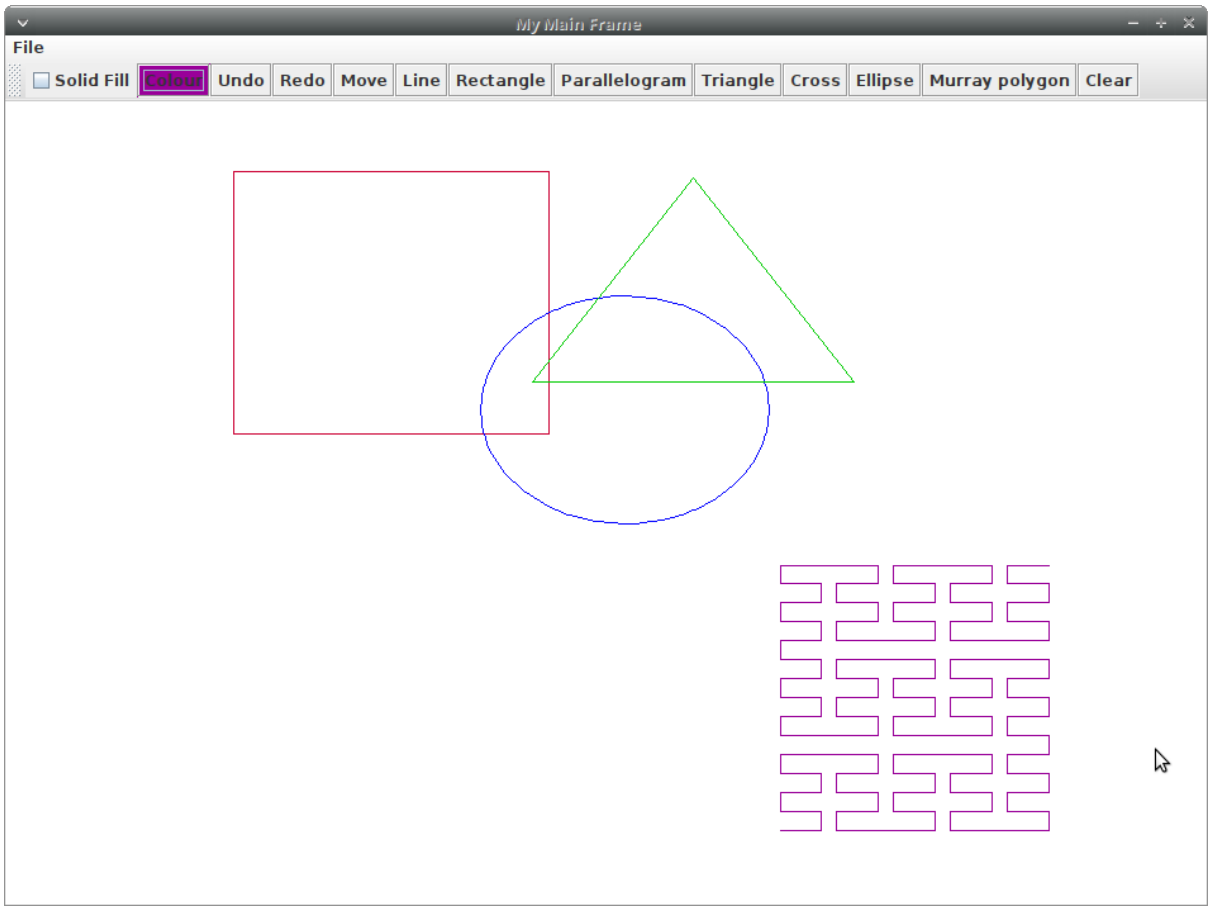

Figure 2: Example GUI layout, showing several shapes and colours. Please don't take this design as a requirement, feel free to be more creative!

they work before submitting. You should include everything necessary to run the tests in your submission, and your readme should explain clearly how to execute them.

Deliverables

For this practical you **must** include a readme file. This file must:

- List all the features you have implemented;
- Explain clearly how to run your program and how to use all its features;
- Give clear instructions on how to run the JUnit tests you have written.

When you are finished, you should submit a zip archive containing your source code, your JUnit tests, and your readme, and submit it to MMS in the usual way.

Marking

A very good attempt at satisfying the basic requirements in an object-oriented fashion, using either the MVC or MD design pattern, can achieve a mark of up to 17. For a 17, you should produce very good, re-usable code which makes proper use of inheritance, association, polymorphism and encapsulation, with very good method decomposition.

To achieve a mark of 18 or above your code must in addition make a very good attempt at one or more of the advanced requirements. Attempting 3 advanced requirements could award a grade of up to 20; however, note that the quality of your core program is the most important aspect of this practical, and adding more and more features is no substitute for good object-oriented design.

See the standard mark descriptors in the School Student Handbook:

https://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/feedback.html#General_Mark_Descriptors

Lateness

The standard penalty for late submission applies (Scheme B: 1 mark per 8-hour period, or part thereof):

https://info.cs.st-andrews.ac.uk/student-handbook/learning-teaching/assessment.html#lateness-penalties

Good Academic Practice

The University policy on Good Academic Practice applies:

https://www.st-andrews.ac.uk/students/rules/academicpractice/