COMP1206 PROGRAMMING II LECTURE NOTES SEMESTER TWO, 2016

comp1206 programming ii - assignment one

(Submission Deadline: 4pm, Friday 18th March 2016)

Please direct any queries regarding these instructions to Dr Julian Rathke (jr2@ecs).

You can expect to receive your mark by: Friday 22nd April 2016. A small amount of individual feedback will be given for this assessment and general feedback may be given in a lecture after the Easter break.

assignment instructions: an interactive fractal explorer

For this assignment I would like you to build a Java application which creates a Graphical User Interface which is capable of rendering a depiction of the well-known fractal, the Mandelbrot Set. In addition to this, the user will be able to a select a point from the Mandelbrot set display using a mouse click. For this selected point, the Julia Set corresponding to that point will be visualised in an adjacent window or panel.

I'll give more details about how to construct this application below. First though I'll explain how the fractals are calculated.

The Mandelbrot Set is actually a set of complex numbers. We can calculate (an approximation to it) using an iterated formula over complex numbers: given any complex number, **c**, we calculate the sequence of values:

Z(0), Z(1), Z(2), Z(3), ... where Z(i+1) = (Z(i) * Z(i)) + **c** and Z(0) = **c**

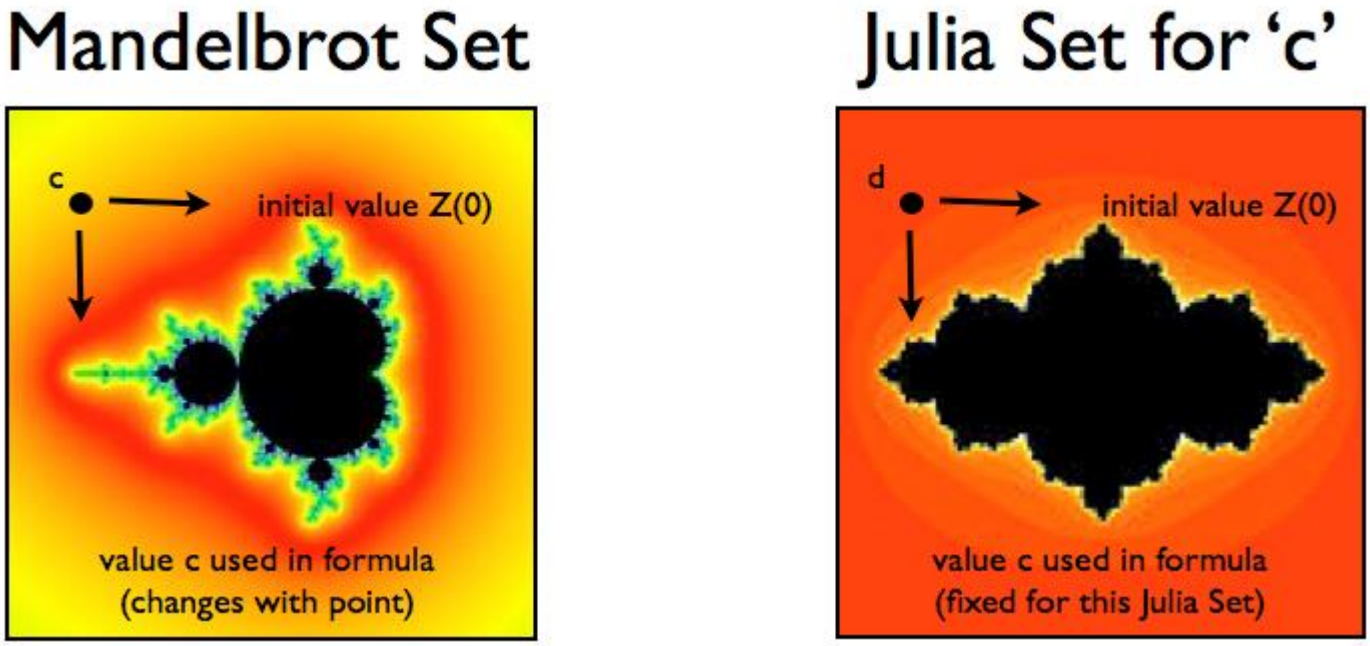
Now, point **c** is said to be in the Mandelbrot Set if the sequence of Z(i) values does not diverge, that is, if the values of Z(i) do not just keep getting bigger and bigger. In practical terms, we are not going to be able to calculate **all** values of Z(i) for each point **c**. Thus we must limit ourselves to some finite subsequence of the Z(i) values. Let's say we calculate up to value Z(n) only (for some fixed n). We could approximate the Mandelbrot Set if we include point **c** whenever the value of Z(i) up to Z(n) has not yet diverged. To test for non-divergence, it is sufficient to check that the **absolute value**(or modulus) of Z(i) (usually written |Z(i)|) is less than **2** for each Z(i) up to Z(n).

To visualise the Mandelbrot set in a GUI, we need to calculate, for each pixel in our display area, which point **c** of the complex plane this pixel represents, and then calculate the sequence

of Z(i) values up to some fixed limit for this value of **c**. If the Z(i) values are all non-divergent then we plot the pixel in some fixed colour. If one of the Z(i) values for point **c** fails the non-divergence test, then we can plot the pixel in some other colour. In fact, it is useful to have range of colours in which to plot the pixel depending on how far in to the sequence of Z(i) values the sequence fails non-divergence. For example, for points which diverge within 10 iterations, I may plot in red. For points which diverge only after 40 iterations, I may plot in orange, etc.

A Julia Set for Complex number **c** is calculated (and displayed) in a very similar way to the Mandelbrot Set, in fact, the sequence of values is given by the same formula: Z(i+1) = (Z(i) * Z(i)) + **c** but this we fix **c** and range over the complex plane with value **d**, say. We then start with Z(0) = **d** rather than Z(0) = **c** above. The point **d** is plotted according to the same non-divergence test as for the Z(i) values. Thus the difference between the Mandelbrot Set and Julia Sets is that for the Mandelbrot Set, the value of **c** changes for each point you are plotting, but for Julia Sets, it is fixed. Here is a diagram to help explain this and show you an example Julia Set (the colours here will most likely differ from

your own solution):



This means that every point **c** in the Mandelbrot set can be used to generate a *different* Julia Set.

Here are the steps to building the application which you must follow:

Part One

Define a Complex numbers class. This class should provide (at least):

- public accessor methods for the real and imaginary parts of the complex number
- A public method `square` which squares this complex number
- A public method `modulusSquared` which returns the square of the modulus of this complex number

- A public method `add(Complex d)` which adds the complex number d to this complex number

Part Two

Build a GUI application which displays the Mandelbrot Set. You should use a `JFrame` for the main window and a `JPanel` for displaying the image. You should provide some text fields to allow the user to see which portion of the complex plane is represented by your display and also to let them change this. I would recommend the portion of the plane stretching from -2 to 2 in the Real Axis and from -1.6 to 1.6 in the Imaginary Axis as the default values. To translate pixel coordinates in your display to points in the complex plane, you will need to know the width and height of your drawing panel.

You should also allow the user to set the number of iterations used in the calculation of the set. Try a default value of 100 iterations - use a lower number if the image is rendered too slowly.

Part Three

Implement an event listener which handles the user click on a point of the Mandelbrot display panel and responds by displaying the Complex number represented by this point in a suitable GUI component.

Let's call this complex number the *user selected point*.Part Four

Extend your GUI application with a display for the Julia Set corresponding to the *user selected point*. For this extra display, fix the portion of the complex plane represented to be the recommended default from Part Two that is: numbers with Real values between -2 to 2 and Imaginary values between -1.6 to 1.6.

Draw the Julia Set corresponding to the *user selected point* in this extra display.

At this point your whole GUI should display, a main display showing a portion of the Mandelbrot Set, text fields with data on which portion of the complex plane is being viewed in the main display, the *user selected point* (if one has been selected) and another display showing the Julia Set for the*user selected point* (if one has been selected).

Part Five

Implement a facility to mark certain Julia Set images as favourites. Allow the user to select to display from a list of favourites (the favourites can just be stored temporarily while the application is running, or in a file for future use).

Part Six

Implement a listener in the main display panel to allow the user to select the portion of the complex plane represented by dragging a rectangular selection with the mouse. This would

allow the user to 'zoom in' on the Mandelbrot set image more easily. Make sure that a clear visual representation of the drag rectangle is provided as the user drags the mouse.

If you have made it this far and completed everything well, then you could expect a decent mark in the 60-70% range. To obtain higher marks than this, read on: Part Seven

Here is a list of possible improvements and extensions (in no particular order). Implement at least three of these for all possible marks:

- In rendering the image of both the Mandelbrot Set and Julia Sets, use a number of helper threads to calculate the image. Draw the image in an offscreen buffer and then display it once all of the helper threads have finished. Be efficient here - don't keep creating new threads each time you want to change the display!
- Allow the user to select different iterative formulas for calculating the fractal image in the main display and Julia Set window. For example, use Z(i+1) = (|Re Z(i)| + i |Im Z(i)|)^2 + c for the "Burning Ship" fractal. See the Wikipedia page on the Mandelbrot Set for more suggestions. Make sure that your code is easily extendible to incorporate other formulas.

- Animate your zooming feature and provide live updates in the Julia Set window as the mouse is moved in the main display.
- Implement Orbit Traps and Region Splits for more interesting images.
- **YOUR OWN IDEA HERE** - feel free to add your own extensions to your solution. Marks will be given in accordance with how difficult we perceive the extension to be to implement.

These extensions are worth up to a maximum total of 6 marks.
Good Practice

Make sure that your code is well organised, concise and well commented. I will award a mark for well presented code that compiles as submitted using Java 1.8 To compile I will use the command line javac command `javac *.java` in your submission directory. If you have used package declarations etc then make sure that your submission directory accounts for these and **make sure that your main class is not within a named package**.

mark scheme

In total there are 20 marks available, which will contribute 40% towards your overall course mark. The breakdown of available marks is as follows:

- Good practice (compilable code): 1 Mark
- Parts One-Three: 7 Marks
- Parts Four-Five 4 Marks
- Part Six: 2 Marks
- Part Seven: 6 Marks

additional information

Submission Information:

Please place of all of your source files in to a single directory and make a zip archive of it. Please do not use RAR archiver and **do not include any compiled bytecode** (i.e. .class files). You should submit your zipped directory.

You should use the automated ECS hand-in facilities found at:https://handin.ecs.soton.ac.uk

Originality of work:

Please be aware of and adhere to the University regulations on collusion and plagiarism, as outlined in the University Calendar on Academic Integrity.http://www.calendar.soton.ac.uk/sectionIV/

Page maintained by Julian Rathke.