CSP1150 Programming Principles

Programming Project – Monte Carlo estimation of areas

In this assessment, you are going to explore using randomized simulation to calculate areas, and make some nice images along the way.

This assessment is worth 15% of your mark for CSP1150.

Please remember when you are working on this project that, unlike the weekly workshop tasks, this project is an **individual** task. You should not be getting help from anyone else except your tutor and lecturer. If you break this rule, you can expect to be in serious trouble for plagiarism.

At the end you will submit two programs, one that estimates the area of an annulus, and one that estimates the area of a Mandelbrot. To help you to get to your final solution, instructions are provided in 4 Stages.

You can start work at any time. Please do not leave it until the last few days – you will probably do very poorly if you do that. Remember that you can ask your lecturer or tutor for help at any time (but no-one else!).

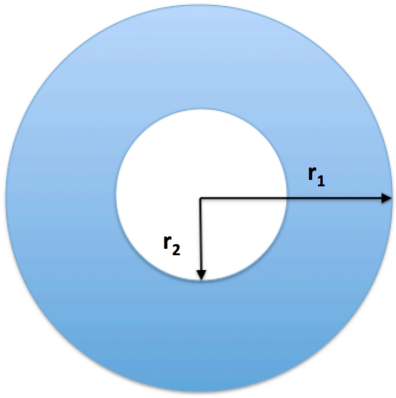
There will be marks for how well your program works, but also for how well it is written. All submissions are done via BlackBoard – go to the Assessments page.

SUBMISSION DEADLINE : Due Midnight Sunday 26th October. If you need an extension for valid reasons, you must request an extension before the deadline. The late penalty is 0.75 marks per day for up to 5 days, and then a mark of 0.

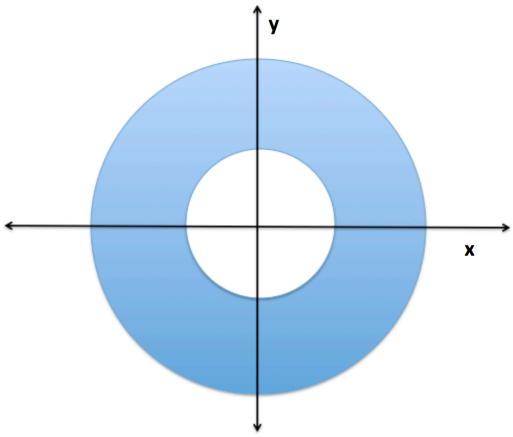
Stage 1 – Approximate area of an annulus

You should be able to complete this Stage after you have learned the material up to Module 5.

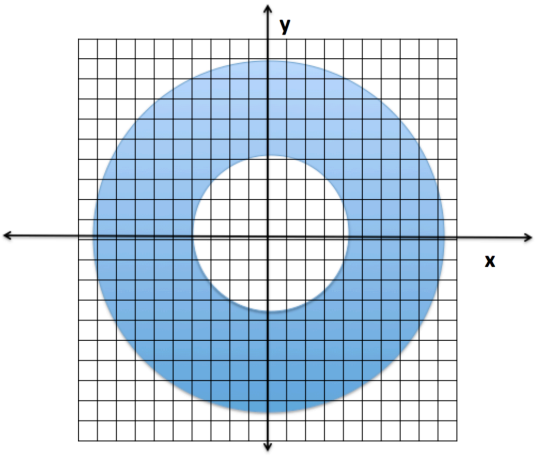
The shape below is called an *annulus*. The *outer radius* is r_1 and the *inner radius* is r_2. Stage 1 is to write a Java program to estimate the area of an annulus.



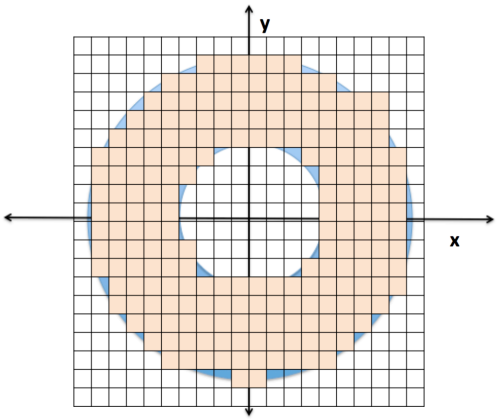
Your task in Stage 1 is to write a Java program that calculates an approximate area for an annulus, based on imagining a grid of small cells laid over a rectangle containing the shape. Let's see how that would work with an annulus with outer radius 2 and inner radius 1. Let's place this annulus on a coordinate system (sort of like a map).



And now imagine a grid laid over the top of that (like the grid lines on a map).



Now (the centre points of) some of these grid cells are on the annulus, and some are not. Let's shade in those cells that are on the annulus.



In this case 199 cells are marked, out of 400 cells (20 across by 20 down). As a fraction (199/400) this is about 0.4975 i.e. about 49.75% of the cells are marked. We can work out the area of the rectangle formed by all the grid cells (it's about 18.5). So the area of the annulus is about 0.4975 times 18.5, which is about **9.2**. We could have used any rectangle that covers the whole annulus for this calculation, and we could have made the grid cells smaller (the smaller they are, the more accurate the calculation should be).

The area of an annulus is $\pi \times (r_1 \times r_1 - r_2 \times r_2)$ where r_1 is the outer radius and r_2 is the inner radius. Using this mathematical formula for the area, we should actually get about **9.43**.

What went wrong? Well, my drawing is not very accurate! But we could do it better: we could write a program to do the calculation for us. In pseudocode, here is what the program should do:

```
BEGIN Area of Annulus
      Set gridSize = 20 (let's use 20 by 20 grid cells)
      Get r1 from user (outer radius)
      Get r2 from user (inner radius)
      Set counter = 0 (will count how many grid cells are on the annulus)
      For each grid cell
            if the centre of the cell is on the annulus
                  Set counter = counter + 1
      Set area = (rectangle area) * counter/(gridSize * gridSize)
      Output "Area : " area
END Area of Annulus
```

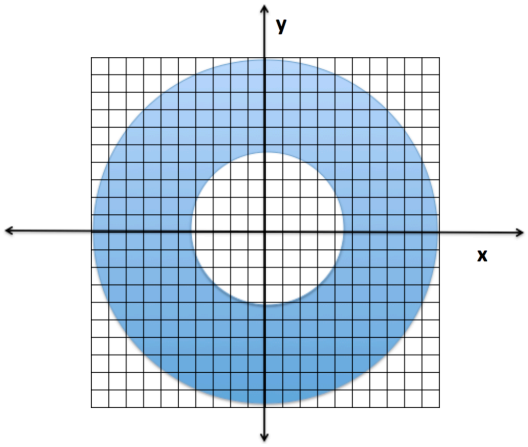
Before we could turn this into a Java program, we have to fill in some of the details.

1. How big should the rectangle of grid cells be?
2. How do we figure out where the centre of each grid cell is?
3. Once we know that, how do we figure out whether it is on the annulus or not?

Let's tackle these one at a time:

1. How big should the rectangle of grid cells be?

Any rectangle that completely covers the annulus will do. Let's say the rectangle goes from minx to maxx in the x direction, and from miny to maxy in the y direction. We might as well use this rectangle:



so minx = -r1, maxx = r1, miny = -r1, maxy = r1.

2. How do we figure out where the centre of each grid cell is?

This is tricky to work out so I'll just give you the answer:

If you start counting the columns of grid cells from the left, starting at zero, then the x-coordinate of a cell is $\min x + (column + 0.5) \times ((\max x - \min x) / gridSize)$.

And if you start counting the rows of grid cells from the bottom, starting at zero, then the y-coordinate of a cell is $\min y + (row + 0.5) \times ((\max y - \min y) / gridSize)$.

And finally

3. Once we know that, how do we figure out whether it is on the annulus or not?

Again, tricky. To be on the annulus, a point has to be inside a circle of the outer radius, but not inside a circle of the inner radius. Using a formula worked out by Pythagoras, this means that if the point has x-coordinate x and y-coordinate y, then for it to be on the annulus:

$x \times x + y \times y < r1 \times r1$ (the point is inside the outer circle), and
$x \times x + y \times y > r2 \times r2$ (the point is outside the inner circle).

Putting all this together in pseudocode:

```
BEGIN Area of Annulus
      Set gridSize = 20 (let's use 20 by 20 grid cells)
      Get r1 from user (outer radius)
      Get r2 from user (inner radius)
      Set minx = -r1
      Set maxx = r1
      Set miny = -r1
      Set maxy = r1
      Set counter = 0 (will count how many grid cells are on the annulus)
      For column = 0 to gridSize-1
            Set x = minx + (column + 0.5)*((maxx-minx)/gridSize)
            For row = 0 to gridSize-1
                  Set y = miny + (row + 0.5)*((maxy-miny)/gridSize)
                  Set test = x*x + y*y
                  if test < r1*r1 and test > r2*r2
                        Set counter = counter + 1
      Set area = (maxx-minx)*(maxy-miny)*counter/(gridSize * gridSize)
      Output "Area : " area
END Area of Annulus
```

Notice that when this is written in Java, it will use nested *for* loops.

Your task for Stage 1 then, is to write and test a Java program based on the pseudocode above. You should also have your program print out the area as calculated by the formula.

Stage 2 – Monte Carlo estimation of the area of an annulus

You should be able to complete this Stage after you have learned the material up to Module 7.

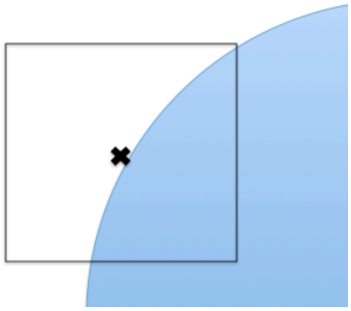
In this Stage, you are going to make the area calculation using a grid more accurate by using random sampling.

Instructions:

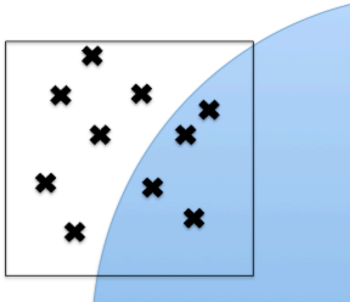
Step 1 Adding random sampling

In Stage 1, you wrote a program to estimate the area of a shape (an annulus) by laying a grid of cells over the shape, and testing whether the centre point of each cell is on the shape or not. One way to make the area estimate more accurate is to test more than one point in each grid cell (not just the centre).

Here's a magnified picture of the idea so far:



Up to now we have just tested the centre point, and either added the whole cell area to the area of the shape, or none of it. A better estimate for the example above might be to add about 40% of the area of this cell. We can estimate this % by scattering a few points inside the cell, and seeing how many are on the shape, like this:



It turns out to be a better idea to randomly scatter points like this, rather than testing a regular pattern of points inside the cell.

To change your program to do this calculation, you first need to work out what changes to make to the pseudocode design from Stage 1. One way to do this is with a two dimensional array, which can be used to count how many random points in each cell are on the shape. Here is a modified design:

```
BEGIN Area of Annulus
      Set gridSize = 20 (let's use 20 by 20 grid cells)
      Create a 2D array, hits, to count how many points are on the shape
      Set samples = 100 (let's use 100 random points inside each cell)
      Get r1 from user (outer radius)
      Get r2 from user (inner radius)
      Set minx = -r1
      Set maxx = r1
      Set miny = -r1
      Set maxy = r1
      Set counter = 0 (will count how many grid cells are on the annulus)
      For column = 0 to gridSize-1
          For row = 0 to gridSize-1
            Set hits[column][row] to 0
            Repeat samples times
                  Set x = minx + (column + random)*((maxx-minx)/gridSize)
                  Set y = miny + (row + random)*((maxy-miny)/gridSize)
                  Set test = x*x + y*y
                  if test < r1*r1 and test > r2*r2
                        Set hits[column][row] = hits[column][row] + 1
            Set hits[column][row] = hits[column][row]/samples
            Set counter = counter + hits[column][row]
      Set area = (maxx-minx)*(maxy-miny)*counter/(gridSize * gridSize)
      Output "Area : " area
END Area of Annulus
```

The main changes here are

1. Adding the constant *samples;*
2. Adding a 2D array (*hit*) to store result for each cell;
3. Changing the 0.5 in the calculation of x and y to *random – random* represents a random value between 0 and 1;
4. Adding another level of nesting to the loop (an extra loop inside the innermost *for* loop). This is shown as a *repeat* in the pseudocode;
5. Moving the calculation of x to inside the innermost loop; and
6. *counter* now totals up fractions instead of just 1's and 0's (so it will need to be a double).

Your first task is to modify your program to implement these changes (and of course test and correct any errors).

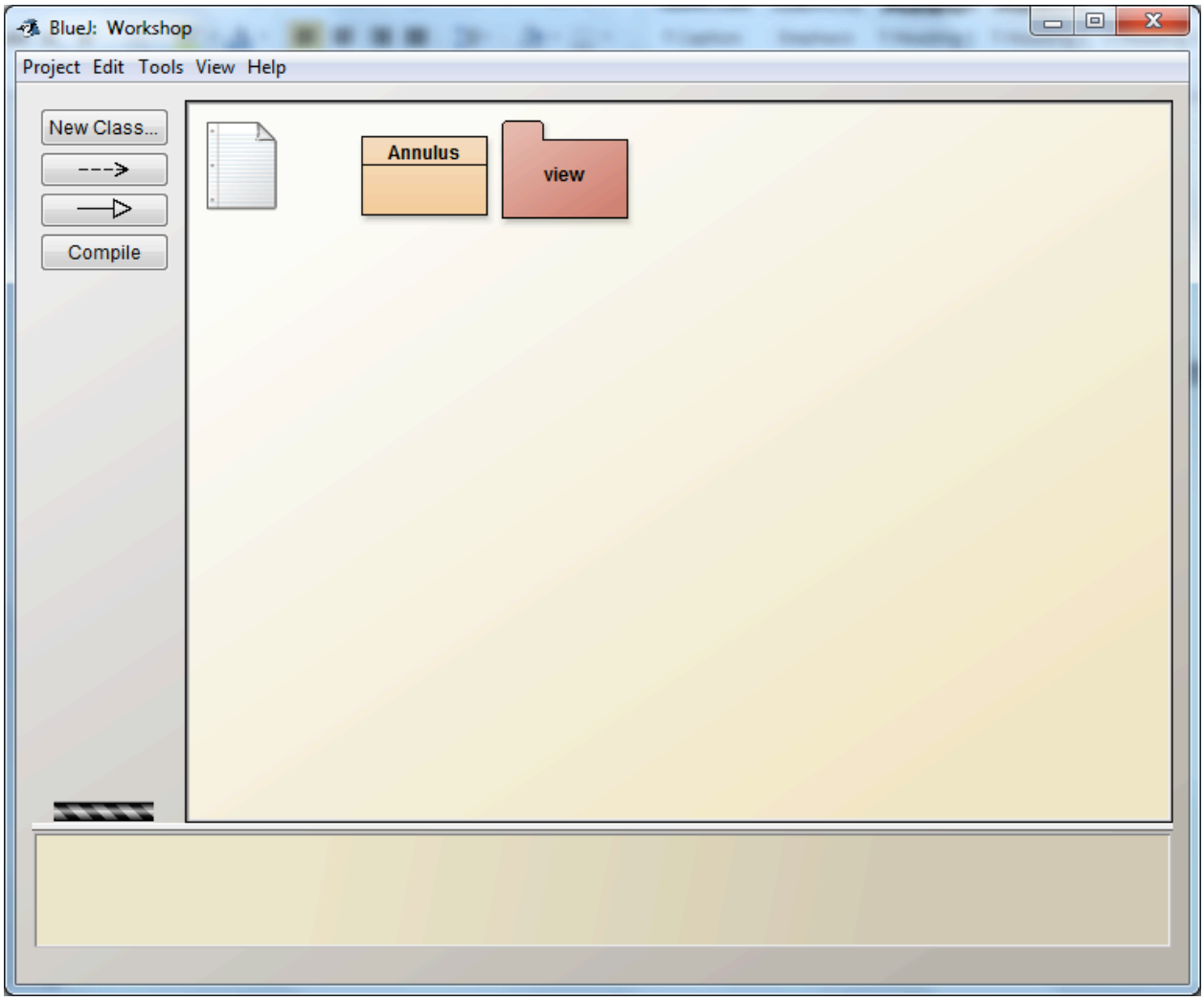
This should be straightforward, except that you need to know how to get a random value between 0 and 1. One way to do that in Java is to replace "0.5" in your code with "Math.random()". This is a subroutine or method provided by Java as part of the Math class. Note also that *counter* will now need to be a double instead of an int.

Step 2 Adding a greyscale image of the shape

Now you have been working hard and it is time you had some fun.

On BlackBoard you will find a file called "viewing.zip". Download it and unzip it. When it is unzipped, you will find a folder called "view". If it is open, close your BlueJ project. Find the folder that contains your project source code (the folder that

has your .java file in it). Copy the “view” folder into the project folder. Now open the project again in BlueJ. Select Edit->New Package and enter the name “view”. You should see something like this:



Now “view” is a Java package that contains some classes that can be used to view a 2D image described by an array of greyscale values.

To use it you will need to add an import:

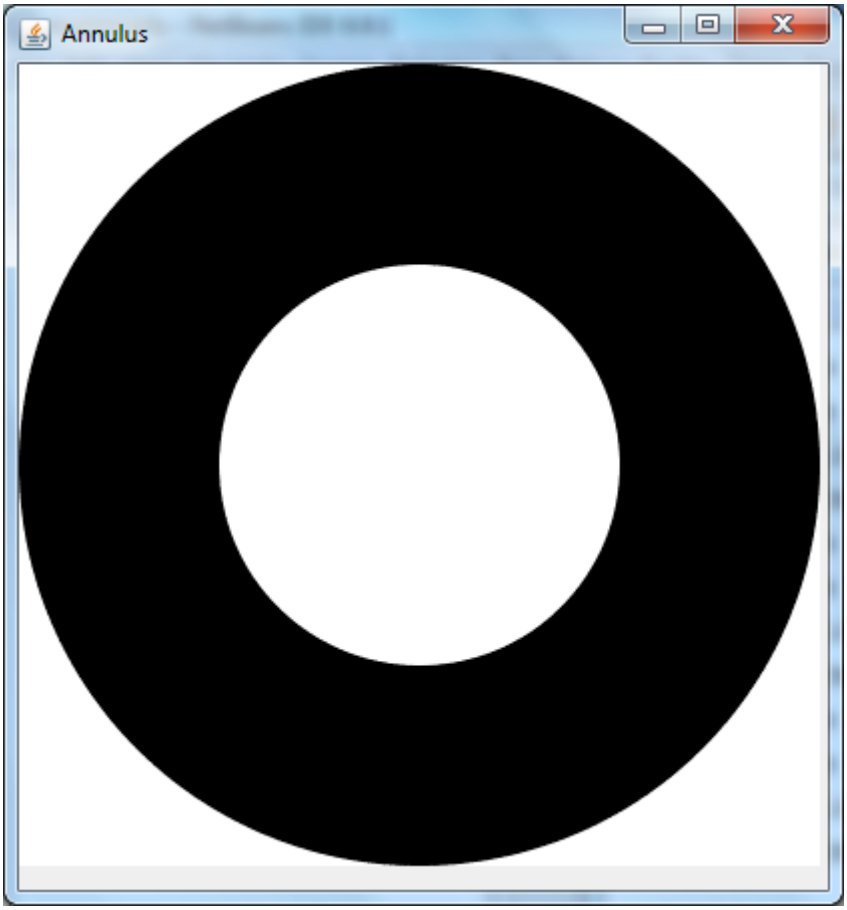
```
import view.*;
```

and add a line like this to the end of your main() method:

```
(new GreyscaleHitViewerFrame(“Annulus”, GRID, GRID)).viewHits(hits);
```

where GRID is the size of your grid (you may have called it something else – and you will probably want to make it something larger than 20 – say 200 or more). If all

is working correctly, when you run your program, a window like this should appear:



Stage 3 – Monte Carlo estimation of the area of a Mandelbrot

You should be able to complete this Stage after you have learned the material up to Module 7.

Now we have two methods for finding areas – use a mathematical formula, or use a grid and some random sampling. One nice thing about the second method is that it can be used for difficult shapes, where it might be hard to find a mathematical formula. All you need is a way to test whether a sample point is on the shape or not.

We can illustrate this by modifying your program to find the area of a complicated shape called a Mandelbrot (read about these shapes by using your googling skills).

Follow these steps:

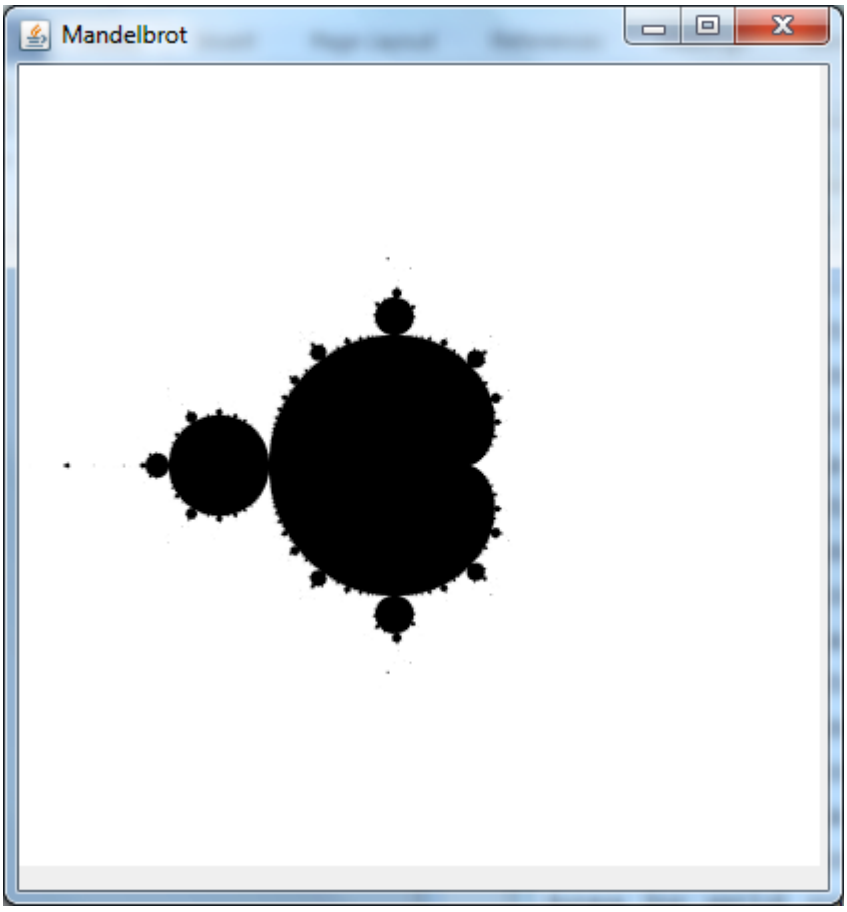
- First, make a new class in your project, called Mandelbrot. Then copy the source for your annulus program into the new class, and edit it to change the name of the class to Mandelbrot.
- Add a new constant, MAX_STEPS = 255
- Remove the code that asks the user for the dimensions of the annulus

- Set MINX = -2, MAXX= 2, MINY = -2, MAXY = 2
- Next, find the place in your code where you test whether or not (x,y) is on the shape, and replace it with this code:

```
boolean inside = true;
int steps = 0;
double px = 0;
double py = 0;

while(steps < MAX_STEPS && inside)
{
    inside = px*px + py*py < 4.0;

    double py1 = 2*px*py + y;
    px = px*px - py*py + x;
    py = py1;

    steps++;
}
```

- Change your code to use the value of *inside* to decide whether to increment the hit count for the cell.

If all goes well, you should see an area of about 1.522, and something like this:



By changing the values of MINX, MAXX, MINY and MAXY in your program, you can zoom in on different parts of the shape. For example, try the values MINX = -0.81, MAXX = -0.79, MINY = 0.17 and MAXY = 0.15. (Note that this will take longer to run).

Stage 4 – Improve your program

You should be able to complete this Stage after you have learned the material up to Module 8.

Before you submit your assignment, please make sure to use what you have learned about methods improve the design of your two programs. You **should not** put all your code into the main() method in each program.

As a suggestion you could have your main() method call the following methods:

```
public double getMinX();
public double getMaxX();
public double getMinY();
public double getMaxY();
public boolean isInside(double x, double y);
```

Also, make sure your code is correctly and consistently indented, your variable, constant, method and class names follow our naming conventions, your methods and classes have Javadoc comments with the correct tags, and your code contains other explanatory comments where they are needed.

Submission

What to submit:

Please submit a zip file containing your source (.java) files. It is your responsibility to make sure you submit the correct files. For example, do not submit your .class files without the corresponding .java files. Do not submit a project folder that does not contain any .java files. Check the file that you intend to submit and **make sure** that it contains your .java files.

Mark allocation (15 marks total):

- Correctness: **10 marks**
  - Annulus program correctly calculates the area of an annulus: **5 marks**
  - Mandelbrot program correctly calculates the area of a Mandelbrot: **5 marks**
- Code quality and programming style: **5 marks**
  - Layout (indenting, use of empty line etc.)
  - Commenting (appropriate use of descriptive comments, including Javadocs)
  - Choice of programmer defined identifiers (use meaningful names)
  - Conformance to naming conventions
  - Appropriate use of object-oriented concepts (methods, classes)

Optional Extra

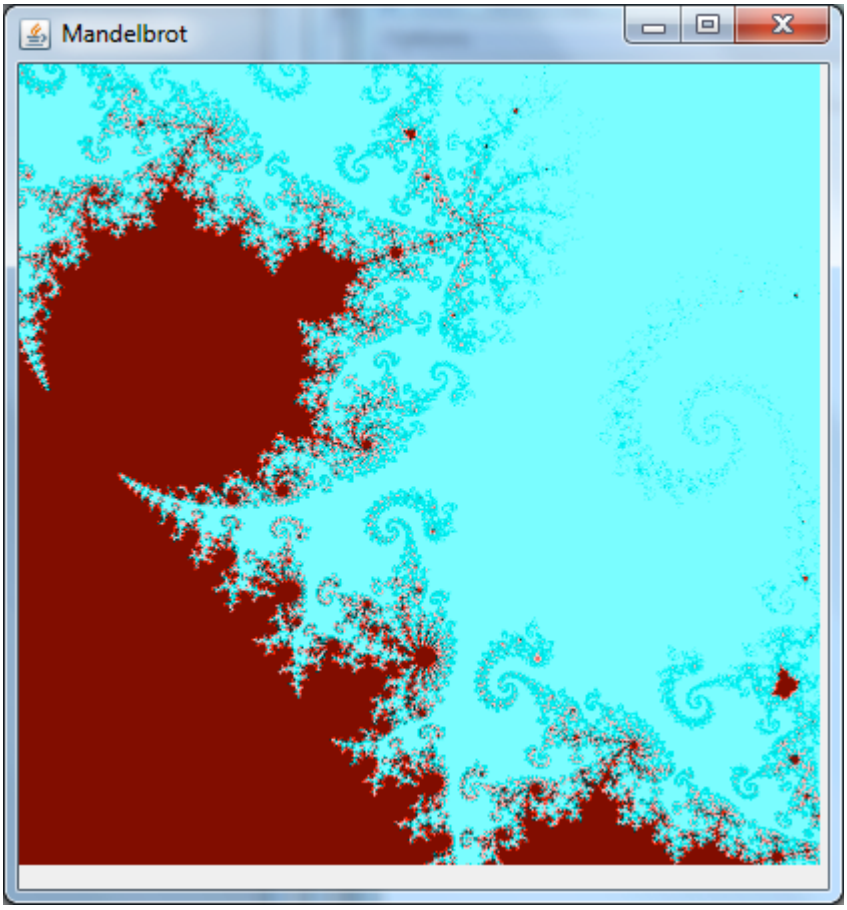
For interested students, there is an optional extra task worth up to 5%.

The requirements are deliberately vague. Use your initiative to fill in the gaps, and make them complete, clear and unambiguous, then check with your lecturer/tutor before proceeding with implementation.

Your task is to design, implement and test some interesting additional functionality for the area calculation program. For this extension, we are only interested in the Mandelbrot shape.

There are two choices for the extension:

1. Add a nice graphical user interface (GUI) from end to the program. Some features that you could consider:
   a. Let the user draw a rectangle on the image, end then recalculate and redraw using only that part of the image inside the rectangle (letting the user zoom in on part of the image)
   b. Provide a “save” function to save the current image to a file.
2. Instead of each point being “inside” or “outside”, count how many steps it takes to escape for the loop `while(steps < MAX_STEPS && inside)`, (this is called the escape time). For each cell, work out the average escape time. Make a new kind of HitViewer class to use this average escape time to choose the colour of that cell in the image. You should be able to produce images like this:



Zip up and submit your BlueJ project, along with a description of the new functionality that you have added, by midnight on 2nd November.