Operating Systems **CSI3131** ***Lab 2b*** *Winter 2018*

Java Threads.

Goals

1. Learn how to use threads – the Java Thread class.
2. Learn how to use thread pools – the Java Executors class.

.

Guideline

For this part of the lab, you will be working with an application that creates an illustration of the Mandelbrot set. The image is generated by painting a set of rectangles in a Windows frame. By using threads (and thread pools) you will be able to visually see the execution of threads. For information on the Mandelbrot set (not quite important for understanding the key idea of the lab but quite interesting), consult the following sites:
http://www.ddewey.net/mandelbrot/ and
https://en.wikipedia.org/wiki/Mandelbrot_set

This interactive site gives you some visual understanding regarding the Mandelbrot set:
http://guciek.github.io/web_mandelbrot.html

Step1 - Generating your first diagram: Unzip the java files from MandelBrot.zip. Compile the following java files using you favourite Java development environment.

MBGlobals.java	- provides a class with global definitions
MBPaint.java	- provides the class for computing and setting colors for pixels and filling a given square of the canvas.
MBCanvas.java	- provides the class that extends Canvas, a component that provides a drawing canvas.
MBFrame.java	- provides the class that extends JFrame, an object for creating a window (an MBCanvas object is added to the window frame).
MandelBrot.java	- The main application – reads arguments to set the global values in MBGlobals, and then creates an MBFrame object to display illustration of Mandelbrot set.

After compiling the java application, experiment with creating different views of the set using the following command:

java MandelBrot <Upper x Coord> <Upper y Coord> <Real Dim> <Pixel Dim> <Fill Dim>

where

<Upper x Coord> and **<Upper y Coord>**: are the coordinates of the upper left hand corner of the diagram in terms of real values. To get an overall view of the set, use an upper corner of -2,2 with a dimension of 4 (see Real Dim).
<Real Dim> - The application displays a square diagram. This argument gives the dimension of the square in terms of real values. Thus, a value of 4 produces a diagram that will be 4 X 4 units. Using -2, 2, as the upper left hand corner coordinate thus produces a diagram that has as coordinates -2,2 (upper left hand corner), 2,2 (upper right hand corner), -2,-2 (lower right hand corner), and (2,-2) (lower right hand corner).
<Pixel Dim> - This is the dimension of the displayed canvas in terms of pixels. This argument controls the size of the window containing the diagram.
<Fill Dim> - This is the dimension of the square for filling in the pixels. In the provided code, recursion is used to create a number of MBPaint objects for filling in the various sections of the Canvas. The objective of the lab is to change the code such that threads are used to fill in these squares.

Try the following parameters:

java MandelBrot -2 2 4 600 50 - For a general view. Modify 600 to get different size windows.

Other interesting views can be generated using the following:

java MandelBrot -2 1 1 600 50
java MandelBrot -1 1 1 600 50
java MandelBrot -2 0.5 1 600 50

Note: To complete Step 2 and Step 3, you only need to modify file MBCanvas.java Mainly focus on and understand function findRectangles

Step 2 - Using Threads : Using the Java Thread Class, modify the provided java code to use threads for filling in the diagram. Try modifying the <Fill dim> argument to see the effect of filling in the diagram. Note that as you reduce this value, the number of threads used is increased.

Step 3 - Using a Thread pool: Java provides a class "Executors" for creating Thread pools. The static method "newFixedThreadPool(int sz)" can be used to create an object of class *ExecutorService* that manages a thread pool consisting of *sz* Threads. The object also manages a queue for queuing tasks when all threads are busy. To create such an object that uses a thread pool of 20 threads, use:

```
ExecutorService thpool = Executors.newFixedThreadPool(20);
```

Tasks can then be executed using:

```
thpool.execute(Runnable task)
```

Note that task is an object that implements the Runnable interface. Consult the Java documentation for details on the use of the Executors class.

Modify the Java code to use a thread pool for executing the MBCompute objects as runnable tasks. You should see the effect of limiting the number of tasks that can be created when comparing to the application in B). Try varying the size of the thread pool to see its effect on the execution.