*Operating Systems* CSI3131

**The Java threads.**

**Goals :**

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

# Description

In this part of the lab, you will be working with an application that draws a representation of the Mandelbrot set. The image will be generated by painting a set of rectangles in a Windows window. By using threads (and thread grouping) you can visually see the execution of the threads. For more information on this MandelBrot set and its illustrations, visit the site:

<http://www.ddewey.net/mandelbrot/>

**Step 1 - Your first image of the Mandelbrot set** : Extract the java files from the MandelBrot.zip archive. Compile the java code with your favorite java development tool:

|  |  |
| --- | --- |
| MBGlobals.java | - Returns a class containing global data |
| MBPaint.java | - Gives a class to calculate the color of the pixels of the illustration (it also colors the pixels) in a given square of the image. |
| MBCanvas.java | - Gives the MBCanvas class (extension of Canvas), a GUI component that allows you to make drawings. |
| MBFrame.java | - Gives the MBFrame class (extension of JFrame), to create a Windows window. An MBCanvas object is added to this window. |
| MandelBrot.java | - We're leaving here. Mandlebrot will translate the arguments of the command to store its values ​​in MBGlobals. Then it will create an MBFrame object to show an image of the set  Mandelbrot according to the arguments of the command. |

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

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

or

**<Upper x Coord>**and <Upper y Coord>: are the coordinates of the upper left corner of the illustration in terms of real values. To have a general view of the whole, use the coordinates -2.2 with a dimension of 4 (see Real Dim).

**<Real Sun>**- The app uses a square as a backdrop. Its value corresponds to the real dimension of the graph of the Mandelbrot set. So the value 4 will give an illustration with the dimensions of 4X4 ​​units. With the coordinates -2.2 for the upper left corner, the diagram will have the following coordinates:

* -2, 2 (upper left corner)
* 2, 2 (upper right corner)
* 2, -2 (lower right corner)
* -2, -2 (lower left corner)

**<Pixel Dim>**- This dimension represents the dimension of the backdrop in terms of pixels. It controls the size of the window produced.

**<Fill Dim>**- This dimension controls the size of the square to fill the pixels of the illustration. With the code provided, recursion is used to create multiple MBPaint objects that fill different sections of the artwork canvas (MBCanvas object). The objective of the lab is to change the code in order to use threads to fill these different squares. Try the following arguments:

**java Lab3 -2 2 4 600 50** - For a general view. Modify 600 to get different window sizes. Other interesting views are possible with the following:

**java Lab3 -2 1 1 600 50** java Lab3 -1 1 1 600 50

**java Lab3 -2 0.5 1,600 50**

**Step 2 - Using the threads:** With Java's Thread class, modify the provided code to use different threads to fill each fill square. Modify the <Fill dim> parameter to see the effect of filling the illustration. Note that when you decrease the value of this parameter, the number of threads used increases.

**Step 3 - Using the thread pool:** Java offers the class

"Executors" for the creation of groups of threads. The static method

"NewFixedThreadPool (int nChildren)" create an object of the ExecutorService class which manages a grouping of nChildren children. To create such an object managing 20 threads, use:

ExecutorService thpool = Executors.newFixedThreadPool (20);

Tasks can be performed with:

thpool.execute (Runnable task)

Note that the "task" task is an object that implements the Runnable interface. See the Java documentation for details of using the Executors class and the ExecutorService interface.

Modify the Java code to use a thread pool that will perform the tasks defined by the MBCompute objects. You should see the effect of limiting the number of threads that can run at a time, especially comparing the behavior with the step version

2). Try varying the size of the pool (number of threads) to see the effect of execution.