Langara
THE COLLEGE OF HIGHER LEARNING.

Department of Computing Science & Information Systems
CPSC 1181
Lab#10
November 12, 2019

Objectives:

Practice thread programming and race condition.

Preparation:

Study thread section from your text book and class notes.

Due date:

Due Date: 11:00 PM on Tuesday November 18, 2019

Where to upload:

zip folder to yourStudentID.zip and upload it to Lab10 in D2L.

Tutorial:

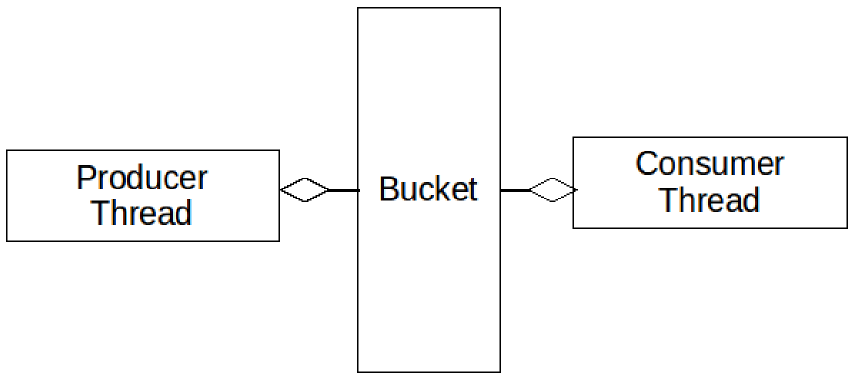
There is no tutorial this week, and attending the lab is not mandatory for this lab session since we have not covered the topic yet due to the long weekend.

Multi-thread programming

The goal of this lab assignment is to create two threads, produce and consumer. The producer thread generates random integer numbers and consumer thread consumes the numbers generated by the producer.

The Bucket class is already developed and you can download it from here. The Bucket class internally uses an ArrayList of integers to add integer numbers to the bucket, finds their median value, and reset the bucket. An instance of this class is shared between both producer and consumer classes.

The rough UML diagram of the the system is shown below.



Phase 1:

Develop class Producer that implements the runnable interface. The class takes an instance of the Bucket class as an argument at construction time. The class performs following tasks *while the Thread is not interrupted*:

- The run() method of the class generates ten random integer numbers in range from 0 to 255 and adds them to the bucket by invoking add(...) method of the Bucket class.
- Put the thread to sleep for 1ms
- Repeats the process while the thread is not interrupted.

Develop class Consumer that implements the runnable interface. The class takes an instance of the Bucket class as an argument at construction time. Note that one single instance of the Bucket class is shared among Producer and Consumer classes. The class performs following tasks *while the Thread is not interrupted*:

- The run() method of the class invokes the median() method of the bucket and prints the returned value on the screen.
- put the thread to sleep for 2ms
- Repeat the process while the thread is not interrupted.

Develop class BucketThreadTester that contains a main method. The class performs following tasks:

- Creates an instance of the Bucket class.
- Creates an instance of the Producer and Consumer classes.
- Starts both of the threads.

Run the BucketThreadTester program. After a few iterations the program will crash with a message like following message:

```
Exception in thread "Thread-1" java.util.ConcurrentModificationException at
java.util.ArrayList.sort(ArrayList.java:1456)at
java.util.Collections.sort(Collections.java:141)
```

The program crashes due to the lack of mechanism to control the problem known as Race-condition.

Pahse2:

Modify the Bucket class by adding ReentrantLock to the class. Implement the locking mechanism to prevent the program from being crashed.

Run the program for at least 15 seconds to be sure that your program is not crashing anymore.

Pahse3:

The program runs indefinitely. Develop a timer in class BucketClassTester that interrupts both producer and consumer after 30 seconds to stop the program.

What to upload

Create one page UML diagram of your design, and save it either as PDF or as word 2003 (JPEG Inserted into word document).

Zip your source files and UML diagram into *your StudentID.zip* file and then upload it to lab10 in D2L Dropbox.

TOTAL MARK: 50

Bonus : [15 marks] Thread Animation:

You are asked to create an animation like this: Sample.mp4

You need to develop following classes:

Develop class Data that implements the runnable interface. The class is partially developed and you can download it from here. The class performs following tasks *while the Thread is not interrupted*:

- The run() method of the class invokes the private setData() method of the class, and then puts the thread to sleep for 50ms
- Repeat the process while the thread is not interrupted

Develop class DisplayPanel by extending it from JPanel and implementing the runnable interface. Note that the difference of this approach with the one provided by your text book. In the example provided by your text book, the thread is not a subclass of JComponent, and it relys on its container to print it. In this approach, the JPanel runs as a thread and it is a self contained thread that displays itself. The class performs following tasks *while the Thread is not interrupted*:

- The class takes a reference of an instance of Data class at construction time.
- Set the size of the panel at construction time by calling:
  - setPreferredSize(new Dimension(300, 300));
- The run() method of the class calls repaint() to repaint the panel, and then puts the thread to sleep for 50ms
- Override the paintComponent of the class that takes an array of integers from Data class by invoking its getData() method, and prints the vertical lines proportional to each element of the array. Do not forget to invoke super.paintComponent(g) method at the beginning of the method.
- Repeat the process while the thread is not interrupted.

Note: Use drawLine(...) method of the Graphics2D class to draw the lines. You do not need to create Line2D objects.

Develop class MainPanel by extending it from JPanel. The class performs the following tasks;

- The class creates an instance of Data class and DisplayPanel and starts them.
- Adds the DisplayPanel to the panel.

Note: You do not need to override the paintComponenet(...) of the MainPanel class, since it is just being used as a container to arrange the DisplayComponenet.

Develop a Frame, set its size to 300x300, and add the MainPanel to the fram.
Compile and run the program.
Optional: For consistency of data and display, it is better to implement locking mechanism, othewise, the data is not exactly what is being displayed. Thought, we still have a race condition, but it is not going to crash your program since one of the threads is just accessor in this case.