# **Object Oriented Programming 2**

# **BBIT 2101**

# **Preet Tailor 094790**

# **Lab 2 Java Threads**

……………………………………………………………………………………………………………………………………………………………

In implementing the Java threads, you may implement the Runnable interface or extend the Thread class.

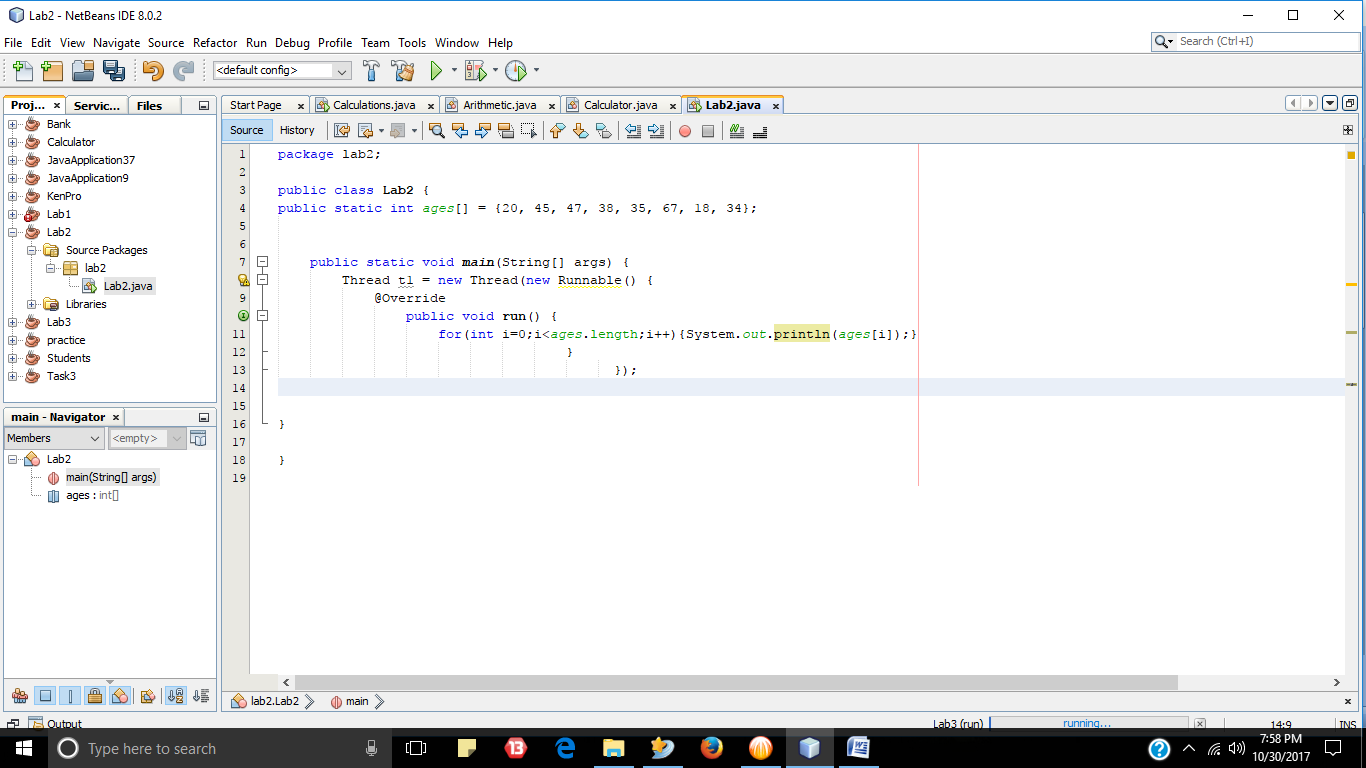
# Section 1

Consider a Java array, which holds ages of people as integer as shown below.

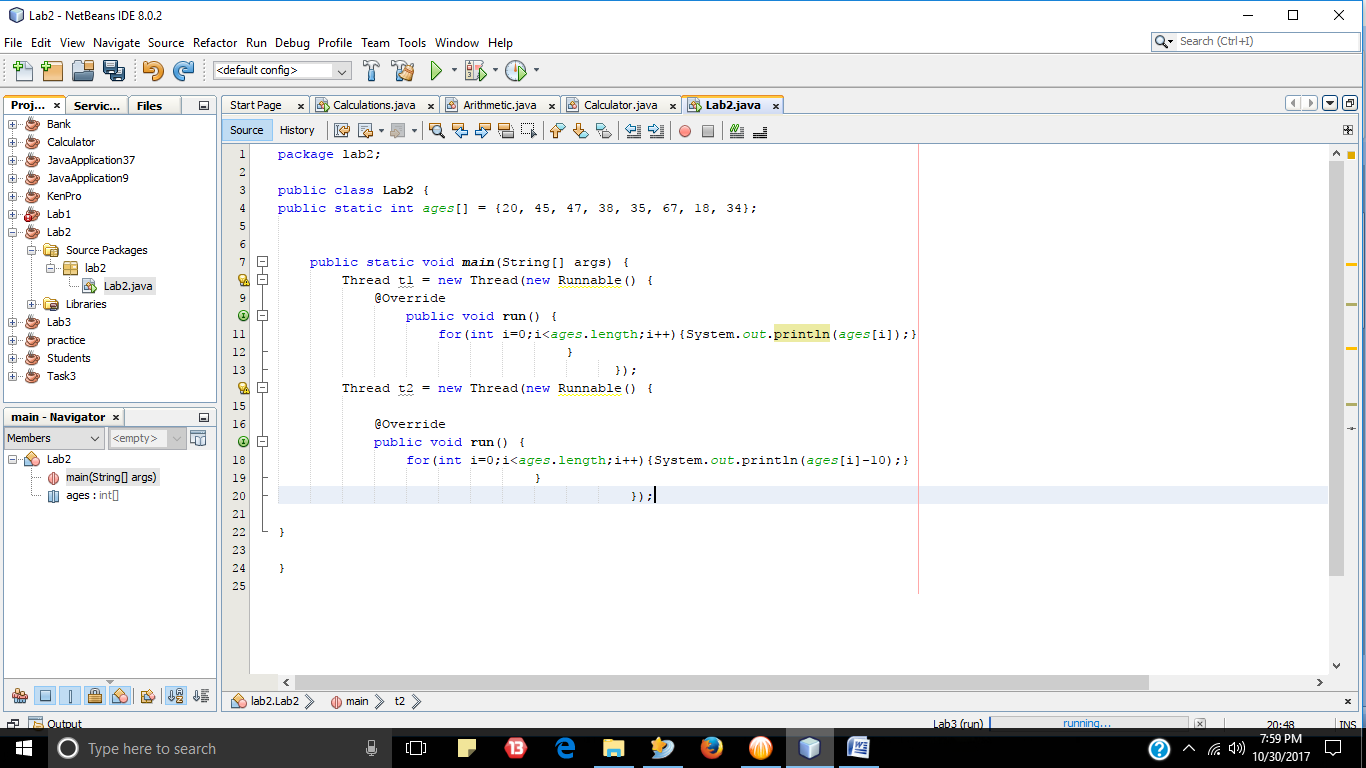
int ages[] = {20, 45, 47, 38, 35, 67, 18, 34}

You are required to write a program that uses the array (above) to perform the following tasks

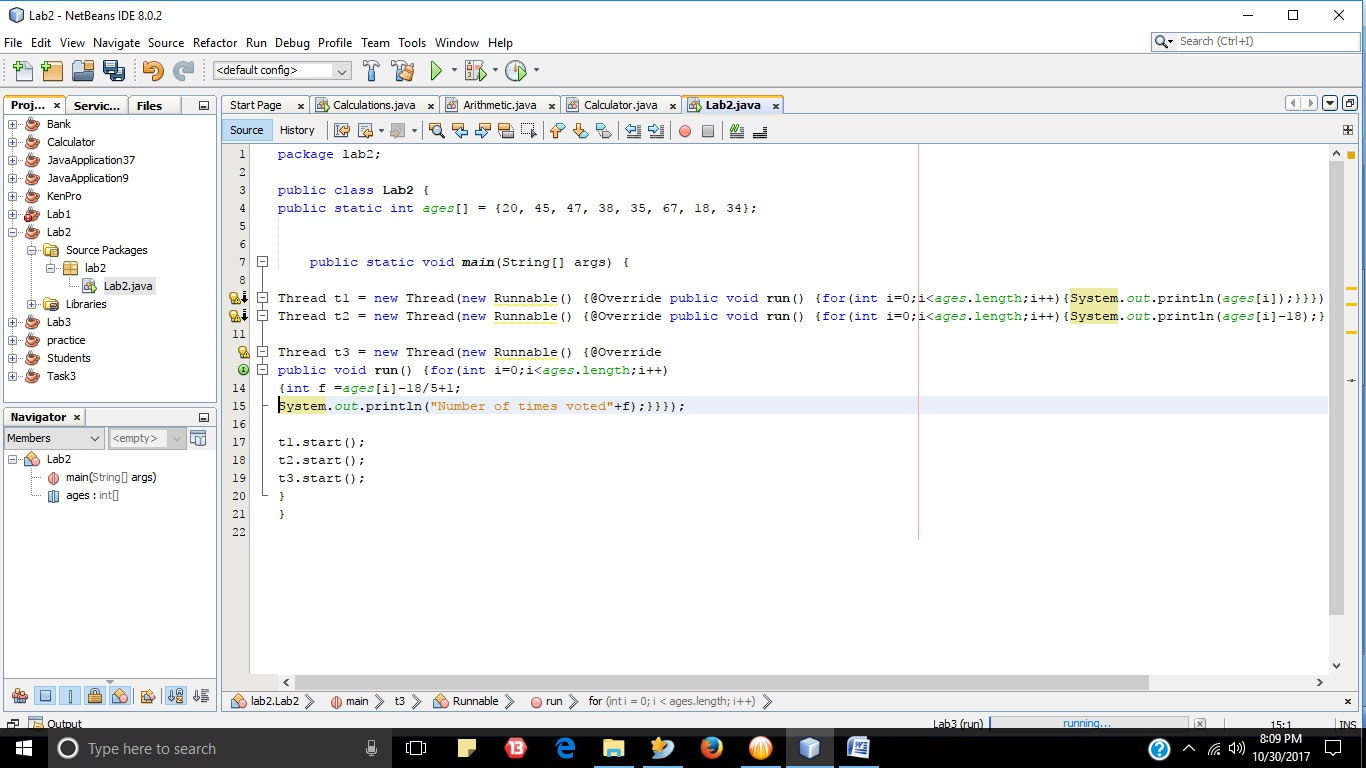
1. Report the ages of people in the array



1. Report the number of years, which have elapsed since they acquired a national identity card (national identity card is acquired at age 18, thereby one is eligible to vote)

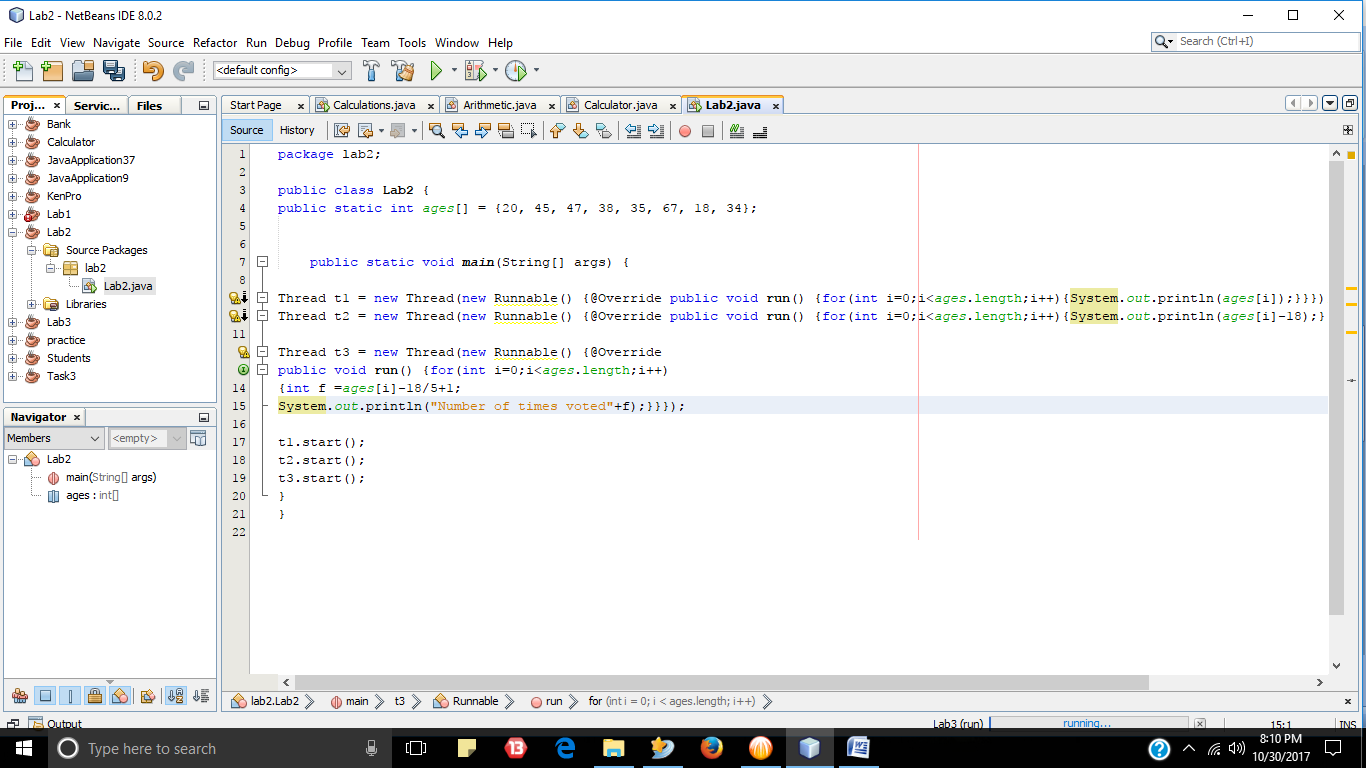


1. The number of times they have voted (assume that all the people whose ages are represented in the array above voted immediately they acquired a national ID and that an elected leader stays in office for 5 years before national election happens again. )

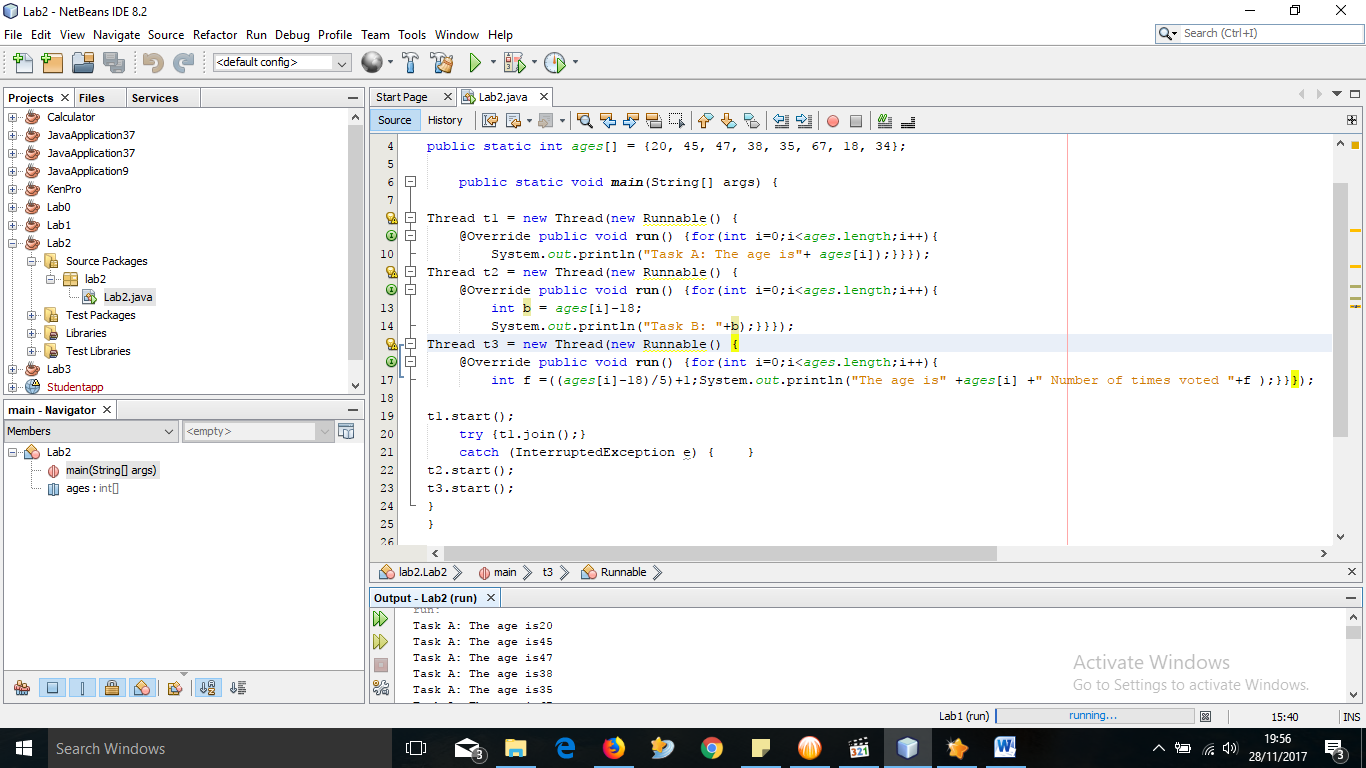


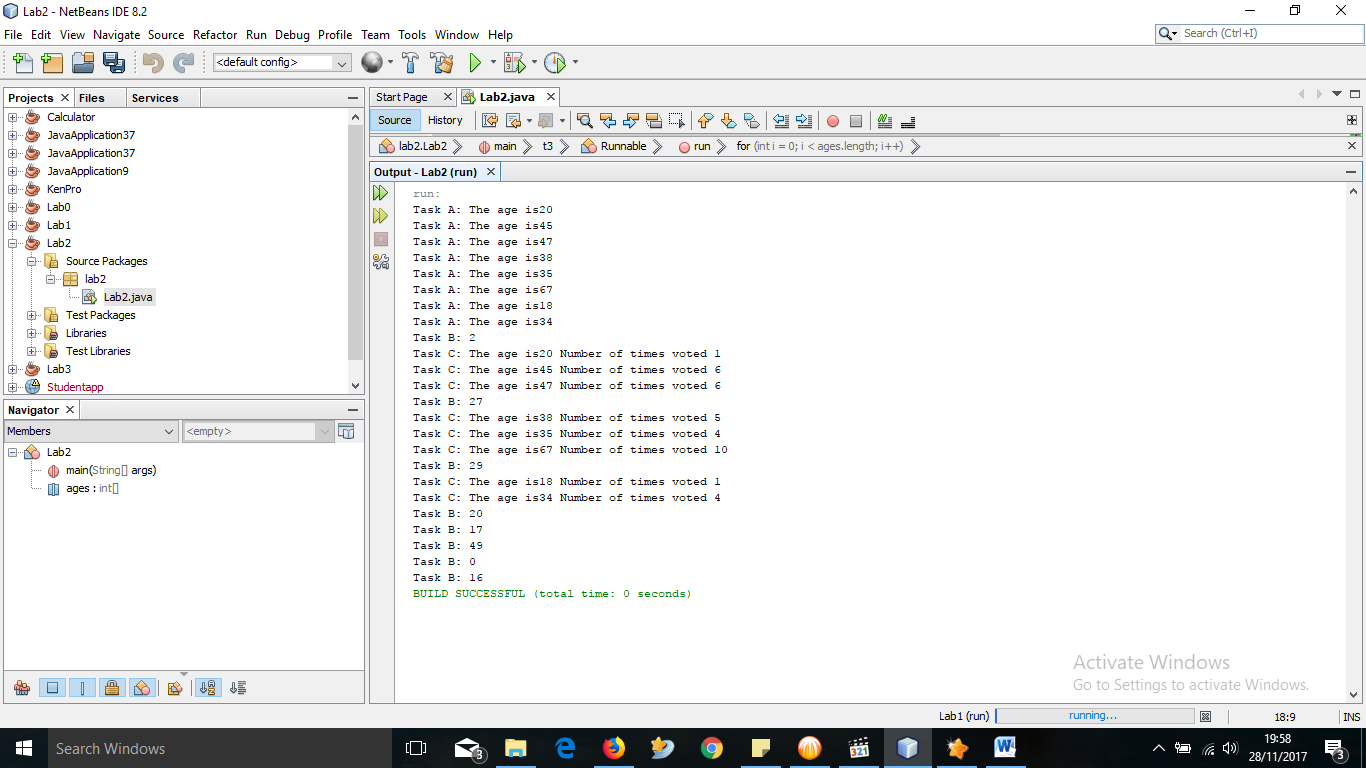
Three different threads must execute the three tasks above under the following scenarios (these scenarios must be implemented in three different java programs)

1. Start all the three threads (A, B & C) at the same time.

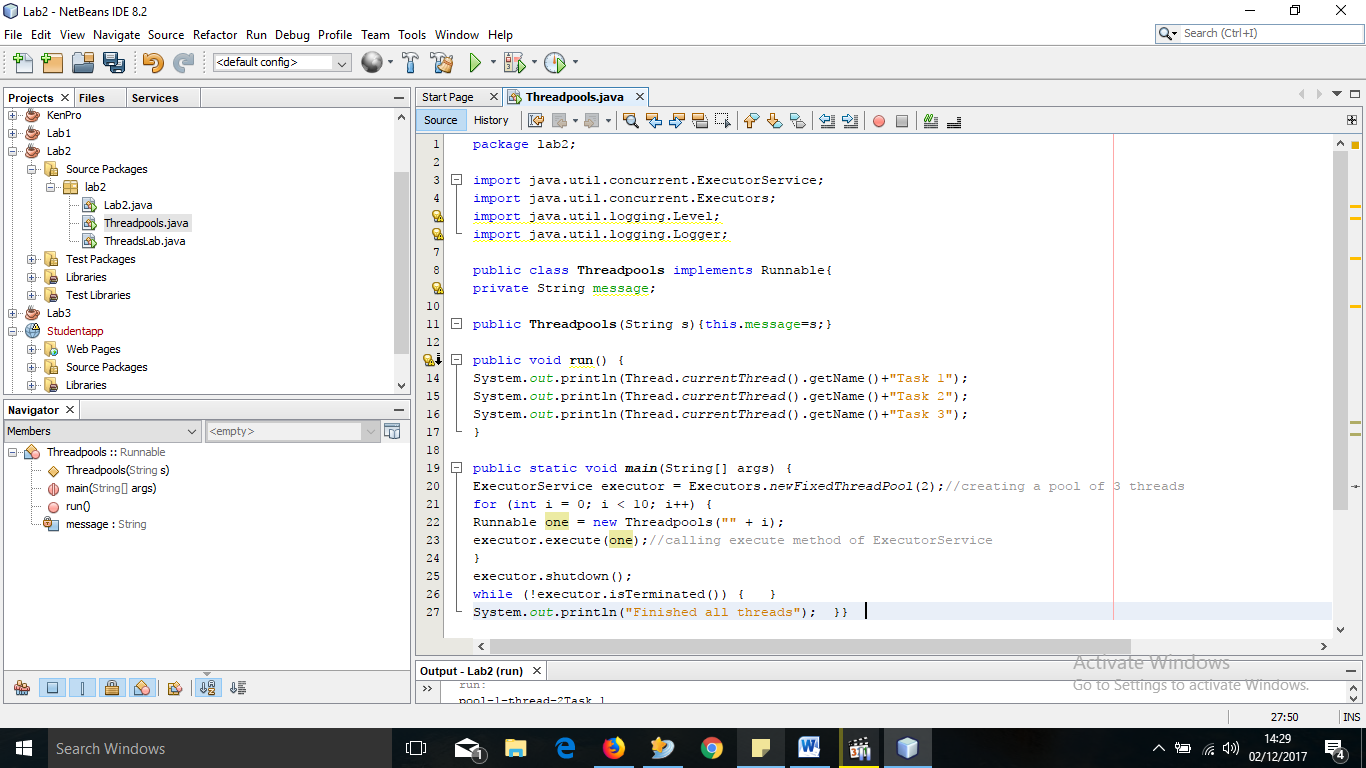


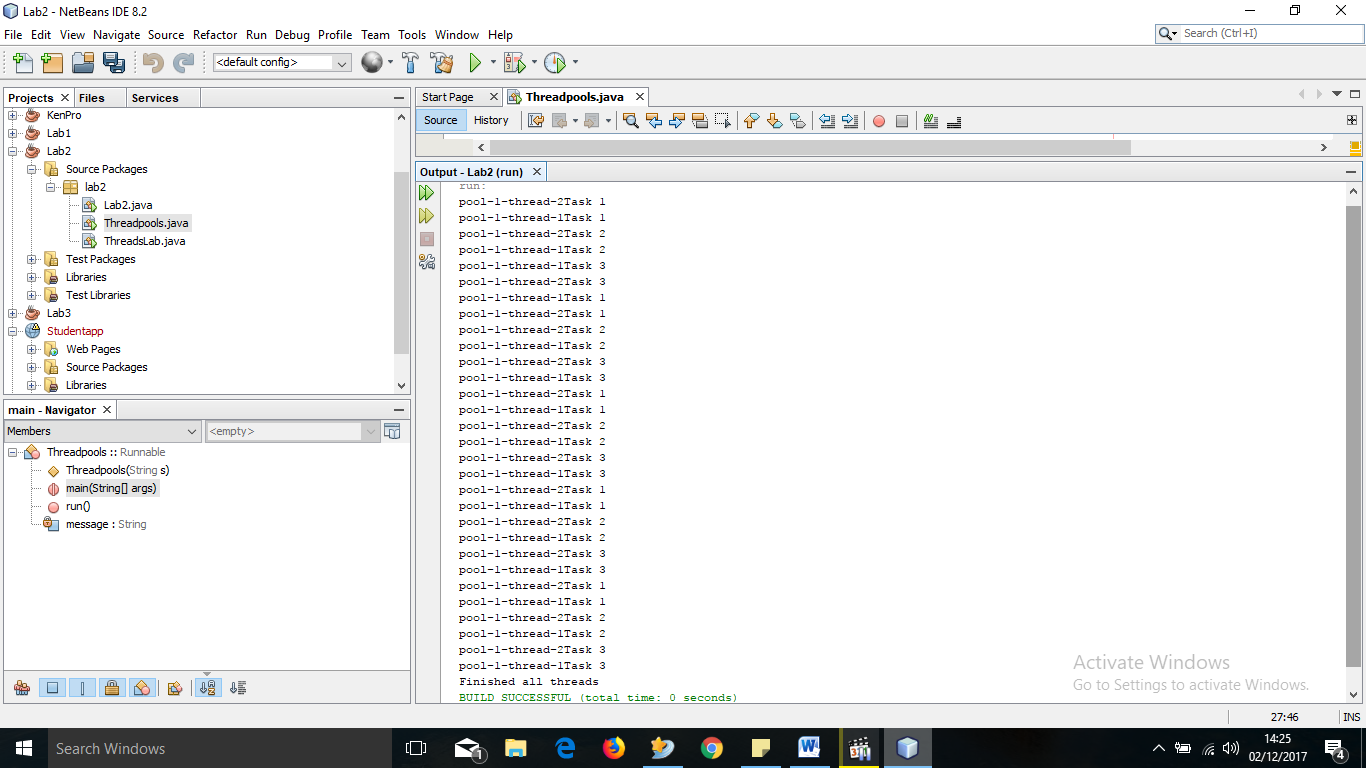
1. Allow thread for tasks A to finish before threads for task B & C execute





1. Create a thread pool of 2 threads and use them to execute the three tasks (Hint: Use the ExecutorService and Executors classes).





# Section 2

Modify you program in section 1 as follows

* Implement all the three functionality mentioned in one method/function
* Synchronize the method
* Allow your three threads to access the same method to complete their specific functions (you can pass some message via the tread class construct to tell the thread what to do see illustration below)

Class MyThread extends Thread {

private String task;

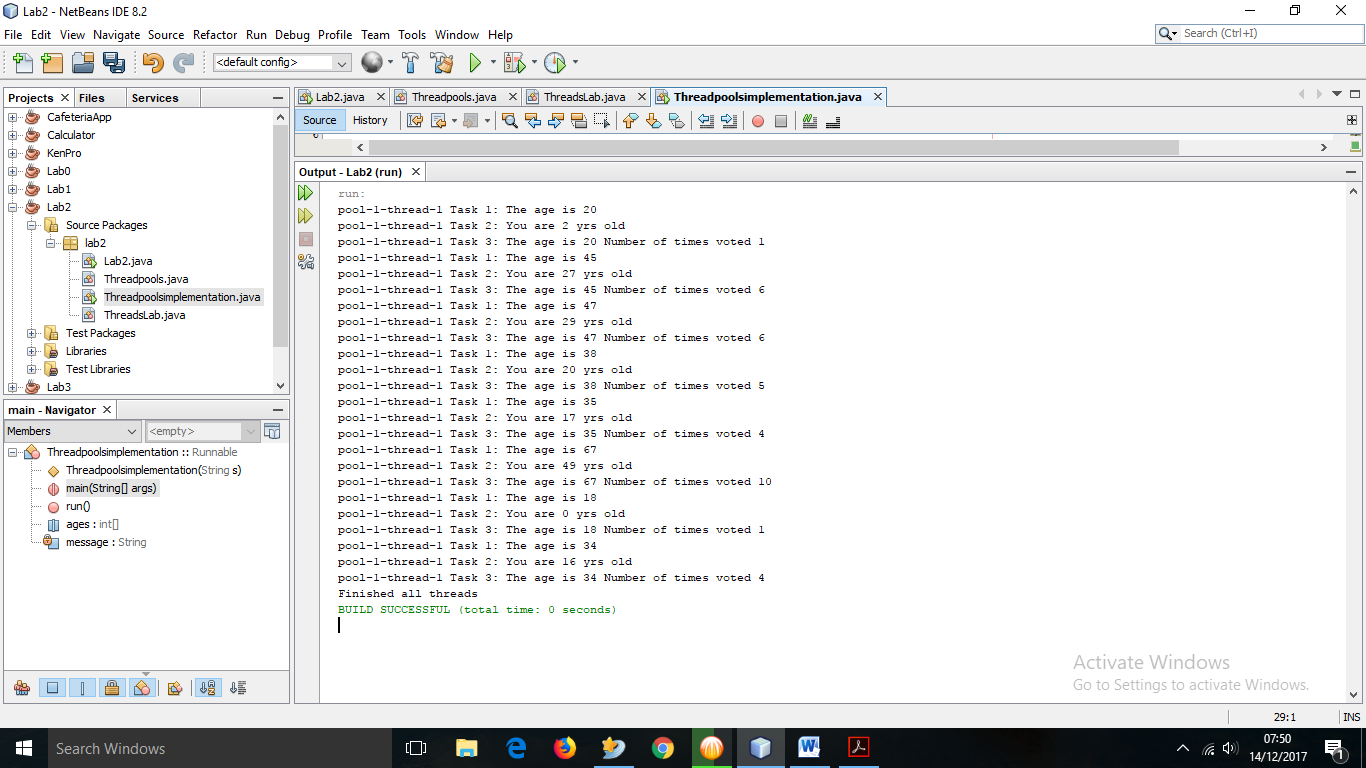
MyThread (String typeOfTask){This.task =typeOfTask;}

}

Thus, when you create instance of MyThread, you pass the name of task

e.g. MyThread t = new MyThread(“A”);

# 



# Section C

**Explain any THREE areas where of thread pools are applied**

Thread pools can be applied in Web servers, database servers, file servers and mail servers.

Servers usually have a large number of short tasks are sent through a network. A request arrives at the server, which might be through a network protocol such as HTTP, FTP, or POP. One way to solve this is by creating a new thread each time a request arrives and service the request in the new thread. Good for prototyping but it has disadvantages:

1. The thread-per-request approach is that the overhead of creating a new thread for each request is significant, a server that created a new thread for each request would spend more time and consume more system resources creating and destroying threads than it would processing actual user requests.
2. The active threads consume system resources. Creating too many threads in one JVM can cause the system to run out of memory or thrash due to excessive memory consumption. To prevent resource thrashing, server applications need some means of limiting how many requests are being processed at any given time.

A thread pool offers a solution to both the problem of thread life-cycle overhead and the problem of resource thrashing.

1. By reusing threads for multiple tasks, the thread-creation overhead is spread over many tasks.
2. As the thread already exists when a request arrives, the delay introduced by thread creation is eliminated.
3. Furthermore, by properly tuning the number of threads in the thread pool, you can prevent resource thrashing by forcing any requests in excess of a certain threshold to wait until a thread is available to process it.

***--This lab is complete--***