Lab 4: Introduction to Java Threads

Purpose:

This lab introduces programming with Java *threads*. You will examine and modify a program that has several threads. You will also compare the behavior of a threaded program on different operating systems. You will write a multi-threaded program to collect word-length frequencies.

Instructions:

1. Open Eclipse, create a project and add `SimpleThread.java` to the project.

   The file `SimpleThread.java` defines a class named `SimpleThread` with a `main` method that creates two `SimpleThread` threads and starts the threads.

   Run the program three times. Do you get the same output each time? Explain.

   Print a copy of an execution of the program. Hand in a copy of the output along with your explanations.

2. Answer the following questions about the Java code in the program you created and ran:
   a) What is the parent class of `SimpleThread`?
   b) What is the purpose of the statement

      `super(name);`

      (in the constructor)?
   c) `getName()` is a method of what class?
   d) Which method defines the behavior of a thread?
   e) How does one start a thread?

3. Modify the program to obtain the names of the threads to be generated from the command line instead of having the names "hard-wired" in the code. (One specifies the command line arguments in Eclipse by editing the run configuration.) Allow the number of threads created to vary with each execution. Thus the program will need to create a separate thread for each command line argument.

   Print a copy of an execution that results when three threads are created. Hand in a copy of the source file with proper documentation along with a copy of the execution.

4. One of Java's advertised features is that it is "cross-platform". This means you should be able to take bytecode that was compiled on one platform and run it on another platform and the program should have the same behavior. Test that this works by copying the class file that was created in step 3 to a Linux box and running it there. (You can use *psftp* to copy the file to *kant*.)

You can run a Java program in Linux with the command `java <classname>` assuming you have the .class file.

Compare the execution of the program in the Linux environment with its behavior in Windows. Do you see any significant differences?

5. I have provided a program (`WordLengthHistogram.java`) that reads a text file and generates a histogram of the word-lengths of the words in the file. Compile and run this program to become familiar with its behavior. It will prompt you for the name of a text file.

   Modify this program to use threads. Define a thread class that reads words from a file and updates frequencies stored in a shared array of integers. A shell for this program is provided below:

```
/* ThreadedHistogram.java
*/
import java.io.*;
public class ThreadedHistogram
{
   public static void main(String[] args)
   {
      int[] counts = new int[21];

      // Insert code here to:
      //    - initialize array
      //    - obtain file names from user
      //    - create two FileReaderThread's, each to read from
      //       a different file
      //    - run the threads
      //    - wait for both threads to terminate
      //       (hint use join method)
      //    - display the histogram
   }
}

class FileReaderThread extends Thread
{
   private String filename;
   private int[] freq;
```

```
    public FileReaderThread(String fname, int[] f)
    {
          filename = fname;
          freq = f;
    }



    public void run()
    {
       System.out.println("Thread to read "
                             + filename + " - started");
      // Insert code here to:
       //     - open file
       //     - read words from file and update array freq
       //     - display message at end of execution with name of
       //        file and number of words read
    }
}
```

Add the required code to the *run()*and the *main()* methods. Note that there is only one array of frequencies that is created in the *main()* method but a reference to the array is passed to each of the threads so they can share it.

Complete the program as described, add documentation to the file (Javadoc the classes and methods), and then run the program on two different text files.

Print an execution of the program and hand it in with your lab.

You will find some sample .txt files on the Moodle page.

6. Modify the ThreadedHistogram program to obtain the filenames from the command line and accommodate any number of file names. Once again a separate thread should be created to handle each filename listed on the command line.

   Make sure the Javadoc is accurate. Print a copy of the modified source code and hand it in with your lab.

Hand In:

Hand in a write-up of the observations you have made (from Steps 1 and 3) along with the execution results. Hand in your answers to the questions in Steps 2 and 4. Hand in a print out of the execution results from Steps 3 and 5. Turn in the programs

`SimpleThread.java` from Step 3 and the `ThreadedHistogram.java` from Step 6. E-mail the two source files to your instructor as well.