Outlab 8 : Java

Please refer to the general instructions and submission guidelines at the end of this document before submitting.

P1. Multi-threading [20 points]

This task is to implement an algorithm that sums up a list of integers in parallel. This is especially beneficial when you have a large number of integers.

For instance;
If the list of numbers is
A = [1 2 3 4 2 3 4 5 3 4 5 6 4 5 6 7]

For a batch size of 4
In round 1
    4 threads are launched to sum numbers of each batch of size 4
    A[0:4], A[4:8], A[8:12], A[12:16] respectively
At the end of round 1:
    We shall have four numbers which are the sums of the sub-arrays.
    10, 14, 18, 22

In round 2
    We launch a new thread to sum up the above partial summations.
At the end of round 2:
    We shall have the result which is the sum of the entire list.

If each thread in a round sums up a batch of p (>1) integers and there are p^n numbers, it takes n rounds to obtain the final result.

Check the diagram in slide 7 of this presentation. This is for p=2 and a max operation.

Your task is to implement this functionality using **Threads** in Java.

Task 1:

Your program is to take as input; n, p and p^n integers from STDIN and output the sum onto STDOUT.

Input:

Write your code in the file **PSum.java** in a class **PSum**.
It will be compiled and executed as follows:
$ javac **PSum.java**
$ java **PSum** < input

Task 2:

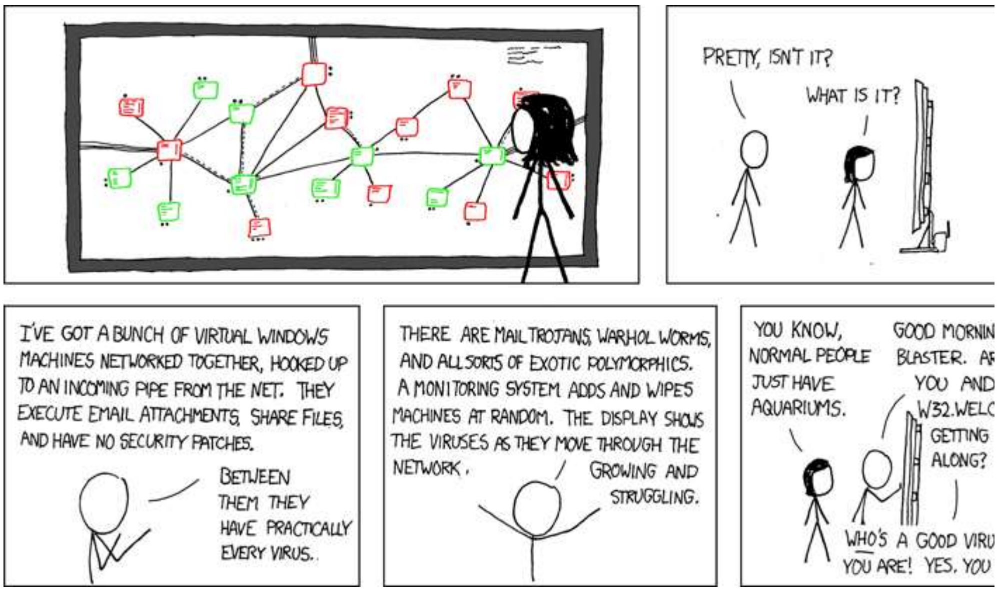
Write a program which implements a sequential algorithm that sums up an array of integers (you needn't submit this program).

Report the (real) timings each of the algorithms takes for a particular instance of the problem in the format specified below in a file, **times.txt** using the *time* command.

times.txt:
parallel: <parallel execution time>
sequential: <sequential execution time>

The instance of the problem for which the times are to be reported are:
n = 10, p = 2
$a_i = i$ for i in $[1\ldots 2^{10}]$

P2. Sockets [15 points]


PRETTY, ISN'T IT?
WHAT IS IT?
I'VE GOT A BUNCH OF VIRTUAL WINDOWS MACHINES NETWORKED TOGETHER, HOOKED UP TO AN INCOMING PIPE FROM THE NET. THEY EXECUTE EMAIL ATTACHMENTS, SHARE FILES, AND HAVE NO SECURITY PATCHES.
BETWEEN THEM THEY HAVE PRACTICALLY EVERY VIRUS.
THERE ARE MAIL TROJANS, WARHOL WORMS, AND ALL SORTS OF EXOTIC POLYMORPHICS. A MONITORING SYSTEM ADDS AND WIPES MACHINES AT RANDOM. THE DISPLAY SHOWS THE VIRUSES AS THEY MOVE THROUGH THE NETWORK,
GROWING AND STRUGGLING.
YOU KNOW, NORMAL PEOPLE JUST HAVE AQUARIUMS.
GOOD MORNIN BLASTER. AR YOU AND W32.WELC GETTING ALONG?
WHO'S A GOOD VIRU YOU ARE! YES, YOU

In this task you will create a simple local server and a simple local client (both written in Java). To do this we use the concept of **Sockets.** A socket is a general networking term etymologically meaning end of a wire. Here, a socket means end of a network.

Remember the server client diagram we saw in outlab6 (Webstack & Git). It consists of a client at one end and a server at another. The server and client are basically sockets. Instead of client written in HTML+JS+CSS and server written in PHP, we are going to write them in Java.
Read about Java sockets online, you should get good references quite easily.

1. It takes in a single argument from command line
2. It waits for a client to connect
3. After a client connects with it,
   a. It reads the data sent by the client (if any)
   b. Then prints the data to the terminal
   c. Then writes back the same data appended with the command line argument to the client using the socket.
      i. For ex, if the argument sent through command line is "**--from-your-loving server**" (without quotes) and the client sends "**Hello-how-are-you?**" (without quotes), the server should send back the the client "**Hello-how-are-you?--from-your-loving-server**" (without quotes)
4. Then it closes the socket to the client and again waits for a new client to connect (in a while loop)

Create a class called **DumbClient** in **DumbClient.java** which does the following

1. Takes in one argument from command line
2. Prints the argument to terminal
3. Sends that argument to the server and gets back the response
4. Prints the response to terminal

You should first start the server and then execute the client. Can you reason why?
If everything is working properly the following should be a sample output.

$ **java EchoServer --from-your-loving-server**	**$java DumbClient Hello-how-are-you?**
Hello-how-are-you?	**Hello-how-are-you?**
	Hello-how-are-you?--from-your-loving-server
	<Maybe some exception>
	$

Notice that server should still be running while client program has ended.

P3. Java Collection Framework [15 points]

The task is to write a program named **FrequencyCollection.java** which reads a plain text file as a string, tokenize it, remove the following **stop words** = {and, the is, in, at, of, his, her, him}, store the remaining words in an ArrayList and process it to display the frequency of each word.

Note :

1. The path of the text file will be given as argument to FrequencyCollection.java
2. The words in the file must be treated to be case insensitive and the output should contain all lowercase words i.e., for ex, “hello” and “Hello” must be treated the same
3. Only words with non-zero frequency should be present in the output and the output should be sorted in lexicographical order of the words.

Output Format :

```
$ javac FrequencyCollection.java
$ java      FrequencyCollection  <inputfile>
<word-1>,<frequency-1>
<word-2>,<frequency-2>
………………...
<word-n>,<frequency-n>
```

General Instructions

- Make sure you know what you write, you might be asked to explain your code at a later point in time
- Grading may be done automatically, so please make sure you stick to naming conventions
- The deadline for this lab is **Monday, 24rd September, 23:55.**

Submission Instructions

After creating your directory, package it into a tarball **`<rollno1>-<rollno2>-<rollno3>-outlab8.tar.gz`** in ascending order. Submit once only per team from the moodle account of smallest roll number.
The directory structure should be as follows (nothing more nothing less)

```
<rollno1>-<rollno2>-<rollno3>-
outlab8
├── P1
│   ├── PSum.java
│   └── times.txt
├── P2
│   ├── EchoServer.java
```

outlab8

Updated automatically every 5 minutes