Full Stack Development/Software Workshop 2

Formative Exercise 7

Based on Week 05 lectures

Topic: Java Threading

Exercise 1.

(Part a) Write a Java program that runs three threads in parallel:

1. Thread 1 prints the number '0' 10 times
2. Thread 2 prints the number '1' 10 times
3. Thread 3 prints the number '2' 10 times

A sample output of the program is (order is not important)

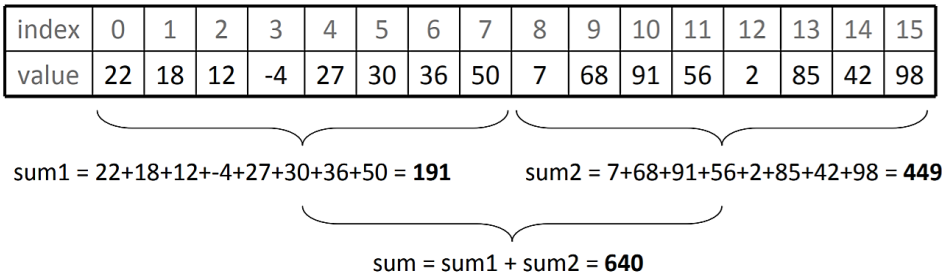
```
22222222221113311111133333333
```

(Part b) Re write the above Java program such that it uses ExecutorService (java.util.concurrent.ExecutorService).

Exercise 2.

[Note: To do this exercise, you have to download Exercise2.java file from Canvas in Week 5 materials]

The purpose of the int sum(int[] a) method inside java class "Exercise2" is to calculate the total sum of all elements of the given array. We want to write a multithreaded version of the sum method that should speed up the calculation by sharing the load between two threads. Both threads can separately compute sum of each half of array and combine the result. Here the threads must wait for each other to complete before the function can combine the results from each thread and return the sum.



The method signature for the multithreaded version of the sum method is

public static int sum_parallel(int[] a)

Write the missing code for int sum_parallel(int[] a) method.

```
public class Exercise2 extends Thread {
	// Computes the total sum of all elements of the given array.
	public static int sum(int[] a) {
		//Code not shown here
	}
	//Computes the total sum of elements of the given array in given
	  Range (min,max)
	public static int sumRange(int[] a, int min, int max) {
		//Code not shown here
	}
	// thread run method
	public void run() {
		//Incomplete. Add Code here
	}
	// Parallel version (two threads)
	public static int sum_parallel(int[] a) {
		//Incomplete. Add Code here
	}
	// Create an array of given length containing random values
	public static int[] createRandomArray(int length) {
		//Code not shown here
	}
	public static void main(String[] args) {
		//Code not shown here
	}
```

Exercise 3. Thread Synchronization

[Note: To do this exercise, you have to download two files Account.java and Exercise3.java file from Canvas in Week 5 materials]

We have a class called Account that represents a bank account. To keep the code short, this account starts with a balance of 50 and can be used only for withdrawals. The account simply reduces the balance by the amount you want to withdraw.

Imagine two people, X and Y, who both have access to the account and want to make withdrawals. But they don't want the account to ever be overdrawn (balance<0). However, when we execute the program Exercise3.java, we can observe that the final balance is negative.

```
Y is going to withdraw
X is going to withdraw
Y completes the withdrawal
Y is going to withdraw
X completes the withdrawal
X is going to withdraw
Y completes the withdrawal
Y is going to withdraw
X completes the withdrawal
X is going to withdraw
Y completes the withdrawal
Not enough in account for Y to withdraw 0
Not enough in account for Y to withdraw 0
X completes the withdrawal
account is overdrawn!
Not enough in account for X to withdraw -10
account is overdrawn!
Not enough in account for X to withdraw -10
account is overdrawn!
```

What is the problem here? Fix the code.