Runnable Thread Vowel Counter and Shared Location

Your cousin Sabari participated in Hackathon(24X2) session organized in his college. He came home very late around 3:00AM after completing the first session. Their project is to compute number of vowels from random web pages extracted from the website. They completed most of the code, But the application takes more time to compute 1000 pages. So he is very upset on and have no clue on improving his code to reduce the computation time.

He called you in the morning to discussion about this problem, You suggest him to use multiple thread to compute number of vowels in parallel. Number of vowels is maintained in a synchronized HashMap since its been accessed by multiple threads.
Caps and small letters are combined together in the map. But he request you to send him a sample code for his references. Can you help him with a reference code as per the below sample input and output to win this hackathan session.

Note:
Use Runnable interface in this example. **Dont use static variables for 5 vowel charecters. Use Map.**

Sample input output:
Enter Number of Counters :
2
Enter text for counter 1 :
Everbody gota learn some time
Enter text for counter 2 :
One,Two,Three,Four,Five,Six,Seven,Eight,Nine,Ten
Vowels count in given text are :
a:2 e:14 i:5 o:6 u:1

Answer:

```
import java.util.*;

class Main {

   public static void main(String[] args) {

      Scanner s = new Scanner(System.in);
      int n,i;
      String str;

      Map<Character, Integer> map = new TreeMap<Character, Integer>();
      map.put('a',0);
      map.put('e',0);
      map.put('i',0);
      map.put('o',0);
      map.put('u',0);

      System.out.println("Enter Number of Counters :");
      n = s.nextInt();
      s.nextLine();
```

```
        Thread t[] = new Thread[n];

        for(i=0;i<n;i++) {

            System.out.println("Enter text for counter "+(i+1)+" :");
            str = s.nextLine().toLowerCase();
            t[i] = new Thread(new Counter(map,str));
        }

        for(i=0;i<n;i++)
            t[i].start();

        System.out.println("Vowels count in given text are :");

        for(Map.Entry<Character,Integer> e:map.entrySet())
            System.out.print(e.getKey()+":"+e.getValue()+" ");

    }
}

import java.util.*;

public class Counter implements Runnable {

    private Map<Character, Integer> map;
    String s;

    public Counter(Map<Character, Integer> map,String s) {

        this.map = map;
        this.s = s;
    }

    public void run() {

        synchronized(map) {
            for(Character c:s.toCharArray()) {
                if(c=='a'||c=='e'||c=='i'||c=='o'||c=='u')
                    map.put(c,map.get(c)+1);
            }
        }
    }
}
```

Character Frequency - Multiple Threads

Your English literature friend is very happy with the code you gave him. Now for his research he used your application to find character frequency in many novels. For larger novels the application takes lot of time for computation. So he called you on a fine sunday to discuss about this with you. He wanted to know whether you can improve the speed of the application.

You decided to modify the application by using multiple threads to reduce the computation time. For this,

accept number of counters or threads at the beginning of the problem and get the string for each counter or thread. Create a thread by extending the Thread class and take the user entered string as input. Each thread calculates the character frequency for the word assigned to that thread. All the counts are stored locally in the thread and once all the threads are completed print the character frequency for each of the thread.

Sample input and output

Enter Number of Counters :
2
Enter text for counter 1 :
FrequencyCounter
Enter text for counter 2 :
JavaTheCompleteReference
Counter 1 Result :
C:1 F:1 c:1 e:3 n:2 o:1 q:1 r:2 t:1 u:2 y:1
Counter 2 Result :
C:1 J:1 R:1 T:1 a:2 c:1 e:7 f:1 h:1 l:1 m:1 n:1 o:1 p:1 r:1 t:1 v:1

Answer:

```
import java.util.*;

class Main {

    public static void main(String[] args) throws Exception {

        Scanner s = new Scanner(System.in);
        int n,i;
        String str;

        Map<Character, Integer> map;

        System.out.println("Enter Number of Counters :");
        n = s.nextInt();
        s.nextLine();

        Counter t[] = new Counter[n];

        for(i=0;i<n;i++) {

            System.out.println("Enter text for counter "+(i+1)+" :");
            str = s.nextLine();
            t[i] = new Counter(new TreeMap<Character, Integer>(),str);
        }

        for(i=0;i<n;i++)
            t[i].start();

        for(i=0;i<n;i++)
            t[i].join();
```

```
        for(i=0;i<n;i++) {
            System.out.println("Counter "+(i+1)+" Result :");
            for(Map.Entry<Character,Integer> e:t[i].getMap().entrySet())
                System.out.print(e.getKey()+":"+e.getValue()+" ");
            System.out.println();
        }
    }
}


import java.util.*;

public class Counter extends Thread {

    private Map<Character, Integer> map;
    private String s;

    public Counter(Map<Character, Integer> map,String s) {

        this.map = map;
        this.s = s;
    }

    public void run() {

        for(Character c:s.toCharArray())
        {
            if(map.containsKey(c))
                map.put(c,map.get(c)+1);
            else
                map.put(c,1);
        }

    }

    public Map<Character, Integer> getMap() {

        return this.map;
    }
}
```

Multi Threading

To illustrate creation of multiple threads in a program performing concurrent operations, let us consider the processing of the following mathematical equation:

p = sin (x) + cos (y) + tan (z)

As these trigonometric functions are independent operations without any dependencies between them, they can be executed concurrently. After that their results can be combined to produce the final result.

All three worker threads are concurrently executed on shared or dedicated CPUs depending on the type of machine. Although the master thread can continue its execution, in this case, it needs to make sure that all operations are completed before combining individual results. This is accomplished by waiting for each thread

to complete by invoking join() method associated with each worker thread.
Refer sample Input and Output.

Input & Output Format:
The main thread is called Main, which acts like a master thread. It creates three worker threads (SineClass, CosClass, and TanClass) and assigns them to compute values for different data inputs.

Keep the name as MathInput Class for getters and setters to get input and pass it to the threads.

Hint:

Use the following import for printing to 2 decimal places.
import java.text.DecimalFormat;
DecimalFormat df = new DecimalFormat("#.##");
System.out.println("Sum of sin, cos, tan = " + df.format(z));

Sample Input and Output :
Enter the Degree for Sin :
45
Enter the Degree for Cos :
30
Enter the Degree for Tan :
30
Sum of sin, cos, tan = 2.15

Answer:

```
import java.util.*;
import java.text.DecimalFormat;
import java.lang.*;

class Main {

    public static void main(String[] args) throws InterruptedException {

        Scanner s = new Scanner(System.in);

        System.out.println("Enter the Degree for Sin : ");
        MathInput sin1 = new MathInput(s.nextDouble());
        System.out.println("Enter the Degree for Cos : ");
        MathInput cos1 = new MathInput(s.nextDouble());
        System.out.println("Enter the Degree for Tan : ");
        MathInput tan1 = new MathInput(s.nextDouble());

        SinClass t1 = new SinClass(sin1);
        CosClass t2 = new CosClass(cos1);
        TanClass t3 = new TanClass(tan1);

        t1.start();

        try {
            t1.join(2000);
```

```
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        t2.start();

        try {
            t1.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        t3.start();

        try {
            t1.join();
            t2.join();
            t3.join();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        double z = sin1.getDegree()+cos1.getDegree()+tan1.getDegree();
        DecimalFormat df = new DecimalFormat("#.##");
        System.out.println("Sum of sin, cos, tan = " + df.format(z));
    }
}

public class SinClass extends Thread {

        MathInput mI;
        public SinClass(MathInput mI) {
                this.mI = mI;
        }
        public void run() {

                mI.setDegree(Math.sin(Math.toRadians(mI.getDegree())));
        }
}

public class TanClass extends Thread {

    MathInput mI;
        public TanClass(MathInput mI) {
                this.mI = mI;
        }
        public void run() {

                mI.setDegree(Math.tan(Math.toRadians(mI.getDegree())));
        }
}public class MathInput {
```

```
	private double degree;

	public double getDegree() {
		return degree;
	}

	public void setDegree(double degree) {
		this.degree = degree;
	}

	public MathInput(double degree) {
		super();
		this.degree = degree;
	}
}

public class CosClass extends Thread {

   MathInput mI;
	public CosClass(MathInput mI) {
		this.mI = mI;
	}
	public void run() {

		mI.setDegree(Math.cos(Math.toRadians(mI.getDegree())));
	}
}
```