Learning Java 9 : Databases and Multithreading in Java

User Assignment-5

Problem Statement:

1. Write a java program to create ten threads and execute them parallelly, where each thread just displays the Hello-Thread-n (where n is the number of thread)

2. Write a java program to read a very large text file which is of the order of 100MB data and count the number of times the word **program** is occurring in that bigdata file.

   To implement this, create four threads by using ExecutorService, where each thread read part of the text file and count the number of times the word is occurring parallelly and integrate all the results into a single value and display it , also calculate the time taken to perform the task.

```java
package com.question5.first;

public class HelloThread {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        A a = new A();
        a.start();
        B b = new B();
        b.start();
        C c = new C();
        c.start();
        D d = new D();
        d.start();
        E e = new E();
        e.start();
        F f = new F();
        f.start();
        G g = new G();
        g.start();
        H h = new H();
        h.start();
        I i = new I();
        i.start();
        J j = new J();
        j.start();

    }

}
class A extends Thread
{
    public void run()
    {
    System.out.println("Hello-Thread: "+Thread.currentThread().getName());

    }

}
class B extends Thread
{
    public void run()
    {
        System.out.println("Hello-Thread: "+Thread.currentThread().getName());

    }
}
class C extends Thread
{
    public void run()
    {
        System.out.println("Hello-Thread: "+Thread.currentThread().getName());
    }

}
class D extends Thread
{
    public void run()
    {
        System.out.println("Hello-Thread: "+Thread.currentThread().getName());
    }
}
class E extends Thread
{
    public void run()
    {
        System.out.println("Hello-Thread: "+Thread.currentThread().getName());
    }
}
class F extends Thread
{
    public void run()
    {
        System.out.println("Hello-Thread: "+Thread.currentThread().getName());
    }
}
class G extends Thread
{
    public void run()
    {
        System.out.println("Hello-Thread: "+Thread.currentThread().getName());
    }
}
class H extends Thread
{
    public void run()
    {
        System.out.println("Hello-Thread: "+Thread.currentThread().getName());
    }
}
class I extends Thread
{
    public void run()
    {
        System.out.println("Hello-Thread: "+Thread.currentThread().getName());
    }
}
class J extends Thread
{
    public void run()
    {
        System.out.println("Hello-Thread: "+Thread.currentThread().getName());
    }
}
```

```java
import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.net.URL;
import java.util.*;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.ExecutorCompletionService;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;

public class Main {
    public static void main(String args[]) throws IOException,
InterruptedException, ExecutionException {
        long startTime = System.currentTimeMillis();
        int threads = 4;
        ExecutorService executorService =
Executors.newFixedThreadPool(threads);
        final ExecutorCompletionService<Map<String, Long>> completionService =
new ExecutorCompletionService<>(executorService);

        List<List<String>> listOfLists = getSplitLists(threads);

        Map<String, Long> allCounts = executeWork(completionService,
listOfLists);

//        LinkedHashMap<String, Long> sortedMap =
allCounts.entrySet().stream()
//            .sorted(Map.Entry.comparingByValue(Comparator.reverseOrder()))
//            .collect(Collectors.toMap(Map.Entry::getKey,
Map.Entry::getValue,
//                (oldValue, newValue) -> oldValue, LinkedHashMap::new));

        long stopTime = System.currentTimeMillis();
        long elapsedTime = stopTime - startTime;
        System.out.println("Total execution time is " + elapsedTime + " ms");

        executorService.shutdown();
    }

    private static Map<String, Long>
executeWork(ExecutorCompletionService<Map<String, Long>> completionService,
List<List<String>> listOfLists) throws InterruptedException,
ExecutionException {
        listOfLists.forEach(sublist -> completionService.submit(new
WordCounter(sublist)));

        Map<String, Long> allCounts = new HashMap<>();
```

```java
package Product4.WordCount;

import java.util.Arrays;
import java.util.List;
import java.util.Map;
import java.util.concurrent.Callable;
import java.util.function.Function;
import java.util.stream.Collectors;

public class WordCounter implements Callable<Map<String, Long>> {
    private final List<String> lineList;

    WordCounter(List<String> lineList) {
        this.lineList = lineList;
    }

    @Override
    public Map<String, Long> call() {
        long startTime = System.currentTimeMillis();
        String threadName = Thread.currentThread().getName();

        Map<String, Long> results = lineList.stream()
            .filter(line -> !line.equals(""))
            .flatMap(line -> Arrays.stream(line.split(" ")))
            .map(word -> word.replaceAll("[^\\w]", ""))
            .filter(word -> !word.equals(""))
            .filter(word -> word.length() > 1)
            .collect(Collectors.groupingBy(Function.identity(),
Collectors.counting()));

        long stopTime = System.currentTimeMillis();
        long elapsedTime = stopTime - startTime;
        System.out.println(threadName + "Finished work in " + elapsedTime + "
ms");

        return results;
    }
}
```