Assignment 5:

User Assignment-4

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.

Source Code:

```
public class Main{
    public static void main(String[] args){
        for(int i=0;i<10;i++){
            Thread t = new Thread(new HelloThread(i));
            t.start();
        }
    }

    static class HelloThread implements Runnable{
        int id;
        HelloThread(int id){
            this.id = id;
        }
        public void run(){
            System.out.println("Hello-Thread-"+id);
        }
    }

}
```

Output:

```
Hello-Thread-5
Hello-Thread-1
Hello-Thread-7
Hello-Thread-4
Hello-Thread-6
Hello-Thread-0
Hello-Thread-9
Hello-Thread-8
Hello-Thread-3
Hello-Thread-2
```

2)

```
import java.io.*;
import java.net.URL;
import java.util.*;
import java.util.concurrent.*;
import java.util.stream.Collectors;

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);

        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<>();
        for (int i = 0; i < listoflists.size(); i++) {
            Map<String, Long> newCounts = completionService.take().get();
            newCounts.forEach((k, v) -> allCounts.merge(k, v, Long::sum));
        }
        return allCounts;
    }

    private static List<List<String>> getSplitlists(int threads) throws
FileNotFoundException {
        URL file_path =
Product4.Hibernate.Main.class.getClassLoader().getResource("words.txt");
        String content = new Scanner(new
File(file_path.getPath())).useDelimiter("\\Z").next();
        List<String> lines = Arrays.asList(content.split("\n"));
        return splitlist(lines, lines.size() / threads);
    }

    private static List<List<String>> splitlist(List<String> originallist, int
partitionSize) {
        List<List<String>> partitions = new LinkedList<>();
        for (int i = 0; i < originallist.size(); i += partitionSize) {
```

```
            partitions.add(originallist.subList(i,
                    Math.min(i + partitionSize, originallist.size())));
        }
        return partitions;
    }
}
```

WordCounter.java

```
import java.util.Arrays;
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 +" ns");
        return results;
    }
}
```