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.

1.)

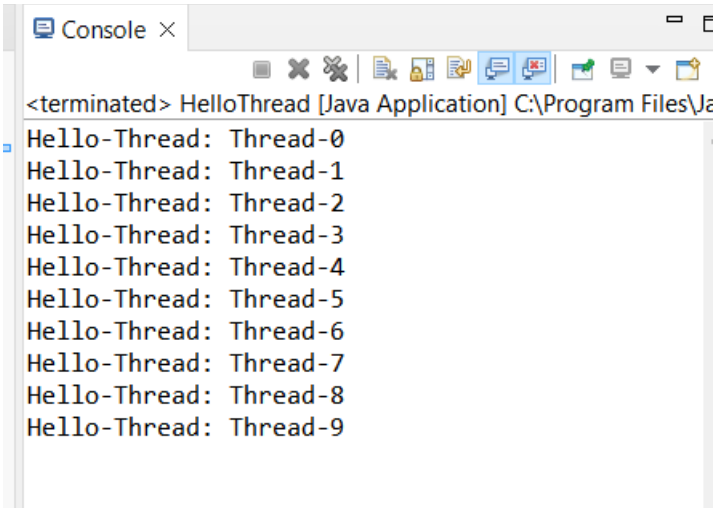
HelloThread.java

```
1  package com.question5.first;
2
3  public class HelloThread {
4
5      public static void main(String[] args) {
6          // TODO Auto-generated method stub
7          A a = new A();
8          a.start();
9          B b = new B();
10         b.start();
11         C c = new C();
12         c.start();
13         D d = new D();
14         d.start();
15         E e = new E();
16         e.start();
17         F f = new F();
18         f.start();
19         G g = new G();
20         g.start();
21         H h = new H();
22         h.start();
23         I i = new I();
24         i.start();
25         J j = new J();
26         j.start();
27
28     }
29
30 }
```

HelloThread.java

```
31 class A extends Thread
32 {
33     public void run()
34     {
35     System.out.println("Hello-Thread: "+Thread.currentThread().getName());
36 
37     }
38 
39 }
40 class B extends Thread
41 {
42     public void run()
43     {
44         System.out.println("Hello-Thread: "+Thread.currentThread().getName());
45 
46     }
47 }
48 class C extends Thread
49 {
50     public void run()
51     {
52         System.out.println("Hello-Thread: "+Thread.currentThread().getName());
53     }
54 
55 }
56 class D extends Thread
57 {
58     public void run()
59     {
60         System.out.println("Hello-Thread: "+Thread.currentThread().getName());
61     }
62 }
63 class E extends Thread
64 {
65     public void run()
66     {
67         System.out.println("Hello-Thread: "+Thread.currentThread().getName());
68     }
69 }
```

```
70 class F extends Thread
71 {
72     public void run()
73     {
74         System.out.println("Hello-Thread: "+Thread.currentThread().getName());
75     }
76 }
77 class G extends Thread
78 {
79     public void run()
80     {
81         System.out.println("Hello-Thread: "+Thread.currentThread().getName());
82     }
83 }
84 class H extends Thread
85 {
86     public void run()
87     {
88         System.out.println("Hello-Thread: "+Thread.currentThread().getName());
89     }
90 }
91 class I extends Thread
92 {
93     public void run()
94     {
95         System.out.println("Hello-Thread: "+Thread.currentThread().getName());
96     }
97 }
98 class J extends Thread
99 {
100    public void run()
101    {
102        System.out.println("Hello-Thread: "+Thread.currentThread().getName());
103    }
104 }
```



2.)

Main.java × WordCounter.java words.txt

```
import java.io.File;

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<>();
        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;
    }
}

```

Main.java WordCounter.java × words.txt

```
package Product4.WordCount;

import java.util.Arrays;

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;
    }
}

```