1. Homework 11

Posted: November/4/2018
Due: November/11/2018 24.00

All homework solutions are due November/11/2018 24.00. I recommend to submit at least one version of all homework solutions long before due date.

1.1. Homework 11.1 (15 Points)

Objective: Working with Sempahores

Grading:
Correctness: You can lose up to 40% if your solution is not correct
Quality: You can lose up to 80% if your solution is poorly designed
Testing: You can lose up to 50% if your solution is not well tested
Explanation: You can lose up to 100% if your solution if you can not explain your solution during the grading session

Homework Description:

You have to modify homework 10.2 (Producer Consumer) from using monitors to using semaphores.

Explanation:

You synschonized 10.2 using monitores. Modify your code so such your are using semaphores.

Your Work:

It might be useful to think about the solution before you start implementing in order to minmize the modification.

Submission:

```
% ssh glados.cs.rit.edu # or use queeg.cs.rit.edu if glados is down
# password
# go to the directory where your solution is ...
% try hpb-grd lab11-1 'All files required'
```

1.2. Homework 11.2 (15 Points)

Objective: Understanding code using lambda expressions

Grading:
Correctness: You can lose up to 40% if your solution is not correct
Quality: You can lose up to 80% if your solution is poorly designed
Testing: You can lose up to 50% if your solution is not well tested
Explanation: You can lose up to 100% if your solution if you can not explain your solution during the grading session

Homework Description:

You have to be able to explain to your grader during your grading session the following code:

```
 1      /*
 2       * Copyright (c) 2014, Oracle and/or its affiliates. All rights reserved.
 3       *
 4       * Redistribution and use in source and binary forms, with or without
 5       * modification, are permitted provided that the following conditions
 6       * are met:
 7       *
 8       *   - Redistributions of source code must retain the above copyright
 9       *     notice, this list of conditions and the following disclaimer.
10       *
11       *   - Redistributions in binary form must reproduce the above copyright
12       *     notice, this list of conditions and the following disclaimer in the
13       *     documentation and/or other materials provided with the distribution.
14       *
15       *   - Neither the name of Oracle nor the names of its
16       *     contributors may be used to endorse or promote products derived
17       *     from this software without specific prior written permission.
18       *
19       * THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS
20       * IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO,
21       * THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
22       * PURPOSE ARE DISCLAIMED.  IN NO EVENT SHALL THE COPYRIGHT OWNER OR
23       * CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
24       * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
25       * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR
26       * PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF
27       * LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING
28       * NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
29       * SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
30       */
31
32      /*
33       * This source code is provided to illustrate the usage of a given feature
34       * or technique and has been deliberately simplified. Additional steps
35       * required for a production-quality application, such as security checks,
36       * input validation, and proper error handling, might not be present in
37       * this sample code.
38       */
39      /*
40       * THIS CODE WAS MODIFIED BY hpb
41       */
```

```
42      import java.io.BufferedReader;
43      import java.io.FileNotFoundException;
44      import java.io.FileReader;
45      import java.io.IOException;
46      import java.util.function.Consumer;
47      import java.util.regex.Pattern;
48
49      public class LE {
50
51          //The number of characters that may be read.
52          private static final int READ_AHEAD_LIMIT = 100_000_000;
53
54          //The pattern for splitting strings by non word characters to get words.
55          private static final Pattern nonWordPattern = Pattern.compile("\\W");
56
57          /**
58           * The main method for the LE program. Run the program with an empty
59           * argument list to see possible arguments.
60           *
61           * @param args the argument list for LE
62           * @throws java.io.IOException If an input exception occurred.
63           */
64          public static void main(String[] args) throws IOException {
65
66              if (args.length != 1) {
67                  usage();
68                  return;
69              }
70
71              try (BufferedReader reader = new BufferedReader(
72                      new FileReader(args[0]))) {
73                  reader.mark(READ_AHEAD_LIMIT);
74                  /*
75                   * Statistics can be gathered in four passes using a built-in API.
76                   * The method demonstrates how separate operations can be
77                   * implemented using a built-in API.
78                   */
79                  collectInFourPasses(reader);
80                  /*
81                   * Usage of several passes to collect data is not the best way.
82                   * Statistics can be gathered by a custom collector in one pass.
83                   */
84                  reader.reset();
85                  collectInOnePass(reader);
86              } catch (FileNotFoundException e) {
87                  usage();
88                  System.err.println(e);
89              }
90          }
91
92          private static void collectInFourPasses(BufferedReader reader)
93                  throws IOException {
94              /*
95               * Input is read as a stream of lines by lines().
```

```
96              * Every line is turned into a stream of chars by the flatMapToInt(...)
97              * method.
98              * Length of the stream is counted by count().
99              */
00             System.out.println("Character count = "
01                     + reader.lines().flatMapToInt(String::chars).count());
02             /*
03              * Input is read as a stream of lines by lines().
04              * Every line is split by nonWordPattern into words by flatMap(...)
05              * method.
06              * Empty lines are removed by the filter(...) method.
07              * Length of the stream is counted by count().
08              */
09             reader.reset();
10             System.out.println("Word count = "
11                     + reader.lines()
12                     .flatMap(nonWordPattern::splitAsStream)
13                     .filter(str -> !str.isEmpty()).count());
14
15             reader.reset();
16             System.out.println("Newline count = " + reader.lines().count());
17
18             reader.reset();
19             System.out.println("Max line length = "
20                     + reader.lines().mapToInt(String::length).max().getAsInt());
21         }
22
23         private static void collectInOnePass(BufferedReader reader) {
24
25             LEStatistics wc = reader.lines().parallel()
26                     .collect(LEStatistics::new,
27                             LEStatistics::accept,
28                             LEStatistics::combine);
29             System.out.println(wc);
30         }
31
32         private static void usage() {
33             System.out.println("Usage: " + LE.class.getSimpleName() + " FILE");
34             System.out.println("Print something");
35         }
36
37         private static class LEStatistics implements Consumer<String> {
38             /*
39              * @implNote This implementation does not need to be thread safe because
40              * the parallel implementation of
41              * {@link java.util.stream.Stream#collect Stream.collect()}
42              * provides the necessary partitioning and isolation for safe parallel
43              * execution.
44              */
45
46             private long count1;
47             private long count3;
48             private long count2;
49             private long count4;
```

```
50
51
52              @Override
53              public void accept(String line) {
54                  count1 += line.length();
55                  count3++;
56                  count2 += nonWordPattern.splitAsStream(line)
57                          .filter(str -> !str.isEmpty()).count();
58                  count4 = Math.max(count4, line.length());
59              }
60
61              public void combine(LEStatistics stat) {
62                  count2 += stat.count2;
63                  count3 += stat.count3;
64                  count1 += stat.count1;
65                  count4 = Math.max(count4, stat.count4);
66              }
67
68              @Override
69              public String toString() {
70                  StringBuilder sb = new StringBuilder();
71                  sb.append("#------LEStatistic------#\n");
72                  sb.append("count 1 = ").append(count1).append('\n');
73                  sb.append("count 2 = ").append(count2).append('\n');
74                  sb.append("count 3 = ").append(count3).append('\n');
75                  sb.append("count 4 = ").append(count4).append('\n');
76                  return sb.toString();
77              }
78          }
79      }
```

Source Code: Src/31/LE.java

Explanation:

Your grader may ask you during the grading session about modifications of the code in order to achieve slighly different requirements.

Submission:

```
% ssh glados.cs.rit.edu # or use queeg.cs.rit.edu if glados is down
# password
# go to the directory where your solution is ...
% try hpb-grd lab11-2 'All files required'
```