Report on s1709906 Inf1OP 2018 Mock Programming Exam

Generated by Automarker on May 04, 2018

Question 1

Submitted Files: OK

Compiling Boss.java alone: OK

Compiling Boss.java with basic tests provided in the exam: OK

Running Basic Tests: All passed

Compiling Boss.java with main tests: OK

Running Main Tests: Passed 27 of 27 main tests

Q1: 25 / 25

Question 2

Submitted Files: OK

Compiling RockPaperScissors.java alone: OK

Compiling RockPaperScissors.java with basic tests provided in the exam: OK

Running Basic Tests: All passed

Compiling RockPaperScissors.java with main tests: OK

Running Main Tests: Passed 14 of 14 main tests

Q2: 25 / 25

Total Marks: 50/50

Submitted Files

Submitted: Boss.java

```
1 class Boss extends Monster {
2  private int stage;
3  private final int initialHealth;
4
5  public Boss(int health, int power) {
6   super(health, power);
7   initialHealth = health;
```

```
  stage = 1;
 }



 public void takeDamage(int damage) {
  super.takeDamage(damage);
  if (!isDefeated()) {
   double healthLeft = health / (double) initialHealth;
   if (stage == 1) {
    if (healthLeft < 0.5 && healthLeft >= 0.2) {
     stage++;
     power *= 2;
    }
    if (healthLeft < 0.2) {
     stage += 2;
     power *= 2;
     power *= 2;
    }
   }
   if (stage == 2) {
    if (healthLeft < 0.2) {
     stage++;
     power*=2;
    }
   }
  }
 }

 public String toString() {
  return String.format(super.toString() + "\nStage: " + stage);
 }

 public static void main(String[] args) {
  Boss b1 = new Boss(600, 20);
  b1.takeDamage(550);
 // b1.takeDamage(130);
  System.out.println(b1.toString());
 }

}
```

Submitted: RockPaperScissors.java

```
import java.util.Map;
import java.util.ArrayList;
import java.util.Hashtable;
import java.util.List;

class RockPaperScissors {

 /**
  * Returns if a given symbol is a valid symbol for the game.
  *
  * @param symbol
  *             the symbol to be checked
  * @return true if the given symbol is valid, false otherwise
```

```
 */
public static boolean isValidSymbol(char symbol) {
 return symbol == 'R' || symbol == 'P' || symbol == 'S';
}

public static List<Matchup> parseMatchups(String[] args) {
 // IMPLEMENT ME
 List<Matchup> matchups = new ArrayList<Matchup>();
 for (String s : args) {
  if (s.length() == 2) {
   int i = 0;
    if (isValidSymbol(s.charAt(i))) {
     Matchup m = new Matchup(s.charAt(0), s.charAt(1));
     matchups.add(m);
     i++;
    }
   }
 }

 return matchups;
}

/**
 * Returns the outcome of a given matchup as String.
 *
 * @param match
 *            The matchup to be decided.
 * @return a String representation of the matchup result; R, P, S or DRAW
 * @throws IllegalArgumentException
 *             if the given matchup parameter is null
 */
public static String decideOutcome(Matchup match) {
 if (match == null)
  throw new IllegalArgumentException("Given␣match␣must␣not␣be␣null!");

 String res = "" + match.getPlayerOne() + match.getPlayerTwo();

 if (res.equals("RP") || res.equals("PR"))
  return "P";
 if (res.equals("RS") || res.equals("SR"))
  return "R";
 if (res.equals("PS") || res.equals("SP"))
  return "S";

 return "DRAW";
}

public static Map<String, Integer> countOutcomes(List<Matchup> matches) {
 Map<String, Integer> result = new Hashtable<String,Integer>();
 int countR = 0;
 int countP = 0;
 int countS = 0;
 int countDraw = 0;

 if (matches.isEmpty() || matches == null) {
  return new Hashtable<String,Integer>();
```

```
70    }
71
72    for (Matchup matchup : matches) {
73     String winner = decideOutcome(matchup);
74     if (winner.equals("S")) {
75      countS++;
76      result.put("S", countS);
77     }
78
79     if (winner.equals("R")) {
80      countR++;
81      result.put("R", countR);
82     }
83
84     if (winner.equals("P")) {
85      countP++;
86      result.put("P", countP);
87     }
88
89     if (winner.equals("DRAW")) {
90      countDraw++;
91      result.put("DRAW", countDraw);
92     }
93    }
94
95    int maximum = 0;
96    String max = "";
97    for (String s : result.keySet()) {
98     if (result.get(s) > maximum) {
99      maximum = result.get(s);
100     max = s;
101    }
102   }
103
104   System.out.println("Most␣outcomes:␣" + max);
105
106   return result;
107  }
108
109  /**
110   * Executes different functionality for the game RockPaperScissors.
111   *
112   * @param args
113   *             A series of matchups encoded as pairs of single characters. The
114   *             series of matchups can be empty.
115   */
116  public static void main(String[] args) {
117 //  String[] a = { "PP", "PS", "SP", "RP", "RR" };
118   List<Matchup> matches = parseMatchups(args);
119
120   System.out.println(matches);
121
122   if (matches != null) {
123    System.out.println(matches.size() + "␣matches␣parsed.");
124
125    System.out.println("\nCount␣outcomes␣per␣symbol...");
```

```
    Map<String, Integer> outcomesPerSymbol = countOutcomes(matches);
    System.out.println(outcomesPerSymbol);
   }
  }

}
```