Report on s1709906 Inf1OP 2017 Mock Programming Exam

Generated by Automarker on May 01, 2018

Question 1

Submitted Files: OK

Compiling ComplexNumbers.java alone: OK

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

Running Basic Tests: All passed

Compiling ComplexNumbers.java with main tests: OK

Running Main Tests: Passed 14 of 18 main tests

Issue: 1.g The main method (1). The text printed out by the main method is different than expected. testMain1 failed. Impact: −1

Issue: 1.g The main method (2). The text printed out by the main method is different than expected. testMain2 failed. Impact: −1

Issue: 1.g The main method (3). The text printed out by the main method is different than expected. testMain3 failed. Impact: −1

Issue: 1.g The main method (4). The text printed out by the main method is different than expected. testMain4 failed. Impact: −1

Q1: 46 / 50

Question 2

Submitted Files: OK

Compiling Student.java alone: OK

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

Running Basic Tests: All passed

Compiling Student.java with main tests: OK

Running Main Tests: Passed 14 of 15 main tests

Issue: 2g) The string returned from the summary method is incorrect. testSummary failed. Impact: −4

Q2: 46 / 50

Total Marks: 92/100

Submitted Files

Submitted: ComplexNumbers.java

```
import java.util.Arrays;

public class ComplexNumbers {

 public static double[] complexAdd(double[] z1, double[] z2) {
  double[] result = new double[z1.length];

  for (int i = 0; i < z1.length; i++) {
   result[i] = z1[i] + z2[i];
  }

  return result;
 }

 public static double[] complexConjugate(double[] z) {
  double[] result = new double[z.length];

  if (z[1] != 0.0) {
   result[0] = z[0];
   result[1] = -z[1];
  } else {
   result[0] = z[0];
   result[1] = z[1];
  }

  return result;
 }

 public static double[] complexMultiply(double[] z1, double[] z2) {
  // (z1[0] + z1[1])(z2[0] + z2[1]) = (z1[0]*z2[0] - z1[1]*z2[1]) + (z1[1]*z2
[0] + z1[0]*z2[1])
  double[] result = new double[z1.length];

  result[0] = z1[0]*z2[0] - z1[1]*z2[1];
  result[1] = z1[1]*z2[0] + z1[0]*z2[1];

  return result;
 }


 public static double[] complexReciprocal(double[] z) {
  double[] result = new double[z.length];

  result[0] = z[0]/(z[0]*z[0] + z[1]*z[1]);
  result[1] = - z[1]/(z[0]*z[0] + z[1]*z[1]);

  return result;
 }

 public static String toString(double[] z) {
  String s = "";

  if (z[0] == 0 && z[1] == 0) {
```

```
    s += z[0];
   } else if (z[0] == 0) {
    s += String.format("%.1fi", z[1]);
   } else if (z[1] == 0) {
    s += String.format("%.1f", z[0]);
   } else if (z[1] < 0) {
    s += String.format("%.1f%.1fi", z[0], z[1]);
   } else if (z[1] > 0) {
    s += String.format("%.1f+%.1fi", z[0], z[1]);
   }

   return s;
  }

  public static double complexMagnitude(double[] z) {
   double result = Math.sqrt(z[0]*z[0] + z[1]*z[1]);
   return result;

  }

  public static double[][] sortByMagnitude(double[][] complexList) {
   Arrays.sort(complexList, (z1, z2) -> Double.compare(complexMagnitude(z1),
  complexMagnitude(z2)));
   return complexList;
  }

 // public static void main(String[] args) {
 //  // complexAdd test
 //  double[] z1 = {1.0,2.0};
 //  double[] z2 = {3.0,4.0};
 //
 //  System.out.println(Arrays.toString(complexAdd(z1,z2)));
 //
 //  // complexConjugate test
 //  double[] z3 = {2.0,0.0};
 //
 //  System.out.println(Arrays.toString(complexConjugate(z1)));
 //  System.out.println(Arrays.toString(complexConjugate(z3)));
 //
 //  // complexMultiply test
 //  System.out.println(Arrays.toString(complexMultiply(z1,z2)));
 //
 //  // complexReciprocal test
 //  System.out.println(Arrays.toString(complexReciprocal(z1)));
 //
 //  // toString test
 //  double[] z4 = {5.6666,-7.12};
 //  double[] z5 = {-3.355,-1.0};
 //  double[] z6 = {-1.0,0.0};
 //  double[] z7 = {0.0,-3.45};
 //  double[] z8 = {0.0,0.0};
 //
 //  System.out.println(toString(z1));
 //  System.out.println(toString(z4));
 //  System.out.println(toString(z5));
 //  System.out.println(toString(z6));
```

```
//  System.out.println(toString(z7));
//  System.out.println(toString(z8));
//
//  // sortByMagnitude test
//  double[][] z9 = {{5.0,-2.0},{1.0,2.0},{0.0,0.0}};
//  for (double[] z : sortByMagnitude(z9) ) {
//   System.out.print(Arrays.toString(z));
//  }
//
//
//
//
// }

 public static void main(String[] args) {
  double[] z1 = new double[2];
  double[] z2 = new double[2];

  for (int i = 0; i < args.length - 2; i++) {
   z1[i] = Double.parseDouble(args[i]);
  }

  for (int i = 0; i < args.length - 2; i++) {
   z2[i] = Double.parseDouble(args[i+2]);
  }

//  System.out.println(Arrays.toString(z1));
//  System.out.println(Arrays.toString(z2));

  // sum
  System.out.println(toString(complexAdd(z1,z2)));
  // conjugate f of z1
  System.out.println(toString(complexConjugate(z1)));
  // product
  System.out.println(toString(complexMultiply(z1,z2)));
  // reciprocal of z1
  System.out.println(toString(complexReciprocal(z1)));
  // conjugate of z1 + z2
  System.out.println(toString(complexConjugate(complexAdd(z1,z2))));
 }

}
```

Submitted: Student.java

```
import java.util.ArrayList;
import java.util.HashMap;

class Student extends Person {

 private ArrayList<Course> activeCourses = new ArrayList<Course>();
 private ArrayList<Course> completedCourses = new ArrayList<Course>();
 private int maxCourses;

 // getters and setters for maxCourses

 public int getMaxCourses() {
```

```
 return maxCourses;
}
public void setMaxCourses(int maxCourses) {
 this.maxCourses = maxCourses;
}

public Student(String firstname, String lastname, ArrayList<Course>
activeCourses, ArrayList<Course> completedCourses, int n) {
 super(firstname, lastname);
 this.activeCourses = activeCourses;
 this.completedCourses = completedCourses;
 setMaxCourses(n);
}

public boolean canEnroll(Course course) {

 boolean a = true;

 ArrayList<String> courseNames = new ArrayList<String>();

 for (Course c : completedCourses) {
  courseNames.add(c.getName());
 }

 for (String c : course.getPrerequisiteCourses()) {
  a = a && courseNames.contains(c);
 }

 boolean b = !activeCourses.contains(course);

 boolean c = !completedCourses.contains(course);

 boolean d = activeCourses.size() < maxCourses;

 boolean flag = a && b && c && d;

 return flag;
}

public boolean enroll(Course course) {
 if (!canEnroll(course)) {
  return false;
 }
 activeCourses.add(course);
 return true;
}

public boolean complete(String courseName) {
 ArrayList<String> activeNames = new ArrayList<String>();

 for (Course c : activeCourses) {
  activeNames.add(c.getName());
 }

 if (activeNames.contains(courseName)) {
  for (Course c : activeCourses) {
```

```
68      if (c.getName().equals(courseName)) {
69       activeCourses.remove(c);
70       completedCourses.add(c);
71       return true;
72      }
73     }
74    }
75    return false;
76   }
77
78   public HashMap<Course,String> getAllCourses() {
79    HashMap<Course,String> courses = new HashMap<Course,String>();
80    for (Course c : activeCourses) {
81     courses.put(c,"active");
82    }
83    for (Course c : completedCourses) {
84     courses.put(c, "completed");
85    }
86
87    return courses;
88   }
89
90   public String summary() {
91    String s = "";
92    for (Course c : getAllCourses().keySet()) {
93     s = s + c + "␣(" + getAllCourses().get(c) + ")" + "/n";
94    }
95
96    return s;
97   }
98
99  }
```