Inf1OP - Mock Exam - April 2017

Instructions

1. All questions are compulsory.
2. Remember that a file that does not compile, does not pass the basic JUnit tests provided, or does not use the default package, will get no marks.
3. This is an open book exam. You may bring your own material on paper. No electronic devices are permitted.
4. Calculators may not be used.

1. Complex Numbers

In this question you will implement routines to work with complex numbers by representing them as arrays. Create a public class **ComplexNumbers**.

A complex number can be expressed in the form $a + bi$, where a and b are real numbers and i is the imaginary unit ($i^2 = -1$). In this expression a is the *real part* and b is the *imaginary part*.

Since the value of the i constant never changes, in order to represent a complex number we need just two values of type **double**. We can use an array of **doubles** of size 2, to store these values, under the assumption that the value at index 0 represents the *real part*, and the value at index 1 represents the *imaginary part* of a complex number.

(a) Addition

In the class ComplexNumbers implement the public static method

double[] complexAdd(double[] z1, double[] z2)

that adds two complex numbers and returns the result. Complex numbers are added by separately adding the real and imaginary parts of the summands:

$$(a + bi) + (c + di) = (a + c) + (b + d)i \tag{1}$$

Expected behaviour: `complexAdd(new double[]`$\{1.0, 2.0\}$`, new double[]`$\{3.0, 4.0\}$)
should return $\{4.0, 6.0\}$ [*5 marks*]

(b) Conjugate

In the class ComplexNumbers implement the public static method

double[] complexConjugate(double[] z)

that returns the complex conjugate of the complex number z passed as the parameter. The complex conjugate of the complex number $z = a + bi$ is defined to be $a - bi$.

Expected behaviour:

`complexConjugate(new double[]`$\{1.0, 2.0\}$) should return $\{1.0, -2.0\}$

`complexConjugate(new double[]`$\{2.0, 0.0\}$) should return $\{2.0, 0.0\}$ [*5 marks*]

(c) Product

In the class ComplexNumbers implement the public static method

double[] complexMultiply(double[] z1, double[] z2)

that multiplies two complex numbers and returns the result. The multiplication of two complex numbers is defined by the following formula:

$$(a + bi)(c + di) = (ac - bd) + (bc + ad)i \tag{2}$$

Expected behaviour: `complexMultiply(new double[]{1.0, 2.0}, new double[]{3.0, 4.0})` should return $\{-5.0, 10.0\}$ [*5 marks*]

(d) Reciprocal

In the class ComplexNumbers implement the public static method

double[] complexReciprocal(double[] z)

that computes the reciprocal of the complex number z passed as the parameter. The reciprocal $\frac{1}{z}$ of the complex number $z = (a + bi)$ is defined by the following formula:

$$\frac{1}{z} = \frac{a}{a^2 + b^2} - \frac{b}{a^2 + b^2}i \tag{3}$$

Expected behaviour: `complexReciprocal(new double[]{1.0, 2.0})` should return $\{0.2, -0.4\}$ [*5 marks*]

(e) String representation

In the class ComplexNumbers implement the public static method

String toString(double[] z)

that returns a String representing the complex number z passed as the parameter. The string should be of the following form:

$a + bi$

where a and b are the values of the real and imaginary parts of the complex number, formatted to include 1 digit after the decimal point.

Expected behaviour:

`toString(new double[]{1.0, 2.0})` should return `"1.0+2.0i"`,
`toString(new double[]{5.6666, −7.12})` should return `"5.7-7.1i"`
`toString(new double[]{−3.355, −1.0})` should return `"-3.4-1.0i"`
`toString(new double[]{−1.0, 0.0})` should return `"-1.0"`
`toString(new double[]{0.0, −3.45})` should return `"-3.5i"`
`toString(new double[]{0.0, 0.0})` should return `"0.0"`

[*10 marks*]

(f) Sorting by magnitude

The magnitude of a complex number $z = a + bi$ is given by the formula $\sqrt{a^2 + b^2}$. In the class ComplexNumbers implement the public static method

double[][] sortByMagnitude(double[][] complexList)

that takes a list of complex numbers as a parameter and returns a list of the same complex numbers, sorted by their magnitude, in the ascending order. If the magnitude of two complex numbers is the same, their position with respect to each other in the sorted list is arbitrary.
Expected behaviour: `sortByMagnitude(new double[][]{{5.0, −2.0}, {1.0, 2.0}, {0.0, 0.0}}` should return $\{\{0.0, 0.0\}, \{1.0, 2.0\}, \{5.0, -2.0\}\}$

[*15 marks*]

(g) The main method

In the class ComplexNumbers implement a main method that reads two complex numbers from the command line. The method should expect 4 arguments of type double from the command line, representing the real and imaginary parts of each of the numbers.

For example, the input of this form:

```
1.0 2.0 3.0 4.0
```

represents the two complex numbers:

$z1 = (1.0 + 2.0i), z2 = (3.0 + 4.0i)$

Next, we will use the implemented routines and print out their results on the screen. In each of the following sub-parts, use the ComplexNumbers.toString(double[] z) method to print out the required number, **surrounded by parenthesis**.

- Print the sum of z1 and z2
- Print the conjugate of z1
- Print the product of z1 and z2
- Print the reciprocal of z1
- Print the conjugate of (z1 + z2)

Expected behaviour:
Invoking:

```
java ComplexNumbers 5.0 6.0 7.0 8.0
```

should print:
(12.0+14.0i)
(5.0-6.0i)
(-13.0+82.0i)
(0.1-0.1i)
(12.0-14.0)

[*5 marks*]

The file you must submit for this question is: **ComplexNumbers.java**
Remember your class should be in the default package.

2. Students and Courses

You are given files **Person.java** and **Course.java** containing the classes **Person** and **Course**. Examine these but do not change them. Your task is to implement the subclass **Student** extending **Person**. A **Person** who is a **Student** can enrol in **Courses**. By examining **Course** note that any course has a name and a array of Strings containing names of prerequisite courses that the student must have previously completed in order to enrol. If the **prerequisiteCourses** list is empty, it means that enrolment into this course doesn't require having completed any other courses.

(a) The Student class
Define the class **Student** extending **Person**. It should have a private instance variable **activeCourses** of type **ArrayList<Course>** which will store the courses that the **Student** has enrolled in. The **Student** class should also have a private instance variable **completedCourses** of type **ArrayList<Course>** that will store the courses already completed by the student int the past. A **Student** should also have a private instance variable **maxCourses** which stores as an **int** the maximum number of active courses that the **Student** can be enrolled into. It should also provide getter and setter methods for the **maxCourses** variable. [*5 marks*]

(b) The constructor
Write a public constructor for **Student**. The constructor must take two **String**s representing the student's first name and last name and pass them to the **Person** constructor. It must also take two **ArrayList<Course>** lists representing the student's active courses, and the courses already completed by the student. It should use these to set the **activeCourses** and **completedCourses** variables. Finally, it should take an integer value and use this to set the **maxCourses** variable. [*5 marks*]

(c) Checking enrolment eligibility
Write public method **canEnroll** that takes a **Course** object and returns a **boolean**. This method should return true if the student can enrol into it, and false otherwise. Specifically, enrolment is allowed if the following requirements are met:

- all the courses on the **prerequisiteCourses** list of the new Course are also in the **completedCourses** list of the Student,
- the Student is not already enrolled into the new Course,
- the Student has not already completed the new Course,
- the Student isn't already enrolled in the maximum number of courses that they are allowed to be (specified by the **maxCourses** variable).

[*10 marks*]

(d) Enrolling into courses
Write public method **enroll** that takes a **Course** object and returns a **boolean**. This method should enrol the Student into the course and return true if the action was successful, and false otherwise. In order to achieve that, the method must:

- check if the student can enrol in the course, using the **canEnroll()** method,
- add the course to the activeCourses list.

[*5 marks*]

(e) Completing courses
Write a public method **complete(String courseName)** that takes a **String** with the name of the course that the student has completed, and returns a **boolean** after performing the following actions:

- the course with the name specified by **courseName** is removed from the activeCourses list,
- the course is added to the completedCourses list.

The method should return true if the completion of the course was successful, and false otherwise (for example - if the student wasn't enrolled in this course in the first place). [*10 marks*]

(f) Retrieving all courses
Write a public method **getAllCourses()** returns a **HashMap<Course,String>** of courses and their status. Each course that the student has completed or is currently pursuing, should be represented as a key in the map, and the value assigned to that key should be either the String `"active"` or the String `"completed"`.

For example, if the **Student's activeCourses** list contains the entries `[<Course>"Advanced Java Programming", <Course>"Machine Learning"]`, and the **completedCourses** list contains the entries `[<Course>"Introduction to Java Programming", <Course>"Basics of Python"]`, the method should return the map:

```
{ { <Course>"Advanced Java Programming" -> "active" },
{ { <Course>"Machine Learning" -> "active" },
{ { <Course>"Introduction to Java Programming" -> "completed" },
{ { <Course>"Basics of Python" -> "completed" } }
```

[*10 marks*]

(g) Student summary

Write a public method **summary()** that returns a **String** object. The String should contain a list of the names of the courses that the Student is currently enrolled in, or has completed in the past. If the course is active, it should be followed by the description " `(active)`". If the course is completed, it should be followed by the description " `(completed)`". Note the space character between the name of the course and the description in the brackets. The active courses should be on top of the list, followed by the completed courses. Don't forget about breaking the line after each course.

For example, if the Student **s** is enrolled in courses: ["Formal Methods", "Artificial Intelligence"] and has completed courses: ["Java Programming", "Software Design" , "Machine Learning",]. Then **System.out.println(s.summary())** should print out:
Formal Methods (active)
Artificial Intelligence (active)
Java Programming (completed)
Software Design (completed)
Machine Learning (completed)

[*5 marks*]

In Summary, **Student** has the following public interface (omitting what is inherited from **Person**) with the behaviour explained above:

public	class Student
	Student(String fn, String ln, ArrayList<Course>active, ArrayList<Course>completed, int mc)
int	getMaxCourses()
void	setMaxCourses(int m)
boolean	canEnroll(Course c)
boolean	enroll(Course c)
boolean	complete(String c)
HashMap<Course, String>	getAllCourses()
String	summary()

Remember, you will need these lines:

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

at the top of your file.
The file you must submit for this question is: **Student.java**
Remember your class should be in the default package.