Construction Of Software Engineering
Suggested Solution to Problem Set 1

No Author Given

Information System of Technology and Design,
Singapore University of Technology and Design,
Singapore

Cohort Exercise 1

Design and implement a program that supports accepting two complex numbers from the user; adding, subtracting, multiplying, and dividing them; and reporting each result to the user.

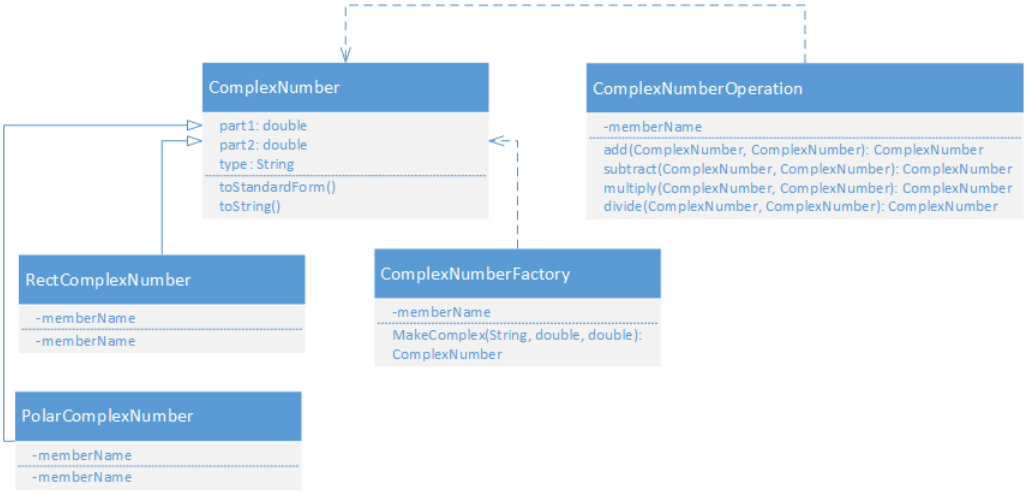

Fig. 1. Class diagram for cohort exercise 1

Listing 1.1. CE1 code

```
1 import java.util.Scanner;
2
3 public class MainClass {
4     public static void main(String[] args) {
5         ComplexNumberFactory factory = new ComplexNumberFactory();
6         Scanner s = new Scanner(System.in);
```

```
7             System.out.println("Please input two complex numbers: (\"rect\"? \"polar\"?
                  part1 part2)");
8             System.out.print("The first one: ");
9             ComplexNumber c1 = factory.MakeComplex(s.next(), s.nextDouble(),
                  s.nextDouble());
10            System.out.print("The second one: ");
11            ComplexNumber c2 = factory.MakeComplex(s.next(), s.nextDouble(),
                  s.nextDouble());
12            System.out.println(c1 + " + " + c2 + " = " + ComplexNumberOperation.add(c1,
                  c2));
13            System.out.println(c1 + " - " + c2 + " = " +
                  ComplexNumberOperation.subtract(c1, c2));
14            System.out.println(c1 + " * " + c2 + " = " +
                  ComplexNumberOperation.multiply(c1, c2));
15            System.out.println(c1 + " / " + c2 + " = " +
                  ComplexNumberOperation.divide(c1, c2));
16        }
17    }
18
19    class ComplexNumber {
20        protected String type = "std";
21        protected double part1;
22        protected double part2;
23        public String toString()
24        {
25            return type + "(" + part1 + ", " + part2 + ")";
26        }
27        public ComplexNumber(String t, double d1, double d2)
28        {
29            type = t;
30            part1 = d1;
31            part2 = d2;
32        }
33        public ComplexNumber(double d1, double d2)
34        {
35            part1 = d1;
36            part2 = d2;
37        }
38        public ComplexNumber toStandardForm()
39        {
40            return this;
41        }
42    }
43
44    class RectComplexNumber extends ComplexNumber {
45        public RectComplexNumber(double d1, double d2)
46        {
47            super("rect", d1, d2);
48        }
49    /*  public PolarComplexNumber toPolarComplexNumber()
```

```
50      {
51          double r = Math.sqrt(part1 * part1 + part2 * part2);
52          double angle = Math.PI / 2;
53          if (part1 != 0) {
54              angle = Math.atan(part2 / part1);
55          }
56          if (part1 < 0)
57              angle += Math.PI;
58          return new PolarComplexNumber(r, angle);
59      }*/
60  }
61
62  class PolarComplexNumber extends ComplexNumber{
63
64      public PolarComplexNumber(double d1, double d2)
65      {
66          super("polar", d1, d2);
67      }
68      public ComplexNumber toStandardForm()
69      {
70          return new ComplexNumber(part1 * Math.cos(part2), part1 * Math.sin(part2));
71      }
72  }
73
74  class ComplexNumberOperation {
75      public static ComplexNumber add(ComplexNumber c1, ComplexNumber c2)
76      {
77          ComplexNumber cc1 = c1.toStandardForm();
78          ComplexNumber cc2 = c2.toStandardForm();
79          return new ComplexNumber(cc1.part1 + cc2.part1, cc1.part2 + cc2.part2);
80      }
81      public static ComplexNumber subtract(ComplexNumber c1, ComplexNumber c2)
82      {
83          ComplexNumber cc1 = c1.toStandardForm();
84          ComplexNumber cc2 = c2.toStandardForm();
85          return new ComplexNumber(cc1.part1 - cc2.part1, cc1.part2 - cc2.part2);
86      }
87      public static ComplexNumber multiply(ComplexNumber c1, ComplexNumber c2)
88      {
89          ComplexNumber cc1 = c1.toStandardForm();
90          ComplexNumber cc2 = c2.toStandardForm();
91          return new ComplexNumber(cc1.part1 * cc2.part1 - cc1.part2 * cc2.part2,
92                  cc1.part1 * cc2.part2 + cc1.part2 * cc2.part1);
93      }
94      public static ComplexNumber divide(ComplexNumber c1, ComplexNumber c2)
95      {
96          ComplexNumber cc1 = c1.toStandardForm();
97          ComplexNumber cc2 = c2.toStandardForm();
98          if (cc2.part1 == 0 && cc2.part2 == 0)
99              return null;
```

```
100             return new ComplexNumber((cc1.part1 * cc2.part1 + cc1.part2 * cc2.part2) /
                    (cc2.part1 * cc2.part1 + cc2.part2 * cc2.part2),
101                     (cc1.part2 * cc2.part1 - cc1.part1 * cc2.part2 ) / (cc2.part1 *
                            cc2.part1 + cc2.part2 * cc2.part2));
102         }
103     }
104
105     class ComplexNumberFactory{
106         ComplexNumber MakeComplex(String type, double d1, double d2)
107         {
108             if (type.equals("polar")) {
109                 return new PolarComplexNumber(d1, d2);
110             }
111             return new RectComplexNumber(d1, d2);
112         }
113     }
```

Cohort Exercise 3

Draw individually a user case diagram for KBO.

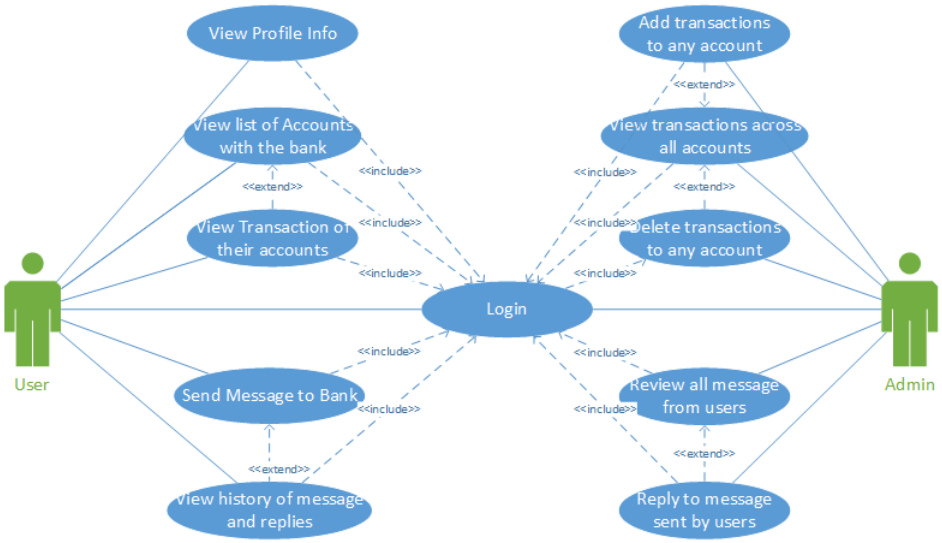

Fig. 2. Class diagram for cohort exercise 3

Cohort Exercise 5

Draw a class diagram for the following scenario. In a university there are different classrooms, offices and departments. A department has a name and it contains many offices. A person working at the university has a unique ID and can be a professor or an employee.

- A professor can be a full, associate or assistant professor and he/she is enrolled in one department.
- Offices and classrooms have a number ID, and a classroom has a number of seats.
- Every employee works in an office.

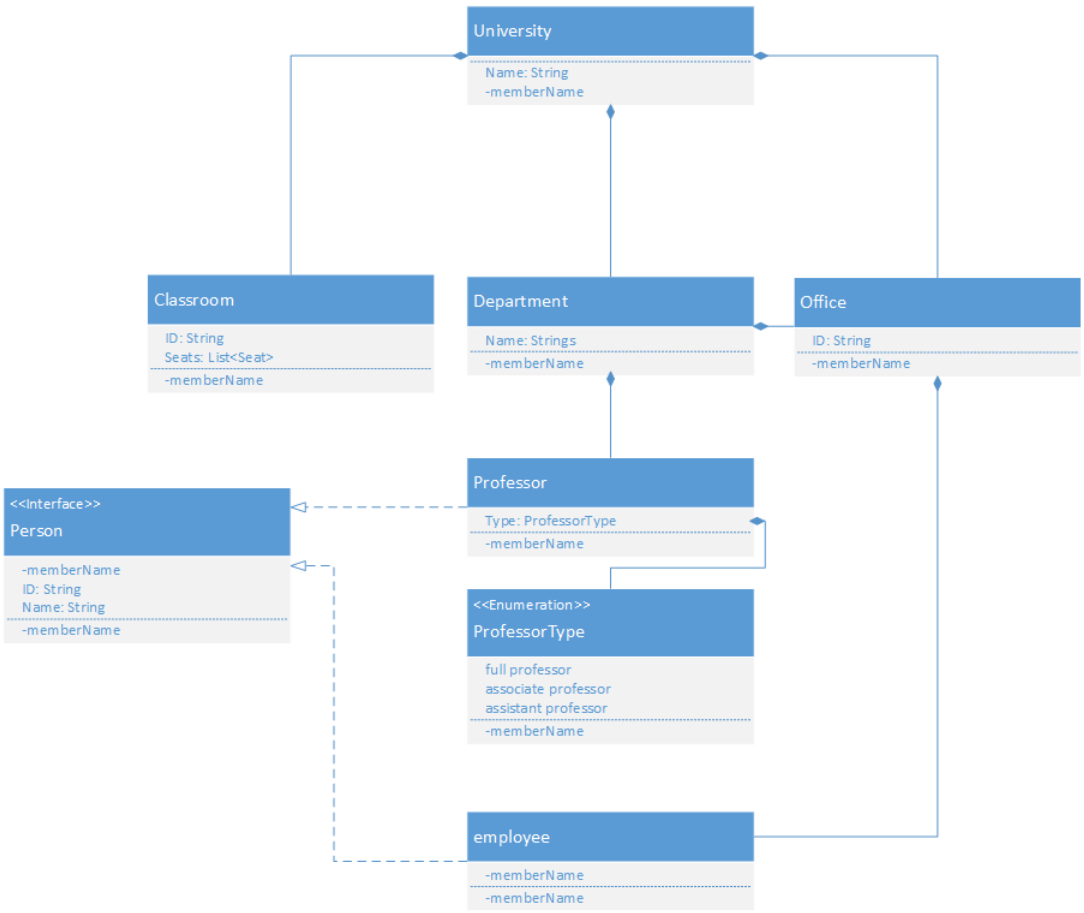

Fig. 3. Class diagram for cohort exercise 5

Cohort Exercise 6

A hardware update wizard can be in three states as follows:
1. Displaying a hardware update window.
2. Searching for new hardware.
3. Displaying new hardware found.
The wizard starts by displaying a hardware update window. While displaying this window, the user can press a ”Search” button to cause the wizard to start searching for new hardware, or the user can press a ”Finish” button to leave the wizard. While the wizard is searching for new hardware, the user may cancel the search at any time. If the user cancels the search, the wizard displays the hardware update window again. When the wizard has completed searching for new hardware, it displays the new hardware found. Draw a state machine diagram that represents the function of the hardware update wizard just described.

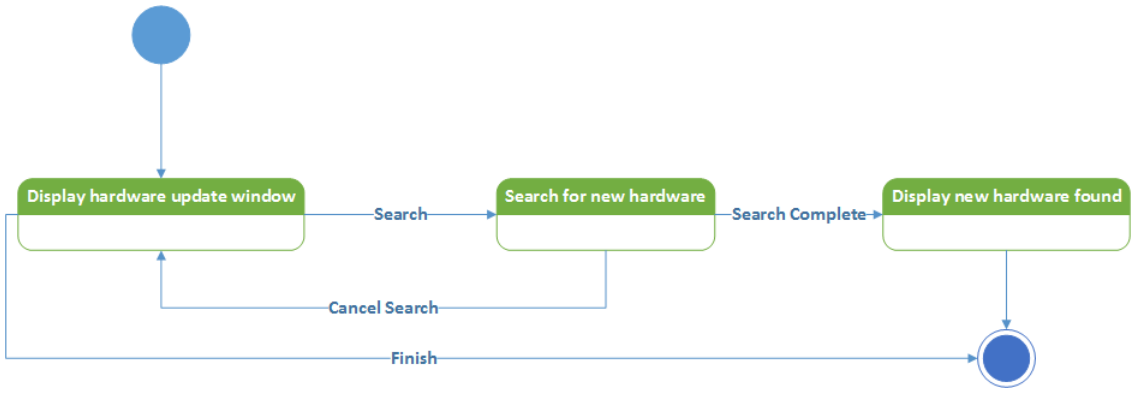

Fig. 4. Class diagram for cohort exercise 6