CSC 1103 OBJECT ORIENTED PROGRAMMING (SEMESTER I, 2018/2019)

Lab #6 : Thinking in Objects

Duration: 1 hour 20 minutes

Objectives:

At the end of this lab session, student shall be able to design java classes and explores the differences between structured and object-oriented programming

Reminder : Use proper header, apply programming styles and follow naming convention

Task 1

Java provides many levels of abstraction, and class abstraction separates class implementation from how the class is used. The details of implementation are encapsulated and hidden from the user. This is known as class encapsulation. See UML class diagram for the Loan class:

Loan	
-annualInterestRate: double	The annual interest rate of the loan (default: 2.5).
-numberOfYears: int	The number of years for the loan (default: 1)
-loanAmount: double	The loan amount (default: 1000).
-loanDate: Date	The date this loan was created.
+Loan()	Constructs a default Loan object.
+Loan(annualInterestRate: double, numberOfYears: int, loanAmount: double)	Constructs a loan with specified interest rate, years, and loan amount.
+getAnnualInterestRate(): double	Returns the annual interest rate of this loan.
+getNumberOfYears(): int	Returns the number of the years of this loan.
+getLoanAmount(): double	Returns the amount of this loan.
+getLoanDate(): Date	Returns the date of the creation of this loan.
+setAnnualInterestRate(annualInterestRate: double): void	Sets a new annual interest rate to this loan.
+setNumberOfYears(numberOfYears: int): void	Sets a new number of years to this loan.
+setLoanAmount(loanAmount: double): void	Sets a new amount to this loan.
+getMonthlyPayment(): double	Returns the monthly payment of this loan.
+getTotalPayment(): double	Returns the total payment of this loan.

1. Run Eclipse on your machine. Create a new Java file and name it TestLoanClass.java

2. Type the following code:

```
 1  import java.util.Scanner;
 2
 3  public class TestLoanClass {
 4    /** Main method */
 5    public static void main(String[] args) {
 6      // Create a Scanner
 7      Scanner input = new Scanner(System.in);
 8
 9      // Enter yearly interest rate
10      System.out.print(
```

```
11        "Enter yearly interest rate, for example, 8.25: ");
12      double annualInterestRate = input.nextDouble();
13
14      // Enter number of years
15      System.out.print("Enter number of years as an integer: ");
16      int numberOfYears = input.nextInt();
17
18      // Enter loan amount
19      System.out.print("Enter loan amount, for example, 120000.95:
");
20      double loanAmount =  input.nextDouble();
21
22      // Create Loan object
23      Loan loan =
24        new Loan(annualInterestRate, numberOfYears, loanAmount);
25
26      // Display loan date, monthly payment, and total payment
27      System.out.printf("The loan was created on %s\n" +
28        "The monthly payment is %.2f\nThe total payment is
%.2f\n",
29        loan.getLoanDate().toString(), loan.getMonthlyPayment(),
30        loan.getTotalPayment());
31    }
32  }
```

3. Create a new Java file and name it Loan.java

4. Type the following code and execute the code when there is no error.

```
 1  public class Loan {
 2    private double annualInterestRate;
 3    private int numberOfYears;
 4    private double loanAmount;
 5    private java.util.Date loanDate;
 6
 7    /** Default constructor */
 8    public Loan() {
 9      this(2.5, 1, 1000);
10    }
11
12    /** Construct a loan with specified annual interest rate,
13        number of years, and loan amount
14      */
15    public Loan(double annualInterestRate, int numberOfYears,
16        double loanAmount) {
17      this.annualInterestRate = annualInterestRate;
18      this.numberOfYears = numberOfYears;
19      this.loanAmount = loanAmount;
20      loanDate = new java.util.Date();
21    }
22
23    /** Return annualInterestRate */
24    public double getAnnualInterestRate() {
25      return annualInterestRate;
26    }
27
28    /** Set a new annualInterestRate */
29    public void setAnnualInterestRate(double annualInterestRate) {
30      this.annualInterestRate = annualInterestRate;
31    }
```

```
 32
 33    /** Return numberOfYears */
 34    public int getNumberOfYears() {
 35      return numberOfYears;
 36    }
 37
 38    /** Set a new numberOfYears */
 39    public void setNumberOfYears(int numberOfYears) {
 40      this.numberOfYears = numberOfYears;
 41    }
 42
 43    /** Return loanAmount */
 44    public double getLoanAmount() {
 45      return loanAmount;
 46    }
 47
 48    /** Set a newloanAmount */
 49    public void setLoanAmount(double loanAmount) {
 50      this.loanAmount = loanAmount;
 51    }
 52
 53    /** Find monthly payment */
 54    public double getMonthlyPayment() {
 55      double monthlyInterestRate = annualInterestRate / 1200;
 56      double monthlyPayment = loanAmount * monthlyInterestRate /
(1 -
 57        (1 / Math.pow(1 + monthlyInterestRate, numberOfYears *
12)));
 58      return monthlyPayment;
 59    }
 60
 61    /** Find total payment */
 62    public double getTotalPayment() {
 63      double totalPayment = getMonthlyPayment() * numberOfYears *
12;
 64      return totalPayment;
 65    }
 66
 67    /** Return loan date */
 68    public java.util.Date getLoanDate() {
 69      return loanDate;
 70    }
 71  }
```

Task 2

The procedural (structural) paradigm focuses on designing methods. The OO paradigm couples data and methods together into objects. Hence, designing software using OO focuses on objects and operations on objects. Now let's redefine the program to compute BMI:

1. Run Eclipse on your machine. Create a new Java file and name it BMI.java

2. Type the following code:

```java
public class BMI {
  private String name;
  private int age;
  private double weight; // in pounds
  private double height; // in inches

  public static final double KILOGRAMS_PER_POUND = 0.45359237;
  public static final double METERS_PER_INCH = 0.0254;

  public BMI(String name, int age, double weight, double height) {
    this.name = name;
    this.age = age;
    this.weight = weight;
    this.height = height;
  }

  public BMI(String name, double weight, double height) {
    this(name, 20, weight, height);
  }

  public double getBMI() {
    double bmi = weight * KILOGRAMS_PER_POUND /
      ((height * METERS_PER_INCH) * (height * METERS_PER_INCH));
    return Math.round(bmi * 100) / 100.0;
  }

  public String getStatus() {
    double bmi = getBMI();
    if (bmi < 18.5)
      return "Underweight";
    else if (bmi < 25)
      return "Normal";
    else if (bmi < 30)
      return "Overweight";
    else
      return "Obese";
  }

  public String getName() {
    return name;
  }

  public int getAge() {
    return age;
  }

  public double getWeight() {
    return weight;
  }

  public double getHeight() {
    return height;
  }
}
```

3. Create a new Java file and name it UseBMIClass.java

4. Type the following code and execute the code when there is no error.

```
1  public class UseBMIClass {
2    public static void main(String[] args) {
3      BMI bmi1 = new BMI("Jep", 18, 145, 70);
4      System.out.println("The BMI for " + bmi1.getName() + " is "
5        + bmi1.getBMI() + " " + bmi1.getStatus());
6
7      BMI bmi2 = new BMI("KakTop", 215, 70);
```

```
8      System.out.println("The BMI for " + bmi2.getName() + " is "
9        + bmi2.getBMI() + " " + bmi2.getStatus());
10   }
11 }
```

Task 3 (Designing the Course class)

1. Create a new Java file and name it **Course**.java

2. Type the following code and execute when there is no error:

```
1  public class Course {
2    private String courseName;
3    private String[] students = new String[100];
4    private int numberOfStudents;
5
6    public Course(String courseName) {
7      this.courseName = courseName;
8    }
9
10   public void addStudent(String student) {
11     students[numberOfStudents] = student;
12     numberOfStudents++;
13   }
14
15   public String[] getStudents() {
16     return students;
17   }
18
19   public int getNumberOfStudents() {
20     return numberOfStudents;
21   }
22
23   public String getCourseName() {
24     return courseName;
25   }
26
27   public void dropStudent(String student) {
28     // Left as an exercise in Exercise 10.9
29   }
30 }
```

3. Create a new Java file and name it **TestCourse**.java

4. Type the following code and execute when there is no error:

```
1  public class TestCourse {
2    public static void main(String[] args) {
3      Course course1 = new Course("Data Structures");
4      Course course2 = new Course("Database Systems");
5
6      course1.addStudent("Yana");
7      course1.addStudent("Sakina");
8      course1.addStudent("Nazli");
9
10     course2.addStudent("Haris");
11     course2.addStudent("Shukri");
12
13     System.out.println("Number of students in course1: "
14       + course1.getNumberOfStudents());
15     String[] students = course1.getStudents();
16     for (int i = 0; i < course1.getNumberOfStudents(); i++)
```

```
17       System.out.print(students[i] + ", ");
18
19     System.out.println();
20     System.out.print("Number of students in course2: "
21       + course2.getNumberOfStudents());
22   }
23 }
```

Exercise 1:(Account class) Design a class named Account that contains:

1. A private int data field named **id** for the account (default 0).
2. A private double data field named **balance** for the account (default 0).
3. A private double data field named **annualInterestRate** that stores the current interest rate (default 0). Assume all accounts have the same interest rate.
4. A private Date data field named **dateCreated** that stores the date when the account was created.
5. A **no-arg constructor** that create a default account.
6. A **constructor** that creates an account with the specified id and initial balance.
7. The **accessor** and **mutator** methods for **id**, **balance**, and **annualInterestRate**.
8. The **accessor** method for **date** Created.
9. A method named **getMonthlyInterestRate**() that returns the monthly interest rate.

10. A method named **getMonthlyInterest**() that returns the monthly interest.
11. A method named **withdraw** that withdraws a specified amount from the account.
12. A method named **deposit** that deposits a specified amount to the account.

Draw the **UML diagram** for the class and then implement the class. (Hints: The method getMonthlyInterest() is to return monthly interest, not the interest rate.

Monthly interest is balance * monthlyInterestRate.

monthlyInterestRate is annualInterestRate/ 12. Note that annualInterestRate is a percentage, e.g., like 45%. You need to divide it by 100.

Write a test program that creates an Account object with an account ID 1122, a balance of RM20,000, and an annual interest rate of 4.5%. Use the withdraw method to withdraw RM2,500, use the deposit method to deposit RM3,000, and print the balance, the monthly interest, and the date when this account was created.

Exercise 2: (*Geometry: the* **MyRectangle2D** *class*) Define the **MyRectangle2D** class that contains:

■ Two **double** data fields named **x** and **y** that specify the center of the rectangle with getter and setter methods. (Assume that the rectangle sides are parallel to **x-** or **y-** axes.)

■ The data fields **width** and **height** with getter and setter methods.

■ A no-arg constructor that creates a default rectangle with (**0**, **0**) for (**x**, **y**) and **1** for both **width** and **height**.

■ A constructor that creates a rectangle with the specified **x**, **y**, **width**, and **height**.

■ A method **getArea()** that returns the area of the rectangle.

■ A method **getPerimeter()** that returns the perimeter of the rectangle.

■ A method **contains(double x, double y)** that returns **true** if the specified point (**x**, **y**) is inside this rectangle (see Figure 1a).

■ A method **contains(MyRectangle2D r)** that returns **true** if the specified rectangle is inside this rectangle (see Figure 1b).

■ A method **overlaps(MyRectangle2D r)** that returns **true** if the specified rectangle overlaps with this rectangle (see Figure 1c).

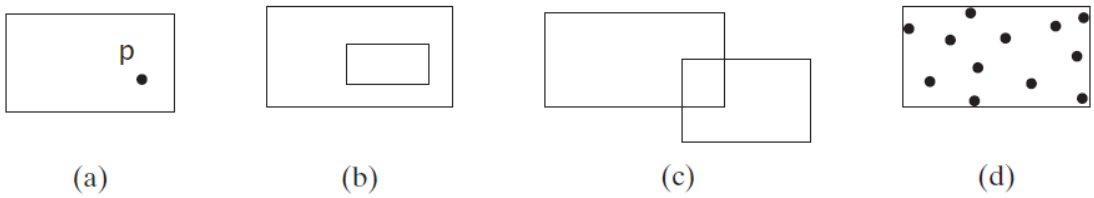

FIGURE 1 A point is inside the rectangle. (b) A rectangle is inside another rectangle. (c) A rectangle overlaps another rectangle. (d) Points are enclosed inside a rectangle.

End of Questions