Lab 4: Methods and constructors overloading

Lab Objective: After this lab, the student should be able to:

- Overload constructors and methods
- Practice constructor chaining
- Code and trace recursive methods
- Encapsulating data
- Write composed classes

Part 1. Constructor overloading

1. A part of the class Student is provided bellow

```
// ------- Class Student ---------------------------------------
public class Student {
    public long id;
    public String name;
    public String major;
    public String department;
    public String[] courses;

    public Student(long id, String name, String title, String department,
Courses []Courses) {…}

    public Student (long id, String name, String title, String department){…}

    public Student (long id, String name, String title) {…}

    public Student (long id, String name) {…}
}
```

a. Complete the constructors code so that:
   - Each constructor has a print instruction to identify it using the number of its parameters
   - Minimize repetitive coding through constructor chaining
b. Code the main method which instantiates several Student objects, each using a different constructor.
c. Run your code and observe how many constructor calls are required for each instantiation.

Part 2. n-sided regular polygon

In an *n*-sided regular polygon, all sides have the same length and all angles have the same degree (i.e., the polygon is both equilateral and equiangular).

- Design a class named **RegularPolygon** that contains:
- A private **int** data field named **n** that defines the number of sides in the polygon with default value **3**.
- A private **double** data field named **side** that stores the length of the side with default value **1**.
- A private **double** data field named **x** that defines the *x*-coordinate of the polygon's center with default value **0**.
- A private **double** data field named **y** that defines the *y*-coordinate of the polygon's center with default value **0**.
- A no-arg constructor that creates a regular polygon with default values.
- A constructor that creates a regular polygon with the specified number of sides and length of side, centered at (**0**, **0**).
- A constructor that creates a regular polygon with the specified number of sides, length of sides, and *x*-and *y*coordinates.
- The accessor and mutator methods for all data fields.
- The method **getPerimeter()** that returns the perimeter of the polygon.
- The method **getArea()** that returns the area of the polygon. The formula for computing the area of a regular polygon is: $area = \frac{n*sides^2}{4*\tan\left(\frac{\pi}{n}\right)}$

Write a test program that creates three **RegularPolygon** objects, created using the no-arg constructor, using **RegularPolygon(6, 4)**, and using **RegularPolygon(10,4, 5.6, 7.8)**. For each object, display its perimeter and area.

Part 3. Account and ATM

Design a class named Account that contains:

- A private int data field named id for the account (default 0).
- A private double data field named balance for the account (default 0).
- A private Date data field named dateCreated that stores the date when the
- account was created.
- A no-arg constructor that creates a default account.
- A constructor that creates an account with the specified id and initial balance.
- The accessor and mutator methods for id, balance, and annualInterestRate.

- The accessor method for dateCreated.
- A method named withdraw that withdraws a specified amount from the account.
- A method named deposit that deposits a specified amount to the account.

- Code a driver code to test your Account class
- Use the created Account class to simulate an ATM machine. Create ten accounts in an array with id 0, 1, . . . , 9, and initial balance $100. The system prompts the user to enter an id. If the id is entered incorrectly, ask the user to enter a correct id. Once an id is accepted, the main menu is displayed as shown in the sample run. You can enter a choice 1 for viewing the current balance, 2 for withdrawing money, 3 for depositing money, and 4 for exiting the main menu. Once you exit, the system will prompt for an id again. Thus, once the system starts, it will not stop.

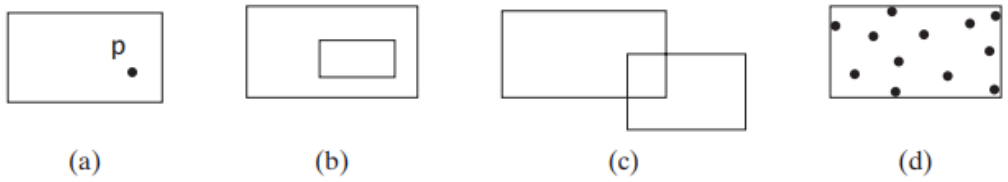
An example of running the code is given bellow.

```
Enter an id: 5
Main menu
1: check balance
2: withdraw
3: deposit
4: exit
Enter a choice: 1
The balance is 100.0
Main menu
1: check balance
2: withdraw
3: deposit
4: exit
Enter a choice: 4
```

Part 4. MyRectangle2D

Define the **MyRectangle2D** class that contains:

- Two **double** data fields named **x** and **y** that specify the center of the rectangle with **get** and **set** methods. (Assume that the rectangle sides are parallel to **x**or **y-** axes.)
- The data fields **width** and **height** with **get** and **set** methods.
- A no-argument 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 provided **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).
- A method **contains(MyRectangle2D r)** that returns **true** if the specified rectangle is inside this rectangle (see Figure).
- A method **overlaps(MyRectangle2D r)** that returns **true** if the specified rectangle overlaps with this rectangle (see Figure).



(a) 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.