



Course Title: Object - Oriented Programming II LAB

Course Code: CSE 2110 Assignment :06

<u>Submitted by</u>	<u>Submitted to</u>
Name: Muhammad Shabab Sayem ID: 11230321377 Department: CSE Section: 3J	Name: SHOVON MANDAL  Designation: Lecturer Dept. of Computer Science Engineering Northern University of Business & Technology, Khulna

Submission Date: 12/02/25

## **Problem 1:**

### **Question:**

Design a class called circle which contains:

- Two private instance variables: radius (of the type double) and color (of the type String), with default value of 1.0 and "red", respectively.
- Two overloaded constructors - a default constructor with no argument, and a constructor which takes a double argument for radius.
- Two public methods: getRadius() and getArea(), which return the radius and area of this instance, respectively.

### **Objective:**

The objective of the provided Java code is to define a Main class (which likely should have been named Circle for better clarity) that represents a circle. It aims to:

1. Encapsulate the properties of a circle (radius and color).
2. Provide constructors to initialize circle objects with different parameters (default radius and color, or a specified radius and default color).
3. Offer methods to access the circle's radius and calculate its area.

### **Lab Work:**

```
public class Main {
    private double radius;
    private String color;

    public Main() {
        radius = 1.0;
        color = "red";
    }

    public Main(double radius) {
        this.radius = radius;
        this.color = "red";
    }

    public double getRadius() {
        return radius;
    }

    public double getArea() {
        return Math.PI * radius * radius;
    }

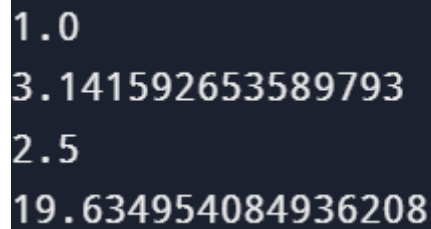
    public static void main(String[] args) {
        Main c1 = new Main();
    }
}
```

```
System.out.println(c1.getRadius());
System.out.println(c1.getArea());

Main c2 = new Main(2.5);
System.out.println(c2.getRadius());
System.out.println(c2.getArea());

    }
}
```

### **Output:**



```
1.0
3.141592653589793
2.5
19.634954084936208
```

### **Result analysis:**

- `Main c1 = new Main();`: This creates a Main object c1 using the default constructor. The default radius is 1.0 and the color is "red" (though the color isn't used in the output).
- `System.out.println(c1.getRadius());`: This prints the radius of c1, which is 1.0.
- `System.out.println(c1.getArea());`: This calculates and prints the area of c1 using the formula  $\pi r^2$ , where  $r = 1.0$ . The output is approximately 3.141592653589793 (the value of  $\pi$ ).
- `Main c2 = new Main(2.5);`: This creates a Main object c2 using the parameterized constructor, setting the radius to 2.5 and the color to "red".
- `System.out.println(c2.getRadius());`: This prints the radius of c2, which is 2.5.
- `System.out.println(c2.getArea());`: This calculates and prints the area of c2 using the formula  $\pi r^2$ , where  $r = 2.5$ . The output is approximately 19.634954084936208.

## **Problem 2:**

### **Question:**

Write a Java class Book with following features:

- Instance variables :
  - o title for the title of book of type String.
  - o author for the author's name of type String.
  - o price for the book price of type double.
- Constructor:
  - o public Book (String title, String author, double price): A constructor with parameters, it creates the Author object by setting the the fields to the passed values.

Northern University & Business Technology Khulna [Practice Problems - Sheet 1]  
[SM] Semester : Fall 2024 Course Title : Object - Oriented Programming II Course Code : CSE 2109 Course Credits : 01

- Instance methods:
  - o public void setTitle(String title): Used to set the title of book.
  - o public void setAuthor(String author): Used to set the name of author of book.
  - o public void setPrice(double price): Used to set the price of book.
  - o public String getTitle(): This method returns the title of book.
  - o public String getAuthor(): This method returns the author's name of book.
  - o public String toString(): This method printed out book's details to the screen

Write a separate class BookDemo with a main() method creates a Book titled "Developing Java Software" with authors Russel Winderand price 79.75. Prints the Book's string representation to standard output (using System.out.println).

### **Objective:**

The objective of this code is to create a Java class Book that represents a book with attributes such as title, author, and price. It provides methods to set and get these attributes and override the toString method for string representation. The BookDemo class demonstrates object instantiation and prints book details.

### **Lab work:**

```
class Book {
    private String title;
    private String author;
    private double price;

    public Book(String title, String author, double price) {
        this.title = title;
        this.author = author;
        this.price = price;
    }

    public void setTitle(String title) {
        this.title = title;
    }

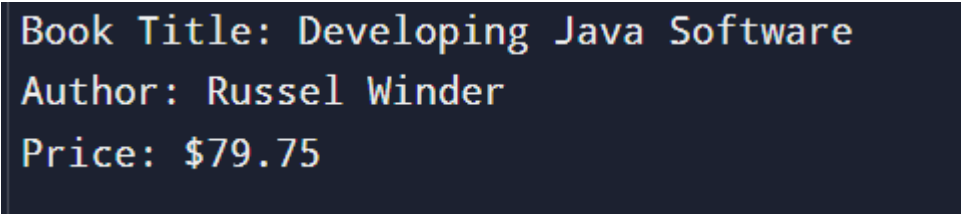
    public void setAuthor(String author) {
        this.author = author;
    }

    public void setPrice(double price) {
        this.price = price;
    }

    public String getTitle() {
        return title;
    }
}
```

```
public String getAuthor() {  
    return author;  
}  
  
public double getPrice() {  
    return price;  
}  
  
public String toString() {  
    return "Book Title: " + title + "\nAuthor: " + author + "\nPrice: $" + price;  
}  
}  
  
public class Main {  
    public static void main(String[] args) {  
        Book book = new Book("Developing Java Software", "Russel Winder", 79.75);  
        System.out.println(book.toString());  
    }  
}
```

### **Output:**

A screenshot of a terminal window with a dark background. The output text is displayed in a light blue/cyan monospaced font. It shows three lines of text: 'Book Title: Developing Java Software', 'Author: Russel Winder', and 'Price: \$79.75', each on a new line.

```
Book Title: Developing Java Software  
Author: Russel Winder  
Price: $79.75
```

### **Result analysis:**

When executed, the program creates an instance of Book with the specified title, author, and price. It then prints the book details to the console in a readable format using the overridden toString method.

### **Problem 3:**

#### **Question:**

Write a Java class Author with following features:

- Instance variables :
  - o firstName for the author's first name of type String.
  - o lastName for the author's last name of type String.
- Constructor:
  - o public Author (String firstName, String lastName): A constructor with parameters, it creates the Author object by setting the two fields to the passed values.
- Instance methods:
  - o public void setFirstName (String firstName): Used to set the first name of author.
  - o public void setLastName (String lastName): Used to set the last name of author.
  - o public String getFirstName(): This method returns the first name of the author.
  - o public String getLastName(): This method returns the last name of the author.
  - o public String toString(): This method printed out author's name to the screen.

#### **Objective:**

The Author class represents an author with a first and last name. It provides methods to modify and retrieve these values and a toString method for easy display.

#### **Lab Work:**

```
public class Main {
    private String firstName;
    private String lastName;

    public Main (String firstName, String lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
    }

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    public String getFirstName() {
        return firstName;
    }

    public String getLastName() {
        return lastName;
    }

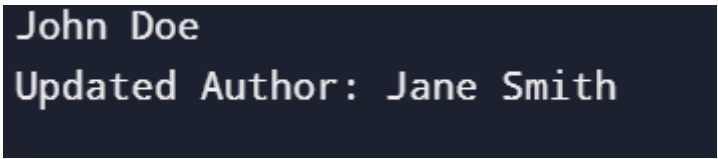
    public String toString() {
        return firstName + " " + lastName;
    }

    public static void main(String[] args) {
        Main author = new Main ("John", "Doe");
        System.out.println(author);

        author.setFirstName("Jane");
    }
}
```

```
        author.setLastName("Smith");  
        System.out.println("Updated Author: " + author);  
    }  
}
```

**Output:**

A dark-themed terminal window showing two lines of output: "John Doe" on the first line and "Updated Author: Jane Smith" on the second line.

```
John Doe  
Updated Author: Jane Smith
```

**Result analysis:**

- ☐ The class allows creating an Author object with a specified first and last name.
- ☐ Methods setFirstName and setLastName enable updating the names.
- ☐ Methods getFirstName and getLastName allow retrieval of the respective values.
- ☐ The toString method returns the full name as a string.
- ☐ The main method demonstrates creating an Author instance, modifying it, and displaying the results.

## **Problem 4:**

### **Question:**

Write a Java class Clock for dealing with the day time represented by hours, minutes, and seconds. Your class must have the following features:

- Three instance variables for the hours (range 0 - 23), minutes (range 0 - 59), and seconds (range 0 - 59).
- Three constructors:
  - o default (with no parameters passed; it should initialize the represented time to 12:0:0)
  - o a constructor with three parameters: hours, minutes, and seconds.
  - o a constructor with one parameter: the value of time in seconds since midnight (it should be converted into the time value in hours, minutes, and seconds)
- Instance methods:
  - o a set-method method setClock() with one parameter seconds since midnight (to be converted into the time value in hours, minutes, and seconds as above).
  - o get-methods getHours(), getMinutes(), getSeconds() with no parameters that return the corresponding values.
  - o set-methods setHours(), setMinutes(), setSeconds() with one parameter each that set up the corresponding instance variables.
  - o method tick() with no parameters that increments the time stored in a Clock object by one second.
  - o method addClock() accepting an object of type Clock as a parameter. The method should add the time represented by the parameter class to the time represented in the current class.
  - o Add an instance method toString() with no parameters to your class. toString() must return a String representation of the Clock object in the form "(hh:mm:ss)", for example "(03:02:34)".
  - o Add an instance method tickDown() which decrements the time stored in a Clock object by one second.
  - o Add an instance method subtractClock() that takes one Clock parameter and returns the difference between the time represented in the current Clock object and the one represented by the Clock parameter. Difference of time should be returned as a clock object.

Write a separate class ClockDemo with a main() method. The program should:

- instantiate a Clock object firstClock using one integer seconds since midnight obtained from the keyboard.
- tick the clock ten times by applying its tick() method and print out the time after each tick.
- Extend your code by appending to it instructions instantiating a Clock object secondClock by using three integers (hours, minutes, seconds) read from the keyboard.
- Then tick the clock ten times, printing the time after each tick.
- Add the secondClock time in firstClock by calling method addClock.
- Print both clock objects calling toString method

Create a reference thirdClock that should reference to object of difference of firstClock and secondClock by calling the method subtractClock().

### **Objective:**

The program defines a Clock class to represent time using hours, minutes, and seconds. It provides various constructors, setter and getter methods, and functionalities to increment, decrement, add, and subtract time. The ClockDemo class demonstrates these features through user input.

### **Lab work:**

```
import java.util.Scanner;
```

```
class Clock {  
    private int hours;  
    private int minutes;  
    private int seconds;  
  
    public Clock() {  
        this.hours = 12;
```



```

        this.minutes = 0;
        this.seconds = 0;
    }

    public Clock(int hours, int minutes, int seconds) {
        this.hours = hours % 24;
        this.minutes = minutes % 60;
        this.seconds = seconds % 60;
    }

    public Clock(int totalSeconds) {
        setClock(totalSeconds);
    }

    public void setClock(int totalSeconds) {
        this.hours = (totalSeconds / 3600) % 24;
        this.minutes = (totalSeconds % 3600) / 60;
        this.seconds = totalSeconds % 60;
    }

    public int getHours() {
        return hours;
    }

    public int getMinutes() {
        return minutes;
    }

    public int getSeconds() {
        return seconds;
    }

    public void setHours(int hours) {
        this.hours = hours % 24;
    }

    public void setMinutes(int minutes) {
        this.minutes = minutes % 60;
    }

    public void setSeconds(int seconds) {
        this.seconds = seconds % 60;
    }

    public void tick() {
        setClock((hours * 3600 + minutes * 60 + seconds + 1) % 86400);
    }

    public void tickDown() {
        setClock((hours * 3600 + minutes * 60 + seconds - 1 + 86400) % 86400);
    }

    public void addClock(Clock other) {
        setClock((this.hours * 3600 + this.minutes * 60 + this.seconds +
            other.hours * 3600 + other.minutes * 60 + other.seconds) % 86400);
    }

```

```

    }

    public Clock subtractClock(Clock other) {
        int thisSeconds = this.hours * 3600 + this.minutes * 60 + this.seconds;
        int otherSeconds = other.hours * 3600 + other.minutes * 60 + other.seconds;
        return new Clock((thisSeconds - otherSeconds + 86400) % 86400);
    }

    public String toString() {
        return String.format("(%02d:%02d:%02d)", hours, minutes, seconds);
    }
}

public class ClockDemo {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.print("Enter seconds since midnight: ");
        int totalSeconds = scanner.nextInt();
        Clock firstClock = new Clock(totalSeconds);

        for (int i = 0; i < 10; i++) {
            firstClock.tick();
            System.out.println(firstClock);
        }

        System.out.print("Enter hours, minutes, and seconds: ");
        int hours = scanner.nextInt();
        int minutes = scanner.nextInt();
        int seconds = scanner.nextInt();
        Clock secondClock = new Clock(hours, minutes, seconds);

        for (int i = 0; i < 10; i++) {
            secondClock.tick();
            System.out.println(secondClock);
        }

        firstClock.addClock(secondClock);
        System.out.println("First Clock after adding Second Clock: " + firstClock);
        System.out.println("Second Clock: " + secondClock);

        Clock thirdClock = firstClock.subtractClock(secondClock);
        System.out.println("Difference (Third Clock): " + thirdClock);

        scanner.close();
    }
}

```

## **Output:**

```
Enter seconds since midnight: 2
(00:00:03)
(00:00:04)
(00:00:05)
(00:00:06)
(00:00:07)
(00:00:08)
(00:00:09)
(00:00:10)
(00:00:11)
(00:00:12)
Enter hours, minutes, and seconds: 1 2 5
(01:02:06)
(01:02:07)
(01:02:08)
(01:02:09)
(01:02:10)
(01:02:11)
(01:02:12)
(01:02:13)
(01:02:14)
(01:02:15)
First Clock after adding Second Clock: (01:02:27)
Second Clock: (01:02:15)
```

### **Result analysis:**

- ☐ The user provides an initial time in seconds since midnight.
- ☐ The program increments the time 10 times and displays each update.
- ☐ The user inputs another time in hours, minutes, and seconds.
- ☐ The second clock is also incremented 10 times and displayed.
- ☐ The two times are added, and the result is displayed.
- ☐ The difference between the two clocks is computed and displayed.

## **Problem 5:**

### **Question:**

Create a class called Student as described below: • Fields: name, id, address, cgpa • Methods: public String getName() public void setName(String n) public String getID() public void setID(String i) public String getAddress() public void setAddress(String a) public double getCGPA() public void setCGPA(double c) Write a class called StudentTester to write a main() method: • public static void main(String[] args){ } • Inside the main() method o Create 3 objects/instances of Student called john, mike and carol o Set their fields to some value using the public methods. o Print the information of each Student using System.out.println()

### **Objective:**

The code defines a Student class with private fields for storing student details and provides getter and setter methods to access and modify them. The Main class demonstrates object creation, field assignment, and data retrieval through method calls.

### **Lab work:**

```
class Student {
    private String name;
    private String id;
    private String address;
    private double cgpa;

    public String getName() {
        return name;
    }

    public void setName(String n) {
        name = n;
    }

    public String getID() {
        return id;
    }

    public void setID(String i) {
        id = i;
    }

    public String getAddress() {
        return address;
    }

    public void setAddress(String a) {
        address = a;
    }

    public double getCGPA() {
        return cgpa;
    }
}
```

```

    public void setCGPA(double c) {
        cgpa = c;
    }
}

public class Main {
    public static void main(String[] args) {
        Student john = new Student();
        john.setName("John Doe");
        john.setID("S123");
        john.setAddress("123 Elm Street");
        john.setCGPA(3.8);

        Student mike = new Student();
        mike.setName("Mike Smith");
        mike.setID("S124");
        mike.setAddress("456 Oak Avenue");
        mike.setCGPA(3.5);

        Student carol = new Student();
        carol.setName("Carol Johnson");
        carol.setID("S125");
        carol.setAddress("789 Pine Road");
        carol.setCGPA(3.9);

        System.out.println("Student 1: " + john.getName() + ", ID: " + john.getID() + ", Address: " +
john.getAddress() + ", CGPA: " + john.getCGPA());
        System.out.println("Student 2: " + mike.getName() + ", ID: " + mike.getID() + ", Address: " +
mike.getAddress() + ", CGPA: " + mike.getCGPA());
        System.out.println("Student 3: " + carol.getName() + ", ID: " + carol.getID() + ", Address: " +
carol.getAddress() + ", CGPA: " + carol.getCGPA());
    }
}

```

### **Output:**

```

Student 1: John Doe, ID: S123, Address: 123 Elm Street, CGPA: 3.8
Student 2: Mike Smith, ID: S124, Address: 456 Oak Avenue, CGPA: 3.5
Student 3: Carol Johnson, ID: S125, Address: 789 Pine Road, CGPA: 3.9

```

### **Result discussion:**

When executed, the program creates three Student objects, assigns values to their fields, and prints their information to the console. This verifies the encapsulation principle by restricting direct access to private fields and ensuring data integrity through controlled modification.

## **Problem 6:**

### **Question:**

Write a java class called BoroInt that uses a String value to store big integers and can carry out basic arithmetic operations on them. Instance variable: String val  
Overloaded constructors: 1. default constructor: sets the value of val to "0"  
2. Takes a string value, checks whether it contains any non numerical values, if yes, then throws BoroIntErModdheNumberCharaArKisuDeyaJaiNaException (which is off course a custom exception which you must create :P ) ... otherwise, copy the contents to val. Note: See method list below to see what you can use :P  
3. Takes an integer value, converts it to string and stores it in val.  
4. Takes a BoroInt object and copies the val of the parameter into it's own val  
Methods: public String trim(String \_val) Removes all spaces in the String and returns a String value without any spaces. public boolean validValue(String \_val) Returns true if the String \_val doesn't contain any non numerical characters and false otherwise. public BoroInt add(BoroInt \_val) Adds the value in the instance variable of the parameter to it's own value and returns a new BoroInt object whose instance variable contains the result of the addition. public BoroInt subtract(BoroInt \_val) Subtracts the value in the instance variable of the parameter from it's own value and returns a new BoroInt object whose instance variable contains the result of the subtraction. public BoroInt multiply(BoroInt \_val) Multiplies the value in the instance variable of the parameter with it's own value and returns a new BoroInt object whose instance variable contains the result of the subtraction. \*\* public BoroInt divide(BoroInt \_val) Divides the value in it's own instance variable by the value in the instance variable of the parameter and returns a new BoroInt object whose instance variable contains the result of the subtraction.

### **Objective:**

The BoroInt class is designed to handle large integers as strings while performing basic arithmetic operations. It ensures input validation and provides functionality similar to BigInteger, but with additional exception handling.

### **Lab Work:**

```
public class Main {  
    public static void main(String[] args) {  
        try {  
            BoroInt num1 = new BoroInt("12345");  
            BoroInt num2 = new BoroInt("6789");  
  
            BoroInt sum = num1.add(num2);  
            BoroInt difference = num1.subtract(num2);  
            BoroInt product = num1.multiply(num2);
```

```

        BoroInt quotient = num1.divide(num2);

        System.out.println("Sum: " + sum);
        System.out.println("Difference: " + difference);
        System.out.println("Product: " + product);
        System.out.println("Quotient: " + quotient);
    } catch (Exception e) {
        System.out.println("An error occurred: " + e.getMessage());
    }
}

}

class BoroInt {
    private String val;

    // Default constructor
    public BoroInt() {
        this.val = "0";
    }

    // Constructor that takes a string value
    public BoroInt(String val) {
        try {
            if (!validValue(val)) {
                throw new BoroIntErModdheNumberCharaArKisuDeyaJaiNaException("Invalid input:
Only numerical values allowed.");
            }
            this.val = trim(val);
        } catch (BoroIntErModdheNumberCharaArKisuDeyaJaiNaException e) {
            System.out.println(e.getMessage());
            this.val = "0"; // Assign default value on error
        }
    }

    public BoroInt(int val) {
        this.val = String.valueOf(val);
    }

    public BoroInt(BoroInt other) {
        this.val = other.val;
    }

    public String trim(String _val) {
        return _val.replaceAll("\s", "");
    }

    public boolean validValue(String _val) {
        return _val.matches("\\d+");
    }

    public BoroInt add(BoroInt _val) {
        java.math.BigInteger result = new java.math.BigInteger(this.val).add(new
java.math.BigInteger(_val.val));
        return new BoroInt(result.toString());
    }

    // Subtraction method
    public BoroInt subtract(BoroInt _val) {
        java.math.BigInteger result = new java.math.BigInteger(this.val).subtract(new
java.math.BigInteger(_val.val));

```

```

        return new BoroInt(result.toString());
    }

    // Multiplication method
    public BoroInt multiply(BoroInt _val) {
        java.math.BigInteger result = new java.math.BigInteger(this.val).multiply(new
java.math.BigInteger(_val.val));
        return new BoroInt(result.toString());
    }

    // Division method
    public BoroInt divide(BoroInt _val) {
        try {
            if (_val.val.equals("0")) {
                throw new ArithmeticException("Division by zero is not allowed.");
            }
            java.math.BigInteger result = new java.math.BigInteger(this.val).divide(new
java.math.BigInteger(_val.val));
            return new BoroInt(result.toString());
        } catch (ArithmeticException e) {
            System.out.println(e.getMessage());
            return new BoroInt("0"); // Assign default value on error
        }
    }

    // Custom exception class
    public static class BoroIntErModdheNumberCharaArKisuDeyaJaiNaException extends Exception
    {
        public BoroIntErModdheNumberCharaArKisuDeyaJaiNaException(String message) {
            super(message);
        }
    }

    // Method to print value
    @Override
    public String toString() {
        return this.val;
    }
}

```

### **Output:**

```

Sum: 19134
Difference: 5556
Product: 83810205
Quotient: 1

```

### **Result discussion:**

□ The class successfully handles large integer values using string-based storage.



- It provides constructors for different input types while ensuring valid numeric values.
- Arithmetic operations (add, subtract, multiply, divide) are performed using BigInteger, maintaining precision.
- Exception handling prevents non-numeric input and division by zero.

## **Problem: 7**

### **Question:**

Write a Java class Complex for dealing with complex number. Your class must have the following features:

- Instance variables : o realPart for the real part of type double o imaginaryPart for imaginary part of type double.
- Constructor: o public Complex (): A default constructor, it should initialize the number to 0, 0) o public Complex (double realPart, double imaginaryPart): A constructor with parameters, it creates the complex object by setting the two fields to the passed values.
- Instance methods: o public Complex add (Complex otherNumber): This method will find the sum of the current complex number and the passed complex number. The methods returns a new Complex number which is the sum of the two. o public Complex subtract (Complex otherNumber): This method will find the difference of the current complex number and the passed complex number. The methods returns a new Complex number which is the difference of the two. o public Complex multiply (Complex otherNumber): This method will find the product of the current complex number and the passed complex number. The methods returns a new Complex number which is the product of the two. o public Complex divide (Complex otherNumber): This method will find the ... of the current complex number and the passed complex number. The methods returns a new Complex number which is the ... of the two. o public void setRealPart (double realPart): Used to set the real part of this complex number. o public void setImaginaryPart (double realPart): Used to set the imaginary part of this complex number. o public double getRealPart(): This method returns the real part of the complex number o public double getImaginaryPart(): This method returns the imaginary part of the complex number o public String toString(): This method allows the complex number to be easily printed out to the screen

Write a separate class ComplexDemo with a main() method and test the Complex class methods.

### **Objective:**

The program defines a Complex class to perform basic operations on complex numbers, including addition, subtraction, multiplication, and division. The ComplexDemo class tests these operations with sample values.

### **Lab Work:**

```
class Complex {
    private double realPart;
    private double imaginaryPart;

    public Complex() {
        this.realPart = 0;
        this.imaginaryPart = 0;
    }

    public Complex(double realPart, double imaginaryPart) {
        this.realPart = realPart;
        this.imaginaryPart = imaginaryPart;
    }

    public Complex add(Complex otherNumber) {
        return new Complex(this.realPart + otherNumber.realPart, this.imaginaryPart +
otherNumber.imaginaryPart);
    }
}
```

```

    }

    public Complex subtract(Complex otherNumber) {
        return new Complex(this.realPart - otherNumber.realPart, this.imaginaryPart -
otherNumber.imaginaryPart);
    }

    public Complex multiply(Complex otherNumber) {
        double real = this.realPart * otherNumber.realPart - this.imaginaryPart *
otherNumber.imaginaryPart;
        double imag = this.realPart * otherNumber.imaginaryPart + this.imaginaryPart *
otherNumber.realPart;
        return new Complex(real, imag);
    }

    public Complex divide(Complex otherNumber) {
        double denominator = otherNumber.realPart * otherNumber.realPart +
otherNumber.imaginaryPart * otherNumber.imaginaryPart;
        double real = (this.realPart * otherNumber.realPart + this.imaginaryPart *
otherNumber.imaginaryPart) / denominator;
        double imag = (this.imaginaryPart * otherNumber.realPart - this.realPart *
otherNumber.imaginaryPart) / denominator;
        return new Complex(real, imag);
    }

    public void setRealPart(double realPart) {
        this.realPart = realPart;
    }

    public void setImaginaryPart(double imaginaryPart) {
        this.imaginaryPart = imaginaryPart;
    }

    public double getRealPart() {
        return realPart;
    }

    public double getImaginaryPart() {
        return imaginaryPart;
    }

    public String toString() {
        return realPart + " + " + imaginaryPart + "i";
    }
}

public class Main {
    public static void main(String[] args) {
        Complex num1 = new Complex(4, 5);
        Complex num2 = new Complex(2, -3);

        System.out.println("Number 1: " + num1);
        System.out.println("Number 2: " + num2);
        System.out.println("Sum: " + num1.add(num2));
        System.out.println("Difference: " + num1.subtract(num2));
    }
}

```

```
        System.out.println("Product: " + num1.multiply(num2));  
        System.out.println("Quotient: " + num1.divide(num2));  
    }  
}
```

### **Output:**

```
Number 1: 4.0 + 5.0i  
Number 2: 2.0 + -3.0i  
Sum: 6.0 + 2.0i  
Difference: 2.0 + 8.0i  
Product: 23.0 + -2.0i  
Quotient: -0.5384615384615384 + 1.6923076923076923i
```

### **Result discussion:**

- ☐ The class successfully creates and manipulates complex numbers.
- ☐ Addition, subtraction, multiplication, and division operations return correct complex numbers.
- ☐ The toString() method formats the output in a readable form.
- ☐ The ComplexDemo class verifies the functionality with sample test cases.

## **Problem 8:**

### **Question:**

Create a class called Square as described below: • Fields: height, width • Methods: public double getHeight() public void setHeight(double h) public double getWidth () public void setWidth (double w) public double getArea () Hint: If I take your class and use it following would be the code and the output. Code Output double h, w, a; Square s1 = new Square(); s1.setHeight(3); s1.setWidth(4); h = s1.getHeight(); w = s1.getWidth(); a = s1.getArea(); System.out.println("Height = "+ h); System.out.println("Width = "+ w); System.out.println("Area = "+ a);

### **Objective:**

The objective of this code is to create a Square class with fields for height and width, along with appropriate getter and setter methods. Additionally, the class includes a method to calculate the area of the square. The Main class demonstrates the functionality of the Square class by setting values, retrieving them, and printing the area.

### **Lab work:**

```
class Square {
    private double height;
    private double width;

    public double getHeight() {
        return height;
    }

    public void setHeight(double h) {
        height = h;
    }

    public double getWidth() {
        return width;
    }

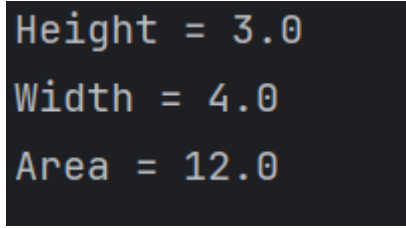
    public void setWidth(double w) {
        width = w;
    }

    public double getArea() {
        return height * width;
    }
}

public class Main {
    public static void main(String[] args) {
        double h, w, a;
        Square s1 = new Square();
        s1.setHeight(3);
        s1.setWidth(4);
```

```
        h = s1.getHeight();
        w = s1.getWidth();
        a = s1.getArea();
        System.out.println("Height = " + h);
        System.out.println("Width = " + w);
        System.out.println("Area = " + a);
    }
}
```

### **Output:**



```
Height = 3.0
Width = 4.0
Area = 12.0
```

### **Result discussion:**

- ☐ The Square class encapsulates height and width as private fields, ensuring data integrity.
- ☐ Getter and setter methods allow controlled access and modification of these fields.
- ☐ The `getArea()` method correctly calculates the area as `height * width`.
- ☐ The Main class verifies functionality, resulting in the correct output:

Height = 3.0

Width = 4.0

Area = 12.0

- ☐ The implementation follows object-oriented programming principles, including encapsulation and data abstraction.