Lab 04
Passing and returning non primitive values from methods

Objective:

The objective of this lab is to teach the students, how the objects can be passed to and returned from the functions.

Activity Outcomes:

After completion of this Lab students will be able to :

- pass objects to methods
- return objects from methods

Instructor Note:

As pre-lab activity, read Chapter 10 from the text book “Introduction to Java Programming”, Y. Daniel Liang, Pearson, 2019.

1) Useful Concepts

In Java, all primitives are passed by value. This means a copy of the value is passed into the method. Objects can be passed natively, just like primitives. It is often misstated that Object parameters are passed by Reference. While it is true that the parameter is a reference to an Object, the reference itself is passed by Value.

Syntax for passing objects to method:

```
     public      void      MethodName      (reference_variable_type
reference_variable) {

     ......

     }
```

Syntax for returning objects from method:

```
     public          reference_variable_type          MethodName
     (reference_variable_type reference_variable) {

          ......

          return reference_variable;

          }
```

2) Solved Lab Activities

Sr.No	*Allocated Time*	*Level of Complexity*	*CLO Mapping*
Activity 1	***10 mins***	***Medium***	***CLO-4***
Activity 2	***20 mins***	***Medium***	***CLO-4***
Activity 3	***20 mins***	***Medium***	***CLO-4***

Activity 1:

Passing object as parameter and change value of its data member.

Solution:

```
class ObjectPass {

    public int value;
    public static void
increment(ObjectPass a) {
        a.value++;
    }
}

public class ObjectPassTest {

    public static void main(String[]
args) {
        ObjectPass p = new
ObjectPass();
        p.value = 5;
        System.out.println("Before
calling: " + p.value); // output is 5
        ObjectPass.increment(p);
        System.out.println("After
calling: " + p.value); // output is 6
    }
}
```

Now it is like the pass was by reference! but the thing is what we pass exactly is a handle of an object, and in the called method a new handle created and pointed to the same object. Now when more than one handles tied to the same object, it is known as **aliasing**. This is the default way Java does when passing the handle to a called method, create alias.

Activity 2*:*

The following activity demonstrates the creation of a method that accepts and returns object.

Solution:

```

class Complex {

    private double real;
    private double imag;

    public Complex() {
        real = 0.0;
        imag = 0.0;
    }

    public Complex(double r, double im) {
        real = r;
        imag = im;
    }

    public Complex Add(Complex b) {
        Complex c_new = new Complex(real +
b.real, imag + b.imag);
        return c_new;
    }

    public void Show() {
        System.out.println(real + imag);
    }
}

public class ComplexTest {

    public static void main(String args[]) {
        Complex C1 = new Complex(11, 2.3);
        Complex C2 = new Complex(9, 2.3);
        Complex C3 = new Complex();
        C3 = C1.Add(C2);
        C3.Show();
    }
}
```

Activity 3:

The following activity demonstrates the creation of a method that accepts two objects.

Solution:

```
class Point {

    private int X;
    private int Y;
    public Point() {
        X = 5;
        Y = 6;
    }

    public Point(int a, int c) {
        X = a;
        Y = c;
    }

    public void setX(int a) {
        X = a;
    }

    public void setY(int c) {
        Y = c;
    }

    public int getX() {
        return X;
    }

    public int getY() {
        return Y;
    }

    public Point Add(Point Pa, Point Pb) {
        Point p_new = new Point(X + Pa.X + Pb.X,
Y + Pa.Y + Pb.Y);
        return p_new;
    }

    public void display() {
        System.out.println(X);
        System.out.println(Y);
    }
}

public class PointTest {
    public static void main(String[] args) {
        Point p1 = new Point(10, 20);
        Point p2 = new Point(30, 40);
        Point p3 = new Point();
        Point p4 = p1.Add(p2, p3);
        p4.display();
    }
}
```

3) Graded Lab Tasks

Note: The instructor can design graded lab activities according to the level of difficulty and complexity of the solved lab activities. The lab tasks assigned by the instructor should be evaluated in the same lab.

Lab Task 1

Create a class ― Distance‖ with two constructors (no argument, and two argument), two data members (feet and inches) . Create setter, getter and display method. Create a method that adds two Distance Objects and returns the added Distance Object.

Lab Task 2

Create an Encapsulated class Book. Its data members are

- *author (String)*
- *chapterNames[100] (String[])*

Create two overloaded constructors, one with no argument and one with two arguments.

Create a method compareBooks that compares the author of two Books and returns true if both books have same author and false otherwise. (This method must manipulate two Book objects)

Create a method compareChapterNames that compares the chapter names of two Books and returns the book with larger chapters. Display the author of the book with greater chapters in main.

Create a runner class that declares two objects of type Book. One object should be declared using no argument constructor and then the parameters should be set through the set() methods. The second object should be declared with argument constructor. Finally the CompareBooks()and compareChapterNames method should be called and the result should be displayed in the runner class.

Lab Task 3

Define a class called Fraction. This class is used to represent a ratio of two integers. Create two constructors, set, get and display function. Include an additional method, ***equals****, that takes as input another Fraction and returns true if the two fractions are identical and false if they are not.*