Dr. B R Ambedkar National Institute of Technology Jalandhar

Java Lab File

ITPC - 226

Submitted To: Ms. Vanitha P S

Submitted By:

Name: Shreya Jathar

Roll No: 20124094

Branch: Information Technology (G4)

Lab No	*Programme Title*	*Date of Implementation*	*Remark*
1	Program to find total, average of given two numbers by using function with default arguments, static data members and this keyword	16-01-2022	
2	Program to illustrate class and objects	24-01-2022	
3	Program to illustrate constructors	30-01-2022	
4	Program to create a class complex with necessary operator overloading and type conversion	08-02-2022	
5	Program that randomly generates complex numbers and performs operations (+, -, *, /) on them	11-02-2022	
6	Program to illustrate inheritance in Java	22-02-2022	
7	Program to demonstrate the concept of polymorphism	24-02-2022	
8	Program to illustrate method overriding	24-02-2022	
9	Programs to illustrate the following classes	17-03-2022	
	Mouse Adapter		
	Key Adapter		
10	Program to illustrate Array Manipulation	01-04-2022	
11	Program to illustrate Layout Manager	13-04-2022	

LAB 1

Program to find total, average of given two numbers by using function with default arguments, static data members and this keyword

LAB 2

Program to illustrate class and objects

Definitions

- Import – Keyword used when importing a package. When we import a package we can use the classes and methods present in that package as a part of our program.

Syntax : import package_name

- Scanner class – Used to get user input. It is found in the java.util package. We need to create an object of scanner class to call the input methods

Syntax : Scanner object_name = new Scanner(System.in)

Where System.in means we are taking input from the standard console

- Class – A class is like a blueprint for an object. It shows us the properties and methods that all objects can have and use. The class keyword is used to define a class.

Syntax : class class_name { //properties and methods }

By convention, class name usually starts with a capital letter

- Float – It is a primitive datatype. Decimal numbers are called floating point 'float'.

Syntax : float variable_name_1, ….., variable_name_n;

- Public – If a method or property is defined as public, it can be accessed outside of the class.
- This – It is a keyword used to refer to the current instance of the class. It can also be used to invoke the constructor of the class.
- Static – A static method can be called without the creation of an object.
- Main – Every program must have a main function. This is where the execution of the program begins.

Syntax : public static void main(String[] args) { //Code }

- New – The new keyword is used to create an object of a class.

Syntax : Class_name object_name = new Class_name(parameters);

Where parameters are used to initialize values of properties by using constructors. Parameters can be 0 or many.

- Println – It is the method used to output information onto the screen. It is a method of the print stream. The print stream is created by calling System.out

Syntax : System.out.println();

Code:

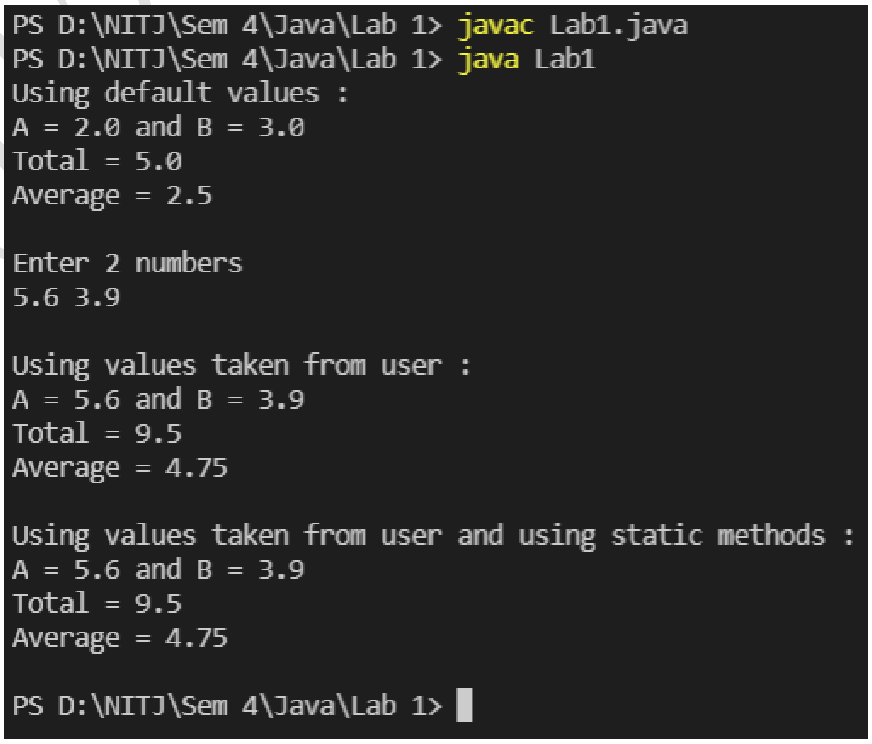
```
/* Program to find
   - Total
   - Average
   of 2 numbers using a function with
   - Default arguments
   - Static data members
   - this keyword */
```

```
import java.util.Scanner;

class Total {

    float A, B;

    public Total() {
        this.A = 2;
        this.B = 3;
    }

    public Total(float x, float y) {
        this.A = x;
        this.B = y;
    }

    float total() {
        return A + B;
    }

    float average() {
        return this.total() / 2;
    }

    //Static methods for the same functions
    public static float total(float A, float B) {
        return A + B;
    }

    public static float average(float A, float B) {
        return total(A, B) / 2;
    }
}

public class Lab1 {
    public static void main(String[] args) {

        try (Scanner sc = new Scanner(System.in)) {
            Total t1 = new Total();

            System.out.println("Using default values : ");
            System.out.println("A = " + t1.A + " and B = " + t1.B);
            System.out.println("Total = " + t1.total());
            System.out.println("Average = " + t1.average());
            System.out.println();

            System.out.println("Enter 2 numbers");
            float A = sc.nextFloat();
            float B = sc.nextFloat();
            System.out.println();

            Total t2 = new Total(A, B);
            System.out.println("Using values taken from user : ");
            System.out.println("A = " + t2.A + " and B = " + t2.B);
            System.out.println("Total = " + t2.total());
            System.out.println("Average = " + t2.average());
            System.out.println();

            System.out.println("Using values taken from user and using static methods : 
");
            System.out.println("A = " + A + " and B = " + B);
            System.out.println("Total = " + Total.total(A, B));
            System.out.println("Average = " + Total.average(A, B));
        }
        System.out.println();
    }
}
```

Explaination:

- In this program, we first create an object t1 of class Total using the default constructor. This assigns the values 2 and 3 to A and B respectively. The total method of class Total is called on this object t1, and the result is printed. Then, we call the average method on t1 and print the result.
- We then take 2 values A and B as input from the user. We create a new object t2 using the parameterized constructor which assigns the values of A and B as given by the user. Then we calculate and print total and average similar to t1.
- The use of static method is shown by calling the static methods total and average with A and B as parameters. This is done without the use of objects to show that we can use static methods without having to create any object.

Output:



```
PS D:\NITJ\Sem 4\Java\Lab 1> javac Lab1.java
PS D:\NITJ\Sem 4\Java\Lab 1> java Lab1
Using default values :
A = 2.0 and B = 3.0
Total = 5.0
Average = 2.5

Enter 2 numbers
5.6 3.9

Using values taken from user :
A = 5.6 and B = 3.9
Total = 9.5
Average = 4.75

Using values taken from user and using static methods :
A = 5.6 and B = 3.9
Total = 9.5
Average = 4.75

PS D:\NITJ\Sem 4\Java\Lab 1> 
```

LAB 3

Program to illustrate constructors

Definitions

Constructor – It is a method in a class which is called during the creation of the class object. The constructor is usually used to initialize values of properties. It can also be used to call methods or output some value during object creation. A constructor does not need to be called separately; it is called during object creation by the compiler. The name of the constructor must be the same as the class name, and the constructor has no return type. The constructor can not be called by us, it is called automatically only when the object is created.

Default constructor – The default constructor does not take any parameters. If we do not create our own constructor in the class, the default constructor is called automatically. We can overload the default constructor to initialize values.

Parameterized constructor – The parameterized constructor takes one or more parameters. These parameter values are used to assign our own values to the properties of an object. We can create multiple parameterized constructors using overloading, to suit our different needs.

Copy constructor – It is called when we try to create an object by taking values of an existing object. We need to create a copy constructor in our class before we use it.

Syntax : Class_name(Class_name instance_name) {

this.property_name = instance_name.property_name; }

Chaining of constructors – In constructor chaining, we call a constructor from within another constructor. We do this using the this keyword. We mostly use this concept in inheritance where we call a constructor of the superclass from the subclass. Which constructor is to be called using this keyword is determined by number and type of the parameters.

Code:

```
/* Program to illustrate constructors */

import java.util.Scanner;

class Shape {
    double side, height;

    public Shape() {
        this(4, 0);

        System.out.println("Chaining back...");
        System.out.println("Inside the default constructor");
    }

    public Shape(double S) {
        this(S, 0);
        System.out.println("Chaining back...");
        System.out.println("Inside the parameterized constructor");
        System.out.println("This constructor has 1 parameter");
    }

    public Shape(double S,  double H) {
        System.out.println("Inside the parameterized constructor");
        System.out.println("This constructor has 2 parameters");

        side = S;
        height = H;
    }

    Shape(Shape s) {
        System.out.println("Inside the copy constructor");

        this.side = s.side;
        this.height = s.height;
    }

    public void printArea() {
        if(this.height == 0) {
            System.out.println("The shape is a square");
            System.out.println("Area = " + side * side);
            System.out.println();
            return;
        }

        System.out.println("The shape is a cuboid");
        System.out.println("Volume = " + side * side * height);
        System.out.println();
    }
}

public class Lab3 {
    public static void main(String[] args) {

        try(Scanner sc = new Scanner(System.in)) {

            //Creating a shape using default constructor
            Shape S1 = new Shape();
            S1.printArea();

            //Creating a shape using parameterized constructor
            //Single parameter
            System.out.println("Enter value of side :");
            double side1 = sc.nextDouble();
            Shape S2 = new Shape(side1);
            S2.printArea();

            //Creating a shape using parameterized constructor
            //Two parameters
            System.out.println("Enter value of side and height:");
            double side2 = sc.nextDouble();
            double height2 = sc.nextDouble();
            Shape S3 = new Shape(side2, height2);
            S3.printArea();

            //Creating a shape using copy constructor
            //S4 is a copy of S3
            Shape S4 = new Shape(S3);
            S4.printArea();
        }
    }
}
```

Explanation:

- In this program, we have a class called shape. This class outputs the area of the shape if it is a square, or the volume if it is a cuboid. The cuboid will have height > 0 and square will have height = 0. By default, an object would be initialized as a square with side 4.
- The concept of constructor chaining has been used. The default constructor and the parameterized constructor with a single parameter call the constructor having 2 parameters to initialize the values of side and height. They both pass value of height as 0.
- First, we create an object S1 using the default constructor. The side is initialized to 4 and height to 0. Constructor with 2 parameters is called using constructor chaining. We can follow the function calls as below :

Shape S1 = new Shape(); -> call to the default constructor.

In the default constructor, we have the call this(4, 0); -> this will call constructor having 2 parameters.

The print statements are encountered and we get an output -> "Inside the parameterized constructor"

"This constructor has 2 parameters"

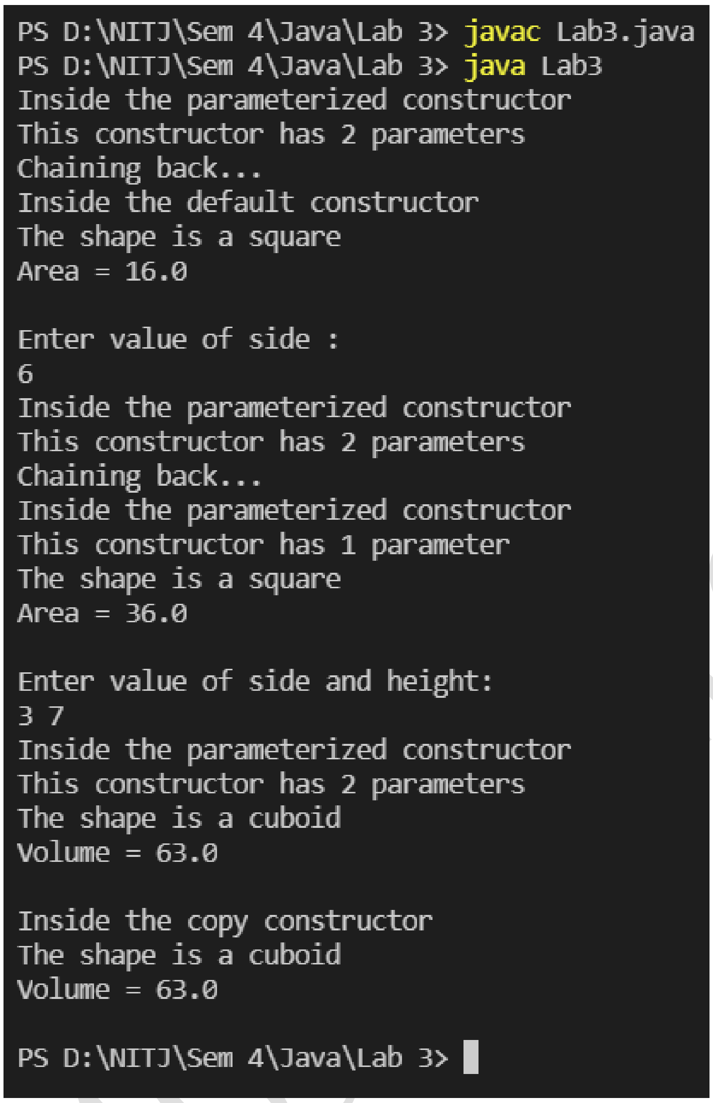
Then, values of side and height are initialized to 4 and 0 respectively.

Control is passed back to the default constructor, where we encounter more print statements that give us the output -> "Chaining back…"

"Inside the default constructor"

- We then call the printArea() method on object S1. First the method checks if it is a square or a cuboid. Since it is a square, the area is printed
- Next, we create an object S2 taking side input from the user. We initialise it using the parameterized constructor having one parameter. We have again used constructor chaining concept. printArea() method is called on S2 and since height is 0, it is a square. Area is printed.
- Object S3 is created as a cuboid by taking non zero height input from the user. If the user chooses to input height as 0, we will get a square again. The printArea function will print the volume of the cuboid S3.
- Object S4 is crested using the copy constructor. It is copied from object S3.

Output:



```
PS D:\NITJ\Sem 4\Java\Lab 3> javac Lab3.java
PS D:\NITJ\Sem 4\Java\Lab 3> java Lab3
Inside the parameterized constructor
This constructor has 2 parameters
Chaining back...
Inside the default constructor
The shape is a square
Area = 16.0

Enter value of side :
6
Inside the parameterized constructor
This constructor has 2 parameters
Chaining back...
Inside the parameterized constructor
This constructor has 1 parameter
The shape is a square
Area = 36.0

Enter value of side and height:
3 7
Inside the parameterized constructor
This constructor has 2 parameters
The shape is a cuboid
Volume = 63.0

Inside the copy constructor
The shape is a cuboid
Volume = 63.0

PS D:\NITJ\Sem 4\Java\Lab 3>
```

LAB 4

Program to create a class complex with necessary operator overloading and type conversion.

Definitions

Private – An access specifier. Member data and member functions specified as private can't be accessed by methods outside the class. They can however be accessed by member functions.

String – String is a datatype. It is a sequence of characters. The ".split" method on a string is used to split a string into parts based on a given regular expression.

parseDouble() – This method in the Double class is used to convert a string into type double.

Constructor – It is a method in a class which is called during the creation of the class object. The constructor is usually used to initialize values of properties. It can also be used to call methods or output some value during object creation. A constructor does not need to be called separately; it is called during object creation by the compiler. The name of the constructor must be the same as the class name, and the constructor has no return type. The constructor can not be called by us, it is called automatically only when the object is created.

Overloading – Overloading is the use of one method to perform different actions. The methods have different signatures (return type, number of parameters, data type of parameters) but the same name.

Operator Overloading – Using an elementary operator, but overloading it to make it behave differently is operator overloading. This concept can't be used in java. Instead, we can define our own methods to act like the operator we want.

Code:

```
/* Program to create complex class
   with operator overloading and type conversion */

import java.util.Scanner;
import java.util.regex.Pattern;

class Complex {
   private double Re, Im;

   public Complex() {
      this(0, 0);
   }

   public Complex(String S) {
      String[] arr = S.split(Pattern.quote("+i"));

      this.Re = Double.parseDouble(arr[0]);
      this.Im = Double.parseDouble(arr[1]);
   }

   public Complex (int Re, int Im) {
      this((double)Re, (double)Im);
   }

   public Complex (double Re, double Im) {
      this.Re = Re;
```

```
        this.Im = Im;
    }

    public void printComplex() {
        System.out.printf("%.2f + %.2fi\n", this.Re, this.Im);
    }

    public static void sum(Complex A, Complex B) {
        Complex C = new Complex();
        C.Re = A.Re + B.Re;
        C.Im = A.Im + B.Im;

        C.printComplex();
    }

    public static void difference(Complex A, Complex B) {
        Complex C = new Complex();
        C.Re = A.Re - B.Re;
        C.Im = A.Im - B.Im;

        C.printComplex();
    }

    public static void product(Complex A, double x) {

        Complex C = new Complex(A.Re * x, A.Im * x);
        C.printComplex();
    }

    public static void product(Complex A, Complex B) {
        Complex C = new Complex();
        C.Re = A.Re * B.Re - A.Im * B.Im;
        C.Im = A.Re * B.Im - B.Re * A.Im;

        C.printComplex();
    }

    public static void quotient(Complex A, double x) {

        Complex C = new Complex(A.Re / x, A.Im / x);
        C.printComplex();
    }

    public static void quotient(Complex A, Complex B) {
        Complex C = new Complex();
        C.Re = (A.Re * B.Re - A.Im * B.Im) / (B.Re * B.Re + B.Im * B.Im);
        C.Im = (A.Re * B.Im + A.Im * B.Re) / (B.Re * B.Re + B.Im * B.Im);

        C.printComplex();
    }
}
```

```
public class Lab4 {
   public static void main(String[] args) {

      try(Scanner sc = new Scanner(System.in)) {

         //Creating a complex number using integer parameters
         System.out.print("Integer values of Re and Im : ");
         int Re1 = sc.nextInt();
         int Im1 = sc.nextInt();

         Complex C1 = new Complex(Re1, Im1);

         //Creating a complex number using double parameters
         System.out.print("Double values of Re and Im : ");
         double Re2 = sc.nextDouble();
         double Im2 = sc.nextDouble();

         Complex C2 = new Complex(Re2, Im2);

         //Creating a complex number using a string
         System.out.print("Type a complex number : ");
         String complex = sc.next();
         Complex C3 = new Complex(complex);
         System.out.println();

         //Printing the 3 complex numbers
         System.out.print("C1 : ");
         C1.printComplex();
         System.out.print("C2 : ");
         C2.printComplex();
         System.out.print("C3 : ");
         C3.printComplex();
         System.out.println();

         //Performing basic operations
         System.out.println("Sum and difference of C1 and C2 :");
         Complex.sum(C1, C2);
         Complex.difference(C1, C2);
         System.out.println();

         System.out.println("Product of C3 and 2.5 :");
         Complex.product(C3, 2.5);
         System.out.println("Product of C1 and C3 :");
         Complex.product(C1, C3);
         System.out.println();

         System.out.println("Quotient of C2 divided by 2.5 :");
         Complex.quotient(C2, 2.5);
         System.out.println("Quotient of C3 divided by C2 :");
         Complex.quotient(C3, C2);
         System.out.println();
      }
```

```
    }
}
```

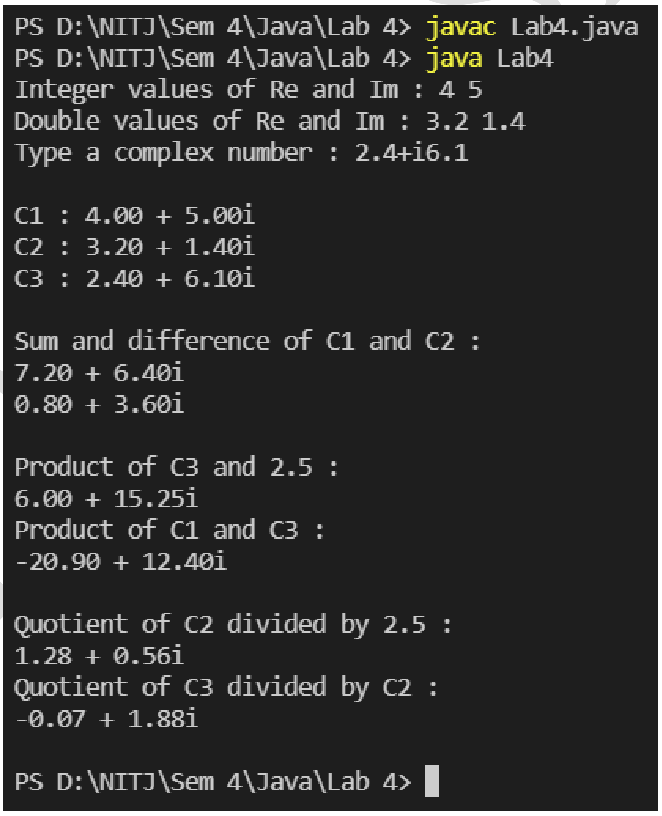
Explanation:

- In this program, we have a class called constructor. This class creates a constructor and contains methods to perform elementary operations '+', '-', 'x', '/' on the complex numbers.
- We can initialize the constructor by entering integer values or double values for the real and imaginary parts, or by typing the constructor as a string.
- We can then perform operations on 2 constructors, or on a constructor and an integer/double.
- The methods for '+', '-', 'x', '/' have been declared as static. This means we don't need to initialize and object to call them. They can be called as

  ClassName.StaticMethod();

- We also have a printConstructor method to print the value of a constructor as a string.

Output:



```
PS D:\NITJ\Sem 4\Java\Lab 4> javac Lab4.java
PS D:\NITJ\Sem 4\Java\Lab 4> java Lab4
Integer values of Re and Im : 4 5
Double values of Re and Im : 3.2 1.4
Type a complex number : 2.4+i6.1

C1 : 4.00 + 5.00i
C2 : 3.20 + 1.40i
C3 : 2.40 + 6.10i

Sum and difference of C1 and C2 :
7.20 + 6.40i
0.80 + 3.60i

Product of C3 and 2.5 :
6.00 + 15.25i
Product of C1 and C3 :
-20.90 + 12.40i

Quotient of C2 divided by 2.5 :
1.28 + 0.56i
Quotient of C3 divided by C2 :
-0.07 + 1.88i

PS D:\NITJ\Sem 4\Java\Lab 4>
```

LAB 5

Program that randomly generates complex numbers and performs operations (+, -, *, /) on them.

Definitions

java.util.Random – It is a library that contains methods to generate random integers, doubles, etc. We use it by initializing an object of type random

Random rand = new Random();

We can then use this object to generate random numbers by calling methods of the Random class.

rand.nextDouble() – This method generates a random number. We specify the range by passing it as a parameter. For instance if we want a random number < 100, we call

rand.nextDouble(100);

We can assign this to a double variable and use it in the future.

Code

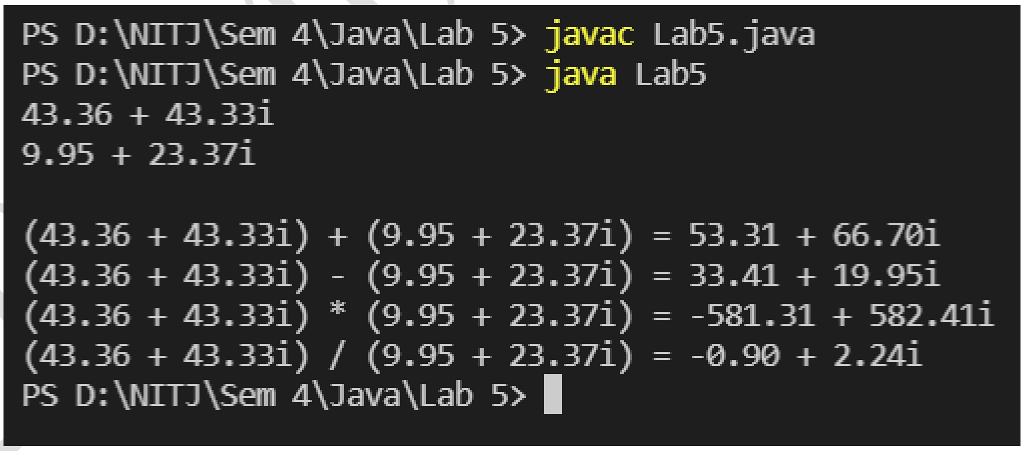
```
/* Program to randomly generate
   complex numbers and perform
   operations on them */

import java.util.Random;

class Complex {
    private double Re, Im;
    Random rand = new Random();

    public Complex() {
        Re = rand.nextDouble(100);
        Im = rand.nextDouble(100);
    }

    public void printComplex() {
        System.out.printf("%.2f + %.2fi", this.Re, this.Im);
    }

    public static void sum(Complex A, Complex B) {
        Complex C = new Complex();
        C.Re = A.Re + B.Re;
        C.Im = A.Im + B.Im;

        System.out.print("(");
        A.printComplex();
        System.out.print(") + (");
        B.printComplex();
        System.out.print(") = ");

        C.printComplex();
    }

    public static void difference(Complex A, Complex B) {
        Complex C = new Complex();
        C.Re = A.Re - B.Re;
        C.Im = A.Im - B.Im;

        System.out.print("(");
        A.printComplex();
        System.out.print(") - (");
        B.printComplex();
        System.out.print(") = ");

        C.printComplex();
    }

    public static void product(Complex A, Complex B) {
        Complex C = new Complex();
        C.Re = A.Re * B.Re - A.Im * B.Im;
        C.Im = A.Re * B.Im - B.Re * A.Im;

        System.out.print("(");
        A.printComplex();
        System.out.print(") * (");
        B.printComplex();
        System.out.print(") = ");

        C.printComplex();
    }

    public static void quotient(Complex A, Complex B) {
        Complex C = new Complex();
        C.Re = (A.Re * B.Re - A.Im * B.Im) / (B.Re * B.Re + B.Im * B.Im);
        C.Im = (A.Re * B.Im + A.Im * B.Re) / (B.Re * B.Re + B.Im * B.Im);

        System.out.print("(");
        A.printComplex();
        System.out.print(") / (");
        B.printComplex();
        System.out.print(") = ");

        C.printComplex();
    }
}

public class Lab5 {
    public static void main(String[] args) {

        //Creating 2 random complex numbers
        Complex C1 = new Complex();
        C1.printComplex();
        System.out.println();

        Complex C2 = new Complex();
        C2.printComplex();
        System.out.println();
        System.out.println();

        //Performing operations on them
        Complex.sum(C1, C2);
        System.out.println();

        Complex.difference(C1, C2);
        System.out.println();

        Complex.product(C1, C2);
        System.out.println();

        Complex.quotient(C1, C2);
        System.out.println();
    }
}
```

Explanation

In this program, we create a class Complex. When we create an object of this class, the constructor is called, and it generates a random complex number. 2 double values are generated which form the real and imaginary parts of our complex number.

We can perform the +, -, *, / operations on our complex number using the methods in the complex class. They print the 2 complex numbers being operated on, separated by the appropriate operator, and then print the result.

Output



```
PS D:\NITJ\Sem 4\Java\Lab 5> javac Lab5.java
PS D:\NITJ\Sem 4\Java\Lab 5> java Lab5
43.36 + 43.33i
9.95 + 23.37i

(43.36 + 43.33i) + (9.95 + 23.37i) = 53.31 + 66.70i
(43.36 + 43.33i) - (9.95 + 23.37i) = 33.41 + 19.95i
(43.36 + 43.33i) * (9.95 + 23.37i) = -581.31 + 582.41i
(43.36 + 43.33i) / (9.95 + 23.37i) = -0.90 + 2.24i
PS D:\NITJ\Sem 4\Java\Lab 5>
```

LAB 6

Program to illustrate inheritance in Java

Definitions

Inheritance – It is one of the main features of OOPs. In inheritance, one class can "inherit" properties (member data and methods) from another class. The class that inherits the properties is the child class(sub-class). The class from which the child inherits properties is the parent class (superclass). In this way, our classes can reuse methods and data from existing classes.

extends – The keyword extends is used to denote when a class is a sub-class. The syntax is

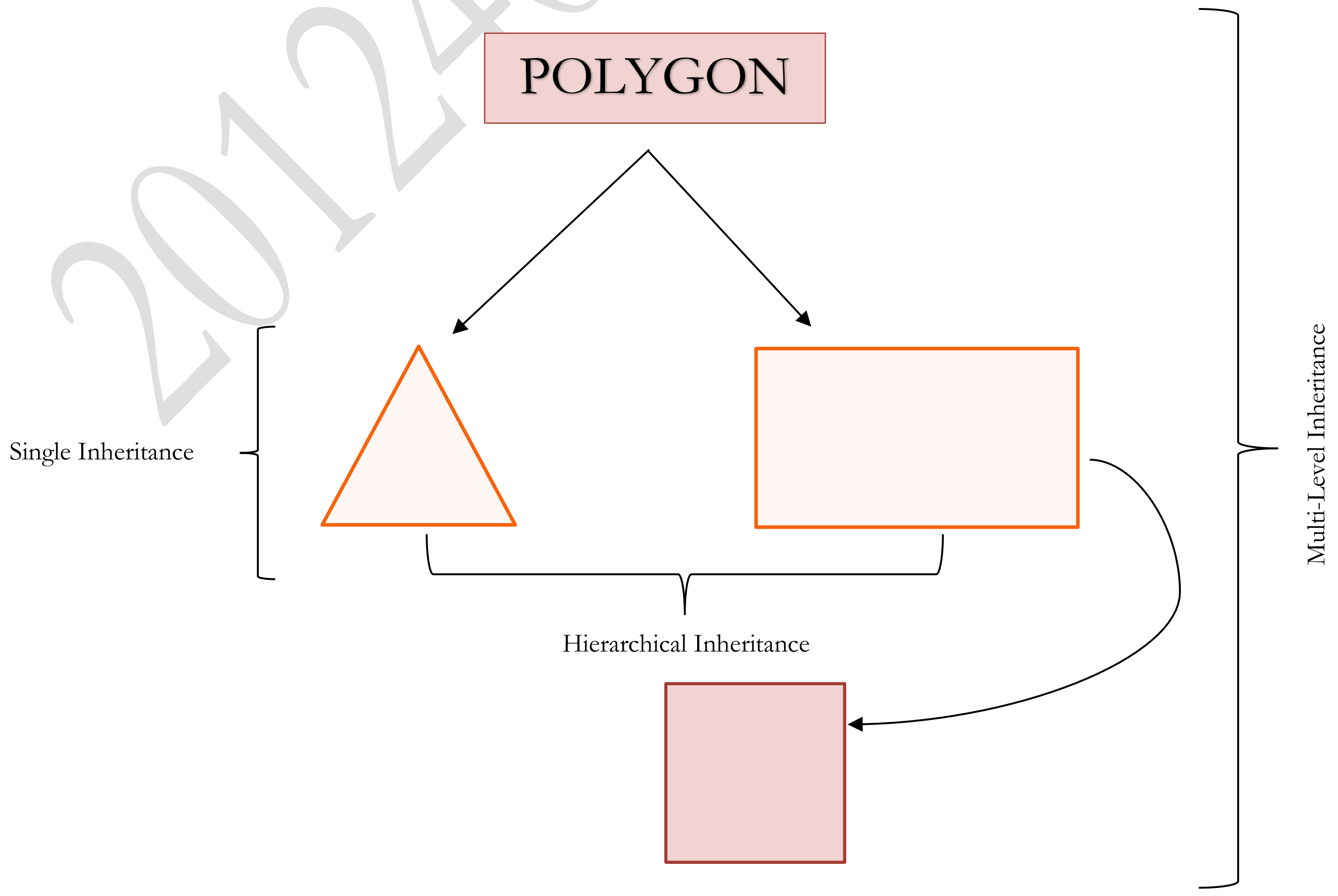
Syntax: class sub-class_name extends parent-class_name {}

super – The super keyword is used to call functions from the parent class.

Types of Inheritance – The types of inheritance in Java are

- Single inheritance – One child inherits from one parent
- Multi-level – One child inherits from one parent, and then acts as a parent for another sub class.
- Hierarchical – One parent class has multiple children

All these types of inheritance are shown in the diagram below.



Code

```
/* Program to illustrate
   inheritance */

class Polygon {

    public Polygon() {
        System.out.println("I am a Polygon");
    }

    public void Sides() {
        System.out.println("I have n sides");
    }

    public void Area() {
        System.out.print("Area of polygon = ");
        System.out.println("(perimeter * apothem) / 2\n");
    }
}

class Triangle extends Polygon{
    public Triangle() {
        System.out.println("I am a triangle");
    }

    public void Sides() {
        super.Sides();
        System.out.println("n = 3");
    }

    public void Area() {
        System.out.println("Area of triangle = (base * height)/2\n");
    }
}

class Rectangle extends Polygon{
    public Rectangle() {
        System.out.println("I am a Rectangle");
    }

    public void Sides() {
        super.Sides();
        System.out.println("n = 4");
    }

    public void Area() {
        System.out.println("Area of rectangle = base * height\n");
    }
}
```

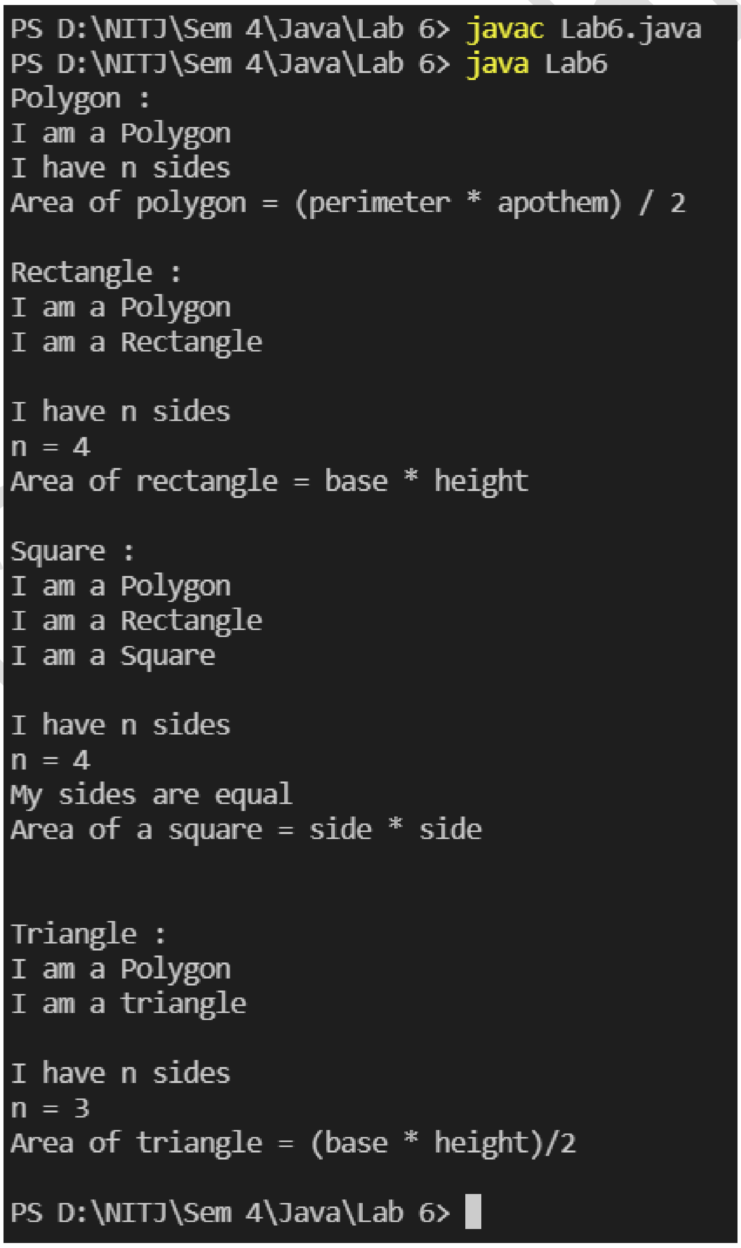
```
class Square extends Rectangle {
    public Square() {
        System.out.println("I am a Square");
    }

    public void Sides() {
        super.Sides();
        System.out.println("My sides are equal");
    }

    public void Area() {
        System.out.println("Area of a square = side * side\n");
    }
}

public class Lab6 {
    public static void main(String[] args) {

        System.out.println("Polygon :");
        Polygon P = new Polygon();
        P.Sides();
        P.Area();

        System.out.println("Rectangle :");
        Rectangle R = new Rectangle();
        System.out.println();
        R.Sides();
        R.Area();

        System.out.println("Square :");
        Square S = new Square();
        System.out.println();
        S.Sides();
        S.Area();
        System.out.println();

        System.out.println("Triangle :");
        Triangle T = new Triangle();
        System.out.println();
        T.Sides();
        T.Area();
    }
}
```

Explanation

- The code has been designed to show all types of inheritance, except multiple inheritance which is not a feature of java.
- The first class we created is Polygon. The constructor of this class prints "I am a Polygon". It also has 2 methods, Sides and Area, which print information about the sides and the area of a polygon respectively.

- The next class is Triangle. When defining it, we write **class Triangle extends Polygon**. This means that Triangle is a child of polygon class. The constructor of Triangle prints “I am a triangle”. But, when we take a look at the output, we see it actually first prints “I am a Polygon” and then in the next line, it prints “I am a Triangle”. This is because, the constructor of the child class always first calls the parent class constructor, then proceeds to execute the code in its own constructor.
- In Triangle class, we again have the methods Area and Sides. The Area method simply prints the formula for area of a triangle. The Sides method first calls **super.Sides()**. This is a call to the Sides method in the Polygon class. This way, we get information on the sides of a polygon and a rectangle.
- The Rectangle class is created the same way as the Triangle class. They both inherit from Polygon. This is an example of Hierarchical inheritance.
- The Square class is next. It inherits from class Rectangle. This is an example of multi-level inheritance, because Rectangle inherits from Polygon and Square inherits from Rectangle.

Output



```
PS D:\NITJ\Sem 4\Java\Lab 6> javac Lab6.java
PS D:\NITJ\Sem 4\Java\Lab 6> java Lab6
Polygon :
I am a Polygon
I have n sides
Area of polygon = (perimeter * apothem) / 2

Rectangle :
I am a Polygon
I am a Rectangle

I have n sides
n = 4
Area of rectangle = base * height

Square :
I am a Polygon
I am a Rectangle
I am a Square

I have n sides
n = 4
My sides are equal
Area of a square = side * side


Triangle :
I am a Polygon
I am a triangle

I have n sides
n = 3
Area of triangle = (base * height)/2

PS D:\NITJ\Sem 4\Java\Lab 6>
```

LAB 7

Program to demonstrate the concept of polymorphism

Definitions

Polymorphism – Polymorphism is another important OOP feature. With the help of this feature, we can perform a single action in multiple ways. In java there are 2 kinds of polymorphism – compile-time and runtime.

Compile-time polymorphism – Compile-time polymorphism is achieved through method overloading. We can have multiple methods with the same name, but with different number/type of parameters and different return type. The method to be used is decided during compile time. It is also called static or early binding

Method overriding - In overriding, we have 2 methods with the same name and same prototype, but in different classes (parent and child class). The implementation of the method is different in both classes. An object of the child class will access the method in the child class and an object of the parent class will access the method in the parent class.

Runtime polymorphism – This is also called dynamic method dispatch. An object of child class is made with a reference to the superclass. When we call methods using this object, it calls the method of the child class. However, if there is an overloaded method where the child class has different number/type of parameters or different return type, then we can call the method of the parent class by passing appropriate parameters.

Static methods cannot be overridden but they can be overloaded.

Code

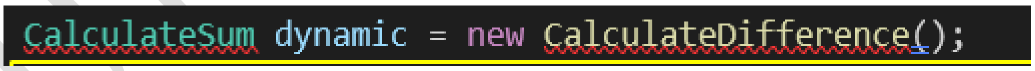
```
/* Program to demonstrate
   the concept of polymorphism */

import java.util.Random;
import java.io.*;

class CalculateSum {

    static int Calculate(int a, int b) {
        return a + b;
    }

    int Calculate(int a, int b, int c) {
        return a + b + c;
    }

    double Calculate(double a, double b) {
        return a + b;
    }
}

class CalculateDifference extends CalculateSum{

    static int Calculate(int a, int b) {
        return a - b;
    }
    double Calculate(double a, double b) {
        return a - b;
    }
}
public class Lab7 {
    public static void main(String[] args) {

        /* We have two classes, calculate sum and
           calculate difference.
           They both have overriden and overloaded funtion Calculate.
           We call this function for both classes to observe
           polymorphism in runtime and compile time. */

        //We first create random integer and double values
        Random rand = new Random();
        int I1 = rand.nextInt(50);
        int I2 = rand.nextInt(50);
        int I3 = rand.nextInt(50);
        double D1 = rand.nextDouble(50);
        double D2 = rand.nextDouble(50);

        //Creating objects of both classes
        CalculateSum CS = new CalculateSum();
        CalculateDifference CD = new CalculateDifference();

        //We output into a file called Polymorphism.txt
        try (FileWriter f = new FileWriter("Polymorphism.txt");
            PrintWriter p = new PrintWriter(f);) {

                p.println("The random integers are: ");
                p.println("I1 = " + I1 + "; I2 = " + I2 + "; I3 = " + I3);

                p.println("The random doubles are: ");
                p.printf("D1 = %.2f; D2 = %.2f\n", D1, D2);
                p.println();

                p.println("Calling Calculate function of class CalculateSum:");
                p.println(I1 + " + " + I2 + " = " + CalculateSum.Calculate(I1, I2));
                p.println(I1 + " + " + I2 + " + " + I3 + " = " + CS.Calculate(I1, I2,
I3));
                p.printf("%.2f + %.2f = %.2f\n", D1, D2, CS.Calculate(D1, D2));
                p.println();

                p.println("Calling Calculate function of class CalculateDifference:");
                p.println(I1 + " - " + I2 + " = " + CalculateDifference.Calculate(I1,
I2));
                p.printf("%.2f - %.2f = %.2f\n", D1, D2, CD.Calculate(D1, D2));
                p.println();

                p.println("Using Dynamic Method Dispatch:");
                CalculateSum dynamic = new CalculateDifference();
                p.println(I2 + " + " + I3 + " = " + dynamic.Calculate(I2, I3));
```

This line gives us a warning because we are trying to override a static method

```
            p.printf("%.2f - %.2f = %.2f\n", D1, D2, dynamic.Calculate(D1, D2));
            p.print(I1 + " + " + I2 + " + " + I3);
            p.print(" = " + dynamic.Calculate(I1, I2, I3));

            System.out.println("File Polymorphism.txt created and written.");


        } catch (IOException i) {
            i.printStackTrace();
        }
    }
}
```

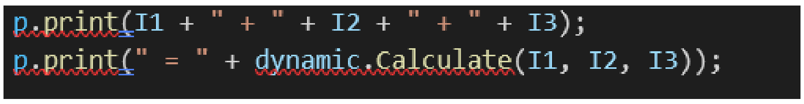
Explanation

- We create 2 classes CalculateSum and CalculateDifference. CalculateDifference is a child of CalculateSum.
- Both classes have methods called Calculate.
- In CalculateSum, we have 1 static method called calculate which takes 2 integer parameters and returns an integer. For the explanation let us call it *Static1*.
- CalculateSum has 2 more methods which are not static.
- The first one takes 3 integers and returns an integer which is the sum of the three and the next takes 2 double values and returns their sum as a double.
- CalculateDifference also has 2 methods. The first one is static and takes 2 integer parameters, returning their difference as an integer. Let us call it *Static2*.
- The other method is not static; it takes 2 doubles and returns their difference as a double.
- We create an object for each class and call the Calculate function for them. We vary the number and type of parameters and get different results. This shows us method overloading.
- We next create an object of CalculateDifference by referencing CalculateSum. We call it dynamic.



```
CalculateSum dynamic = new CalculateDifference();
```

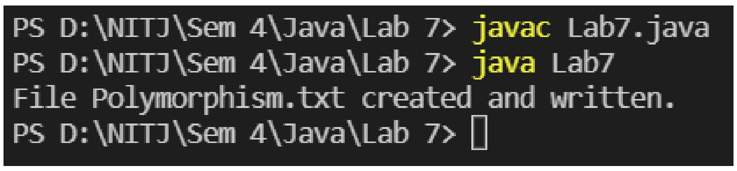
- When we try to call dynamic.Calculate(I2, I3), we get a warning. This is because we are trying to override the static function from parent class in the child class. Since this is not allowed, the output will be a sum of integers instead of a difference. i.e., Static1 is called instead of Static2.
- However, when we call dynamic.Calculate(D1, D2), it calls the function present in the child class and gives us the difference.
- We can also call the method from the parent class which takes 3 integer parameters.



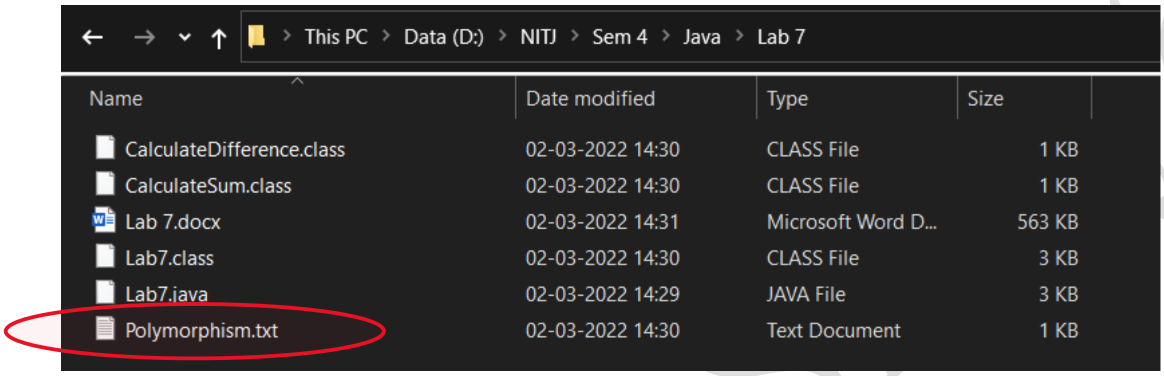
```
p.print(I1 + " + " + I2 + " + " + I3);
p.print(" = " + dynamic.Calculate(I1, I2, I3));
```

- This gives us the sum. This is an example of dynamic method dispatch. It occurs during runtime.

Output

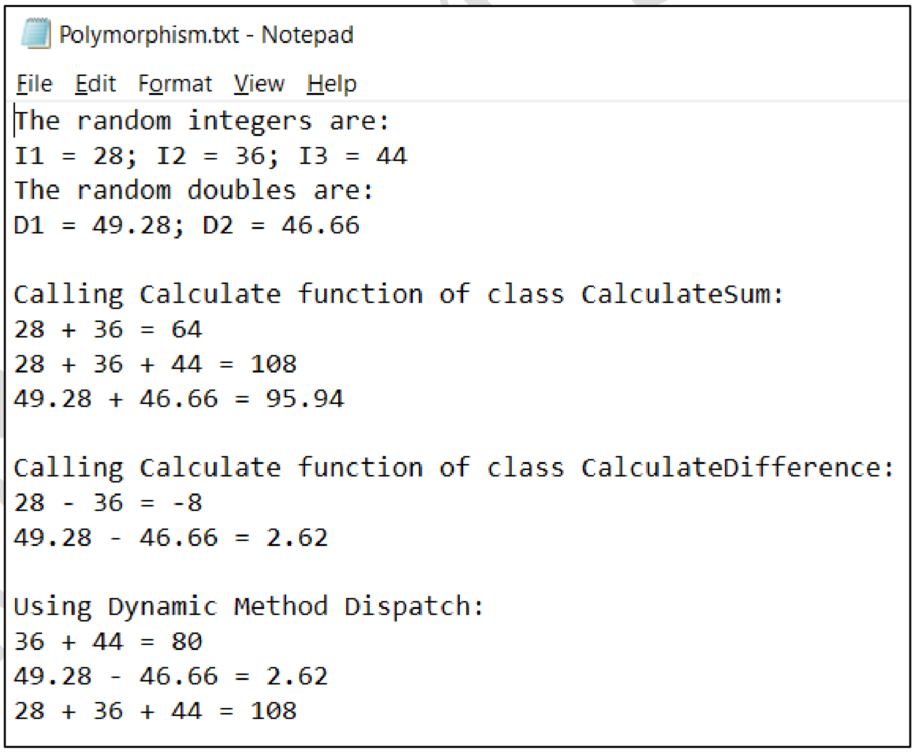


```
PS D:\NITJ\Sem 4\Java\Lab 7> javac Lab7.java
PS D:\NITJ\Sem 4\Java\Lab 7> java Lab7
File Polymorphism.txt created and written.
PS D:\NITJ\Sem 4\Java\Lab 7> 
```



This PC › Data (D:) › NITJ › Sem 4 › Java › Lab 7

Name	Date modified	Type	Size
CalculateDifference.class	02-03-2022 14:30	CLASS File	1 KB
CalculateSum.class	02-03-2022 14:30	CLASS File	1 KB
Lab 7.docx	02-03-2022 14:31	Microsoft Word D...	563 KB
Lab7.class	02-03-2022 14:30	CLASS File	3 KB
Lab7.java	02-03-2022 14:29	JAVA File	3 KB
Polymorphism.txt	02-03-2022 14:30	Text Document	1 KB



```
The random integers are:
I1 = 28; I2 = 36; I3 = 44
The random doubles are:
D1 = 49.28; D2 = 46.66

Calling Calculate function of class CalculateSum:
28 + 36 = 64
28 + 36 + 44 = 108
49.28 + 46.66 = 95.94

Calling Calculate function of class CalculateDifference:
28 - 36 = -8
49.28 - 46.66 = 2.62

Using Dynamic Method Dispatch:
36 + 44 = 80
49.28 - 46.66 = 2.62
28 + 36 + 44 = 108
```

LAB 8

Program to illustrate method overriding

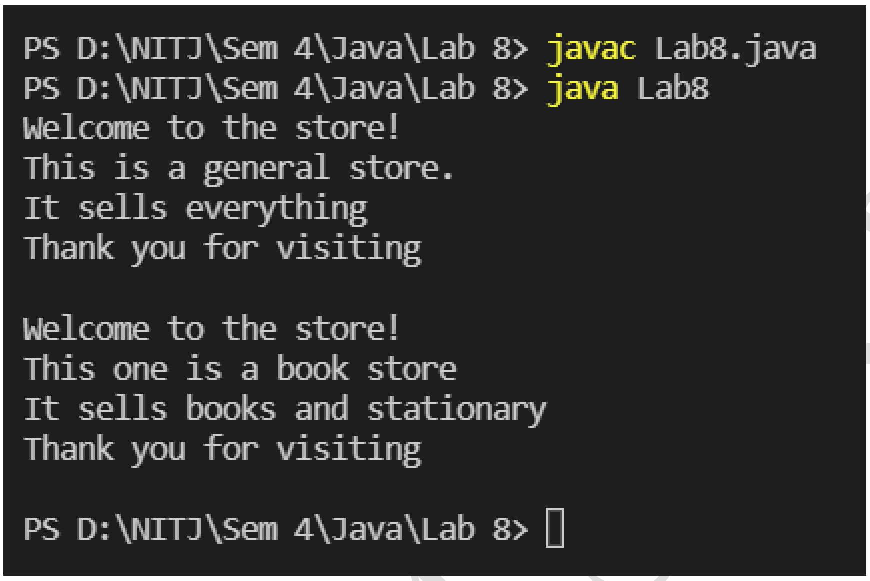
Definitions

Overriding – When a child class and a parent class have the same method, but the method does different things, it is called method overriding. The method of the parent class can be accessed by objects of parent class, and methods of the child class are accessed by objects of the child class.

Code

```
/* Program to illustrate
   method overriding */

class Store {

    public Store() {
        System.out.println("Welcome to the store!");
    }

    public void Sell() {
        System.out.println("This is a general store.");
        System.out.println("It sells everything");
    }

    protected void Exit() {
        System.out.println("Thank you for visiting");
    }
}

class BookStore extends Store {

    public BookStore() {
        System.out.println("This one is a book store");
    }

    public void Sell() {
        System.out.println("It sells books and stationary");
    }
}

public class Lab8 {
    public static void main(String[] args) {

        Store S = new Store();
        S.Sell();
        S.Exit();
        System.out.println();

        BookStore BS = new BookStore();
        BS.Sell();
        BS.Exit();
        System.out.println();
    }
}
```

Output



```
PS D:\NITJ\Sem 4\Java\Lab 8> javac Lab8.java
PS D:\NITJ\Sem 4\Java\Lab 8> java Lab8
Welcome to the store!
This is a general store.
It sells everything
Thank you for visiting

Welcome to the store!
This one is a book store
It sells books and stationary
Thank you for visiting

PS D:\NITJ\Sem 4\Java\Lab 8> 
```

LAB 9

Programs to illustrate the following classes

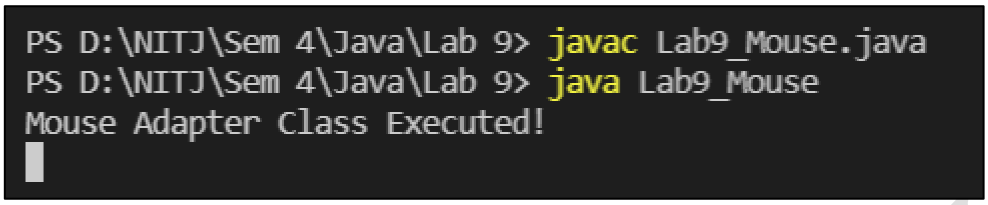
Mouse Adapter

Code

```java
import java.awt.*;
import java.awt.event.*;

public class Lab9_Mouse {

    public class mouseAdapterClass extends MouseAdapter {

        Frame f;

        public mouseAdapterClass() {
            f = new Frame("Mouse Adapter");
            f.setSize(500, 500);
            f.setLayout(null);
            f.setVisible(true);
            f.addMouseListener(this);

            f.addWindowListener(new WindowAdapter() {
                public void windowClosing(WindowEvent windowEvent) {
                   System.exit(0);
                }
            });
        }

        public void mouseClicked (MouseEvent e) {
            Graphics g = f.getGraphics();

            Color randomColor = new Color((int)(Math.random() * 0x1000000));
            g.setColor (randomColor);
            g.fillRect (e.getX(), e.getY(), 30, 30);
        }
    }

    public static void main(String[] args) {
        Lab9_Mouse L = new Lab9_Mouse();

        mouseAdapterClass M = L.new mouseAdapterClass();
        M.getClass();
        System.out.println("Mouse Adapter Class Executed!");
    }
}
```

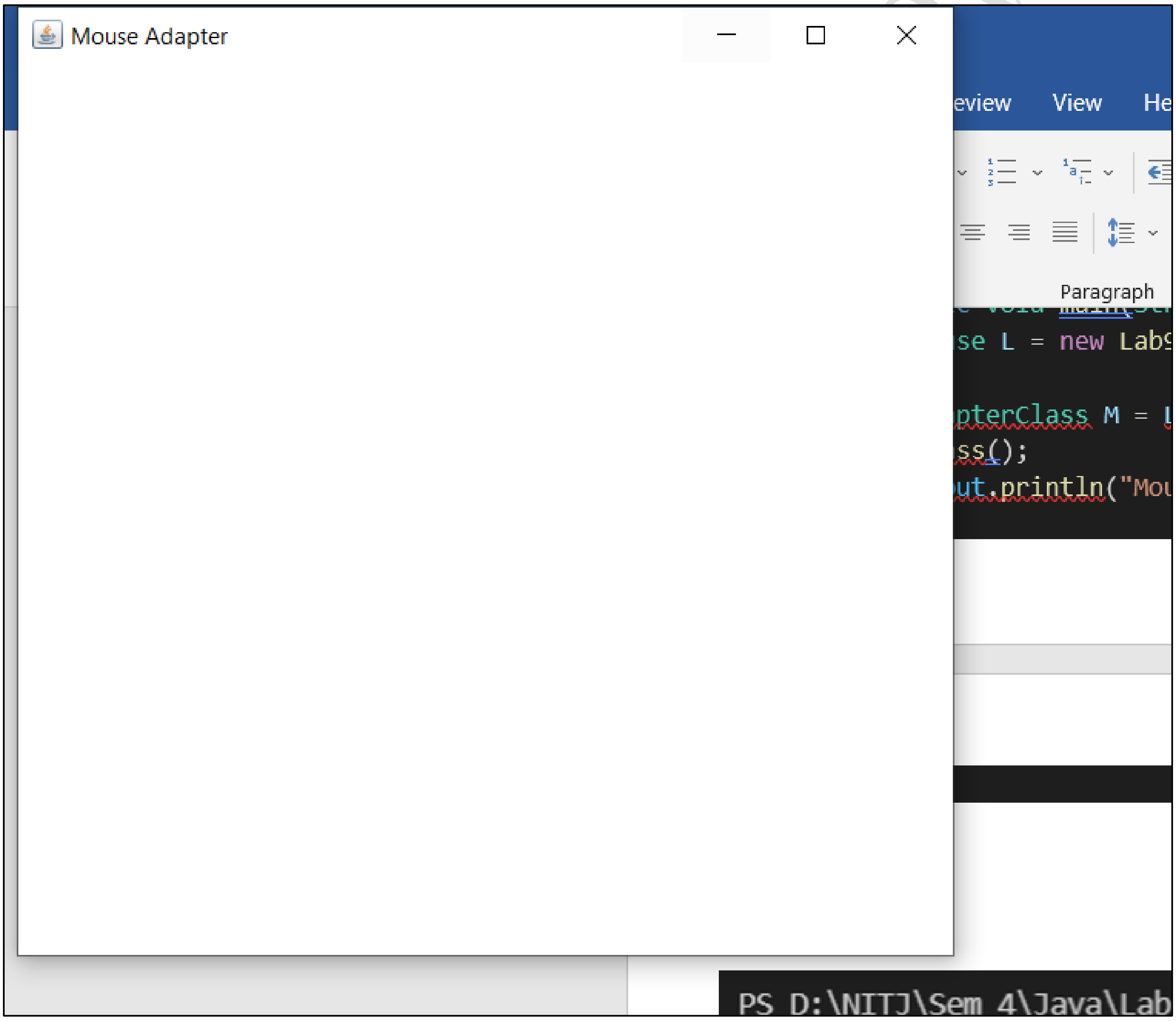
Output

1. Console output:

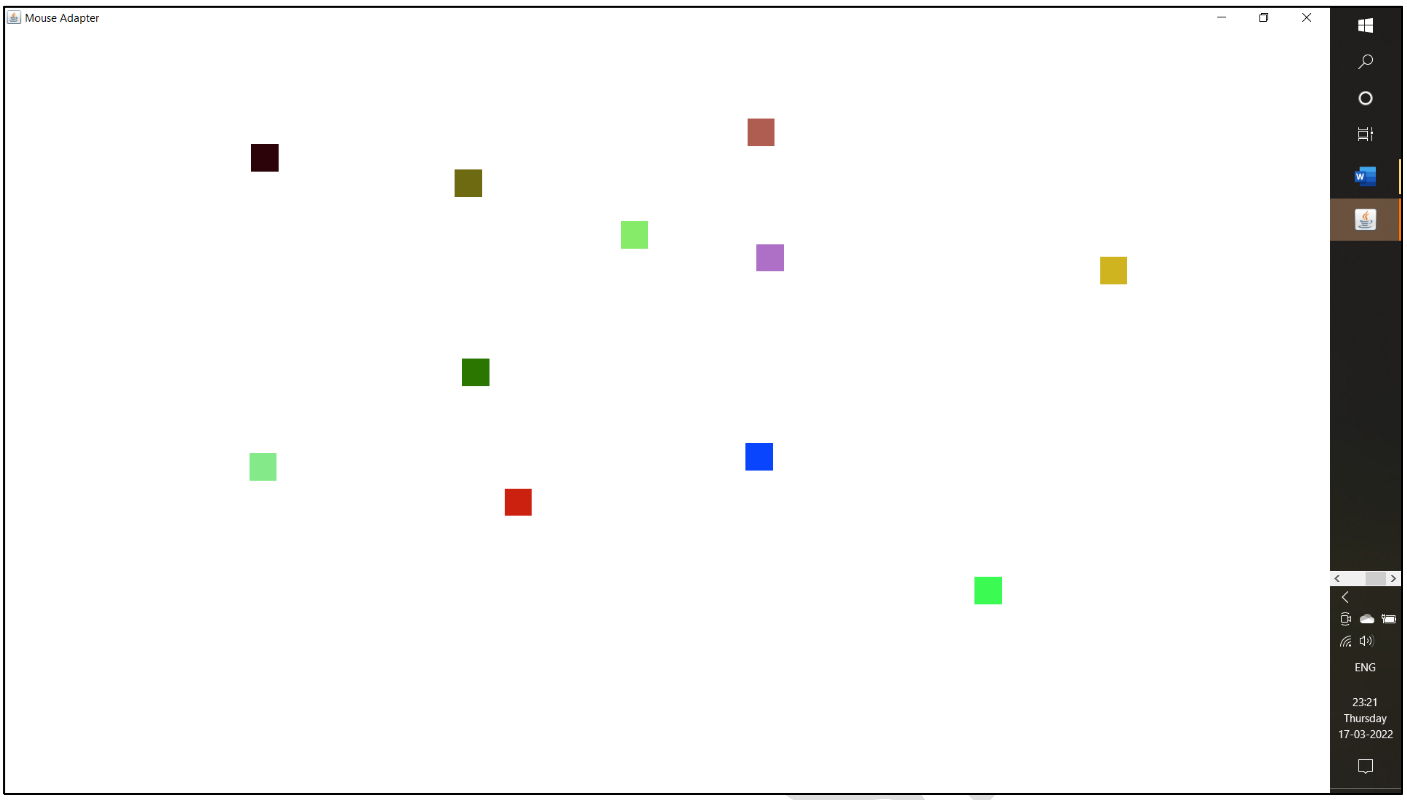


```
PS D:\NITJ\Sem 4\Java\Lab 9> javac Lab9_Mouse.java
PS D:\NITJ\Sem 4\Java\Lab 9> java Lab9_Mouse
Mouse Adapter Class Executed!
```

2. The new frame created:



3. The frame in full screen after clicking a few spots
4. The frame can be closed by clicking the cross on top right, or by typing ctrl+c in the console


Mouse Adapter
ENG
23:21
Thursday
17-03-2022

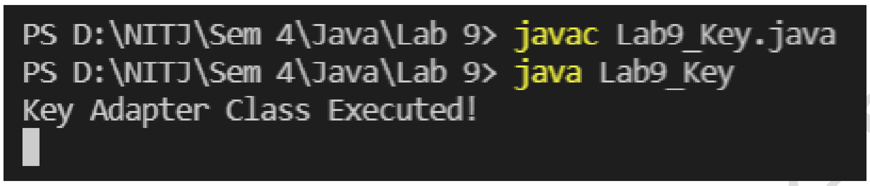
Key Adapter

Code

```java
import java.awt.*;
import java.awt.event.*;

public class Lab9_Key {

    public class keyAdapterClass extends KeyAdapter {

        Frame f;
        TextField textArea;
        Label label;

        public keyAdapterClass() {

            f = new Frame("Key Adapter");

            f.addWindowListener(new WindowAdapter() {
                public void windowClosing(WindowEvent windowEvent) {
                    System.exit(0);
                }
            });

            textArea = new TextField();
            textArea.setBounds(50, 50, 400, 200);
            textArea.setBackground(Color.LIGHT_GRAY);
            textArea.addKeyListener(this);
            label = new Label();
            label.setBounds(0, 100, 500, 500);
            label.setAlignment(Label.CENTER);

            f.setSize(500, 500);
            f.setLayout(null);
            f.add(textArea);
            f.add(label);
            f.setVisible(true);
        }

        public void keyPressed(KeyEvent e) {
            if(e.getKeyCode() == KeyEvent.VK_ENTER) {
                label.setText("Entered text: " + textArea.getText());
            }
        }
    }

    public static void main(String[] args) {
        Lab9_Key L = new Lab9_Key();

        keyAdapterClass K = L.new keyAdapterClass();
        K.getClass();
        System.out.println("Key Adapter Class Executed!");
    }
}
```

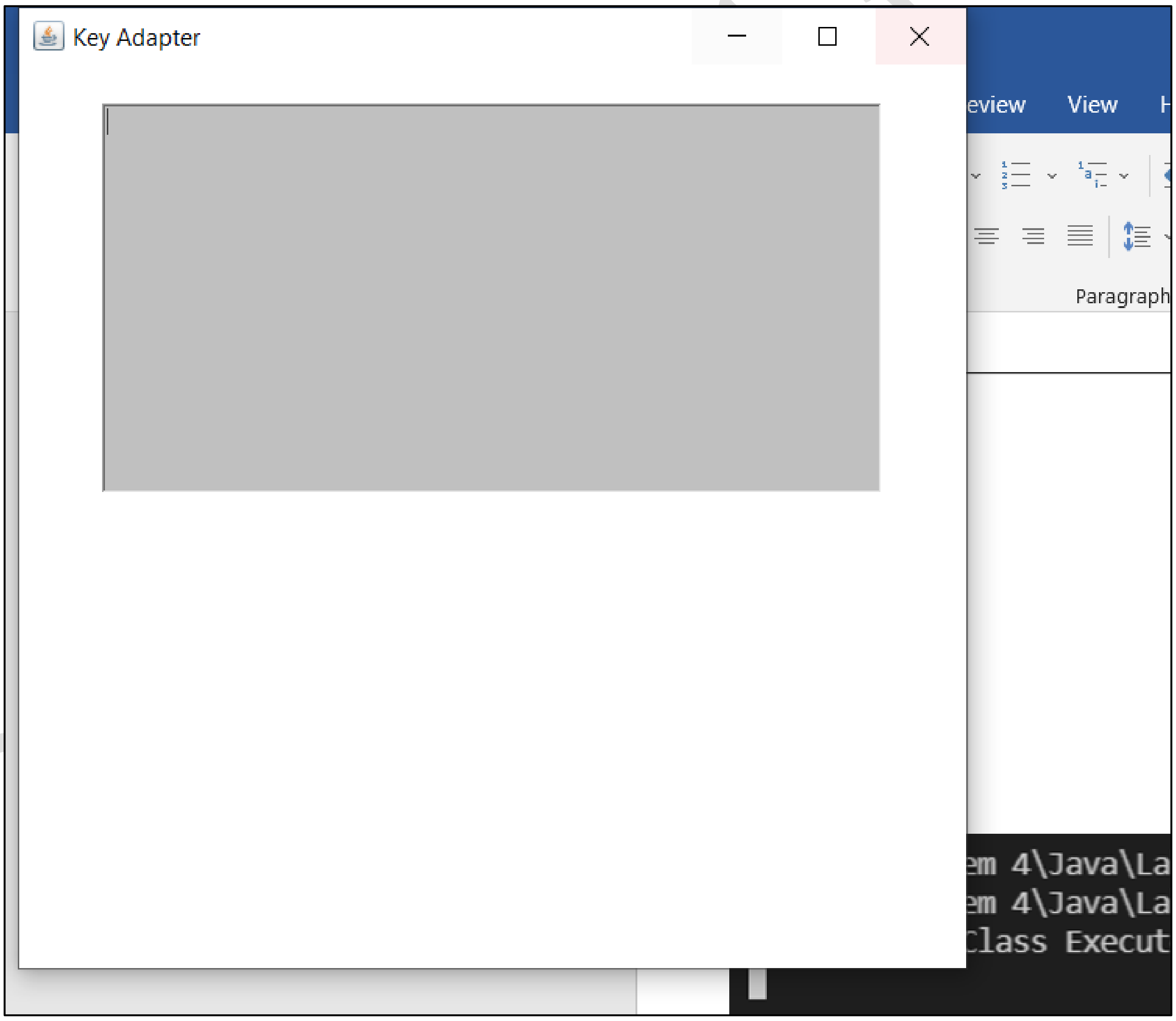
Output

1. Console Output:



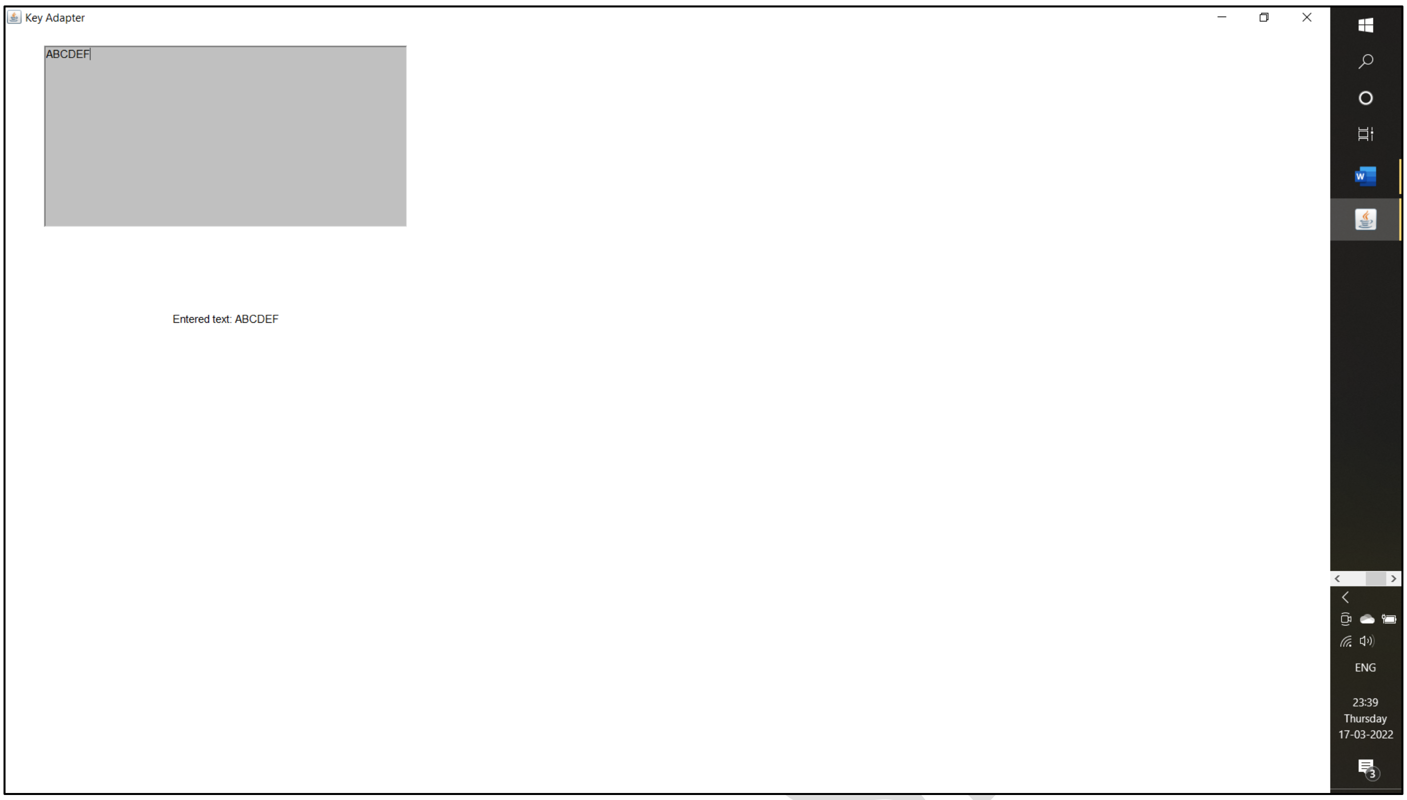
```
PS D:\NITJ\Sem 4\Java\Lab 9> javac Lab9_Key.java
PS D:\NITJ\Sem 4\Java\Lab 9> java Lab9_Key
Key Adapter Class Executed!
```

2. The new frame created:

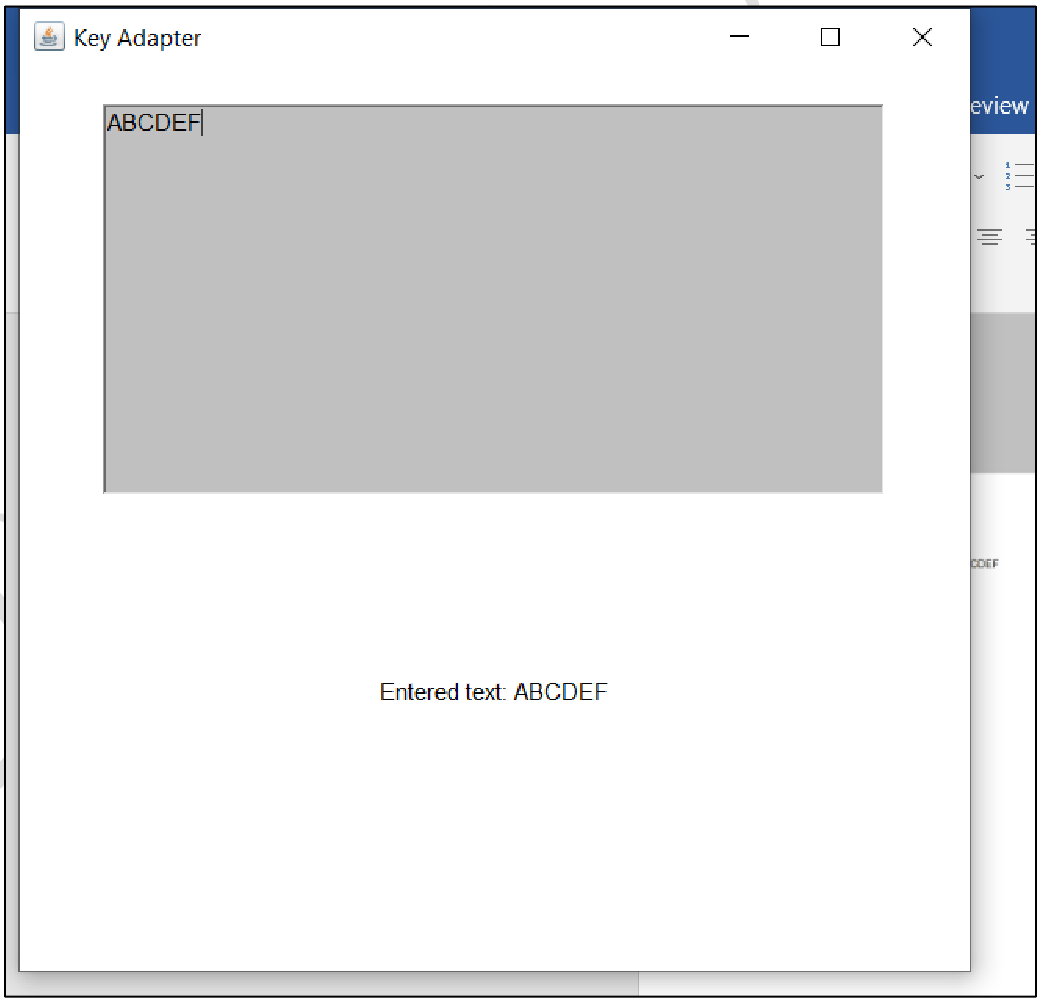


```
em 4\Java\La
em 4\Java\La
Class Execut
```

3. The frame in full screen with some letters typed in
4. After typing ABCDEF, enter was clicked to get the output



Entered text: ABCDEF



Entered text: ABCDEF

5. The Frame is exited by clicking the cross on top right, or by typing ctrl+c in the console

LAB 10

Program to illustrate Array Manipulation

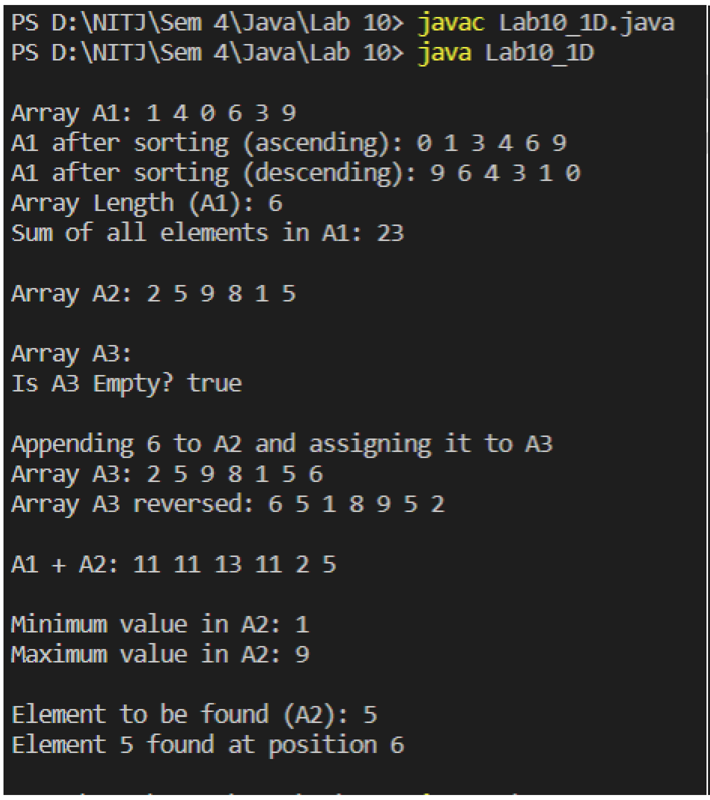
1. 1-Dimensional Array

Code

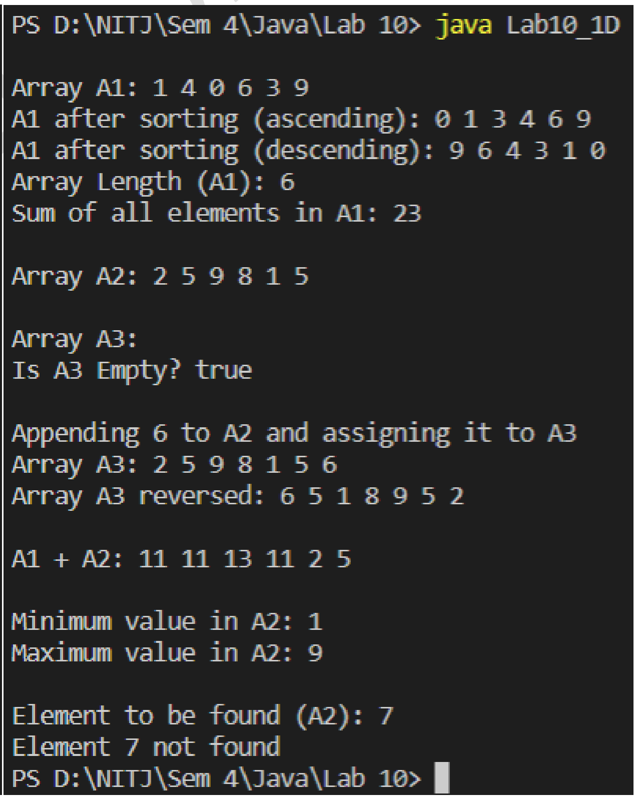
```
import java.util.Arrays;
import java.util.Collections;
import java.util.Scanner;

/* Program to illustrate
   Array Manipulations */

public class Lab10_1D {
    public static void main(String[] args) {

        try (Scanner sc = new Scanner(System.in)) {

            Integer[] A1 = new Integer[] {1, 4, 0, 6, 3, 9};
            int[] A2 = new int[] {2, 5, 9, 8, 1, 5};
            int[] A3 = new int[] {};
            int i, sum = 0;

            System.out.print("\nArray A1: ");
            for(int k: A1) {
                System.out.print(k + " ");
                sum = sum + k;
            }

            //Sorting an array
            Arrays.sort(A1);
            System.out.print("\nA1 after sorting (ascending): ");
            for(int k: A1)
                System.out.print(k + " ");

            Arrays.sort(A1, Collections.reverseOrder());
            System.out.print("\nA1 after sorting (descending): ");
            for(int k: A1) {
                System.out.print(k + " ");
            }

            //Finding length of an array
            System.out.println("\nArray Length (A1): " + A1.length);
            //Finding sum of all elements of an array
            System.out.println("Sum of all elements in A1: " + sum);

            System.out.print("\nArray A2: ");
            for(int k: A2) {
                System.out.print(k + " ");
            }

            System.out.print("\n\nArray A3: ");
            for(int k: A3) {
                System.out.print(k + " ");
            }

            System.out.print("\nIs A3 Empty? ");
            System.out.println(A3.length == 0 || A3 == null);

            //Appending an element to an array
            System.out.println("\nAppending 6 to A2 and assigning it to A3");
            A3 = Arrays.copyOf(A2, A2.length + 1);
            A3[A3.length - 1] = 6;
            System.out.print("Array A3: ");
            for(int k: A3) {
                System.out.print(k + " ");
            }

            int temp, len = A3.length;
            //Reversing an array
            for(i = 0; i < len / 2; i++) {
                temp = A3[i];
                A3[i] = A3[len - 1 - i];
                A3[len - 1 - i] = temp;
            }

            System.out.print("\nArray A3 reversed: ");
            for(int k: A3) {
                System.out.print(k + " ");
            }

            //Adding two arrays
            System.out.print("\n\nA1 + A2: ");
            if(A1.length != A2.length)
                System.out.println("Addition not possible!");
            else {
                for(i = 0; i < A2.length; i++)
                    System.out.printf("%d ", (A1[i] + A2[i]));
            }

            //Finding min and max values of array
            int min = A2[0], max = A2[0];
            for(int k: A2) {
                if(k < min)
                    min = k;

                if(k > max)
                    max = k;
            }
            System.out.println("\n\nMinimum value in A2: " + min);
            System.out.println("Maximum value in A2: " + max);

            //Finding a given element in an array, and printing its position
            int pos = -1;
            System.out.print("\nElement to be found (A2): ");
            int element = sc.nextInt();
```

```java
            for(i = 0; i < A2.length; i++) {
                if(A2[i] == element)
                    pos = i;
            }

            if(pos == -1)
                System.out.println("Element " + element + " not found");
            else {
                pos ++;
                System.out.print("Element " + element);
                System.out.println(" found at position " + pos);
                System.out.println();
            }
        }
    }
}
```

Output



```
PS D:\NITJ\Sem 4\Java\Lab 10> javac Lab10_1D.java
PS D:\NITJ\Sem 4\Java\Lab 10> java Lab10_1D

Array A1: 1 4 0 6 3 9
A1 after sorting (ascending): 0 1 3 4 6 9
A1 after sorting (descending): 9 6 4 3 1 0
Array Length (A1): 6
Sum of all elements in A1: 23

Array A2: 2 5 9 8 1 5

Array A3:
Is A3 Empty? true

Appending 6 to A2 and assigning it to A3
Array A3: 2 5 9 8 1 5 6
Array A3 reversed: 6 5 1 8 9 5 2

A1 + A2: 11 11 13 11 2 5

Minimum value in A2: 1
Maximum value in A2: 9

Element to be found (A2): 5
Element 5 found at position 6
```



```
PS D:\NITJ\Sem 4\Java\Lab 10> java Lab10_1D

Array A1: 1 4 0 6 3 9
A1 after sorting (ascending): 0 1 3 4 6 9
A1 after sorting (descending): 9 6 4 3 1 0
Array Length (A1): 6
Sum of all elements in A1: 23

Array A2: 2 5 9 8 1 5

Array A3:
Is A3 Empty? true

Appending 6 to A2 and assigning it to A3
Array A3: 2 5 9 8 1 5 6
Array A3 reversed: 6 5 1 8 9 5 2

A1 + A2: 11 11 13 11 2 5

Minimum value in A2: 1
Maximum value in A2: 9

Element to be found (A2): 7
Element 7 not found
PS D:\NITJ\Sem 4\Java\Lab 10>
```

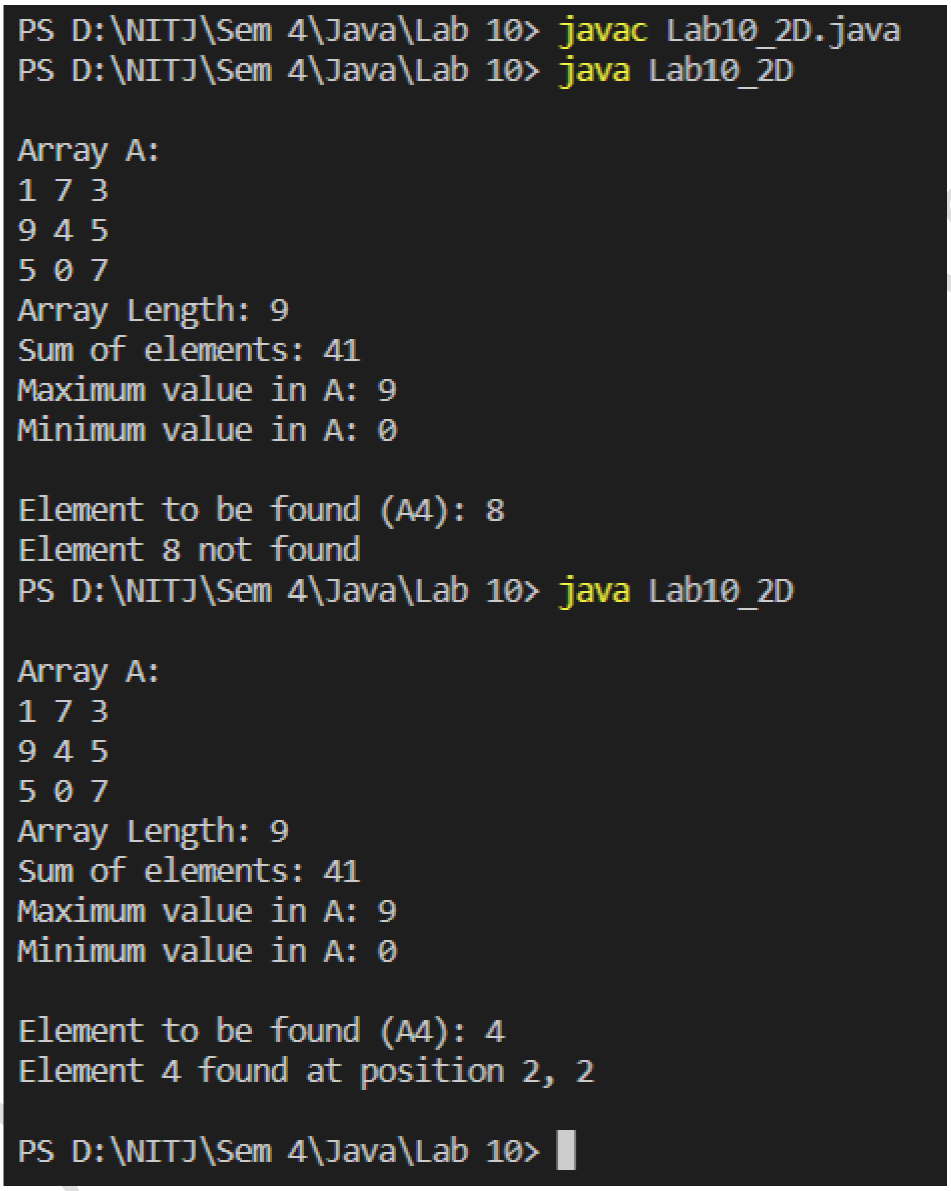
2. 2-Dimensional Array

Code

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

```
public class Lab10_2D {
    public static void main(String[] args) {

        int[][] A = new int[][] {{1, 7, 3}, {9, 4, 5}, {5, 0, 7}};
        int i, j;
        int sum = 0;
        int min = A[0][0], max = A[0][0];

        try(Scanner sc = new Scanner(System.in)) {
            System.out.println("\nArray A: ");
            for(i = 0; i < A.length; i++) {
                for(j = 0; j < A[i].length; j++) {
                    sum += A[i][j];
                    System.out.print(A[i][j] + " ");

                    if(A[i][j] < min)
                        min = A[i][j];

                    if(A[i][j] > max)
                        max = A[i][j];
                }
                System.out.println();
            }

            System.out.println("Array Length: " + A.length * (A[0].length));
            System.out.println("Sum of elements: " + sum);
            System.out.println("Maximum value in A: " + max);
            System.out.println("Minimum value in A: " + min);

            //Finding a given element in an array, and printing its position
            int posi = -1, posj = -1;
            System.out.print("\nElement to be found (A4): ");
            int element = sc.nextInt();

            for(i = 0; i < A.length; i++) {
                for(j = 0; j < A[i].length; j++) {
                    if(A[j][i] == element) {
                        posi = i;
                        posj = j;
                    }
                }
            }

            if(posi == -1)
                System.out.println("Element " + element + " not found");
            else {
                posi ++; posj ++;
                System.out.print("Element " + element);
                System.out.println(" found at position " + posj + ", " + posi);
                System.out.println();
            }
        }
    }
}
```

Output



```
PS D:\NITJ\Sem 4\Java\Lab 10> javac Lab10_2D.java
PS D:\NITJ\Sem 4\Java\Lab 10> java Lab10_2D

Array A:
1 7 3
9 4 5
5 0 7
Array Length: 9
Sum of elements: 41
Maximum value in A: 9
Minimum value in A: 0

Element to be found (A4): 8
Element 8 not found
PS D:\NITJ\Sem 4\Java\Lab 10> java Lab10_2D

Array A:
1 7 3
9 4 5
5 0 7
Array Length: 9
Sum of elements: 41
Maximum value in A: 9
Minimum value in A: 0

Element to be found (A4): 4
Element 4 found at position 2, 2

PS D:\NITJ\Sem 4\Java\Lab 10>
```

LAB 11

Program to illustrate Layout Manager

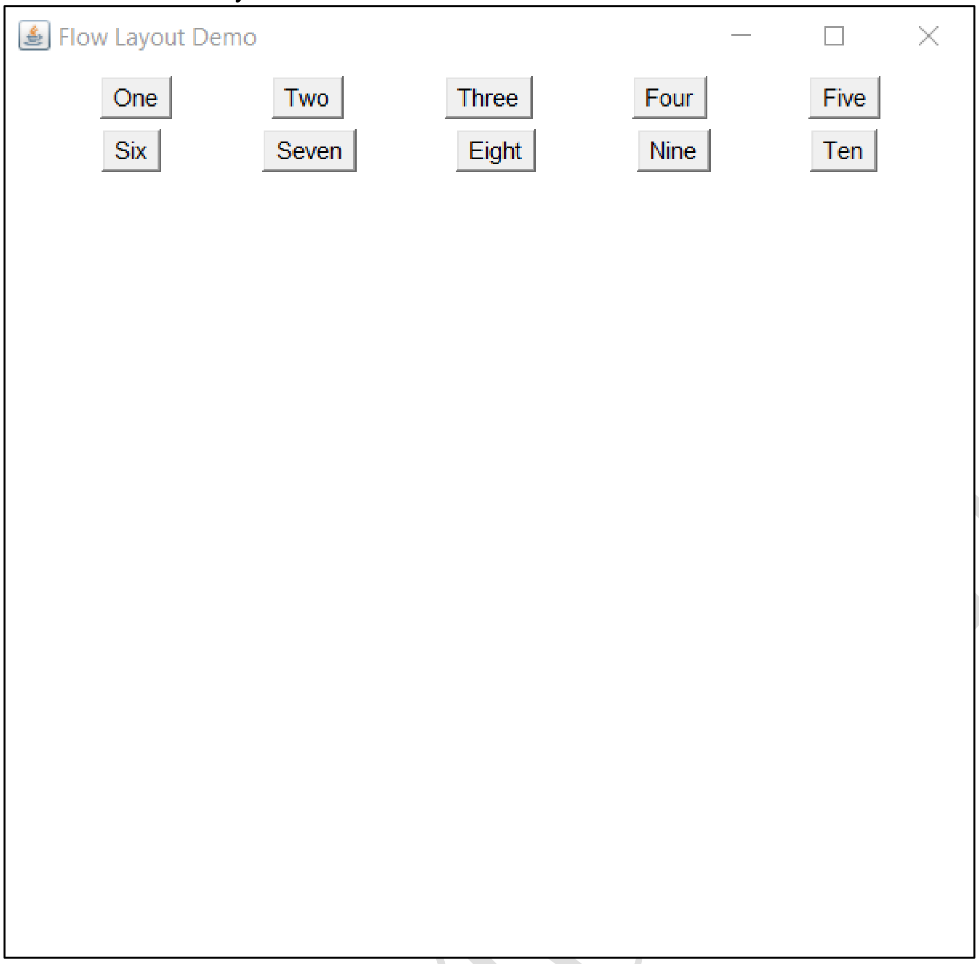
Code

```
import javax.swing.*;
import java.util.Scanner;
import java.awt.*;
import java.awt.event.*;

class flowLayout extends Frame {

    Button b1, b2, b3, b4, b5, b6, b7, b8, b9, b10;
    public flowLayout() {

        super("Flow Layout Demo");

        b1=new Button("One");
        b2=new Button("Two");
        b3=new Button("Three");

        b4=new Button("Four");
        b5=new Button("Five");
        b6=new Button("Six");

        b7=new Button("Seven");
        b8=new Button("Eight");
        b9=new Button("Nine");
        b10=new Button("Ten");

        FlowLayout fl=new FlowLayout();
        fl.setAlignment(FlowLayout.CENTER);
        fl.setHgap(50);
        setLayout(fl);

        add(b1);
        add(b2);
        add(b3);
        add(b4);
        add(b5);
        add(b6);
        add(b7);
        add(b8);
        add(b9);
        add(b10);
    }
}

class borderLayout extends Frame {

    Button b1, b2, b3, b4, b5, b6, b7, b8, b9, b10;
    public borderLayout() {

        super("Border Layout Demo");
        b1=new Button("One");
        b2=new Button("Two");
        b3=new Button("Three");

        b4=new Button("Four");
        b5=new Button("Five");
        b6=new Button("Mon");
        b7=new Button("Tue");

        b8=new Button("Wed");
        b9=new Button("Thru");
        b10=new Button("Fri");

        add(b2, BorderLayout.WEST);
        add(b3, BorderLayout.EAST);
        add(b4, BorderLayout.NORTH);
        add(b5, BorderLayout.SOUTH);

        Panel p=new Panel(new GridLayout());
        p.add(b6);
        p.add(b7);
        p.add(b8);
        p.add(b9);
        p.add(b10);
        add(p, BorderLayout.CENTER);

    }
}

class gridLayout extends Frame {

    Button b1, b2, b3, b4, b5, b6, b7, b8, b9, b10;
    public gridLayout() {

        super("Grid Layout Demo");
        b1=new Button("One");
        b2=new Button("Two");
        b3=new Button("Three");
        b4=new Button("Four");
        b5=new Button("Five");
        b6=new Button("Six");
        b7=new Button("Seven");
        b8=new Button("Eight");
        b9=new Button("Nine");
        b10=new Button("Ten");

        GridLayout gl=new GridLayout(4, 3);
        setLayout(gl);

        add(b1);
        add(b2);
        add(b3);
        add(b4);
        add(b5);
        add(b6);
        add(b7);
        add(b8);
        add(b9);
        add(b10);
    }
}

class gridBagLayout extends Frame {

    Button b1, b2, b3, b4, b5, b6, b7, b8, b9, b10;
    public gridBagLayout() {

        super("GridBag Layout Demo");
        b1=new Button("One");
        b2=new Button("Two");
        b3=new Button("Three");
        b4=new Button("Four");
        b5=new Button("Five");
        b6=new Button("Six");

        GridBagLayout gb=new GridBagLayout();
        GridBagConstraints gbc=new GridBagConstraints();

        setLayout(gb);
        gbc.gridx=1;
        gbc.gridy=1;

        add(b1, gbc);
        gbc.gridx=2;
        gbc.gridy=2;

        add(b2, gbc);
        gbc.gridx=3;
        gbc.gridy=3;

        add(b3, gbc);
        gbc.gridx=4;
        gbc.gridy=4;

        add(b4, gbc);
        gbc.gridx=5;
        gbc.gridy=5;

        add(b5, gbc);
        gbc.gridx=6;
        gbc.gridy=6;
        add(b6, gbc);
    }
}
```

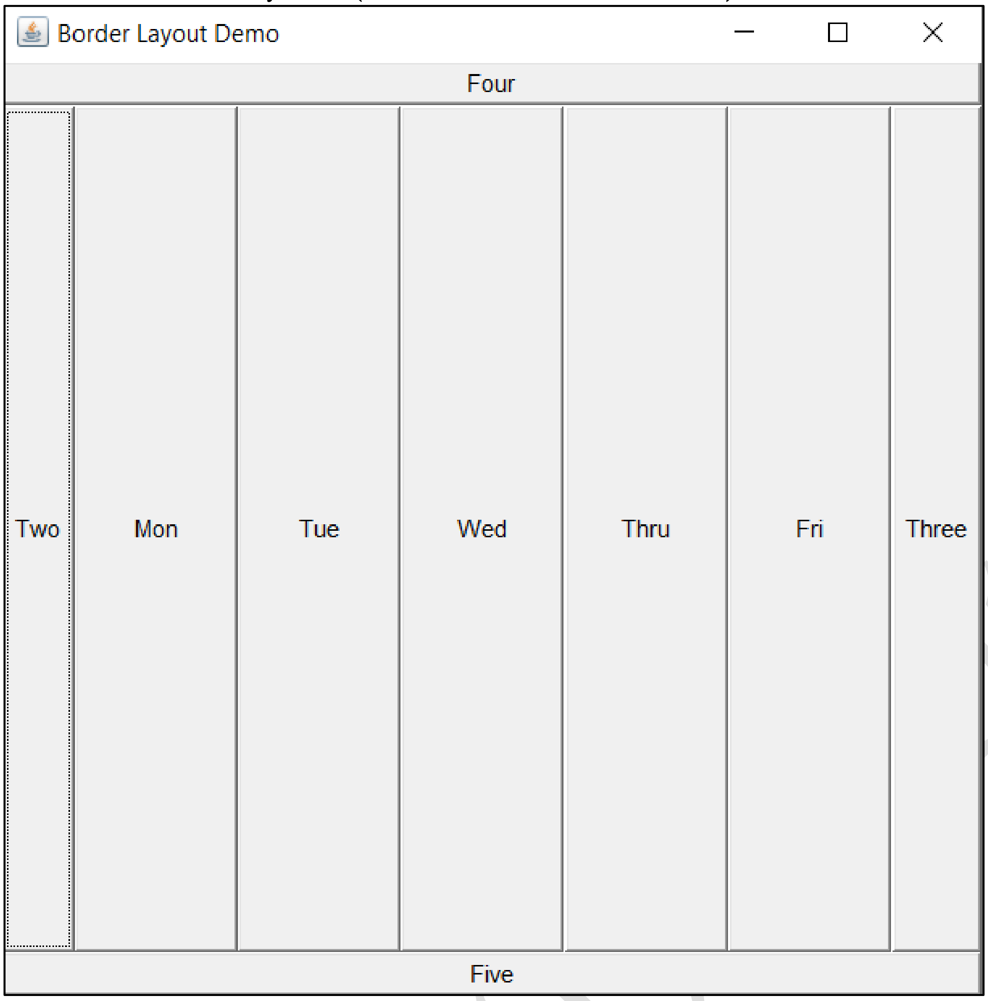
```
class cardLayout extends Frame {

    JFrame frame = new JFrame("CardLayout demo");
    JPanel panelCont = new JPanel();
    JPanel panelFirst = new JPanel();
    JPanel panelSecond = new JPanel();
    JButton buttonOne = new JButton("Switch to second panel/workspace");
    JButton buttonSecond = new JButton("Switch to first panel/workspace");
    CardLayout cl = new CardLayout();

    public cardLayout() {

        panelCont.setLayout(cl);
        panelFirst.add(buttonOne);
        panelSecond.add(buttonSecond);
        panelFirst.setBackground(Color.BLUE);
        panelSecond.setBackground(Color.GREEN);

        panelCont.add(panelFirst, "1");
        panelCont.add(panelSecond, "2");
        cl.show(panelCont, "1");

        buttonOne.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent arg0) {
                cl.show(panelCont, "2");
            }
        });

        buttonSecond.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent arg0) {
                cl.show(panelCont, "1");
            }
        });

        frame.add(panelCont);
        frame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        frame.pack();
        frame.setVisible(true);
    }
}

    public class Lab11 {
        public static void main(String[] args) {
            flowLayout f = new flowLayout();
            f.setSize(500, 500);
            f.setVisible(true);
            borderLayout b = new borderLayout();
            b.setSize(500, 500);
            b.setVisible(true);
            gridLayout g = new gridLayout();
            g.setSize(500, 500);
            g.setVisible(true);
            gridBagLayout gb = new gridBagLayout();
            gb.setSize(500, 500);
            gb.setVisible(true);
            cardLayout cd = new cardLayout();
        }
    }
```

Output

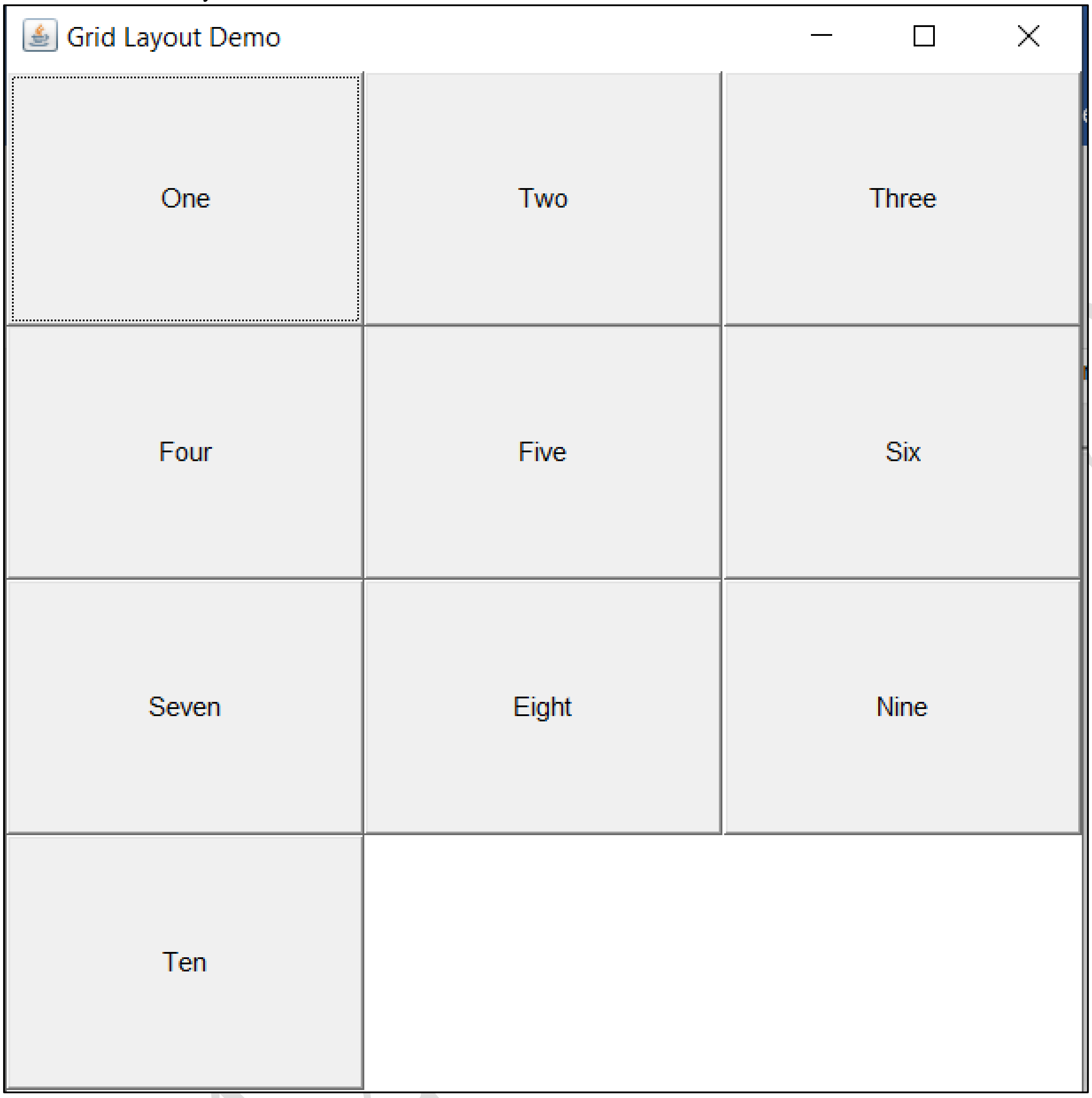
1. Flow Layout:



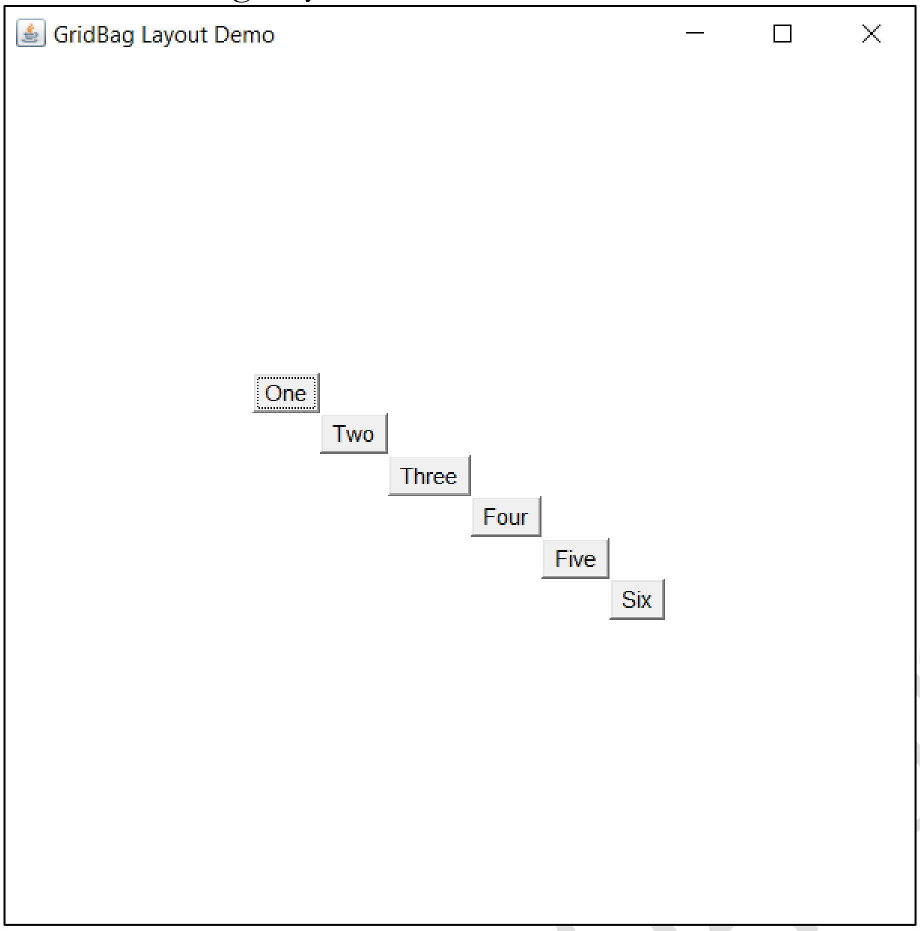
2. Border Layout (with a Panel in Centre):



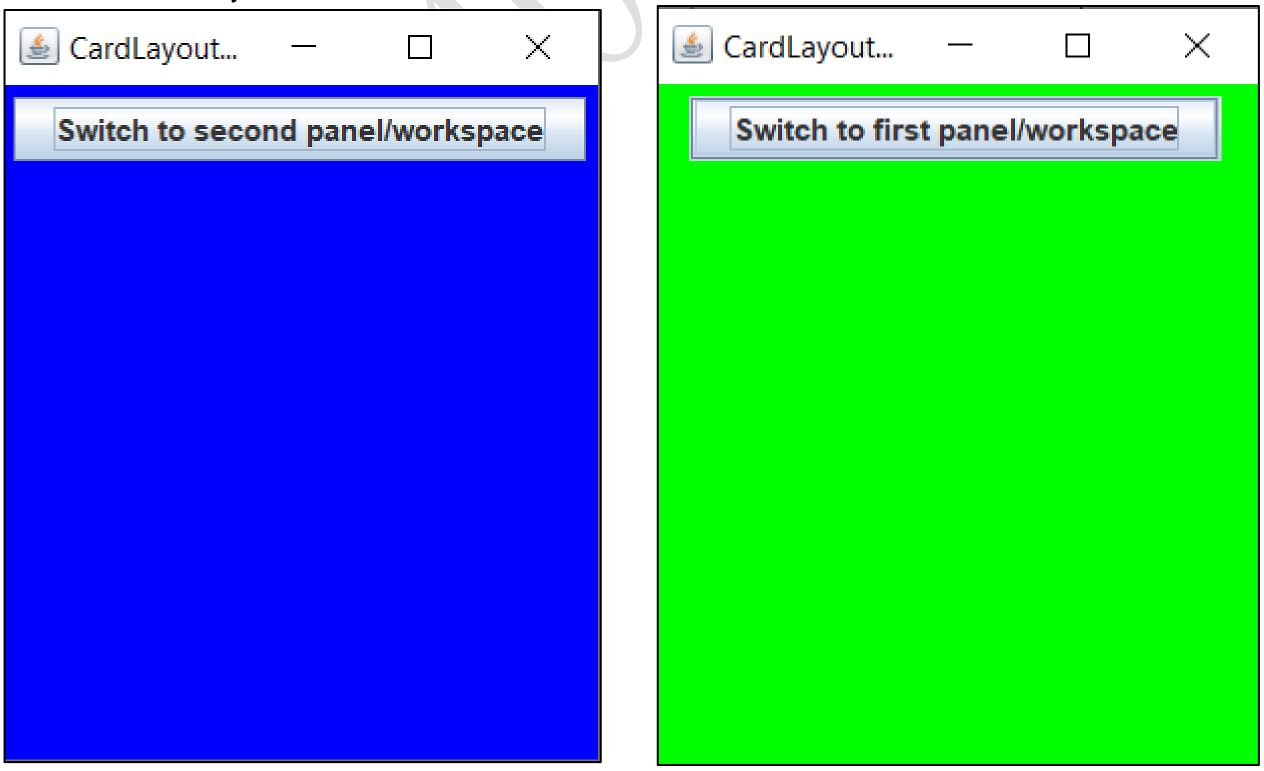
3. Grid Layout:



4. Grid-Bag Layout:



5. Card Layout:


CardLayout...
Switch to second panel/workspace
CardLayout...
Switch to first panel/workspace