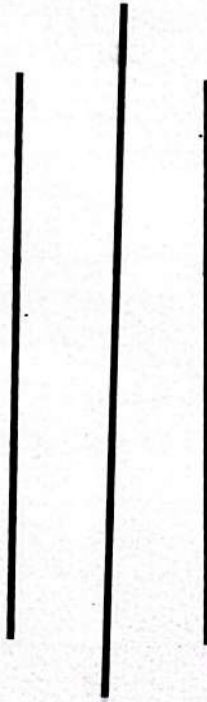


AADIKAVI BHANUBHAKTA CAMPUS
Vyas-01, Bigyanchour, Tanahun



Lab report on
Java Programming Language [ICT Ed. 455]

Submitted by:

Name: Rajip Ranabhat

Roll No.: 80288022

Submitted To:

Er. Bibek Bhujel

Subject Teacher

TABLE OF CONTENTS

S.N.	Title	Lab Date	Page No.	Lab Submission Date	Remarks
1.	Program to demonstrate the basic structure of a Java program.	2082-08-23	1	2082-09-02	Bebeky Singh
2.	Basic data types in Java.	2082-09-02	2-3	2082-09-09	Bebeky Singh
3.	Program to demonstrate type-casting in Java.	2082-09-09	4-5	2082-09-23	Bebeky Singh
4.	Program to demonstrate loop using For Buzz in Java.	2082-09-23	6-7	2082-10-07	Bebeky Singh
5.	Program to demonstrate inheritance in Java.	2082-10-07	8-9	2082-10-24	Bebeky Singh
6.	Program to demonstrate polymorphism in Java.	2082-10-07	10-11	2082-10-24	Bebeky Singh
7.	Program to demonstrate exception handling in Java.	2082-10-24	12-13	2082-10-21	
8.	Program to demonstrate JFrame and JPanel in Java.	2082-10-21	14-15	2082-10-28	
9.	Program to demonstrate Border layout in Java.	2082-10-28	16-17	2082-11-13	

TABLE OF CONTENTS

S.N.	Title	Lab Date	Page No.	Lab Submission Date	Remarks
10.	Program to demonstrate Box Layout in Java.	2082-12-23	18-19	2082-11-27	
11.	Program to demonstrate Grid layout in Java.	2082-11-27	20-21	2082-12-11	
12.	Program to demonstrate GridBag layout in Java.	2082-12-11	22-23	2082-12-18	
23.	Program to demonstrate Flow layout in Java.	2082-12-18	24-25	2082-12-25	

TITLE : PROGRAM TO DEMONSTRATE THE BASIC STRUCTURE OF A JAVA PROGRAM.

OBJECTIVES:

- To write and execute a simple Java program
- To understand the basic structure and syntax of a Java program.

TOOLS USED:

- Visual Studio Code

PROBLEM ANALYSIS:

Java programs follow a specific structure that includes class declaration, main method, and statements for execution. This lab focuses on analyzing the basic structure of a Java program by writing and executing a simple code to understand program flow and syntax.

SOURCE CODE:

```
public class Main {  
    public static void main (String[] args)  
    {  
        System.out.println("Rajip Ranabhat");  
    }  
}
```

OUTPUT:

Rajip Ranabhat

DISCUSSION AND CONCLUSION:

In this lab, a simple Java program was successfully written and executed using Visual Studio Code. The program helped in understanding the essential components of Java such as class definition and the main method.

TITLE: BASIC DATA TYPES IN JAVA.

OBJECTIVES:

- To understand different basic data types available in Java.

TOOLS USED:

- Visual Studio Code

PROBLEM ANALYSIS:

Java provides various basic data types such as int, float, double, char, and boolean to store different kinds of data. Each data type has its own size and range.

SOURCE CODE:

```
public class DataTypes {  
    public static void main (String[] args) {  
        int age = 20;  
        float height = 5.8f;  
        double salary = 25000.50;  
        char grade = 'A';  
        boolean isStudent = true;  
  
        System.out.println ("Age: " + age);  
        System.out.println ("Height: " + height);  
        System.out.println ("Salary: " + salary);  
        System.out.println ("Grade: " + grade);  
        System.out.println ("Is Student: " + isStudent);  
    }  
}
```

3 3

OUTPUT:

Age: 20

Height: 5.8

Salary: 25000.5

Grade: A

Is student: true

DISCUSSION AND CONCLUSION:

In this lab, different ~~base~~ data types of Java were used and displayed successfully. The program helped in understanding how primitive data types are declared, initialized, and printed.

TITLE: PROGRAM TO DEMONSTRATE TYPE CASTING IN JAVA.

OBJECTIVES:

- To understand the concept of type casting in Java.
- To perform implicit and explicit type casting using a Java program.

TOOLS USED:

- Visual Studio Code.

PROBLEM ANALYSIS:

Type casting in Java is the process of converting one data type into another. There are two types of type casting: implicit and explicit. Implicit casting is done automatically, while explicit casting requires manual conversion.

SOURCE CODE:

```
public class TypeCasting {  
    public static void main (String[] args) {  
        int a = 10;  
        double b = a;           // Implicit  
        double x = 20.75;  
        int y = (int) x;        // Explicit  
        System.out.println ("Implicit Casting:");  
        System.out.println ("Integer value: " + a);  
        System.out.println ("Double value: " + b);  
        System.out.println ("Explicit Casting:");  
        System.out.println ("Double value: " + x);  
        System.out.println ("Integer value: " + y);  
    }  
}
```

3 3

OUTPUT:

Implicit Casting:

Integer value: 10

Double value: 10.0

Explicit Casting:

Double value: 20.75

Integer value: 20

DISCUSSION AND CONCLUSION:

In this lab, both implicit and explicit type casting were successfully demonstrated. The program showed how data can be converted from one type to another.

TITLE: PROGRAM TO DEMONSTRATE LOOP USING FIZZ BUZZ IN JAVA

OBJECTIVES:

- To understand the use of loops in Java.
- To implement a Fizz Buzz program using conditional statements and loops.

TOOLS USED:

- Visual Studio Code

PROBLEM ANALYSIS:

Looping is used to execute a block of code repeatedly. In this program, a loop is used to iterate numbers from 1 to 50. For each number, if it is divisible by 3, "Fizz" is printed; if it is divisible by 5, "Buzz" is printed; and if divisible by both 3 and 5, "FizzBuzz" is printed. Otherwise, the number itself is displayed.

SOURCE CODE:

```
public class FizzBuzz {  
    public static void main (String[] args) {  
        for (int i = 1; i <= 50; i++) {  
            if (i % 3 == 0 && i % 5 == 0) {  
                System.out.println ("FizzBuzz");  
            } else if (i % 3 == 0) {  
                System.out.println ("Fizz");  
            } else if (i % 5 == 0) {  
                System.out.println ("Buzz");  
            } else {  
                System.out.println (i);  
            }  
        }  
    }  
}
```

3 3 3 3

OUTPUT:

1
2
Fizz
4
Buzz
Fizz
7
8
Fizz
Buzz
11
Fizz
13
14
FizzBuzz

DISCUSSION AND CONCLUSION:

In this lab, a loop was successfully used to implement the Fizz Buzz program. The program demonstrated how looping and conditional statements work together.

TITLE: PROGRAM TO DEMONSTRATE INHERITANCE IN JAVA.

OBJECTIVES:

- To understand the concept of inheritance in Java.
- To implement inheritance using a simple Java program.

TOOLS USED:

- Visual Studio Code

PROBLEM ANALYSIS:

Inheritance is a feature of OOP where one class acquires the properties and methods of another class. The parent class is called the superclass and the child class is called the subclass.

SOURCE CODE:

```
class Animal {  
    void eat() {  
        System.out.println("Animal is eating");  
    }  
}  
  
class Dog extends Animal {  
    void bark() {  
        System.out.println("Dog is barking");  
    }  
}  
  
public class Inheritance {  
    public static void main(String[] args) {  
        Dog d = new Dog();  
        d.eat();  
        d.bark();  
    }  
}
```

OUTPUT:

Animal is eating
Dog is barking

DISCUSSION AND CONCLUSION:

In this lab, inheritance was successfully demonstrated using superclass and subclass. The program showed how a child class can access methods of a parent class.

TITLE: PROGRAM TO DEMONSTRATE POLYMORPHISM IN JAVA.

OBJECTIVES:

- To understand the concept of polymorphism in Java.
- To implement method overloading (Compile-time polymorphism).
- To implement method overriding (Run-time polymorphism).

TOOLS USED:

- Visual Studio Code

PROBLEM ANALYSIS:

Polymorphism means "one name, many forms". It is achieved through method overloading (same method name with different parameters) and method overriding (same method with different implementation in a subclass).

SOURCE CODE:

Method Overriding

```
class Animal {  
    void sound() {  
        System.out.println("Animal makes sound");  
    }  
}
```

```
class Cat extends Animal {  
    void sound() {  
        System.out.println("Cat meows");  
    }  
}
```

```
public class Polymorphism {  
    public static void main (String[] args) {  
        Animal a = new Cat();  
        a.sound();  
    }  
}
```

Method Overloading

```
public class PolymorphismDemo {  
    int add (int a, int b) {  
        return a+b;  
    }  
}
```

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

```
public static void main (String[] args) {  
    PolymorphismDemo obj = new PolymorphismDemo();  
    System.out.println ("Sum of 2 integers: " +  
        obj.add (10, 20));  
    System.out.println ("Sum of 3 integers: " +  
        obj.add (20, 20, 30));  
}
```

OUTPUT:

Method Overriding

Cat meows

Method Overloading

sum of 2 integers: 30

sum of 3 integers: 60

DISCUSSION AND CONCLUSION:

In this lab, polymorphism was successfully demonstrated using both method overloading and method overriding, showing flexible behavior in Java.

TITLE: PROGRAM TO DEMONSTRATE EXCEPTION HANDLING USING TRY-CATCH IN JAVA.

OBJECTIVES:

- To understand exception handling in Java.
- To handle runtime errors using try-catch.

TOOLS USED:

- Visual Studio Code

PROBLEM ANALYSIS:

Exception handling is used to handle runtime errors so that the program does not crash. The try block contains risky code, and the catch block handles the exception. This lab demonstrates how errors like division by zero can be handled.

SOURCE CODE:

```
public class TryCatch {  
    public static void main (String[] args) {  
        try {  
            int a = 20, b = 0;  
            int result = a/b;  
            System.out.println("Result: " + result);  
        } catch (ArithmeticException e) {  
            System.out.println("Error: Division by  
            zero");  
        }  
        System.out.println("Program continues...")  
    }  
}
```

OUTPUT:

Errors: Division by zero
Program continues...

DISCUSSION AND CONCLUSION:

This lab demonstrated exception handling using try-catch. Errors were handled properly without stopping program execution.

TITLE: PROGRAM TO DEMONSTRATE JFrame AND JPanel IN JAVA.

OBJECTIVES:

- To understand GUI components in Java.
- To create a window using JFrame and JPanel.

TOOLS USED:

- IntelliJ IDEA

PROBLEM ANALYSIS:

JFrame is used to create the main window, while JPanel is used to organize components inside the frame. This lab demonstrates creating a simple GUI window with a panel and button.

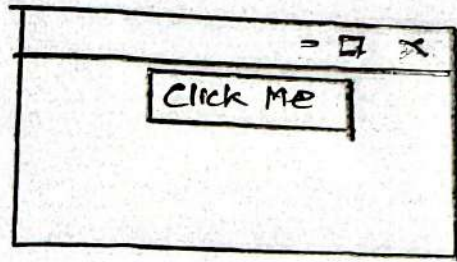
SOURCE CODE:

```
import javax.swing.*;

public class FramePanel {
    public static void main (String[] args) {
        JFrame frame = new JFrame();
        JPanel panel = new JPanel();
        JButton button = new JButton("Click Me");
        panel.add(button);
        frame.add(panel);
        frame.setSize(300,200);
        frame.setVisible(true);
    }
}
```

3 3

OUTPUT:



DISCUSSION AND CONCLUSION:

This lab demonstrated how JFrame and JPanel are used to create and organize GUI components in Java.

TITLE : PROGRAM TO DEMONSTRATE BORDER LAYOUT IN JAVA.

OBJECTIVES:

- To understand BorderLayout.
- To arrange components in five regions.

TOOLS USED:

- IntelliJ IDEA

PROBLEM ANALYSIS:

BorderLayout divides the container in 5 regions: North, South, East, West and Center. Components are added to these regions. This layout helps in organizing components clearly based on their position in the container.

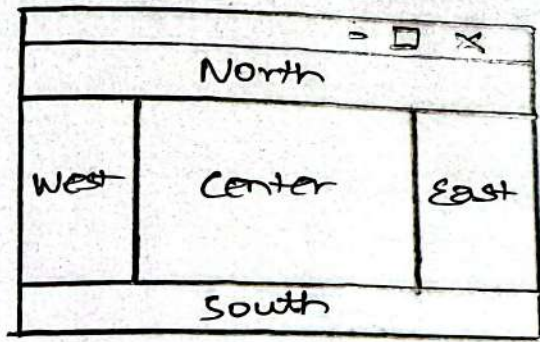
SOURCE CODE:

```
import javax.swing.*;
import java.awt.*;

public class BorderLayout {
    public static void main (String[] args) {
        JFrame frame = new JFrame();
        frame.setLayout (new BorderLayout());
        frame.add (new JButton ("North"),
            BorderLayout.NORTH);
        frame.add (new JButton ("South"),
            BorderLayout.SOUTH);
        frame.add (new JButton ("East"),
            BorderLayout.EAST);
        frame.add (new JButton ("West"),
            BorderLayout.WEST);
        frame.add (new JButton ("Center"),
            BorderLayout.CENTER);
        frame.setSize (300, 200);
        frame.setVisible (true);
    }
}
```

3

OUTPUT:



DISCUSSION AND CONCLUSION:

This lab demonstrated how BorderLayout arranges components in 5 different regions. It also showed how each region behaves differently in terms of size and positioning.

TITLE: PROGRAM TO DEMONSTRATE BOX LAYOUT IN JAVA.

OBJECTIVES:

- To understand Box Layout.
- To arrange components vertically or horizontally.

TOOLS USED:

- IntelliJ IDEA

PROBLEM ANALYSIS:

BoxLayout arranges components in a single row or column. It is useful for stacking components neatly. This layout is helpful when a simple vertical or horizontal arrangement is required.

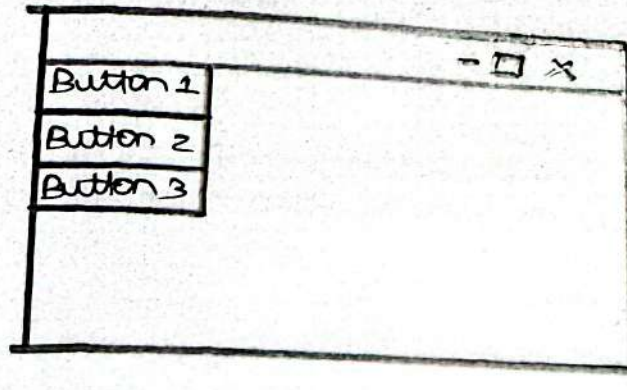
SOURCE CODE:

```
import javax.swing.*;
import java.awt.*;

public class BoxLayout {
    public static void main(String[] args) {
        JFrame frame = new JFrame();
        frame.setLayout(new BoxLayout(frame.
            getContentPane(), BoxLayout.Y_AXIS));
        frame.add(new JButton("Button 1"));
        frame.add(new JButton("Button 2"));
        frame.add(new JButton("Button 3"));
        frame.setSize(300, 200);
        frame.setVisible(true);
    }
}
```

3 3

OUTPUT:



DISCUSSION AND CONCLUSION:

This lab demonstrated how BoxLayout arranges components in a vertical or horizontal manner. It also helped in understanding alignment and spacing of components in a container.

TITLE: PROGRAM TO DEMONSTRATE GRIDLAYOUT IN JAVA.

OBJECTIVES:

- To understand GridLayout.
- To arrange components in rows and columns.

TOOLS USED:

- IntelliJ IDEA

PROBLEM ANALYSIS:

GridLayout arranges components in a grid format where all cells are equal in size. Each component occupies one cell. This layout is useful for creating structured and uniform interfaces.

SOURCE CODE:

```
import javax.swing.*;
import java.awt.*;

public class GridLayout {
    public static void main(String[] args) {
        JFrame frame = new JFrame();
        frame.setLayout(new GridLayout(2, 2));
        frame.add(new JButton("1"));
        frame.add(new JButton("2"));
        frame.add(new JButton("3"));
        frame.add(new JButton("4"));
        frame.setSize(300, 200);
        frame.setVisible(true);
    }
}
```

OUTPUT:

- □ x	
1	2
3	4

DISCUSSION AND CONCLUSION:

This lab demonstrated how GridLayout arranges components evenly in rows and columns. It also shows how equal spacing makes the interface simple and organized.

TITLE: PROGRAM TO DEMONSTRATE GRID BAG LAYOUT IN JAVA.

OBJECTIVES:

- TO understand GridBagLayout.
- TO arrange components using constraints.

TOOLS USED:

- INTELLIJ IDEA.

PROBLEM ANALYSIS:

GridBagLayout is a flexible layout manager that allows components of different sizes using constraints like grid position. It provides more control compared to other layouts.

SOURCE CODE:

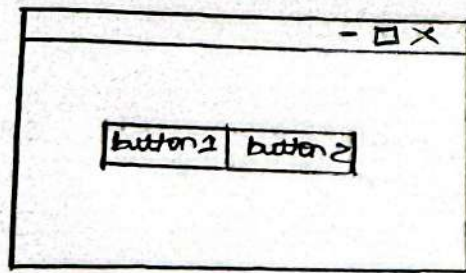
```
import javax.swing.*;
import java.awt.*;

public class GridBagLayout {
    public static void main (String[] args) {
        JFrame frame = new JFrame();
        frame.setSize (300, 300);
        frame.setLocationRelativeTo (null);
        frame.setLayout (new GridBagLayout());
        GridBagConstraints gbc = new GridBag-
            constraints();

        gbc.gridx = 0;
        gbc.gridy = 0;
        JButton button1 = new JButton ("button 1");
        frame.add (button1, gbc);

        gbc.gridx = 1;
        - gbc.gridy = 0;
        JButton button2 = new JButton ("button 2");
        frame.add (button2, gbc);
        frame.setVisible (true);
    }
}
```

OUTPUT:



DISCUSSION AND CONCLUSION:

This lab demonstrated flexible arrangement of components using GridBag Layout. It also showed how constraints help in controlling position and size of components.

TITLE: PROGRAM TO DEMONSTRATE FLOW LAYOUT IN JAVA.

OBJECTIVES:

- TO understand FlowLayout.
- TO arrange components in a flow manner.

TOOLS USED:

- IntelliJ IDEA.

PROBLEM ANALYSIS:

FlowLayout arranges components from left to right and moves to the next line when space is full. It is simple and commonly used layout. This layout behaves like text flow in a paragraph.

SOURCE CODE:

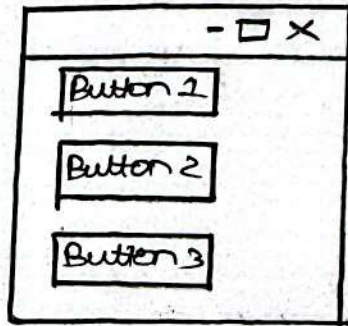
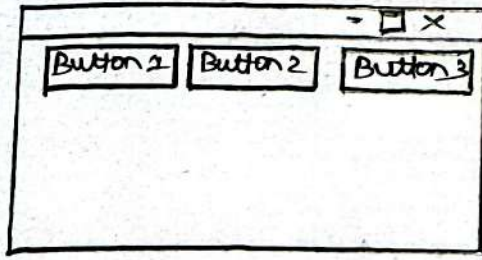
```
import javax.swing.*;
import java.awt.*;

public class FlowLayout {
    public static void main (String [] args) {
        JFrame frame = new JFrame();
        frame.setLayout (new FlowLayout());
        frame.add (new JButton ("Button 1"));
        frame.add (new JButton ("Button 2"));
        frame.add (new JButton ("Button 3"));

        frame.setSize (300, 200);
        frame.setVisible (true);
    }
}
```

3 3

OUTPUT:



A window with buttons arranged in a flowing manner.

DISCUSSION AND CONCLUSION;

This lab demonstrated how FlowLayout arranges components in a simple left to right flow. It also showed how components automatically adjust based on available space.