SRM Institute of Science and Technology

Department of Computer Applications

Delhi – Meerut Road, Sikri Kalan, Ghaziabad, Uttar Pradesh – 201204

Circular – 2023-24

BCA DS 1st Sem

Programming Using Java (UDS23101J)

Lab Manual

Laboratory 1: Learning to work with Java IDE and Writing Simple Conversion Programs

Title

Temperature Conversion Program (Celsius to Fahrenheit)

Aim

To write a Java program that converts temperature from Celsius to Fahrenheit using a Java IDE.

Procedure

1. Open your Java IDE (e.g., Eclipse or IntelliJ IDEA).
2. Create a new Java project named TemperatureConverter.
3. Create a new class named TempConverter in the project.
4. Write a program to accept a temperature in Celsius from the user and convert it to Fahrenheit using the formula: F = (C * 9/5) + 32.
5. Compile and run the program to test the conversion.

Source Code

java
```
import java.util.Scanner;

public class TempConverter {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter temperature in Celsius: ");
        double celsius = sc.nextDouble();
        double fahrenheit = (celsius * 9 / 5) + 32;
        System.out.println(celsius + "°C is equal to " + fahrenheit +
"°F");
        sc.close();
    }
}
```

Input

plain
```
Enter temperature in Celsius: 25
```

Expected Output

plain
```
25°C is equal to 77.0°F
```

Laboratory 2: Operators

Title

Arithmetic Operations Using Operators

Aim

To write a Java program that performs basic arithmetic operations (addition, subtraction, multiplication, division) using operators.

Procedure

1. Open your Java IDE.
2. Create a new Java project named ArithmeticOps.
3. Create a new class named ArithmeticCalculator.
4. Write a program to accept two numbers from the user and perform addition, subtraction, multiplication, and division.
5. Display the results of each operation.

Source Code

java
```
import java.util.Scanner;

public class ArithmeticCalculator {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter first number: ");
        double num1 = sc.nextDouble();
        System.out.print("Enter second number: ");
        double num2 = sc.nextDouble();

        System.out.println("Addition: " + (num1 + num2));
        System.out.println("Subtraction: " + (num1 - num2));
        System.out.println("Multiplication: " + (num1 * num2));
        System.out.println("Division: " + (num1 / num2));
        sc.close();
    }
}
```

Input

plain
```
Enter first number: 10
Enter second number: 5
```

Expected Output

plain
```
Addition: 15.0
Subtraction: 5.0
Multiplication: 50.0
Division: 2.0
```

Laboratory 3: Arrays, Control Statements

Title

Finding the Largest Element in an Array

Aim

To write a Java program that finds the largest element in an array using control statements.

Procedure

1. Open your Java IDE.
2. Create a new Java project named ArrayLargest.
3. Create a new class named LargestElement.
4. Write a program to accept an array of integers from the user and find the largest element using a loop.
5. Display the largest element.

Source Code

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

public class LargestElement {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the size of the array: ");
        int size = sc.nextInt();
        int[] arr = new int[size];

        System.out.println("Enter " + size + " elements:");
        for (int i = 0; i < size; i++) {
            arr[i] = sc.nextInt();
        }

        int largest = arr[0];
        for (int i = 1; i < size; i++) {
            if (arr[i] > largest) {
                largest = arr[i];
            }
        }
        System.out.println("Largest element: " + largest);
        sc.close();
    }
}
```

Input

plain
```
Enter the size of the array: 4
Enter 4 elements:
12 45 7 23
```

Expected Output

plain

```
Largest element: 45
```

Laboratory 4: Classes and Objects

Title

Creating a Student Class with Objects

Aim

To write a Java program that demonstrates the creation of a Student class and its objects.

Procedure

1. Open your Java IDE.
2. Create a new Java project named StudentClassDemo.
3. Create a new class named Student.
4. Define attributes (e.g., name, roll number) and a method to display student details.
5. Create objects of the Student class in the main method and display their details.

Source Code

java
```java
public class Student {
    String name;
    int rollNo;

    public Student(String name, int rollNo) {
        this.name = name;
        this.rollNo = rollNo;
    }

    public void display() {
        System.out.println("Name: " + name + ", Roll No: " + rollNo);
    }

    public static void main(String[] args) {
        Student s1 = new Student("Alice", 101);
        Student s2 = new Student("Bob", 102);
        s1.display();
        s2.display();
    }
}
```

Input

No user input required.

Expected Output

plain
```
Name: Alice, Roll No: 101
Name: Bob, Roll No: 102
```

Laboratory 5: Overloading Methods and Constructors, finalize() Method

Title

Method and Constructor Overloading with finalize() Method

Aim

To write a Java program that demonstrates method overloading, constructor overloading, and the use of the finalize() method.

Procedure

1. Open your Java IDE.
2. Create a new Java project named OverloadingDemo.
3. Create a new class named Shape.
4. Define overloaded constructors and methods to calculate the area of different shapes (e.g., circle, rectangle).
5. Override the finalize() method to display a message when the object is garbage collected.

Source Code

java
```java
public class Shape {
    Shape() {
        System.out.println("Default Shape constructor called");
    }

    Shape(double radius) {
        System.out.println("Area of Circle: " + calculateArea(radius));
    }

    Shape(double length, double breadth) {
        System.out.println("Area of Rectangle: " + calculateArea(length,
breadth));
    }

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

    double calculateArea(double length, double breadth) {
        return length * breadth;
    }

    @Override
    protected void finalize() throws Throwable {
        System.out.println("Shape object is being garbage collected");
        super.finalize();
    }

    public static void main(String[] args) {
        Shape circle = new Shape(5.0);
        Shape rectangle = new Shape(4.0, 6.0);
        circle = null;
        System.gc(); // Request garbage collection
    }
}
```

Input

No user input required.

Expected Output

plain

```
Area of Circle: 78.53981633974483
Area of Rectangle: 24.0
Shape object is being garbage collected
```

Laboratory 6: String Class, Command Line Arguments

Title

String Manipulation Using Command Line Arguments

Aim

To write a Java program that manipulates strings passed as command line arguments using the String class.

Procedure

1. Open your Java IDE.
2. Create a new Java project named StringCommandLine.
3. Create a new class named StringManipulator.
4. Write a program to accept strings as command line arguments, concatenate them, and display their length.
5. Run the program by passing arguments via the IDE's run configuration.

Source Code

java
```
public class StringManipulator {
    public static void main(String[] args) {
        if (args.length < 2) {
            System.out.println("Please provide at least two strings as 
arguments");
            return;
        }

        String concatenated = args[0] + " " + args[1];
        System.out.println("Concatenated String: " + concatenated);
        System.out.println("Length of Concatenated String: " + 
concatenated.length());
    }
}
```

Input

Command line arguments: Hello World

Expected Output

plain
```
Concatenated String: Hello World
Length of Concatenated String: 11
```

Laboratory 7: Inheritance, Method Overriding, Abstract Classes and Methods

Title

Demonstrating Inheritance and Method Overriding with Abstract Classes

Aim

To write a Java program that demonstrates inheritance, method overriding, and the use of abstract classes and methods.

Procedure

1. Open your Java IDE.
2. Create a new Java project named InheritanceDemo.
3. Create an abstract class Animal with an abstract method sound().
4. Create two subclasses Dog and Cat that inherit from Animal and override the sound() method.
5. Create objects of Dog and Cat in the main method and call their sound() methods.

Source Code

java
```java
abstract class Animal {
    abstract void sound();
}

class Dog extends Animal {
    @Override
    void sound() {
        System.out.println("Dog barks");
    }
}

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

public class InheritanceDemo {
    public static void main(String[] args) {
        Animal dog = new Dog();
        Animal cat = new Cat();
        dog.sound();
        cat.sound();
    }
}
```

Input

No user input required.

Expected Output

plain

```
Dog barks
Cat meows
```

Laboratory 8: Packages and Interfaces

Title

Using Packages and Interfaces for a Calculator

Aim

To write a Java program that demonstrates the use of packages and interfaces by creating a simple calculator.

Procedure

1. Open your Java IDE.
2. Create a new Java project named CalculatorPackageDemo.
3. Create a package named calc.
4. Define an interface Operations with methods for addition and subtraction.
5. Create a class Calculator in the calc package that implements the Operations interface.
6. Write a main class to test the calculator.

Source Code

java
```java
package calc;

interface Operations {
    int add(int a, int b);
    int subtract(int a, int b);
}

class Calculator implements Operations {
    public int add(int a, int b) {
        return a + b;
    }

    public int subtract(int a, int b) {
        return a - b;
    }
}

public class CalculatorDemo {
    public static void main(String[] args) {
        calc.Calculator calc = new calc.Calculator();
        System.out.println("Addition: " + calc.add(10, 5));
        System.out.println("Subtraction: " + calc.subtract(10, 5));
    }
}
```

Input

No user input required.

Expected Output

plain

```
Addition: 15
Subtraction: 5
```

Laboratory 9: Exception Handling

Title

Handling Arithmetic and Array Index Exceptions

Aim

To write a Java program that demonstrates exception handling for arithmetic and array index out of bounds exceptions.

Procedure

1. Open your Java IDE.
2. Create a new Java project named ExceptionHandlingDemo.
3. Create a new class named ExceptionDemo.
4. Write a program that attempts to divide a number by zero and access an invalid array index.
5. Use try-catch blocks to handle the exceptions and display appropriate messages.

Source Code

java
```
public class ExceptionDemo {
    public static void main(String[] args) {
        int[] arr = {1, 2, 3};
        try {
            // Arithmetic exception
            int result = 10 / 0;
            System.out.println("Result: " + result);
        } catch (ArithmeticException e) {
            System.out.println("Error: Division by zero");
        }

        try {
            // Array index out of bounds
            System.out.println(arr[5]);
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Error: Invalid array index");
        }
    }
}
```

Input

No user input required.

Expected Output

plain
```
Error: Division by zero
Error: Invalid array index
```

Laboratory 10: Multi-threading

Title

Creating Multiple Threads for Counting

Aim

To write a Java program that demonstrates multi-threading by creating two threads to count numbers in different ranges.

Procedure

1. Open your Java IDE.
2. Create a new Java project named MultiThreadDemo.
3. Create a new class named Counter that extends Thread.
4. Write a program to create two threads: one to count from 1 to 5, and another to count from 6 to 10.
5. Start both threads and observe the output.

Source Code

java
```
class Counter extends Thread {
    int start, end;

    Counter(String name, int start, int end) {
        super(name);
        this.start = start;
        this.end = end;
    }

    public void run() {
        for (int i = start; i <= end; i++) {
            System.out.println(getName() + ": " + i);
            try {
                Thread.sleep(500);
            } catch (InterruptedException e) {
                System.out.println("Thread interrupted");
            }
        }
    }
}

public class MultiThreadDemo {
    public static void main(String[] args) {
        Counter thread1 = new Counter("Thread-1", 1, 5);
        Counter thread2 = new Counter("Thread-2", 6, 10);
        thread1.start();
        thread2.start();
    }
}
```

Input

No user input required.

Expected Output

plain
```
Thread-1: 1
Thread-2: 6
Thread-1: 2
Thread-2: 7
Thread-1: 3
Thread-2: 8
Thread-1: 4
Thread-2: 9
Thread-1: 5
Thread-2: 10
```

(Note: The order of thread execution may vary due to thread scheduling.)

Laboratory 11: Legacy Classes and Interfaces

Title

Using Vector (Legacy Class) for Dynamic Array

Aim

To write a Java program that demonstrates the use of the legacy Vector class to store and manipulate a dynamic array.

Procedure

1. Open your Java IDE.
2. Create a new Java project named LegacyVectorDemo.
3. Create a new class named VectorDemo.
4. Write a program to create a Vector, add elements, and display its contents.
5. Use methods like addElement() and elementAt() to manipulate the Vector.

Source Code

java
```java
import java.util.Vector;

public class VectorDemo {
    public static void main(String[] args) {
        Vector<Integer> vector = new Vector<>();
        vector.addElement(10);
        vector.addElement(20);
        vector.addElement(30);

        System.out.println("Vector elements:");
        for (int i = 0; i < vector.size(); i++) {
            System.out.println(vector.elementAt(i));
        }
    }
}
```

Input

No user input required.

Expected Output

plain
```
Vector elements:
10
20
30
```

Laboratory 12: Utility Classes and Simple Applet Programs

Title

Creating a Simple Applet to Display a Message

Aim

To write a Java program that creates a simple applet to display a message using the Applet class.

Procedure

1. Open your Java IDE.
2. Create a new Java project named SimpleAppletDemo.
3. Create a new class named MessageApplet that extends Applet.
4. Override the paint() method to display a message on the applet window.
5. Create an HTML file to load the applet (Note: Applets are deprecated; this is for educational purposes).

Source Code

java
```java
import java.applet.Applet;
import java.awt.Graphics;

public class MessageApplet extends Applet {
    public void paint(Graphics g) {
        g.drawString("Hello, Welcome to Java Applet!", 50, 50);
    }
}
```

Input

No user input required.

Expected Output

A graphical window displaying:

plain
```
Hello, Welcome to Java Applet!
```

(Note: Applets require a browser or applet viewer to run, which may not be supported in modern environments.)

Laboratory 13: Event Handling

Title

Handling Mouse Click Events in an Applet

Aim

To write a Java program that demonstrates event handling by detecting mouse clicks in an applet.

Procedure

1. Open your Java IDE.
2. Create a new Java project named EventHandlingApplet.
3. Create a new class named MouseClickApplet that extends Applet.
4. Implement the MouseListener interface to handle mouse click events.
5. Display a message when the mouse is clicked.

Source Code

java
```java
import java.applet.Applet;
import java.awt.Graphics;
import java.awt.event.MouseListener;
import java.awt.event.MouseEvent;

public class MouseClickApplet extends Applet implements MouseListener {
    String message = "";

    public void init() {
        addMouseListener(this);
    }

    public void paint(Graphics g) {
        g.drawString(message, 50, 50);
    }

    public void mouseClicked(MouseEvent e) {
        message = "Mouse clicked at: " + e.getX() + ", " + e.getY();
        repaint();
    }

    public void mousePressed(MouseEvent e) {}
    public void mouseReleased(MouseEvent e) {}
    public void mouseEntered(MouseEvent e) {}
    public void mouseExited(MouseEvent e) {}
}
```

Input

Click the mouse at coordinates (100, 120).

Expected Output

A graphical window displaying:

```plain
Mouse clicked at: 100, 120
```

(Note: Applets require a browser or applet viewer to run, which may not be supported in modern environments.)

Laboratory 14: AWT Controls

Title

Creating a Simple AWT Application with Buttons

Aim

To write a Java program that creates a simple AWT application with buttons and handles button click events.

Procedure

1. Open your Java IDE.
2. Create a new Java project named AWTButtonDemo.
3. Create a new class named ButtonDemo that extends Frame.
4. Add two buttons to the frame and handle their click events to display a message.
5. Set up the frame and make it visible.

Source Code

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

public class ButtonDemo extends Frame implements ActionListener {
    Button btn1, btn2;
    Label label;

    ButtonDemo() {
        setLayout(new FlowLayout());
        btn1 = new Button("Click Me");
        btn2 = new Button("Clear");
        label = new Label("Click a button");

        add(btn1);
        add(btn2);
        add(label);

        btn1.addActionListener(this);
        btn2.addActionListener(this);

        setSize(300, 200);
        setVisible(true);

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

    public void actionPerformed(ActionEvent e) {
        if (e.getSource() == btn1) {
            label.setText("Button 1 Clicked!");
        } else if (e.getSource() == btn2) {
            label.setText("Cleared!");
        }
    }

    public static void main(String[] args) {
        new ButtonDemo();
    }
}
```

Input

Click the "Click Me" button, then the "Clear" button.

Expected Output

- After clicking "Click Me": Label displays Button 1 Clicked!
- After clicking "Clear": Label displays Cleared!

Laboratory 15: Layout Managers, Byte and Character Streams

Title

Using Layout Managers and Character Streams

Aim

To write a Java program that demonstrates the use of layout managers in AWT and character streams to read from a file.

Procedure

1. Open your Java IDE.
2. Create a new Java project named LayoutStreamDemo.
3. Create a new class named LayoutStreamDemo that extends Frame.
4. Use BorderLayout to arrange AWT components (e.g., labels, text fields).
5. Simulate reading from a file using a StringReader (since actual file I/O may not be supported in some environments).
6. Display the read content in a text area.

Source Code

java
```
import java.awt.*;
import java.io.*;

public class LayoutStreamDemo extends Frame {
    TextArea textArea;

    LayoutStreamDemo() {
        setLayout(new BorderLayout());
        Label label = new Label("File Content:");
        textArea = new TextArea(10, 30);

        add(label, BorderLayout.NORTH);
        add(textArea, BorderLayout.CENTER);

        // Simulate file content using StringReader
        String simulatedFileContent = "This is a sample file content.\nLine
2 of the file.";
        try (StringReader sr = new StringReader(simulatedFileContent);
             BufferedReader br = new BufferedReader(sr)) {
            String line;
            while ((line = br.readLine()) != null) {
                textArea.append(line + "\n");
            }
        } catch (IOException e) {
            textArea.append("Error reading content: " + e.getMessage());
        }

        setSize(400, 300);
        setVisible(true);

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

    public static void main(String[] args) {
        new LayoutStreamDemo();
    }
}
```

Input

No user input required (simulated file content).

Expected Output

A window with a text area displaying:

plain
```
This is a sample file content.
Line 2 of the file.
```