SRM Institute of Science and Technology

Department of Computer Applications

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

Circular – 2020-21

MCA GAI 1st semester

Object Oriented Programming using Java (PGI20C01J)

Lab Manual

Lab 1: Discussing the real-world examples of OOP concepts and applications

Title

Real-World Examples of OOP Concepts and Applications

Aim

To understand the fundamental concepts of Object-Oriented Programming (OOP) and their applications in real-world scenarios.

Procedure

1. **Introduction to OOP:** Briefly discuss the paradigm shift from procedural to object-oriented programming.
2. **Core OOP Concepts:**
    - **Encapsulation:** Explain how data and methods are bundled together within a class, and access to data is restricted, promoting data hiding. Provide a real-world example like a "Car" object where engine details (data) are encapsulated within the car, and interaction happens via methods like `start()` or `accelerate()`.
    - **Inheritance:** Describe how new classes can inherit properties and behaviors from existing classes, promoting code reusability. Illustrate with an example like `Dog` and `Cat` classes inheriting from an `Animal` class, sharing common attributes (e.g., `name`, `age`) and methods (e.g., `eat()`).
    - **Polymorphism:** Explain the ability of an object to take on many forms. Discuss method overloading (same method name, different parameters) and method overriding (subclass providing a specific implementation for a method defined in its superclass). Use an example where different `Shape` objects (Circle, Rectangle) can all have a `draw()` method, but their implementations differ.
    - **Abstraction:** Define abstraction as the process of hiding complex implementation details and showing only the necessary features of an object. Discuss abstract classes and interfaces. A real-world example could be a "Remote Control" where you interact with buttons (abstract actions) without knowing the complex circuitry inside.
3. **Real-World Applications:**
    - **Software Development:** How OOP is used in building large, complex software systems, making them modular, maintainable, and scalable.

- **Game Development:** Representing game entities (characters, items, enemies) as objects.
- **Database Systems:** Modeling real-world entities as objects that can be stored and retrieved from databases.
- **UI/UX Design:** Components in graphical user interfaces (buttons, text fields, windows) are often objects.

Source Code

N/A (Conceptual discussion)

Input

N/A

Expected Output

A clear understanding and discussion of OOP concepts with relevant real-world examples.

Lab 2: Implement a program to demonstrate the use of casting for converting data types

Title

Data Type Casting in Java

Aim

To implement a Java program that demonstrates both widening (implicit) and narrowing (explicit) data type casting.

Procedure

1. **Widening Conversion (Implicit Casting):**
    - Declare a variable of a smaller data type (e.g., `int`).
    - Assign its value to a variable of a larger data type (e.g., `double`). Observe that no explicit cast is required.
2. **Narrowing Conversion (Explicit Casting):**
    - Declare a variable of a larger data type (e.g., `double`).
    - Assign its value to a variable of a smaller data type (e.g., `int`). Observe that an explicit cast `(targetType)` is required.
    - Demonstrate potential data loss during narrowing conversion.
3. **Object Casting (Upcasting and Downcasting):**
    - (Optional, for advanced understanding) Create a simple class hierarchy (e.g., `Animal` and `Dog`).
    - Demonstrate upcasting (assigning a subclass object to a superclass reference) which is implicit.
    - Demonstrate downcasting (assigning a superclass reference to a subclass object) which requires explicit casting and a runtime check (`instanceof`).

Source Code

```
// Filename: DataTypeCasting.java
public class DataTypeCasting {
    public static void main(String[] args) {
        // --- Widening Conversion (Implicit Casting) ---
        System.out.println("--- Widening Conversion (Implicit) ---");
        int myInt = 100;
        long myLong = myInt; // int to long (implicit)
        float myFloat = myLong; // long to float (implicit)
        double myDouble = myFloat; // float to double (implicit)

        System.out.println("Original int: " + myInt);
        System.out.println("int to long: " + myLong);
        System.out.println("long to float: " + myFloat);
        System.out.println("float to double: " + myDouble);
        System.out.println();

        // --- Narrowing Conversion (Explicit Casting) ---
        System.out.println("--- Narrowing Conversion (Explicit) ---");
        double largeDouble = 123.456;
        int convertedInt = (int) largeDouble; // double to int (explicit)
        System.out.println("Original double: " + largeDouble);
```

```
        System.out.println("double to int (data loss expected): " +
convertedInt); // Decimal part is truncated

        float largeFloat = 300.75f;
        byte convertedByte = (byte) largeFloat; // float to byte (explicit)
        System.out.println("Original float: " + largeFloat);
        System.out.println("float to byte (potential overflow/underflow): " +
convertedByte); // Value might wrap around

        char myChar = 'A';
        int charToInt = myChar; // char to int (implicit, ASCII value)
        System.out.println("Original char: " + myChar);
        System.out.println("char to int: " + charToInt);

        int intToChar = 66;
        char convertedChar = (char) intToChar; // int to char (explicit, ASCII
value)
        System.out.println("Original int: " + intToChar);
        System.out.println("int to char: " + convertedChar);
    }
}
```

Input

No specific input is required as values are hardcoded in the program.

Expected Output

```
--- Widening Conversion (Implicit) ---
Original int: 100
int to long: 100
long to float: 100.0
float to double: 100.0

--- Narrowing Conversion (Explicit) ---
Original double: 123.456
double to int (data loss expected): 123
Original float: 300.75
float to byte (potential overflow/underflow): 44
Original char: A
char to int: 65
Original int: 66
int to char: B
```

Lab 3: Implement the concept of String Handling functions

Title

String Handling Functions in Java

Aim

To implement and demonstrate various commonly used String handling functions in Java.

Procedure

1. **String Declaration and Initialization:** Declare String variables using literal and `new` keyword.
2. **Basic Operations:**
    - `length()`: Get the length of a string.
    - `charAt(int index)`: Get the character at a specific index.
    - `substring(int beginIndex)` / `substring(int beginIndex, int endIndex)`: Extract substrings.
3. **Comparison and Search:**
    - `equals(Object another)` / `equalsIgnoreCase(String another)`: Compare strings for equality.
    - `indexOf(String str)` / `lastIndexOf(String str)`: Find the first/last occurrence of a substring.
    - `startsWith(String prefix)` / `endsWith(String suffix)`: Check if a string starts/ends with a prefix/suffix.
    - `contains(CharSequence s)`: Check if a string contains a sequence of characters.
4. **Modification and Concatenation:**
    - `concat(String str)`: Concatenate strings.
    - `replace(char oldChar, char newChar)` / `replace(CharSequence target, CharSequence replacement)`: Replace characters or substrings.
    - `trim()`: Remove leading and trailing whitespace.
5. **Case Conversion:**
    - `toUpperCase()`: Convert to uppercase.
    - `toLowerCase()`: Convert to lowercase.
6. **Splitting Strings:**
    - `split(String regex)`: Split a string into an array of strings based on a delimiter.

Source Code

```
// Filename: StringHandling.java
public class StringHandling {
    public static void main(String[] args) {
        String str1 = "Hello, Java!";
        String str2 = "hello, java!";
        String str3 = "  Programming is fun.  ";

        System.out.println("--- Basic String Operations ---");
        System.out.println("str1: \"" + str1 + "\"");
        System.out.println("Length of str1: " + str1.length());
        System.out.println("Character at index 7 in str1: " + str1.charAt(7));
        System.out.println("Substring of str1 from index 7: " +
str1.substring(7));
```

```
        System.out.println("Substring of str1 from index 0 to 5: " +
str1.substring(0, 5));
        System.out.println();

        System.out.println("--- String Comparison and Search ---");
        System.out.println("str1 equals str2: " + str1.equals(str2));
        System.out.println("str1 equalsIgnoreCase str2: " +
str1.equalsIgnoreCase(str2));
        System.out.println("Index of 'Java' in str1: " + str1.indexOf("Java"));
        System.out.println("Index of 'o' from end in str1: " +
str1.lastIndexOf("o"));
        System.out.println("str1 starts with 'Hello': " +
str1.startsWith("Hello"));
        System.out.println("str1 ends with 'Java!': " + str1.endsWith("Java!"));
        System.out.println("str1 contains 'llo': " + str1.contains("llo"));
        System.out.println();

        System.out.println("--- String Modification and Concatenation ---");
        String concatenatedStr = str1.concat(" Welcome!");
        System.out.println("Concatenated string: " + concatenatedStr);
        String replacedStr = str1.replace('o', 'X');
        System.out.println("Replaced 'o' with 'X' in str1: " + replacedStr);
        String trimmedStr = str3.trim();
        System.out.println("Original str3: \"" + str3 + "\"");
        System.out.println("Trimmed str3: \"" + trimmedStr + "\"");
        System.out.println();

        System.out.println("--- Case Conversion ---");
        System.out.println("str1 to Uppercase: " + str1.toUpperCase());
        System.out.println("str1 to Lowercase: " + str1.toLowerCase());
        System.out.println();

        System.out.println("--- Splitting Strings ---");
        String sentence = "Java is a powerful programming language.";
        String[] words = sentence.split(" ");
        System.out.println("Original sentence: \"" + sentence + "\"");
        System.out.println("Words after splitting by space:");
        for (String word : words) {
            System.out.println(word);
        }
    }
}
```

Input

No specific input is required as strings are hardcoded in the program.

Expected Output

```
--- Basic String Operations ---
str1: "Hello, Java!"
Length of str1: 12
Character at index 7 in str1: J
Substring of str1 from index 7: Java!
Substring of str1 from index 0 to 5: Hello

--- String Comparison and Search ---
str1 equals str2: false
str1 equalsIgnoreCase str2: true
Index of 'Java' in str1: 7
Index of 'o' from end in str1: 7
str1 starts with 'Hello': true
str1 ends with 'Java!': true
str1 contains 'llo': true
```

```
--- String Modification and Concatenation ---
Concatenated string: Hello, Java! Welcome!
Replaced 'o' with 'X' in str1: HellX, Java!
Original str3: "  Programming is fun.  "
Trimmed str3: "Programming is fun."

--- Case Conversion ---
str1 to Uppercase: HELLO, JAVA!
str1 to Lowercase: hello, java!

--- Splitting Strings ---
Original sentence: "Java is a powerful programming language."
Words after splitting by space:
Java
is
a
powerful
programming
language.
```

Lab 4: Demonstrate Multi Level Inheritance Implement Dynamic Method Dispatch

Title

Multi-Level Inheritance and Dynamic Method Dispatch

Aim

To demonstrate the concepts of multi-level inheritance and dynamic method dispatch (runtime polymorphism) in Java.

Procedure

1. **Multi-Level Inheritance:**
    - Create a `Grandparent` class with a method (e.g., `displayGrandparent()`).
    - Create a `Parent` class that extends `Grandparent` and adds its own method (e.g., `displayParent()`).
    - Create a `Child` class that extends `Parent` and adds its own method (e.g., `displayChild()`).
    - Show how a `Child` object can access methods from `Grandparent` and `Parent`.
2. **Dynamic Method Dispatch:**
    - In the `Grandparent`, `Parent`, and `Child` classes, define a method with the same signature (e.g., `showInfo()`). This demonstrates method overriding.
    - In the `main` method, create references of the `Grandparent` type.
    - Assign objects of `Parent` and `Child` classes to these `Grandparent` references.
    - Call the overridden method (`showInfo()`) using these `Grandparent` references. Observe that the actual method executed depends on the type of the object being referred to at runtime, not the type of the reference variable.

Source Code

```
// Filename: InheritanceAndPolymorphism.java

// Grandparent class
class Grandparent {
    void displayGrandparent() {
        System.out.println("This is the Grandparent class.");
    }

    void showInfo() {
        System.out.println("Grandparent's showInfo method.");
    }
}

// Parent class extending Grandparent (first level of inheritance)
class Parent extends Grandparent {
    void displayParent() {
        System.out.println("This is the Parent class.");
    }

    // Overriding showInfo method
    @Override
    void showInfo() {
        System.out.println("Parent's showInfo method.");
```

```
    }
}

// Child class extending Parent (second level of inheritance - multi-level)
class Child extends Parent {
    void displayChild() {
        System.out.println("This is the Child class.");
    }

    // Overriding showInfo method
    @Override
    void showInfo() {
        System.out.println("Child's showInfo method.");
    }
}

public class InheritanceAndPolymorphism {
    public static void main(String[] args) {
        System.out.println("--- Multi-Level Inheritance Demonstration ---");
        Child c = new Child();
        c.displayGrandparent(); // Method from Grandparent
        c.displayParent();      // Method from Parent
        c.displayChild();       // Method from Child
        System.out.println();

        System.out.println("--- Dynamic Method Dispatch Demonstration ---");

        Grandparent gpRef; // Reference of Grandparent type

        // Case 1: Grandparent reference referring to Grandparent object
        gpRef = new Grandparent();
        gpRef.showInfo(); // Calls Grandparent's showInfo()

        // Case 2: Grandparent reference referring to Parent object
        gpRef = new Parent();
        gpRef.showInfo(); // Calls Parent's showInfo() (Dynamic Method Dispatch)

        // Case 3: Grandparent reference referring to Child object
        gpRef = new Child();
        gpRef.showInfo(); // Calls Child's showInfo() (Dynamic Method Dispatch)

        System.out.println("\nNote: In Dynamic Method Dispatch, the method to be
executed is determined at runtime based on the actual object type, not the
reference type.");
    }
}
```

Input

No specific input is required.

Expected Output

```
--- Multi-Level Inheritance Demonstration ---
This is the Grandparent class.
This is the Parent class.
This is the Child class.

--- Dynamic Method Dispatch Demonstration ---
Grandparent's showInfo method.
Parent's showInfo method.
Child's showInfo method.
```

Note: In Dynamic Method Dispatch, the method to be executed is determined at runtime based on the actual object type, not the reference type.

Lab 5: Write a Java program that demonstrates the use of try-catch blocks to handle built-in exceptions

Title

Exception Handling with `try-catch` Blocks

Aim

To write a Java program that demonstrates the use of `try-catch` blocks to gracefully handle common built-in exceptions.

Procedure

1. **`ArithmeticException`:**
    - Create a scenario where division by zero occurs.
    - Enclose the problematic code within a `try` block.
    - Provide a `catch` block specifically for `ArithmeticException` to handle the error and print a user-friendly message.
2. **`ArrayIndexOutOfBoundsException`:**
    - Create an array and attempt to access an index that is out of its bounds.
    - Enclose this code within a `try` block.
    - Add a `catch` block for `ArrayIndexOutOfBoundsException` to manage the error.
3. **`NumberFormatException`:**
    - Attempt to convert a non-numeric string to an integer using `Integer.parseInt()`.
    - Place this in a `try` block.
    - Catch `NumberFormatException` to inform the user about invalid input.
4. **Multiple `catch` blocks:** Demonstrate how to use multiple `catch` blocks for different exception types.
5. **`finally` block (Optional but Recommended):** Explain and demonstrate the `finally` block, which executes regardless of whether an exception occurred or was caught.

Source Code

```
// Filename: ExceptionHandling.java
import java.util.InputMismatchException;
import java.util.Scanner;

public class ExceptionHandling {
    public static void main(String[] args) {
        // --- Demonstrating ArithmeticException ---
        System.out.println("--- ArithmeticException Example ---");
        try {
            int numerator = 10;
            int denominator = 0; // This will cause an ArithmeticException
            int result = numerator / denominator;
            System.out.println("Result of division: " + result); // This line
will not be executed
        } catch (ArithmeticException e) {
            System.err.println("Error: Cannot divide by zero! " +
e.getMessage());
        } finally {
            System.out.println("Arithmetic operation attempt finished.\n");
        }
```

```
        // --- Demonstrating ArrayIndexOutOfBoundsException ---
        System.out.println("--- ArrayIndexOutOfBoundsException Example ---");
        int[] numbers = {1, 2, 3, 4, 5};
        try {
            System.out.println("Accessing element at index 2: " + numbers[2]);
            System.out.println("Accessing element at index 10: " + numbers[10]);
// This will cause an exception
            System.out.println("This line will not be executed.");
        } catch (ArrayIndexOutOfBoundsException e) {
            System.err.println("Error: Array index is out of bounds! " +
e.getMessage());
        } finally {
            System.out.println("Array access attempt finished.\n");
        }

        // --- Demonstrating NumberFormatException ---
        System.out.println("--- NumberFormatException Example ---");
        String strNum = "abc";
        try {
            int parsedInt = Integer.parseInt(strNum);
            System.out.println("Parsed integer: " + parsedInt); // This line
will not be executed
        } catch (NumberFormatException e) {
            System.err.println("Error: Invalid number format for parsing! " +
e.getMessage());
        } finally {
            System.out.println("String to integer conversion attempt
finished.\n");
        }

        // --- Demonstrating InputMismatchException with Scanner ---
        System.out.println("--- InputMismatchException Example ---");
        Scanner scanner = new Scanner(System.in);
        try {
            System.out.print("Enter an integer: ");
            int userInput = scanner.nextInt(); // Expects an integer
            System.out.println("You entered: " + userInput);
        } catch (InputMismatchException e) {
            System.err.println("Error: Invalid input. Please enter an integer. "
+ e.getMessage());
        } finally {
            System.out.println("User input attempt finished.");
            scanner.close(); // Always close resources in finally block
        }
    }
}
```

Input

For the `InputMismatchException` example, you will be prompted to enter an integer.

- **Valid Input:** `123`
- **Invalid Input:** `abc`

Expected Output

```
--- ArithmeticException Example ---
Error: Cannot divide by zero! / by zero
Arithmetic operation attempt finished.

--- ArrayIndexOutOfBoundsException Example ---
Accessing element at index 2: 3
```

```
Error: Array index is out of bounds! Index 10 out of bounds for length 5
Array access attempt finished.

--- NumberFormatException Example ---
Error: Invalid number format for parsing! For input string: "abc"
String to integer conversion attempt finished.

--- InputMismatchException Example ---
Enter an integer: abc  (User enters 'abc' here)
Error: Invalid input. Please enter an integer. null
User input attempt finished.
```

(If valid input `123` is given for `InputMismatchException`)

```
--- InputMismatchException Example ---
Enter an integer: 123
You entered: 123
User input attempt finished.
```

Lab 6: Develop a program that converts the character encoding of a text file using Input Stream Reader and Output Stream Writer.

Title

Character Encoding Conversion of a Text File

Aim

To develop a Java program that reads a text file with a specified character encoding and writes its content to a new file with a different character encoding, utilizing `InputStreamReader` and `OutputStreamWriter`.

Procedure

1. **Prepare Input File:** Create a sample text file (e.g., `input.txt`) with some content. Ensure it's saved with a specific encoding (e.g., UTF-8).
2. **Define Encodings:** Specify the input and output character encodings (e.g., "UTF-8" for input, "UTF-16" for output).
3. **Read with `InputStreamReader`:**
    - Open a `FileInputStream` for the input file.
    - Wrap it with an `InputStreamReader`, specifying the input encoding. This converts bytes from the file into characters using the given encoding.
4. **Write with `OutputStreamWriter`:**
    - Open a `FileOutputStream` for the output file.
    - Wrap it with an `OutputStreamWriter`, specifying the desired output encoding. This converts characters into bytes using the given encoding before writing to the file.
5. **Read and Write Loop:** Read characters from the `InputStreamReader` and write them to the `OutputStreamWriter` character by character or in chunks.
6. **Close Resources:** Ensure all streams are properly closed in a `finally` block or using try-with-resources.
7. **Verification:** After running the program, open the output file with a text editor that allows checking file encoding to verify the conversion.

Source Code

```
// Filename: FileEncodingConverter.java
import java.io.*;
import java.nio.charset.StandardCharsets;

public class FileEncodingConverter {

    public static void main(String[] args) {
        String inputFileName = "input.txt";
        String outputFileName = "output.txt";
        String inputEncoding = StandardCharsets.UTF_8.name(); // Example: UTF-8
        String outputEncoding = StandardCharsets.UTF_16.name(); // Example: UTF-
16

        // 1. Create a dummy input.txt file for demonstration
        createDummyInputFile(inputFileName, inputEncoding);
```

```
        System.out.println("Attempting to convert file encoding from " +
inputEncoding + " to " + outputEncoding);
        System.out.println("Reading from: " + inputFileName);
        System.out.println("Writing to: " + outputFileName);

        try (
            // Setup InputStreamReader for reading with specified input encoding
            FileInputStream fis = new FileInputStream(inputFileName);
            InputStreamReader isr = new InputStreamReader(fis, inputEncoding);

            // Setup OutputStreamWriter for writing with specified output
encoding
            FileOutputStream fos = new FileOutputStream(outputFileName);
            OutputStreamWriter osw = new OutputStreamWriter(fos,
outputEncoding);

            // Use a BufferedReader for efficient character reading
            BufferedReader reader = new BufferedReader(isr);
            // Use a BufferedWriter for efficient character writing
            BufferedWriter writer = new BufferedWriter(osw)
        ) {
            String line;
            while ((line = reader.readLine()) != null) {
                writer.write(line);
                writer.newLine(); // Write newline character
            }
            System.out.println("File encoding conversion successful!");

        } catch (IOException e) {
            System.err.println("An I/O error occurred: " + e.getMessage());
            e.printStackTrace();
        }
    }

    // Helper method to create a dummy input file
    private static void createDummyInputFile(String fileName, String encoding) {
        try (FileOutputStream fos = new FileOutputStream(fileName);
             OutputStreamWriter osw = new OutputStreamWriter(fos, encoding);
             BufferedWriter writer = new BufferedWriter(osw)) {
            writer.write("This is a test string with some special characters:");
            writer.newLine();
            writer.write("你好 (Chinese), こんにちは (Japanese), 안녕하세요
(Korean)");
            writer.newLine();
            writer.write("ÄÖÜäöüß (German), éàç (French)");
            writer.newLine();
            System.out.println("Dummy input file '" + fileName + "' created with
" + encoding + " encoding.");
        } catch (IOException e) {
            System.err.println("Error creating dummy input file: " +
e.getMessage());
            e.printStackTrace();
        }
    }
}
```

Input

A file named `input.txt` will be automatically created by the program with UTF-8 encoding for demonstration purposes. Its content will be:

```
This is a test string with some special characters:
你好 (Chinese), こんにちは (Japanese), 안녕하세요 (Korean)
```

```
ÄÖÜäöüß (German), éàç (French)
```

Expected Output

Console Output:

```
Dummy input file 'input.txt' created with UTF-8 encoding.
Attempting to convert file encoding from UTF-8 to UTF-16
Reading from: input.txt
Writing to: output.txt
File encoding conversion successful!
```

File Output (`output.txt`): A new file named `output.txt` will be created in the same directory. Its content will be identical to `input.txt`, but its internal character encoding will be UTF-16. You can verify this by opening `output.txt` in a text editor like Notepad++ or VS Code and checking its encoding status.

Lab 7: Create an interactive applet that takes user input and performs some action based on it

Title

Interactive Applet with User Input

Aim

To create a basic interactive Java Applet that takes user input from a text field and displays it upon a button click.

Procedure

1. **Applet Class:** Create a class that extends `java.applet.Applet`.
2. **GUI Components:**
    - Declare `TextField` and `Button` components.
    - In the `init()` method, initialize these components.
    - Add them to the applet's layout.
3. **Event Handling:**
    - Implement the `ActionListener` interface in your applet class.
    - Register the applet as an `ActionListener` for the button using `button.addActionListener(this)`.
    - Override the `actionPerformed(ActionEvent e)` method.
    - Inside `actionPerformed()`, check if the event source is your button. If so, retrieve the text from the `TextField` using `getText()` and display it (e.g., using `Graphics.drawString()` or by updating another label).
4. **HTML for Applet:** Create an HTML file to embed and run the applet.

Note on Applets: Java Applets are largely deprecated and not supported by modern web browsers due to security concerns and the rise of more modern web technologies (like HTML5, CSS3, JavaScript). This exercise is primarily for understanding historical Java GUI programming concepts. You will need a Java-enabled browser or an Applet Viewer to run this.

Source Code

```
// Filename: InteractiveApplet.java
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;

/*
<applet code="InteractiveApplet.class" width="400" height="200">
</applet>
*/

public class InteractiveApplet extends Applet implements ActionListener {
    TextField inputField;
    Button submitButton;
    String message = "Enter your name and click Submit";

    // init() method is called when the applet is loaded
    public void init() {
        // Set background color
```

```
        setBackground(Color.LIGHT_GRAY);

        // Create a TextField for user input
        inputField = new TextField(20); // 20 columns wide

        // Create a Button
        submitButton = new Button("Submit");

        // Add components to the applet
        add(new Label("Your Name:")); // A simple label
        add(inputField);
        add(submitButton);

        // Register the applet as an ActionListener for the button
        submitButton.addActionListener(this);
    }

    // paint() method is called to draw on the applet window
    public void paint(Graphics g) {
        // Set font for the message
        g.setFont(new Font("Arial", Font.BOLD, 16));
        // Draw the message below the input field and button
        g.drawString(message, 50, 150);
    }

    // actionPerformed() method is called when an action event occurs (e.g.,
button click)
    public void actionPerformed(ActionEvent ae) {
        if (ae.getSource() == submitButton) {
            // Get text from the input field
            String name = inputField.getText();
            if (name.isEmpty()) {
                message = "Please enter something!";
            } else {
                message = "Hello, " + name + "!";
            }
            // Repaint the applet to display the updated message
            repaint();
        }
    }
}
```

HTML File to run the Applet (save as `InteractiveApplet.html`):

```
<!DOCTYPE html>
<html>
<head>
    <title>Interactive Applet</title>
</head>
<body>
    <h1>My Interactive Java Applet</h1>
    <p>This applet demonstrates basic user input and interaction.</p>
    <applet code="InteractiveApplet.class" width="400" height="200">
        Your browser does not support Java applets.
    </applet>
    <p>Instructions: Type your name in the text field and click "Submit".</p>
</body>
</html>
```

Input

1. Compile `InteractiveApplet.java`: `javac InteractiveApplet.java`
2. Run using `appletviewer`: `appletviewer InteractiveApplet.html`

3. In the applet window, type text into the text field (e.g., "Alice").
4. Click the "Submit" button.

Expected Output

Initially, the applet window will show a text field, a "Submit" button, and the message "Enter your name and click Submit".

After typing "Alice" and clicking "Submit", the message at the bottom of the applet window will change to:

```
Hello, Alice!
```

If you leave the field empty and click submit, it will show:

```
Please enter something!
```

Lab 8: Develop a real-world AWT applications such as calculators, text editors, and image viewers

Title

Basic AWT Calculator Application

Aim

To develop a simple calculator application using Java's Abstract Window Toolkit (AWT) that performs basic arithmetic operations.

Procedure

1. **Frame Setup:** Create a class that extends `java.awt.Frame` or `java.awt.Dialog` to serve as the main window.
2. **Layout Manager:** Use a suitable layout manager (e.g., `BorderLayout` and `GridLayout`) to arrange components. `BorderLayout` for the display field and `GridLayout` for the buttons.
3. **Display Field:** Add a `TextField` (or `Label`) at the top to show numbers and results.
4. **Buttons:** Create `Button` components for digits (0-9), arithmetic operators (+, -, *, /), decimal point, clear, and equals.
5. **Event Handling:**
    - Implement `ActionListener` to handle button clicks.
    - Register the `ActionListener` for each button.
    - In the `actionPerformed(ActionEvent e)` method:
        - Identify which button was clicked using `e.getActionCommand()`.
        - Implement the calculator logic:
            - Append digits to the display.
            - Store the first operand and operator when an operator button is pressed.
            - Perform the calculation when the equals button is pressed.
            - Handle the clear button to reset the calculator.
6. **Window Closing:** Add a `WindowListener` (or `WindowAdapter`) to handle closing the frame gracefully.

Source Code

```
// Filename: AWTCalculator.java
import java.awt.*;
import java.awt.event.*;

public class AWTCalculator extends Frame implements ActionListener {
    TextField display;
    Button[] buttons;
    String[] buttonLabels = {
        "7", "8", "9", "/",
        "4", "5", "6", "*",
        "1", "2", "3", "-",
        "C", "0", "=", "+"
    };

    double operand1 = 0;
    String operator = "";
```

```
    boolean newNumber = true; // Flag to indicate if a new number is being
entered

    public AWTCalculator() {
        setTitle("Simple AWT Calculator");
        setSize(300, 400);
        setLayout(new BorderLayout());
        setBackground(Color.DARK_GRAY); // Dark background for the frame

        // Display field
        display = new TextField("0");
        display.setEditable(false); // User cannot type directly
        display.setFont(new Font("Arial", Font.BOLD, 24));
        display.setBackground(Color.LIGHT_GRAY);
        display.setForeground(Color.BLACK);
        display.setAlignment(TextField.RIGHT); // Align text to the right
        add(display, BorderLayout.NORTH);

        // Button panel
        Panel buttonPanel = new Panel();
        buttonPanel.setLayout(new GridLayout(4, 4, 5, 5)); // 4x4 grid with gaps
        buttonPanel.setBackground(Color.GRAY); // Background for button panel

        buttons = new Button[buttonLabels.length];
        for (int i = 0; i < buttonLabels.length; i++) {
            buttons[i] = new Button(buttonLabels[i]);
            buttons[i].setFont(new Font("Arial", Font.BOLD, 18));
            buttons[i].addActionListener(this);

            // Set button colors
            if (buttonLabels[i].matches("[0-9]")) {
                buttons[i].setBackground(Color.WHITE);
            } else if (buttonLabels[i].matches("[\\+\\-\\*\\/]")) {
                buttons[i].setBackground(new Color(255, 165, 0)); // Orange for
operators
            } else if (buttonLabels[i].equals("=")) {
                buttons[i].setBackground(new Color(0, 150, 0)); // Green for
equals
                buttons[i].setForeground(Color.WHITE);
            } else if (buttonLabels[i].equals("C")) {
                buttons[i].setBackground(new Color(200, 0, 0)); // Red for clear
                buttons[i].setForeground(Color.WHITE);
            }
            buttonPanel.add(buttons[i]);
        }
        add(buttonPanel, BorderLayout.CENTER);

        // Handle window closing
        addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent we) {
                dispose(); // Close the frame
            }
        });

        setVisible(true);
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        String command = e.getActionCommand();

        if ("0123456789.".contains(command)) {
            // It's a digit or decimal point
            if (newNumber) {
                display.setText(command);
                newNumber = false;
            } else {
```

```
                if (command.equals(".") && display.getText().contains(".")) {
                    // Do nothing if decimal already exists
                } else {
                    display.setText(display.getText() + command);
                }
            }
        } else if ("+-*/".contains(command)) {
            // It's an operator
            operand1 = Double.parseDouble(display.getText());
            operator = command;
            newNumber = true; // Ready for the next number
        } else if (command.equals("=")) {
            // It's the equals button
            if (!operator.isEmpty()) {
                double operand2 = Double.parseDouble(display.getText());
                double result = 0;
                try {
                    switch (operator) {
                        case "+":
                            result = operand1 + operand2;
                            break;
                        case "-":
                            result = operand1 - operand2;
                            break;
                        case "*":
                            result = operand1 * operand2;
                            break;
                        case "/":
                            if (operand2 == 0) {
                                display.setText("Error: Div by 0");
                                newNumber = true;
                                operator = "";
                                return;
                            }
                            result = operand1 / operand2;
                            break;
                    }
                    display.setText(String.valueOf(result));
                } catch (NumberFormatException ex) {
                    display.setText("Error");
                }
                operator = ""; // Reset operator
                newNumber = true; // Result is displayed, ready for new
calculation
            }
        } else if (command.equals("C")) {
            // Clear button
            display.setText("0");
            operand1 = 0;
            operator = "";
            newNumber = true;
        }
    }

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

Input

Interact with the calculator by clicking the buttons. Example sequence:

1. Click `1`

2. Click `2`
3. Click `+`
4. Click `3`
5. Click `=`

Expected Output

A standalone AWT window will appear, resembling a basic calculator. Following the example input sequence:

1. Display shows `1`
2. Display shows `12`
3. Display shows `12` (operator is stored internally)
4. Display shows `3`
5. Display shows `15.0`

Lab 9: Handling Mouse and Key events, Adapter classes (Demonstrate)

Title

Mouse and Key Event Handling with Adapter Classes

Aim

To demonstrate how to handle mouse events (clicks, movements) and key events (presses, releases) in a Java AWT/Swing application, specifically utilizing adapter classes for convenience.

Procedure

1. **Frame Setup:** Create a `Frame` or `JFrame` to serve as the main window.
2. **Panel for Drawing/Interaction:** Add a `Panel` or `JPanel` to the frame where events will be detected and messages displayed.
3. **Mouse Events:**
    - Implement `MouseListener` and `MouseMotionListener` interfaces, or extend `MouseAdapter`.
    - Register the listener to the panel using `addMouseListener()` and `addMouseMotionListener()`.
    - Override methods like `mouseClicked()`, `mousePressed()`, `mouseReleased()`, `mouseMoved()`, `mouseDragged()`.
    - Display information about the event (e.g., coordinates, type of event) on the console or directly on the panel.
4. **Key Events:**
    - Implement `KeyListener` interface, or extend `KeyAdapter`.
    - Register the listener to the panel using `addKeyListener()`. (Note: The component must have focus to receive key events).
    - Override methods like `keyPressed()`, `keyReleased()`, `keyTyped()`.
    - Display information about the key pressed (e.g., character, key code).
5. **Focus Handling:** Ensure the panel can gain focus (e.g., by calling `setFocusable(true)` and `requestFocusInWindow()` for Swing, or just `requestFocus()` for AWT).
6. **Window Closing:** Add a `WindowListener` (or `WindowAdapter`) to handle closing the frame.

Source Code

```
// Filename: EventHandlingDemo.java
import java.awt.*;
import java.awt.event.*;
import javax.swing.*; // Using Swing for better component handling and focus

public class EventHandlingDemo extends JFrame {

    private JLabel mouseStatusLabel;
    private JLabel keyStatusLabel;
    private JPanel drawingPanel;

    public EventHandlingDemo() {
        setTitle("Event Handling Demo (Mouse & Keyboard)");
        setSize(600, 400);
```

```
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new BorderLayout());

        // --- Status Labels ---
        JPanel statusPanel = new JPanel(new GridLayout(2, 1));
        mouseStatusLabel = new JLabel("Mouse Status: No event yet.");
        keyStatusLabel = new JLabel("Key Status: No event yet. Click here to
gain focus!");
        mouseStatusLabel.setHorizontalAlignment(SwingConstants.CENTER);
        keyStatusLabel.setHorizontalAlignment(SwingConstants.CENTER);
        statusPanel.add(mouseStatusLabel);
        statusPanel.add(keyStatusLabel);
        add(statusPanel, BorderLayout.SOUTH);

        // --- Drawing Panel for Mouse Events ---
        drawingPanel = new JPanel() {
            @Override
            protected void paintComponent(Graphics g) {
                super.paintComponent(g);
                // Optional: Draw something on the panel
                g.setColor(Color.BLUE);
                g.drawRect(10, 10, getWidth() - 20, getHeight() - 20);
                g.drawString("Move mouse, click, or press keys here.", 100,
100);
            }
        };
        drawingPanel.setBackground(Color.LIGHT_GRAY);
        drawingPanel.setFocusable(true); // Make panel focusable for key events
        add(drawingPanel, BorderLayout.CENTER);

        // --- Mouse Event Handling using MouseAdapter ---
        drawingPanel.addMouseListener(new MouseAdapter() {
            @Override
            public void mouseClicked(MouseEvent e) {
                mouseStatusLabel.setText("Mouse Clicked at (" + e.getX() + ", "
+ e.getY() + ")");
                drawingPanel.requestFocusInWindow(); // Request focus on click
for key events
            }

            @Override
            public void mousePressed(MouseEvent e) {
                mouseStatusLabel.setText("Mouse Pressed at (" + e.getX() + ", "
+ e.getY() + ")");
            }

            @Override
            public void mouseReleased(MouseEvent e) {
                mouseStatusLabel.setText("Mouse Released at (" + e.getX() + ", "
+ e.getY() + ")");
            }
        });

        drawingPanel.addMouseMotionListener(new MouseMotionAdapter() {
            @Override
            public void mouseMoved(MouseEvent e) {
                mouseStatusLabel.setText("Mouse Moved to (" + e.getX() + ", " +
e.getY() + ")");
            }

            @Override
            public void mouseDragged(MouseEvent e) {
                mouseStatusLabel.setText("Mouse Dragged to (" + e.getX() + ", "
+ e.getY() + ")");
            }
        });
```

```
        // --- Key Event Handling using KeyAdapter ---
        drawingPanel.addKeyListener(new KeyAdapter() {
            @Override
            public void keyPressed(KeyEvent e) {
                keyStatusLabel.setText("Key Pressed: " +
KeyEvent.getKeyText(e.getKeyCode()) + " (Code: " + e.getKeyCode() + ")");
            }

            @Override
            public void keyReleased(KeyEvent e) {
                keyStatusLabel.setText("Key Released: " +
KeyEvent.getKeyText(e.getKeyCode()));
            }

            @Override
            public void keyTyped(KeyEvent e) {
                keyStatusLabel.setText("Key Typed: '" + e.getKeyChar() + "'");
            }
        });

        // Request focus initially so key events can be captured right away
        addWindowListener(new WindowAdapter() {
            @Override
            public void windowOpened(WindowEvent e) {
                drawingPanel.requestFocusInWindow();
            }
        });

        setVisible(true);
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(() -> new EventHandlingDemo());
    }
}
```

Input

1. Run the program.
2. Move your mouse cursor over the gray panel.
3. Click and drag the mouse on the panel.
4. Click anywhere on the gray panel to ensure it has focus.
5. Press various keys on your keyboard (letters, numbers, special keys like Enter, Shift, Ctrl).

Expected Output

A Swing window will appear.

- **Mouse Events:** The "Mouse Status" label at the bottom will update dynamically, showing the current mouse coordinates and the type of mouse event (Moved, Clicked, Pressed, Released, Dragged).
- **Key Events:** After clicking the panel to give it focus, the "Key Status" label will update to show which key was pressed, released, or typed, along with its key code.

Example Console Output (if `System.out.println` were used, but here it's on GUI): (Imagine these messages appearing in the GUI labels)

```
Mouse Moved to (150, 100)
Mouse Clicked at (200, 150)
Key Pressed: A (Code: 65)
Key Typed: 'a'
```

```
Key Released: A
Key Pressed: Enter (Code: 10)
Key Released: Enter
```

Lab 10: Write a Java program to execute a simple SQL query to retrieve data from a table and display the results.

Title

JDBC - Simple SQL Query Execution and Data Retrieval

Aim

To write a Java program using JDBC (Java Database Connectivity) to connect to a database, create a table, insert sample data, execute a simple SQL `SELECT` query, and display the retrieved results.

Procedure

1. **Database Setup:** For simplicity and portability, we will use an in-memory H2 database. No external server installation is required.
2. **JDBC Driver:** The H2 database driver will be included as a dependency (or downloaded manually if not using a build tool).
3. **Establish Connection:**
    - Load the JDBC driver class (though often not strictly necessary with modern JDBC 4.0+ drivers).
    - Obtain a `Connection` object using `DriverManager.getConnection()`, providing the database URL, username, and password.
4. **Create Statement:** Create a `Statement` object from the `Connection` to execute SQL queries.
5. **Execute DDL/DML (Optional, for setup):** Execute `CREATE TABLE` and `INSERT` statements to prepare the data.
6. **Execute Query:** Execute a `SELECT` query using `statement.executeQuery()`. This returns a `ResultSet` object.
7. **Process Results:**
    - Iterate through the `ResultSet` using `resultSet.next()`.
    - Retrieve data from each column using appropriate `get` methods (e.g., `getString()`, `getInt()`) by column name or index.
    - Print the retrieved data to the console.
8. **Close Resources:** Crucially, close all JDBC resources (`ResultSet`, `Statement`, `Connection`) in reverse order of creation, typically within `finally` blocks or using try-with-resources to prevent resource leaks.

Source Code

```
// Filename: SimpleJDBCQuery.java
import java.sql.*; // Import all classes from java.sql package

public class SimpleJDBCQuery {

    // JDBC URL for H2 in-memory database. DB_CLOSE_DELAY=-1 keeps the DB open
    // as long as the JVM is running, useful for quick testing.
    static final String JDBC_URL = "jdbc:h2:mem:testdb;DB_CLOSE_DELAY=-1";
    static final String USER = "sa";
    static final String PASSWORD = "";

    public static void main(String[] args) {
        Connection conn = null;
```

```
        Statement stmt = null;
        ResultSet rs = null;

        try {
            // 1. Establish a connection
            // No need to explicitly load driver for H2 with modern JDBC (Java
6+)
            System.out.println("Connecting to database...");
            conn = DriverManager.getConnection(JDBC_URL, USER, PASSWORD);
            System.out.println("Connection established successfully.");

            // 2. Create a Statement
            stmt = conn.createStatement();

            // 3. Execute DDL (Create Table)
            System.out.println("\nCreating table 'employees'...");
            String createTableSQL = "CREATE TABLE employees (" +
                                    "id INT PRIMARY KEY AUTO_INCREMENT," +
                                    "name VARCHAR(255) NOT NULL," +
                                    "age INT," +
                                    "department VARCHAR(255)" +
                                    ")";
            stmt.executeUpdate(createTableSQL);
            System.out.println("Table 'employees' created.");

            // 4. Execute DML (Insert Data)
            System.out.println("\nInserting data into 'employees'...");
            String insertSQL1 = "INSERT INTO employees (name, age, department)
VALUES ('Alice', 30, 'HR')";
            String insertSQL2 = "INSERT INTO employees (name, age, department)
VALUES ('Bob', 24, 'Engineering')";
            String insertSQL3 = "INSERT INTO employees (name, age, department)
VALUES ('Charlie', 35, 'Sales')";
            stmt.executeUpdate(insertSQL1);
            stmt.executeUpdate(insertSQL2);
            stmt.executeUpdate(insertSQL3);
            System.out.println("Data inserted successfully.");

            // 5. Execute a simple SELECT query
            System.out.println("\nRetrieving data from 'employees'...");
            String selectSQL = "SELECT id, name, age, department FROM employees
WHERE age > 25";
            rs = stmt.executeQuery(selectSQL);

            // 6. Process the results
            System.out.println("--- Employees (Age > 25) ---");
            boolean foundRecords = false;
            while (rs.next()) {
                foundRecords = true;
                // Retrieve by column name or index
                int id = rs.getInt("id");
                String name = rs.getString("name");
                int age = rs.getInt("age");
                String department = rs.getString("department");

                System.out.println("ID: " + id + ", Name: " + name + ", Age: " +
age + ", Department: " + department);
            }

            if (!foundRecords) {
                System.out.println("No records found matching the criteria.");
            }

        } catch (SQLException se) {
            // Handle errors for JDBC
            System.err.println("JDBC Error: " + se.getMessage());
            se.printStackTrace();
```

```
        } catch (Exception e) {
            // Handle other errors
            System.err.println("General Error: " + e.getMessage());
            e.printStackTrace();
        } finally {
            // 7. Close resources in reverse order
            try {
                if (rs != null) rs.close();
            } catch (SQLException se2) {
                // Do nothing
            }
            try {
                if (stmt != null) stmt.close();
            } catch (SQLException se2) {
                // Do nothing
            }
            try {
                if (conn != null) conn.close();
                System.out.println("\nDatabase connection closed.");
            } catch (SQLException se) {
                se.printStackTrace();
            }
        }
    }
}
```

Input

No specific input is required. The program connects to an in-memory H2 database, creates a table, inserts data, and queries it.

To run this, you need the H2 database JAR in your classpath. You can download it from the H2 website or add it as a Maven/Gradle dependency. For manual compilation and execution:

1. Download `h2-<version>.jar` (e.g., `h2-2.2.224.jar`).
2. Compile: `javac -cp h2-<version>.jar SimpleJDBCQuery.java`
3. Run: `java -cp .;h2-<version>.jar SimpleJDBCQuery` (on Windows) or `java -cp .:h2-<version>.jar SimpleJDBCQuery` (on Linux/macOS)

Expected Output

```
Connecting to database...
Connection established successfully.

Creating table 'employees'...
Table 'employees' created.

Inserting data into 'employees'...
Data inserted successfully.

Retrieving data from 'employees'...
--- Employees (Age > 25) ---
ID: 1, Name: Alice, Age: 30, Department: HR
ID: 3, Name: Charlie, Age: 35, Department: Sales

Database connection closed.
```

Lab 11: Write a Java program to establish a socket connection between a client and a server on different machines using Socket and Server Socket classes.

Title

Socket Programming - Client-Server Communication

Aim

To write two separate Java programs, a Server and a Client, that establish a basic TCP socket connection and exchange simple messages, demonstrating the use of `ServerSocket` and `Socket` classes.

Procedure

Server Program:

1. **Create `ServerSocket`:** Instantiate a `ServerSocket` object, specifying a port number on which the server will listen for incoming client connections.
2. **Listen for Connection:** Call `serverSocket.accept()` which blocks until a client connects. This method returns a `Socket` object representing the connection to the client.
3. **Get I/O Streams:** Obtain `InputStream` and `OutputStream` (or `BufferedReader`/`PrintWriter` for text) from the client `Socket` to send and receive data.
4. **Communicate:** Send a message to the client and/or receive a message from the client.
5. **Close Resources:** Close the `Socket` and `ServerSocket` in a `finally` block to release resources.

Client Program:

1. **Create `Socket`:** Instantiate a `Socket` object, providing the server's IP address (or hostname) and the port number. This attempts to connect to the server.
2. **Get I/O Streams:** Obtain `InputStream` and `OutputStream` from the `Socket` to send and receive data.
3. **Communicate:** Send a message to the server and/or receive a message from the server.
4. **Close Resources:** Close the `Socket` in a `finally` block.

Source Code

1. Server Program (Save as `SimpleServer.java`)

```
// Filename: SimpleServer.java
import java.io.*;
import java.net.*;

public class SimpleServer {
    private static final int PORT = 12345; // Port number for the server

    public static void main(String[] args) {
        ServerSocket serverSocket = null;
        Socket clientSocket = null;
        BufferedReader in = null;
```

```
        PrintWriter out = null;

        try {
            // 1. Create a ServerSocket
            serverSocket = new ServerSocket(PORT);
            System.out.println("Server started. Listening on port " + PORT +
"...");

            // 2. Listen for a client connection
            clientSocket = serverSocket.accept(); // Blocks until a client
connects
            System.out.println("Client connected from: " +
clientSocket.getInetAddress().getHostAddress());

            // 3. Get I/O streams from the client socket
            in = new BufferedReader(new
InputStreamReader(clientSocket.getInputStream()));
            out = new PrintWriter(clientSocket.getOutputStream(), true); //
'true' for auto-flush

            // 4. Communicate
            String clientMessage = in.readLine(); // Read message from client
            System.out.println("Received from client: " + clientMessage);

            String serverResponse = "Hello from Server! Your message was
received.";
            out.println(serverResponse); // Send response to client
            System.out.println("Sent to client: " + serverResponse);

        } catch (IOException e) {
            System.err.println("Server error: " + e.getMessage());
            e.printStackTrace();
        } finally {
            // 5. Close resources
            try {
                if (out != null) out.close();
                if (in != null) in.close();
                if (clientSocket != null) clientSocket.close();
                if (serverSocket != null) serverSocket.close();
                System.out.println("Server resources closed.");
            } catch (IOException e) {
                System.err.println("Error closing server resources: " +
e.getMessage());
            }
        }
    }
}
```

2. Client Program (Save as `SimpleClient.java`)

```
// Filename: SimpleClient.java
import java.io.*;
import java.net.*;

public class SimpleClient {
    private static final String SERVER_IP = "localhost"; // Use "localhost" for
same machine, or server's IP
    private static final int PORT = 12345; // Must match the server's port

    public static void main(String[] args) {
        Socket socket = null;
        BufferedReader in = null;
        PrintWriter out = null;

        try {
```

```
            // 1. Create a Socket to connect to the server
            System.out.println("Attempting to connect to server at " + SERVER_IP
+ ":" + PORT + "...");
            socket = new Socket(SERVER_IP, PORT);
            System.out.println("Connected to server.");

            // 2. Get I/O streams from the socket
            in = new BufferedReader(new
InputStreamReader(socket.getInputStream()));
            out = new PrintWriter(socket.getOutputStream(), true); // 'true' for
auto-flush

            // 3. Communicate
            String clientMessage = "Hello from Client!";
            out.println(clientMessage); // Send message to server
            System.out.println("Sent to server: " + clientMessage);

            String serverResponse = in.readLine(); // Read response from server
            System.out.println("Received from server: " + serverResponse);

        } catch (UnknownHostException e) {
            System.err.println("Server host not found: " + SERVER_IP + " - " +
e.getMessage());
        } catch (IOException e) {
            System.err.println("Client I/O error: " + e.getMessage());
            e.printStackTrace();
        } finally {
            // 4. Close resources
            try {
                if (out != null) out.close();
                if (in != null) in.close();
                if (socket != null) socket.close();
                System.out.println("Client resources closed.");
            } catch (IOException e) {
                System.err.println("Error closing client resources: " +
e.getMessage());
            }
        }
    }
}
```

Input

To run these programs:

1. **Compile both files:** `javac SimpleServer.java SimpleClient.java`
2. **Start the Server first:** `java SimpleServer` The server will print "Server started. Listening on port 12345...". It will then wait for a client connection.
3. **Run the Client (in a *separate* terminal/command prompt window):** `java SimpleClient` The client will attempt to connect to the server.

Expected Output

Server Console Output:

```
Server started. Listening on port 12345...
Client connected from: 127.0.0.1
Received from client: Hello from Client!
Sent to client: Hello from Server! Your message was received.
Server resources closed.
```

Client Console Output:

```
Attempting to connect to server at localhost:12345...
Connected to server.
Sent to server: Hello from Client!
Received from server: Hello from Server! Your message was received.
Client resources closed.
```

(Note: If running on different machines, replace "localhost" in `SimpleClient.java` with the actual IP address of the machine running `SimpleServer.java`.)

Lab 12: Develop a Java RMI application with a clear separation of client and server components.

Title

Java RMI - Remote Method Invocation Application

Aim

To develop a Java RMI (Remote Method Invocation) application that demonstrates how a client can invoke methods on a remote object running on a server, with a clear separation of client and server components.

Procedure

1. **Define Remote Interface:**
   - Create an interface that extends `java.rmi.Remote`.
   - Declare the methods that will be callable remotely.
   - Each method must declare `throws RemoteException`.
2. **Implement Remote Object (Server Side):**
   - Create a class that implements the remote interface.
   - This class must extend `java.rmi.server.UnicastRemoteObject` (or use `exportObject` explicitly).
   - Implement the remote methods.
   - The constructor must declare `throws RemoteException`.
3. **Server Program:**
   - Create an instance of the remote object implementation.
   - Bind this remote object to a name in the RMI Registry using `Naming.rebind()` (or `LocateRegistry`).
   - The RMI Registry must be running (either as a separate process `rmiregistry` or embedded in the server).
4. **Client Program:**
   - Look up the remote object from the RMI Registry using `Naming.lookup()`.
   - Cast the returned object to the remote interface type.
   - Invoke the remote methods as if they were local methods.
5. **Compile and Run:**
   - Compile all Java files.
   - Generate stubs (if using older RMI versions, `rmic` was needed; modern RMI often generates them dynamically).
   - Start the RMI Registry.
   - Start the Server.
   - Run the Client.

Source Code

1. Remote Interface (Save as `CalculatorService.java`)

```
// Filename: CalculatorService.java
import java.rmi.Remote;
import java.rmi.RemoteException;
```

```
// This interface defines the methods that can be called remotely.
public interface CalculatorService extends Remote {
    double add(double a, double b) throws RemoteException;
    double subtract(double a, double b) throws RemoteException;
    double multiply(double a, double b) throws RemoteException;
    double divide(double a, double b) throws RemoteException;
}
```

2. Remote Object Implementation (Server Side) (Save as `CalculatorServiceImpl.java`)

```
// Filename: CalculatorServiceImpl.java
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

// This class implements the remote interface and provides the actual logic.
// It extends UnicastRemoteObject to make it a remote object.
public class CalculatorServiceImpl extends UnicastRemoteObject implements
CalculatorService {

    // Constructor must throw RemoteException
    public CalculatorServiceImpl() throws RemoteException {
        super(); // Calls the constructor of UnicastRemoteObject
    }

    // Implement the remote methods
    @Override
    public double add(double a, double b) throws RemoteException {
        System.out.println("Server: add(" + a + ", " + b + ") called.");
        return a + b;
    }

    @Override
    public double subtract(double a, double b) throws RemoteException {
        System.out.println("Server: subtract(" + a + ", " + b + ") called.");
        return a - b;
    }

    @Override
    public double multiply(double a, double b) throws RemoteException {
        System.out.println("Server: multiply(" + a + ", " + b + ") called.");
        return a * b;
    }

    @Override
    public double divide(double a, double b) throws RemoteException {
        System.out.println("Server: divide(" + a + ", " + b + ") called.");
        if (b == 0) {
            throw new RemoteException("Cannot divide by zero!");
        }
        return a / b;
    }
}
```

3. Server Program (Save as `RMIServer.java`)

```
// Filename: RMIServer.java
import java.rmi.Naming;
import java.rmi.registry.LocateRegistry;

public class RMIServer {
    public static void main(String[] args) {
        try {
            // Create an instance of the remote object implementation
```

```
            CalculatorService calculator = new CalculatorServiceImpl();

            // Start the RMI Registry on the default port (1099)
            // If rmiregistry is already running separately, this line can be
omitted.
            LocateRegistry.createRegistry(1099);
            System.out.println("RMI Registry created on port 1099.");

            // Bind the remote object to a name in the RMI Registry
            // The name "CalculatorService" will be used by the client to look
up the object.
            Naming.rebind("CalculatorService", calculator);
            System.out.println("CalculatorService bound in RMI Registry.");
            System.out.println("Server is ready.");

        } catch (Exception e) {
            System.err.println("Server exception: " + e.toString());
            e.printStackTrace();
        }
    }
}
```

4. Client Program (Save as `RMIClient.java`)

```
// Filename: RMIClient.java
import java.rmi.Naming;

public class RMIClient {
    public static void main(String[] args) {
        try {
            // Look up the remote object from the RMI Registry
            // The URL format is "rmi://<server_ip>:<port>/<object_name>"
            // For localhost and default port 1099, it's
"rmi://localhost/CalculatorService"
            CalculatorService calculator = (CalculatorService)
Naming.lookup("rmi://localhost/CalculatorService");
            System.out.println("Client: Connected to CalculatorService.");

            // Invoke remote methods
            double sum = calculator.add(10, 5);
            System.out.println("Client: 10 + 5 = " + sum);

            double difference = calculator.subtract(10, 5);
            System.out.println("Client: 10 - 5 = " + difference);

            double product = calculator.multiply(10, 5);
            System.out.println("Client: 10 * 5 = " + product);

            double quotient = calculator.divide(10, 5);
            System.out.println("Client: 10 / 5 = " + quotient);

            // Demonstrate exception handling for remote methods
            try {
                calculator.divide(10, 0); // This will throw a RemoteException
            } catch (Exception e) {
                System.err.println("Client: Caught expected exception: " +
e.getMessage());
            }

        } catch (Exception e) {
            System.err.println("Client exception: " + e.toString());
            e.printStackTrace();
        }
    }
}
```

Input

To run this RMI application, follow these steps in separate terminal windows:

1. **Compile all Java files:** `javac CalculatorService.java CalculatorServiceImpl.java RMIServer.java RMIClient.java`
2. **Start the RMI Registry:**
    - If you *don't* want the server to embed the registry (as shown in `RMIServer.java`), you can start it separately: `rmiregistry` (This command should be run from the directory where your compiled `.class` files are located, or ensure your classpath is set correctly).
    - *Note:* The provided `RMIServer.java` *includes* `LocateRegistry.createRegistry(1099);`, so you might not need to run `rmiregistry` separately if you run the server first. If you get `java.rmi.server.ExportException: Port already in use: 1099`, it means `rmiregistry` is already running.
3. **Start the Server:** `java RMIServer` The server will print messages indicating it has started and bound the service.
4. **Run the Client:** `java RMIClient` The client will connect to the server and invoke the remote methods.

Expected Output

Server Console Output:

```
RMI Registry created on port 1099.
CalculatorService bound in RMI Registry.
Server is ready.
Server: add(10.0, 5.0) called.
Server: subtract(10.0, 5.0) called.
Server: multiply(10.0, 5.0) called.
Server: divide(10.0, 5.0) called.
Server: divide(10.0, 0.0) called.
```

Client Console Output:

```
Client: Connected to CalculatorService.
Client: 10 + 5 = 15.0
Client: 10 - 5 = 5.0
Client: 10 * 5 = 50.0
Client: 10 / 5 = 2.0
Client: Caught expected exception: Cannot divide by zero!
```

Lab 13: Write a servlet that demonstrates the usage of these key classes/interfaces for handling HTTP requests and responses.

Title

Basic Servlet - HTTP Request and Response Handling

Aim

To write a Java Servlet that demonstrates how to handle incoming HTTP GET requests and generate an HTTP response, utilizing key servlet classes and interfaces.

Procedure

1. **Servlet Class:** Create a Java class that extends `javax.servlet.http.HttpServlet`.
2. **Override `doGet()`:** Override the `doGet(HttpServletRequest request, HttpServletResponse response)` method. This method is invoked when an HTTP GET request is made to the servlet.
3. **Request Handling:**
    - Use the `HttpServletRequest` object to retrieve information about the incoming request (e.g., parameters, headers, client IP).
    - Example: `request.getParameter("name")` to get a query parameter.
4. **Response Generation:**
    - Use the `HttpServletResponse` object to set the response content type (e.g., `response.setContentType("text/html")`).
    - Obtain a `PrintWriter` object from the response (`response.getWriter()`) to write the response content back to the client.
    - Write HTML content (or plain text) to the `PrintWriter`.
5. **Deployment Descriptor (`web.xml`):** Configure the servlet mapping in `WEB-INF/web.xml` to associate a URL pattern with your servlet class.
6. **Deployment:** Package the servlet (and `web.xml`) into a WAR file and deploy it to a Java web server (like Apache Tomcat).

Note on Servlets: To run this, you need a Java EE compatible web server (like Apache Tomcat). You will need to set up a web project and deploy the servlet.

Source Code

1. Servlet Class (Save as `SimpleHttpServlet.java`)

```
// Filename: SimpleHttpServlet.java
import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

// Extend HttpServlet to handle HTTP requests
public class SimpleHttpServlet extends HttpServlet {
```

```
    // Override the doGet method to handle HTTP GET requests
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse
response)
            throws ServletException, IOException {

        // 1. Set the content type of the response
        // This tells the browser what kind of content it's receiving
        response.setContentType("text/html");

        // 2. Get a PrintWriter object to write the response back to the client
        PrintWriter out = response.getWriter();

        // 3. Retrieve information from the request
        String userName = request.getParameter("name"); // Get a query parameter
named 'name'
        if (userName == null || userName.isEmpty()) {
            userName = "Guest"; // Default if no name is provided
        }
        String clientIP = request.getRemoteAddr(); // Get client's IP address

        // 4. Generate the HTML response
        out.println("<!DOCTYPE html>");
        out.println("<html>");
        out.println("<head>");
        out.println("<title>Simple Servlet Response</title>");
        out.println("<style>");
        out.println("body { font-family: Arial, sans-serif; background-color:
#f4f4f4; color: #333; text-align: center; padding-top: 50px; }");
        out.println("h1 { color: #0056b3; }");
        out.println("p { font-size: 1.1em; }");
        out.println(".container { background-color: #fff; margin: 20px auto;
padding: 30px; border-radius: 8px; box-shadow: 0 2px 4px rgba(0,0,0,0.1); max-
width: 600px; }");
        out.println("</style>");
        out.println("</head>");
        out.println("<body>");
        out.println("<div class='container'>");
        out.println("<h1>Hello, " + userName + "!</h1>");
        out.println("<p>This is a simple Java Servlet demonstrating HTTP request
and response handling.</p>");
        out.println("<p>Your IP Address: <strong>" + clientIP +
"</strong></p>");
        out.println("<p>Current Time on Server: <strong>" + new java.util.Date()
+ "</strong></p>");
        out.println("<p>Try adding a parameter:
<code>/simple?name=YourName</code></p>");
        out.println("</div>");
        out.println("</body>");
        out.println("</html>");

        // 5. Close the PrintWriter (optional, as it's typically handled by the
container)
        out.close();
    }

    // You can also override doPost for POST requests, etc.
    // @Override
    // protected void doPost(HttpServletRequest request, HttpServletResponse
response)
    //         throws ServletException, IOException {
    //     // ... handle POST requests ...
    // }
}
```

2. Deployment Descriptor (`WEB-INF/web.xml`)

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
                             http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
         version="4.0">

    <display-name>SimpleServletApp</display-name>

    <servlet>
        <servlet-name>SimpleServlet</servlet-name>
        <servlet-class>SimpleHttpServlet</servlet-class>
    </servlet>

    <servlet-mapping>
        <servlet-name>SimpleServlet</servlet-name>
        <url-pattern>/simple</url-pattern>
    </servlet-mapping>

</web-app>
```

Input

Deployment Steps (using Apache Tomcat as an example):

1. **Create Project Structure:**

```
SimpleServletApp/
├── src/
│   └── SimpleHttpServlet.java
└── WebContent/
    └── WEB-INF/
        └── web.xml
```

8. **Compile:** Place `servlet-api.jar` (from Tomcat's `lib` directory) in your classpath when compiling `SimpleHttpServlet.java`. `javac -cp "path/to/tomcat/lib/servlet-api.jar" src/SimpleHttpServlet.java`
9. **Package (WAR file):** Create a WAR file from the `WebContent` directory. `jar -cvf SimpleServletApp.war -C WebContent/ .`
10. **Deploy:** Copy `SimpleServletApp.war` to Tomcat's `webapps` directory. Tomcat will automatically deploy it.
11. **Access in Browser:** Open your web browser and navigate to:
    - `http://localhost:8080/SimpleServletApp/simple` (for default name "Guest")
    - `http://localhost:8080/SimpleServletApp/simple?name=JohnDoe` (to pass a name)

Expected Output

Browser Output (for `http://localhost:8080/SimpleServletApp/simple?name=JohnDoe`):

```
<!DOCTYPE html>
<html>
<head>
<title>Simple Servlet Response</title>
<style>
body { font-family: Arial, sans-serif; background-color: #f4f4f4; color: #333;
text-align: center; padding-top: 50px; }
h1 { color: #0056b3; }
```

```
p { font-size: 1.1em; }
.container { background-color: #fff; margin: 20px auto; padding: 30px; border-
radius: 8px; box-shadow: 0 2px 4px rgba(0,0,0,0.1); max-width: 600px; }
</style>
</head>
<body>
<div class='container'>
<h1>Hello, JohnDoe!</h1>
<p>This is a simple Java Servlet demonstrating HTTP request and response
handling.</p>
<p>Your IP Address: <strong>127.0.0.1</strong></p> <p>Current Time on Server:
<strong>Thu May 22 11:39:00 IST 2025</strong></p> <p>Try adding a parameter:
<code>/simple?name=YourName</code></p>
</div>
</body>
</html>
```

The page will display a greeting, the client's IP address, and the current server time.

Lab 14: Develop a servlet that interacts with a database using JDBC to retrieve data from a table and display it in a web page.

Title

Servlet with JDBC Database Interaction

Aim

To develop a Java Servlet that connects to a database using JDBC, retrieves data from a table, and dynamically generates an HTML web page to display the retrieved information.

Procedure

1. **Servlet Setup:** Create a Java class extending `javax.servlet.http.HttpServlet`.
2. **JDBC Connection:**
    - Inside the `doGet()` method (or a helper method called from `doGet()`), establish a JDBC connection to the database. For simplicity, we'll use an in-memory H2 database again.
    - Ensure the JDBC driver JAR (e.g., `h2-<version>.jar`) is available in your web application's `WEB-INF/lib` directory.
3. **Database Operations:**
    - Create a `Statement` object.
    - Execute a `SELECT` SQL query to retrieve data.
    - Process the `ResultSet` to extract data.
4. **HTML Generation:**
    - Set the response content type to `text/html`.
    - Obtain a `PrintWriter`.
    - Dynamically generate HTML content, including an HTML table, to display the data retrieved from the database.
5. **Error Handling:** Implement `try-catch` blocks for `SQLException` and `IOException` to handle database and I/O errors gracefully.
6. **Resource Closing:** Ensure all JDBC resources (`ResultSet`, `Statement`, `Connection`) are closed in `finally` blocks or using try-with-resources.
7. **Deployment:** Configure `web.xml` and deploy the WAR file to a web server (like Tomcat).

Source Code

1. Servlet Class (Save as `DatabaseServlet.java`)

```
// Filename: DatabaseServlet.java
import java.io.IOException;
import java.io.PrintWriter;
import java.sql.*;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class DatabaseServlet extends HttpServlet {
```

```
    // JDBC URL for H2 in-memory database
    private static final String JDBC_URL =
"jdbc:h2:mem:servletdb;DB_CLOSE_DELAY=-1";
    private static final String USER = "sa";
    private static final String PASSWORD = "";

    @Override
    public void init() throws ServletException {
        super.init();
        // Initialize database (create table and insert data) when servlet
starts
        try (Connection conn = DriverManager.getConnection(JDBC_URL, USER,
PASSWORD);
             Statement stmt = conn.createStatement()) {

            // Create table if it doesn't exist
            stmt.executeUpdate("CREATE TABLE IF NOT EXISTS products (" +
                               "id INT PRIMARY KEY AUTO_INCREMENT," +
                               "name VARCHAR(255) NOT NULL," +
                               "price DOUBLE," +
                               "quantity INT" +
                               ")");

            // Insert some sample data (only if table is empty)
            ResultSet rs = stmt.executeQuery("SELECT COUNT(*) FROM products");
            rs.next();
            if (rs.getInt(1) == 0) {
                stmt.executeUpdate("INSERT INTO products (name, price, quantity)
VALUES ('Laptop', 1200.00, 10)");
                stmt.executeUpdate("INSERT INTO products (name, price, quantity)
VALUES ('Mouse', 25.50, 100)");
                stmt.executeUpdate("INSERT INTO products (name, price, quantity)
VALUES ('Keyboard', 75.00, 50)");
                System.out.println("Sample data inserted into 'products'
table.");
            }

        } catch (SQLException e) {
            System.err.println("Database initialization error: " +
e.getMessage());
            e.printStackTrace();
            throw new ServletException("Could not initialize database", e);
        }
    }

    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse
response)
            throws ServletException, IOException {

        response.setContentType("text/html");
        PrintWriter out = response.getWriter();

        Connection conn = null;
        Statement stmt = null;
        ResultSet rs = null;

        try {
            // Establish JDBC connection
            conn = DriverManager.getConnection(JDBC_URL, USER, PASSWORD);
            stmt = conn.createStatement();

            // Execute SQL query to retrieve data
            String selectSQL = "SELECT id, name, price, quantity FROM products";
            rs = stmt.executeQuery(selectSQL);

            // Generate HTML response
```

```
            out.println("<!DOCTYPE html>");
            out.println("<html>");
            out.println("<head>");
            out.println("<title>Product List</title>");
            out.println("<style>");
            out.println("body { font-family: Arial, sans-serif; background-
color: #f4f4f4; color: #333; text-align: center; padding-top: 50px; }");
            out.println("h1 { color: #0056b3; }");
            out.println("table { width: 80%; border-collapse: collapse; margin:
20px auto; background-color: #fff; box-shadow: 0 2px 4px rgba(0,0,0,0.1);
border-radius: 8px; overflow: hidden; }");
            out.println("th, td { border: 1px solid #ddd; padding: 12px; text-
align: left; }");
            out.println("th { background-color: #007bff; color: white; }");
            out.println("tr:nth-child(even) { background-color: #f2f2f2; }");
            out.println("</style>");
            out.println("</head>");
            out.println("<body>");
            out.println("<h1>Products in Database</h1>");
            out.println("<table>");

out.println("<tr><th>ID</th><th>Name</th><th>Price</th><th>Quantity</th></tr>");

            // Process ResultSet and populate HTML table
            while (rs.next()) {
                int id = rs.getInt("id");
                String name = rs.getString("name");
                double price = rs.getDouble("price");
                int quantity = rs.getInt("quantity");

                out.println("<tr>");
                out.println("<td>" + id + "</td>");
                out.println("<td>" + name + "</td>");
                out.println("<td>$" + String.format("%.2f", price) + "</td>");
                out.println("<td>" + quantity + "</td>");
                out.println("</tr>");
            }

            out.println("</table>");
            out.println("<p>Data retrieved from in-memory H2 database.</p>");
            out.println("</body>");
            out.println("</html>");

        } catch (SQLException e) {
            System.err.println("Database access error: " + e.getMessage());
            e.printStackTrace();
            out.println("<h1>Error: Could not retrieve data from
database.</h1>");
            out.println("<p>" + e.getMessage() + "</p>");
        } finally {
            // Close JDBC resources
            try {
                if (rs != null) rs.close();
                if (stmt != null) stmt.close();
                if (conn != null) conn.close();
            } catch (SQLException se) {
                System.err.println("Error closing JDBC resources: " +
se.getMessage());
            }
        }
    }
}
```

2. Deployment Descriptor (`WEB-INF/web.xml`)

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
                             http://xmlns.jcp.org/xml/ns/javaee/web-app_4_0.xsd"
         version="4.0">

    <display-name>DatabaseWebApp</display-name>

    <servlet>
        <servlet-name>ProductServlet</servlet-name>
        <servlet-class>DatabaseServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>

    <servlet-mapping>
        <servlet-name>ProductServlet</servlet-name>
        <url-pattern>/products</url-pattern>
    </servlet-mapping>

</web-app>
```

Input

Deployment Steps (using Apache Tomcat as an example):

1. **Create Project Structure:**

```
DatabaseWebApp/
├── src/
│   └── DatabaseServlet.java
└── WebContent/
    ├── WEB-INF/
    │   ├── lib/
    │   │   └── h2-<version>.jar  (e.g., h2-2.2.224.jar)
    │   └── web.xml
    └── index.html (optional, for a welcome page)
```

11. **Download H2 JAR:** Download the H2 database JAR file and place it in `WebContent/WEB-INF/lib/`.
12. **Compile:** Place `servlet-api.jar` (from Tomcat's `lib`) and `h2-<version>.jar` in your classpath when compiling `DatabaseServlet.java`. `javac -cp "path/to/tomcat/lib/servlet-api.jar;path/to/h2-<version>.jar" src/DatabaseServlet.java`
13. **Package (WAR file):** `jar -cvf DatabaseWebApp.war -C WebContent/ .`
14. **Deploy:** Copy `DatabaseWebApp.war` to Tomcat's `webapps` directory.
15. **Access in Browser:** Open your web browser and navigate to: `http://localhost:8080/DatabaseWebApp/products`

Expected Output

A web page will be displayed in your browser, showing an HTML table with product data retrieved from the in-memory H2 database.

```
<!DOCTYPE html>
<html>
<head>
<title>Product List</title>
<style>
body { font-family: Arial, sans-serif; background-color: #f4f4f4; color: #333;
text-align: center; padding-top: 50px; }
```

```
h1 { color: #0056b3; }
table { width: 80%; border-collapse: collapse; margin: 20px auto; background-
color: #fff; box-shadow: 0 2px 4px rgba(0,0,0,0.1); border-radius: 8px;
overflow: hidden; }
th, td { border: 1px solid #ddd; padding: 12px; text-align: left; }
th { background-color: #007bff; color: white; }
tr:nth-child(even) { background-color: #f2f2f2; }
</style>
</head>
<body>
<h1>Products in Database</h1>
<table>
<tr><th>ID</th><th>Name</th><th>Price</th><th>Quantity</th></tr>
<tr><td>1</td><td>Laptop</td><td>$1200.00</td><td>10</td></tr>
<tr><td>2</td><td>Mouse</td><td>$25.50</td><td>100</td></tr>
<tr><td>3</td><td>Keyboard</td><td>$75.00</td><td>50</td></tr>
</table>
<p>Data retrieved from in-memory H2 database.</p>
</body>
</html>
```

Lab 15: Implement a JSP page that displays the current date and time dynamically using JSP scriptlets.

Title

JSP - Dynamic Date and Time Display

Aim

To implement a JavaServer Pages (JSP) page that dynamically displays the current date and time on a web page using JSP scriptlets.

Procedure

1. **Create JSP File:** Create a file with a `.jsp` extension (e.g., `currentTime.jsp`).
2. **Import Classes:** Use the JSP `page` directive to import necessary Java classes (e.g., `java.util.Date`).
3. **Scriptlet (`<% ... %>`):** Use a JSP scriptlet to embed Java code directly into the HTML.
    - Inside the scriptlet, create an instance of `java.util.Date`.
4. **Expression (`<%= ... %>`):** Use a JSP expression to output the value of a Java expression directly into the HTML.
    - Display the `Date` object using an expression.
5. **Deployment:** Place the `.jsp` file directly into your web application's root directory or a subdirectory within `WebContent` (or `webapps/<your_app_name>`). Deploy the web application to a JSP-enabled web server (like Apache Tomcat).

Source Code

1. JSP File (Save as `currentTime.jsp` in your web application's root)

```
<%-- Filename: currentTime.jsp --%>
<%@ page language="java" contentType="text/html; charset=UTF-8"
pageEncoding="UTF-8"%>
<%@ page import="java.util.Date" %> <%-- Import the Date class --%>
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Current Date and Time</title>
    <style>
        body {
            font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;
            background-color: #e0f2f7; /* Light blue background */
            color: #2c3e50; /* Dark text */
            text-align: center;
            padding-top: 80px;
            display: flex;
            flex-direction: column;
            align-items: center;
            justify-content: center;
            min-height: 100vh;
            margin: 0;
        }
        .container {
            background-color: #ffffff;
```

```
            border-radius: 15px;
            box-shadow: 0 8px 16px rgba(0, 0, 0, 0.2);
            padding: 40px 60px;
            max-width: 600px;
            animation: fadeIn 1s ease-out;
        }
        h1 {
            color: #007bff; /* Primary blue */
            margin-bottom: 25px;
            font-size: 2.5em;
            text-shadow: 1px 1px 2px rgba(0,0,0,0.1);
        }
        p {
            font-size: 1.4em;
            color: #555;
            line-height: 1.6;
        }
        .date-time {
            font-weight: bold;
            color: #dc3545; /* Red for emphasis */
            font-size: 1.6em;
            margin-top: 20px;
            padding: 10px 20px;
            background-color: #f8d7da; /* Light red background */
            border-radius: 8px;
            display: inline-block;
            border: 1px solid #dc3545;
        }
        @keyframes fadeIn {
            from { opacity: 0; transform: translateY(-20px); }
            to { opacity: 1; transform: translateY(0); }
        }
    </style>
</head>
<body>
    <div class="container">
        <h1>Current Server Date and Time</h1>
        <p>This page dynamically displays the current date and time on the
server.</p>

        <%
            // JSP Scriptlet: Java code executed on the server side
            Date now = new Date(); // Create a Date object
        %>

        <p class="date-time">
            <%= now %> <%-- JSP Expression: Outputs the value of 'now' --%>
        </p>

        <p>Refresh the page to see the time update!</p>
    </div>
</body>
</html>
```

Input

Deployment Steps (using Apache Tomcat as an example):

1. **Create Project Structure:**
2. DynamicTimeApp/
3. └── WebContent/
4. └── currentTime.jsp

5. **Package (WAR file):** `jar -cvf DynamicTimeApp.war -C WebContent/ .`
6. **Deploy:** Copy `DynamicTimeApp.war` to Tomcat's `webapps` directory.
7. **Access in Browser:** Open your web browser and navigate to:
   `http://localhost:8080/DynamicTimeApp/currentTime.jsp`

Expected Output

A web page will be displayed in your browser, showing a heading and the current date and time on the server. Each time you refresh the page, the displayed time will update.

Example Browser Output:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Current Date and Time</title>
    <style>
        /* ... (CSS styles as in Source Code) ... */
    </style>
</head>
<body>
    <div class="container">
        <h1>Current Server Date and Time</h1>
        <p>This page dynamically displays the current date and time on the
server.</p>

        <p class="date-time">
            Thu May 22 11:39:00 IST 2025 </p>

        <p>Refresh the page to see the time update!</p>
    </div>
</body>
</html>
```

The exact date and time will vary based on when you access the page.