

**Tribhuvan University**

**Faculty of Humanities and Social Sciences**

A LAB REPORT on

**ADVANCE JAVA PROGRAMMING (CACS354)**

**Submitted to**

**Department of Computer Application**

**Divya Gyan College**

**Putalisadak, Kathmandu**

*In partial fulfillment of the requirements for the Bachelor in Computer Application*

**Submitted By:**

Susheel Maharjan (6-2-751-31-2020)

Under the Supervision of

**Ram Gautam**

# Lab 1

## Question 1 Write a GUI application to find sum and difference of two integer numbers. Use two text fields for input and third test field for output and require two buttons for one for addition and another for difference operation, result should be displayed on third textfield.

**OBJECTIVES:**

1. To implement swing to create a GUI application to find the sum and difference of two integer numbers.
2. To create your own custom Exception class.

**THEORY**

Swing is an addition to the Abstract Window Toolkit [AWT] and a library for Java Foundation Classes [JFC]. Compared to AWT, Java Swing has significantly more capability, additional components, enhanced component features, and superior event handling with drag-and-drop support. An exception In Java, propagation is the process of passing an exception from the point of occurrence to a higher-level technique that deals with it. When a rule is broken and is not contained in that method, it moves up the call stack to the person making that call approach. This keeps on until the exception is handled or caught, or it reaches the main method, if uncaught, resulting in the program's termination.

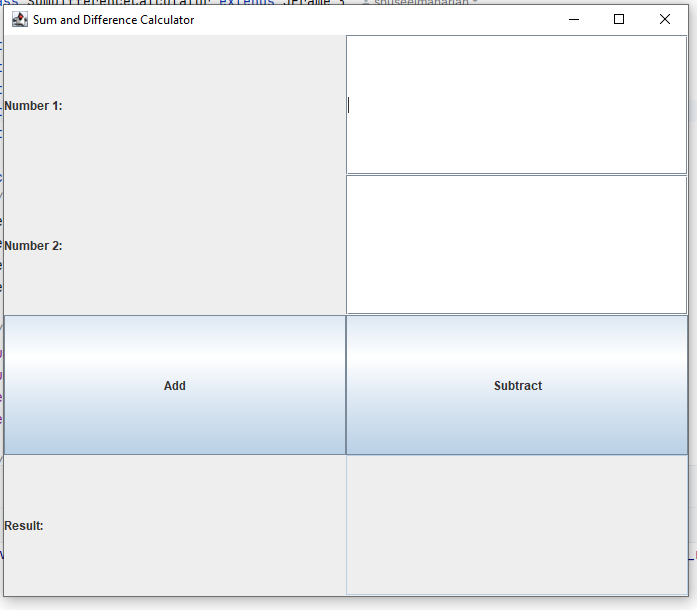
**#Source Code.**

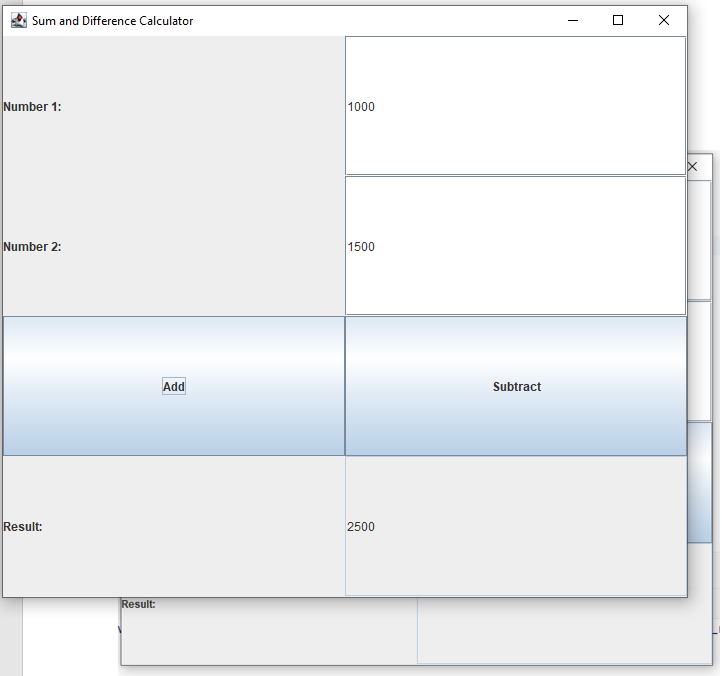
# Calculator Implementation

import javax.swing.\*;  
import java.awt.\*;  
import java.awt.event.ActionEvent;  
import java.awt.event.ActionListener;  
  
public class SumDifferenceCalculator extends JFrame {  
  
 private JTextField number1Field;  
 private JTextField number2Field;  
 private JTextField resultField;  
 private JButton addButton;  
 private JButton subtractButton;  
  
 public SumDifferenceCalculator() {  
 *// Create the frame* setTitle("Sum and Difference Calculator");  
 setSize(700, 600);  
 setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
 setLayout(new GridLayout(4, 2));  
  
 *// Create the text fields* number1Field = new JTextField();  
 number2Field = new JTextField();  
 resultField = new JTextField();  
 resultField.setEditable(false);  
  
 *// Create the buttons* addButton = new JButton("Add");  
 subtractButton = new JButton("Subtract");  
  
 *// Add action listeners to buttons* addButton.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 performAddition();  
 }  
 });  
  
 subtractButton.addActionListener(new ActionListener() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 performSubtraction();  
 }  
 });  
  
 *// Add components to the frame* add(new JLabel("Number 1:"));  
 add(number1Field);  
 add(new JLabel("Number 2:"));  
 add(number2Field);  
 add(addButton);  
 add(subtractButton);  
 add(new JLabel("Result:"));  
 add(resultField);  
  
 *// Set the frame to be visible* setVisible(true);  
 }  
  
 private void performAddition() {  
 try {  
 int num1 = Integer.*parseInt*(number1Field.getText());  
 int num2 = Integer.*parseInt*(number2Field.getText());  
 int sum = num1 + num2;  
 resultField.setText(String.*valueOf*(sum));  
 } catch (NumberFormatException e) {  
 JOptionPane.*showMessageDialog*(this,   
 "Please enter valid integers.",  
 "Error", JOptionPane.*ERROR\_MESSAGE*);  
 }  
 }  
  
 private void performSubtraction() {  
 try {  
 int num1 = Integer.*parseInt*(number1Field.getText());  
 int num2 = Integer.*parseInt*(number2Field.getText());  
 int difference = num1 - num2;  
 resultField.setText(String.*valueOf*(difference));  
 } catch (NumberFormatException e) {  
 JOptionPane.*showMessageDialog*(this,   
 "Please enter valid integers.",   
 "Error", JOptionPane.*ERROR\_MESSAGE*);  
 }  
 }  
  
 public static void main(String[] args) {  
 *// Create and display the GUI* SwingUtilities.*invokeLater*(new Runnable() {  
 @Override  
 public void run() {  
 new SumDifferenceCalculator();  
 }  
 });  
 }  
}

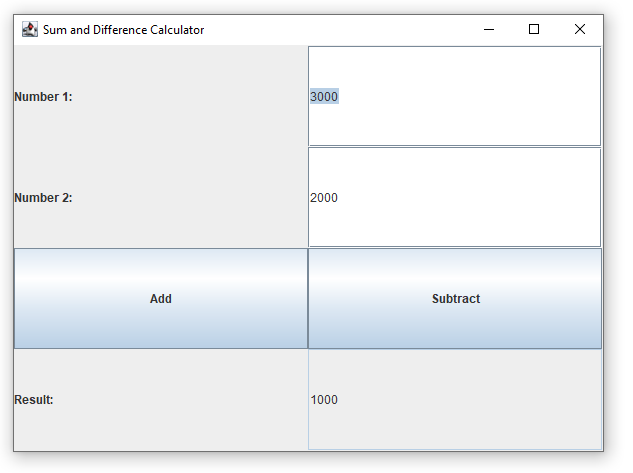
Output

Layout Design Output



Adding Two Numbers 1000+1500 with result 2500  


Subtracting two numbers 3000-2000 with result 1000



**CONCLUSION:**

In conclusion, we created a GUI application to find the sum & difference of two integer numbers using Swing.

## Question 2: What is Exception Propagation? How this can be done in Java programing? Show with proper example after making own custom Exception class.

**THEORY**

Exception propagation refers to movement of exception event from nested try or nested methods calls. A try block can be nested within another try block. Similarly, a method can call another method where each method can handle exception independently or can throw checked/unchecked exceptions. Whenever an exception is raised within a nested try block/method, its exception is pushed to Stack.

**Source Code:**

**#src/Main.java**

import custom.InvalidAgeException;  
import main.AgeValidator;  
  
import java.util.Scanner;  
  
public class Main {  
 public static void main(String[] args) {  
 AgeValidator validator = new AgeValidator();  
  
 System.*out*.println("Enter an age:");  
 Scanner input = new Scanner(System.*in*);  
 int age = input.nextInt();  
  
 try{  
 validator.validateAge(age);  
 }catch (InvalidAgeException e) {  
 System.*err*.println("Exception caught: " + e.getMessage());  
 }  
 }  
}

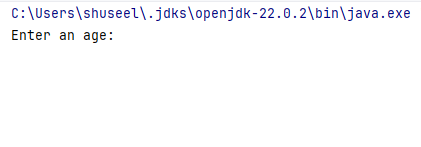
**#src/main/InvalidAgeException.java**

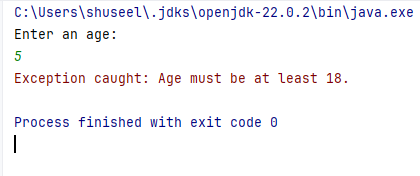
package custom;  
  
public class InvalidAgeException extends Exception{  
 public InvalidAgeException(String message){  
 super(message);  
 }  
}

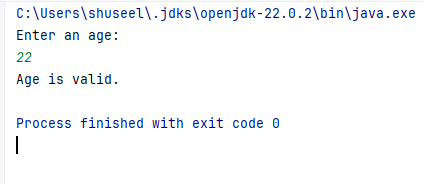
**#src/main/AgeValidator.java**

package main;  
import custom.InvalidAgeException;  
public class AgeValidator {  
 public void validateAge(int age) throws InvalidAgeException {  
 if (age < 0) {  
 throw new InvalidAgeException("Age cannot be negative.");  
 } else if (age < 18) {  
 throw new InvalidAgeException("Age must be at least 18.");  
 }  
 System.*out*.println("Age is valid.");  
 }  
}

Output:







**CONCLUSION**

In conclusion, we created a custom Exception class to demonstrate Exception Propagation.

# Lab 2

## Question 1: How do you execute SQL statement using JDBC, explain with step to be followed? Explain with suitable example.

Steps to Execute SQL Statements Using JDBC

1. Load the JDBC Driver:

* Before establishing a connection to the database, the appropriate JDBC driver needs to be loaded. This step is not strictly required in newer versions of Java, but it's good practice to explicitly load the driver.
* Use Class.forName("com.mysql.cj.jdbc.Driver"); for MySQL.

1. Establish a Connection:

* The next step is to establish a connection to the database using DriverManager.getConnection(). This method requires the database URL, username, and password.
* Example: Connection conn = DriverManager.getConnection(DB\_URL, USER, PASS);

1. Create a Statement:

* To execute SQL queries, you need to create a Statement or PreparedStatement object.
* Statement is generally used for simple SQL queries without parameters.
* Example: Statement stmt = conn.createStatement();

1. Execute the SQL Query:

* Depending on the type of SQL statement (e.g., SELECT, INSERT, UPDATE, DELETE), you use different methods:
* executeQuery(String sql) for SELECT queries, which returns a ResultSet.
* executeUpdate(String sql) for INSERT, UPDATE, or DELETE queries, which returns an integer representing the number of rows affected.
* execute(String sql) for executing any kind of SQL statement, useful for statements that return multiple results.
* Example: ResultSet rs = stmt.executeQuery("SELECT \* FROM Students");

5. Process the Result (if any):

* If the SQL statement is a SELECT query, the results are returned in a ResultSet object. You can iterate through this ResultSet to process the data.
* Example: while(rs.next()) { /\* Process each row \*/ }

1. Close the Resources:

* It's crucial to close all the resources (ResultSet, Statement, Connection) after their usage to free up database resources.
* Example: rs.close(); stmt.close(); conn.close();

In SQL, a "statement" is a single line or a group of lines representing a specific action or operation performed on a database. SQL statements are used to interact with the database, retrieve data, modify data, create or alter database structures, and more. SQL statements are essential for managing and querying relational databases effectively. A PreparedStatement in SQL is a precompiled SQL statement that allows you to execute the same SQL query with different parameter values repeatedly. It is a feature provided by database APIs like JDBC, which improves performance and security by separating the SQL query from the user-supplied parameters. PreparedStatement helps prevent SQL injection attacks and enhances database query execution efficiency by reducing the need for query recompilation, making it a valuable tool for database interactions in applications. JavaBeans are classes that encapsulate many objects into a single object (the bean). It is a Java class that should follow the following conventions:

1. Must implement Serializable.

2. It should have a public no-arg constructor.

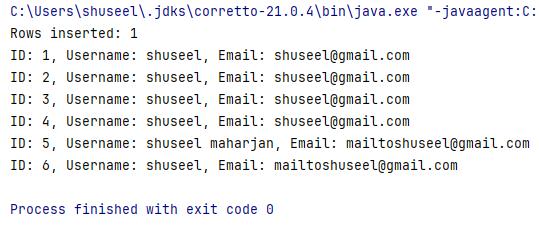
3. All properties in java bean must be private with public getters and setter methods.

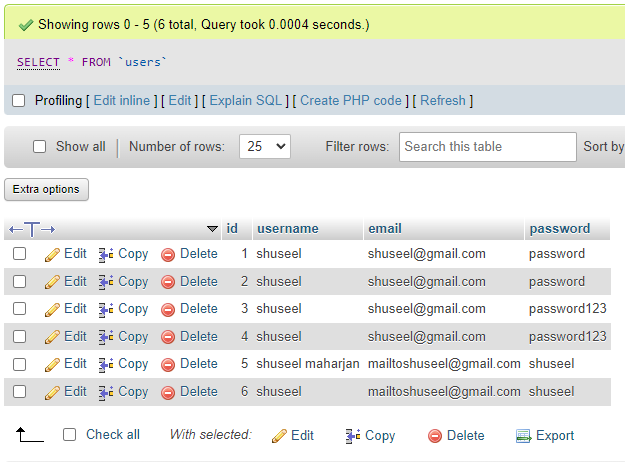
#Source Code

**# JDBC Program to Retrieve Data from a Database**

import java.sql.\*;  
import java.sql.Statement;  
  
  
public class SimpleJdbc {  
  
 public static void main(String[] args) {  
 String url = "jdbc:mysql://localhost:3306/fms";  
 String username = "root";  
 String password = "";  
  
 try {  
 *//Load the JDBC driver* Class.*forName*("com.mysql.cj.jdbc.Driver");  
  
 *//Establish the connection* Connection connection = DriverManager.*getConnection*(url, username, password);  
  
 *//Create a statement* Statement statement = connection.createStatement();  
  
 *//Execute Insert Query* String insertSql = "INSERT INTO users " +  
 "(username, email, password) VALUES " +  
 "('shuseel', 'mailtoshuseel@gmail.com', 'shuseel')";  
 int rowsInserted = statement.executeUpdate(insertSql);  
 System.*out*.println("Rows inserted: " + rowsInserted);  
  
 *//Execute Select Statement* String selectSql = "SELECT id, username, email FROM users";  
 ResultSet resultSet = statement.executeQuery(selectSql);  
  
 *//Process the Result* while (resultSet.next()) {  
 int id = resultSet.getInt("id");  
 String uname = resultSet.getString("username");  
 String email = resultSet.getString("email");  
  
 System.*out*.println("ID: " + id + ", Username: " + uname + ", Email: " + email);  
 }  
  
 *//Close resourses* resultSet.close();  
 statement.close();  
 connection.close();  
  
 } catch (Exception e) {  
 e.printStackTrace();  
 }  
 }  
}

Output:



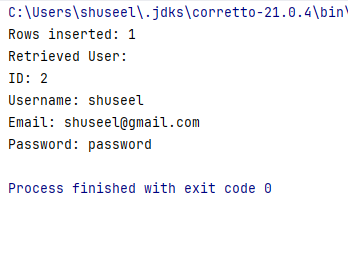


**# Program to Show Statement and PreparedStatement Interface in Java**

#PreparedStatementExample.java

import java.sql.\*;  
  
public class PreparedStatementExample {  
  
 private static final String *URL* = "jdbc:mysql://localhost:3306/fms";  
 private static final String *USER* = "root";  
 private static final String *PASSWORD* = "";  
  
 public void addUser(User user) {  
 String sql = "INSERT INTO users (username, email, password) VALUES (?, ?, ?)";  
 try (Connection conn = DriverManager.*getConnection*(*URL*, *USER*, *PASSWORD*);  
 PreparedStatement stmt = conn.prepareStatement(sql)) {  
  
 stmt.setString(1, user.getUsername());  
 stmt.setString(2, user.getEmail());  
 stmt.setString(3, user.getPassword());  
  
 int rowsInserted = stmt.executeUpdate();  
 System.*out*.println("Rows inserted: " + rowsInserted);  
  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
  
 public User getUser(int id) {  
 String sql = "SELECT \* FROM users WHERE id = ?";  
 User user = null;  
 try (Connection conn = DriverManager.*getConnection*(*URL*, *USER*, *PASSWORD*);  
 PreparedStatement stmt = conn.prepareStatement(sql)) {  
  
 stmt.setInt(1, id);  
 ResultSet rs = stmt.executeQuery();  
  
 if (rs.next()) {  
 user = new User();  
 user.setId(rs.getInt("id"));  
 user.setUsername(rs.getString("username"));  
 user.setEmail(rs.getString("email"));  
 user.setPassword(rs.getString("password"));  
 }  
  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 return user;  
 }  
  
 public static void main(String[] args) {  
 PreparedStatementExample example = new PreparedStatementExample();  
  
 *//create new user* User newUser = new User();  
 newUser.setUsername("shuseel");  
 newUser.setEmail("shuseel@gmail.com");  
 newUser.setPassword("password123");  
  
 *//add user to database* example.addUser(newUser);  
  
 *//print user from database* User retrievedUser = example.getUser(2);  
 if (retrievedUser != null) {  
 System.*out*.println("Retrieved User:");  
 System.*out*.println("ID: " + retrievedUser.getId());  
 System.*out*.println("Username: " + retrievedUser.getUsername());  
 System.*out*.println("Email: " + retrievedUser.getEmail());  
 System.*out*.println("Password: " + retrievedUser.getPassword());  
 } else {  
 System.*out*.println("User not found.");  
 }  
 }  
}  
  
class User {  
 private int id;  
 private String username;  
 private String email;  
 private String password;  
  
 *// Getters and setters* public int getId() { return id; }  
 public void setId(int id) { this.id = id; }  
  
 public String getUsername() { return username; }  
 public void setUsername(String username) { this.username = username; }  
  
 public String getEmail() { return email; }  
 public void setEmail(String email) { this.email = email; }  
  
 public String getPassword() { return password; }  
 public void setPassword(String password) { this.password = password; }  
}

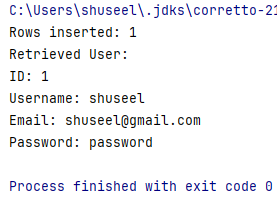
Output



#StatementExample.java

import java.sql.\*;  
  
public class StatementExample {  
  
 private static final String *URL* = "jdbc:mysql://localhost:3306/fms";  
 private static final String *USER* = "root";  
 private static final String *PASSWORD* = "";  
  
 *//method to insert user* public void addUser(User user) {  
 String sql = "INSERT INTO users (username, email, password) VALUES ('"  
 + user.getUsername() + "', '"  
 + user.getEmail() + "', '"  
 + user.getPassword() + "')";  
 try (Connection conn = DriverManager.*getConnection*(*URL*, *USER*, *PASSWORD*);  
 Statement stmt = conn.createStatement()) {  
  
 int rowsInserted = stmt.executeUpdate(sql);  
 System.*out*.println("Rows inserted: " + rowsInserted);  
  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
  
 *//method to call database user with id* public User getUser(int id) {  
 String sql = "SELECT \* FROM users WHERE id = " + id;  
 User user = null;  
 try (Connection conn = DriverManager.*getConnection*(*URL*, *USER*, *PASSWORD*);  
 Statement stmt = conn.createStatement();  
 ResultSet rs = stmt.executeQuery(sql)) {  
  
 if (rs.next()) {  
 user = new User();  
 user.setId(rs.getInt("id"));  
 user.setUsername(rs.getString("username"));  
 user.setEmail(rs.getString("email"));  
 user.setPassword(rs.getString("password"));  
 }  
  
 } catch (SQLException e) {  
 e.printStackTrace();  
 }  
 return user;  
 }  
  
 public static void main(String[] args) {  
 StatementExample example = new StatementExample();  
  
 *//create new user* User newUser = new User();  
 newUser.setUsername("shuseel");  
 newUser.setEmail("shuseel@gmail.com");  
 newUser.setPassword("password");  
  
 *//add user to database* example.addUser(newUser);  
  
 *//retrive user with userid* User retrievedUser = example.getUser(1);  
 if (retrievedUser != null) {  
 System.*out*.println("Retrieved User:");  
 System.*out*.println("ID: " + retrievedUser.getId());  
 System.*out*.println("Username: " + retrievedUser.getUsername());  
 System.*out*.println("Email: " + retrievedUser.getEmail());  
 System.*out*.println("Password: " + retrievedUser.getPassword());  
 } else {  
 System.*out*.println("User not found.");  
 }  
 }  
}

Output:



3. Write a program to simple, Boolean, and indexed property used in Java Bean

#Source Code.

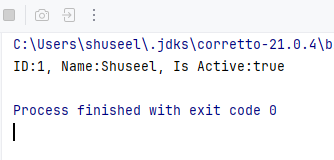
#src/Main.java

public class Main {  
 public static void main(String[] args) {  
 Employee e=new Employee();  
 e.setId(1);  
 e.setName("Shuseel");  
 e.setActive(true);  
 System.*out*.println("ID:"+e.getId()+  
 ", Name:" + e.getName() +  
 ", Is Active:"+e.isActive());  
 }  
}

#src/Employee.java

import java.io.Serializable;  
  
public class Employee implements Serializable{  
 private int id;  
  
 private String name;  
  
 private boolean active;  
  
 public Employee(){  
  
 }  
  
 public void setId(int id){  
 this.id=id;  
 }  
  
 public int getId(){  
 return id;  
 }  
  
 public void setName(String name){  
 this.name=name;  
  
 }  
 public String getName(){  
 return name;  
 }  
  
 public boolean isActive() {  
 return active;  
 }  
  
 public void setActive(boolean active) {  
 this.active = active;  
 }  
}

Output:



# LAB 3

**OBJECTIVES**

1. To create a web application using JSP
2. To write client and server application in RMI

**THEORY**

JSP technology is used to create web application just like Servlet technology. It can be thought of as an extension to Servlet because it provides more functionality than servlet such as expression language, JSTL, etc.

The RMI (Remote Method Invocation) is an API that provides a mechanism to create distributed application in java. The RMI allows an object to invoke methods on an object running in another JVM.

**# JSP Web App**

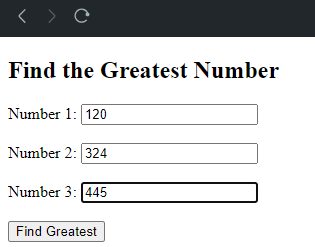
#Source Code

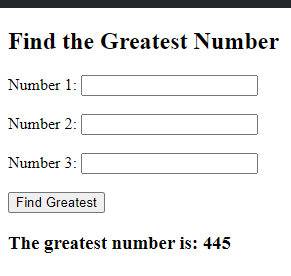
<%@ page language="java" contentType="text/html; charset=UTF-8" pageEncoding="UTF-8"%>  
<!DOCTYPE html>  
<html>  
<head>  
 <title>Greatest Number Finder</title>  
</head>  
<body>  
 <h2>Find the Greatest Number</h2>  
 <form action="index.jsp" method="post">  
 <label for="num1">Number 1:</label>  
 <input type="text" id="num1" name="num1" required>  
 <br><br>  
 <label for="num2">Number 2:</label>  
 <input type="text" id="num2" name="num2" required>  
 <br><br>  
 <label for="num3">Number 3:</label>  
 <input type="text" id="num3" name="num3" required>  
 <br><br>  
 <input type="submit" value="Find Greatest">  
 </form>  
  
 <%  
 // Retrieve numbers from the request  
 String num1Str = request.getParameter("num1");  
 String num2Str = request.getParameter("num2");  
 String num3Str = request.getParameter("num3");  
  
 if (num1Str != null && num2Str != null && num3Str != null) {  
 try {  
 // Convert input strings to integers  
 int num1 = Integer.parseInt(num1Str);  
 int num2 = Integer.parseInt(num2Str);  
 int num3 = Integer.parseInt(num3Str);  
  
 // Find the greatest number  
 int greatest = num1;  
 if (num2 > greatest) {  
 greatest = num2;  
 }  
 if (num3 > greatest) {  
 greatest = num3;  
 }  
  
 // Display the result  
 out.println("<h3>The greatest number is: " + greatest + "</h3>");  
 } catch (NumberFormatException e) {  
 out.println("<h3>Please enter valid numbers.</h3>");  
 }  
 }  
 %>  
</body>  
</html>

#Output

Lab 1 Question 2 Output





#Result:  


**# RMI Server**

*// SellingPrice.java*package org.example.lab3.RMI.server;  
import java.rmi.Remote;  
import java.rmi.RemoteException;  
public interface SellingPrice extends Remote {  
 double calculateSellingPrice() throws RemoteException;  
}

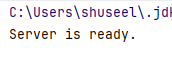
*// SellingPriceImpl.java*package org.example.lab3.RMI.server;  
import java.rmi.RemoteException;  
import java.rmi.server.UnicastRemoteObject;  
public class SellingPriceImpl extends UnicastRemoteObject implements SellingPrice {  
 protected SellingPriceImpl() throws RemoteException {  
 super();  
 }  
 @Override  
 public double calculateSellingPrice() throws RemoteException {  
 double costPrice = 5000;  
 double discountAmt = 50;  
 return costPrice - discountAmt;  
 }  
}

*// RMIServer.java*package org.example.lab3.RMI.server;  
import java.rmi.registry.LocateRegistry;  
import java.rmi.registry.Registry;  
public class RMIServer {  
 public static void main(String[] args) {  
 try {  
*// Create the remote object* SellingPriceImpl obj = new SellingPriceImpl();  
*// Bind the remote object in the registry* Registry registry = LocateRegistry.*createRegistry*(9000);  
 registry.rebind("Selling Price", obj);  
 System.*out*.println("Server is ready...");  
*// Keep the server running indefinitely* synchronized (RMIServer.class) {  
 RMIServer.class.wait();  
 }  
 } catch (Exception e) {  
 System.*err*.println("Server exception: " + e);  
 }  
 }  
}

**# RMI Client**

*// RMIClient.java*package org.example.lab3.RMI.client;  
import org.example.lab3.RMI.server.SellingPrice;  
import java.rmi.registry.LocateRegistry;  
import java.rmi.registry.Registry;  
public class RMIClient {  
 public static void main(String[] args) {  
 try {  
*// Get the registry* Registry registry = LocateRegistry.*getRegistry*("localhost", 9000);  
*// Look up the remote object* SellingPrice stub = (SellingPrice) registry.lookup("Selling Price");  
*// Call the remote method* String response = String.valueOf(stub.calculateSellingPrice());  
 System.*out*.println("Selling Price = " + response);  
 } catch (Exception e) {  
 System.*err*.println("Client exception: " + e);  
 }  
 }  
}

**Output:**

****

**CONCLUSION**

In conclusion, we employed JSP to create a web application that finds and displays the highest number from provided inputs, and we used RMI to build a client and server setup for calculating the selling price of an item