Divya Gyan College
Putalisadak, Kathmandu

Lab Report
of
Advanced Java Programming

Submitted To:
Mr. Siris Timilsina

Submitted By:
Name: Bibash Bogati
Roll no: 07

Acknowledgement

I would like to express my heartfelt gratitude to my supervisor for his invaluable guidance and unwavering support throughout this project. His feedback and encouragement were pivotal in shaping the direction and outcomes of my research.

I am also thankful to my friends and classmates for their collaboration and enthusiasm, which made this project an enjoyable and enriching experience.

Additionally, I extend my appreciation to the college faculty for providing access to essential resources and facilities that contributed to the success of this study.

To all those mentioned above, your contributions have had a lasting impact on my academic journey, and I am truly thankful for your support.

Thank you,

Bibash Bogati

1. ## To create GUI applications using swing, event handling, and layout management

SwingEventLayouting: This term represents the core concepts in Java Swing for creating dynamic and interactive user interfaces. It encompasses the skills and knowledge required to effectively place and organize components within a graphical user interface (GUI) using Swing. The process involves understanding event handling mechanisms, which enable developers to respond to user interactions such as button clicks, mouse movements, and keyboard inputs. Through SwingEventLayouting, developers gain the ability to create responsive and user-friendly applications with adaptable layouts that adjust dynamically to various screen sizes and resolutions, providing a seamless user experience.

Objective:

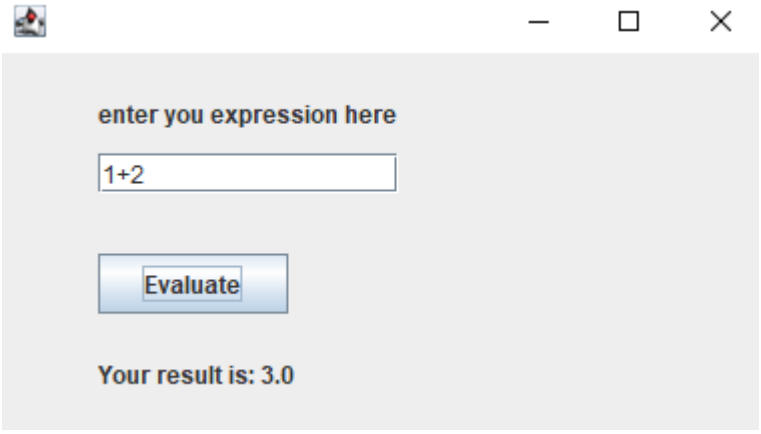
- To learn how to create swing application

Source Code

```
package com.javalab.ui;

public class ExpressionEvaluator {


   public static double evaluateExpression(String expression) {
      char[] tokens = expression.toCharArray();

      // Initialize variables to store the current number and operator
      double currentNumber = 0;
      char currentOperator = '+';
      double result = 0;

      for (int i = 0; i < tokens.length; i++) {
         // Skip whitespace characters
         if (tokens[i] == ' ')
            continue;

         // If the current token is a digit, extract the whole number
         if (Character.isDigit(tokens[i])) {
            StringBuilder sb = new StringBuilder();
            while (i < tokens.length && (Character.isDigit(tokens[i]) || tokens[i] == '.')) {
               sb.append(tokens[i]);
               i++;
            }
            i--; // Decrement i as it gets incremented in the loop
            currentNumber = Double.parseDouble(sb.toString());
         } else {
            // If the current token is an operator, evaluate the previous expression and update
the current operator
            switch (currentOperator) {
               case '+':
                  result += currentNumber;
```

```
                    break;
                case '-':
                    result -= currentNumber;
                    break;
                case '*':
                    result *= currentNumber;
                    break;
                case '/':
                    // Check for division by zero
                    if (currentNumber != 0) {
                        result /= currentNumber;
                    } else {
                        System.out.println("Error: Division by zero is not allowed.");
                        return 0;
                    }
                    break;
            }
            currentNumber = 0;
            currentOperator = tokens[i];
        }
    }

    // Evaluate the last expression
    switch (currentOperator) {
        case '+':
            result += currentNumber;
            break;
        case '-':
            result -= currentNumber;
            break;
        case '*':
            result *= currentNumber;
            break;
        case '/':
            if (currentNumber != 0) {
                result /= currentNumber;
            } else {
                System.out.println("Error: Division by zero is not allowed.");
                return 0;
            }
            break;
    }

    return result;
  }
}
```

Output:



2. To create an application to work with databases.

JDBC (Java Database Connectivity) is a Java API that facilitates Java programs to interact with databases, enabling them to execute various operations like querying, updating, and managing data. It offers a standard way for Java applications to connect to diverse databases.

Objective:

- To learn how to create java application which works with database

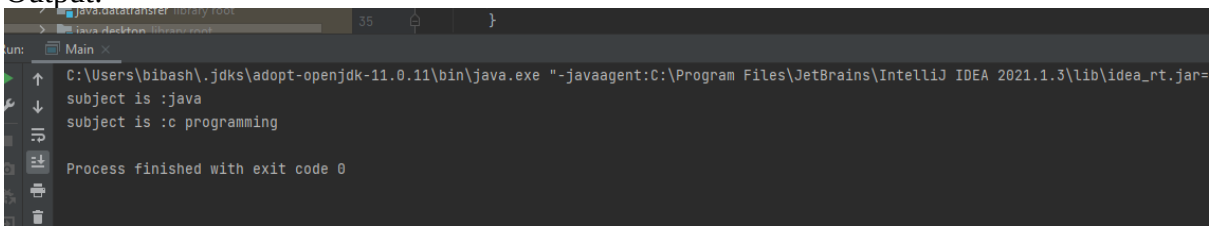
Source Code:

Connection.java

```
package com.javalab;

import com.mysql.cj.jdbc.Driver;
import com.mysql.cj.protocol.Resultset;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;

public class Main {

   public static void main(String[] args) {
      try {
         Class.forName("com.mysql.cj.jdbc.Driver");
         Connection connection =
DriverManager.getConnection("jdbc:mysql://localhost:3306/javalab", "root", "");
         String sql = "select * from subjects";

         PreparedStatement preparedStatement = null;

         ResultSet resultset = null;
```

```
            preparedStatement = connection.prepareStatement(sql);
            resultset = preparedStatement.executeQuery();

            while (resultset.next()) {
                int id = resultset.getInt("id");
                String name = resultset.getString("name");

                System.out.println("subject is :" +name);
            }
        } catch (Exception exception) {
            exception.printStackTrace();
        }
    }
}
```

Output:



3. Creating a JavaBean

JavaBean: In Java, a JavaBean is like a data container that organizes and holds information. It follows specific rules to make it easy for other parts of the program to access and modify the data.

Objective:

- To learn how to create a JavaBean.

Source Code:

```
package com.example.simplewebapp.beans;

public class ReviewBean {
    private String userName;
    private String userEmail;
    private String reviewTitle;
    private String reviewText;

    public String getUserName() {
        return userName;
    }

    public void setUserName(String userName) {
        this.userName = userName;
```

```
    }

    public String getUserEmail() {
        return userEmail;
    }

    public void setUserEmail(String userEmail) {
        this.userEmail = userEmail;
    }

    public String getReviewTitle() {
        return reviewTitle;
    }

    public void setReviewTitle(String reviewTitle) {
        this.reviewTitle = reviewTitle;
    }

    public String getReviewText() {
        return reviewText;
    }

    public void setReviewText(String reviewText) {
        this.reviewText = reviewText;
    }
}
```

4. To create a server-side web-application using Servlet and JSP.

Servlet and JSP are key components of Java web development.

Servlet: A servlet is a Java class that serves as a server-side component, handling client requests and generating dynamic responses. It acts as the backbone of Java web applications, allowing efficient processing of user interactions and data manipulation.

JSP: JSP (JavaServer Pages) is a technology that simplifies the creation of dynamic web pages by integrating Java code directly into HTML templates. This enables developers to separate the presentation logic from the business logic, enhancing the maintainability and readability of web applications.

Objective:

- To learn how to work with servlet and jsp

Source Code:

ReviewController.java:

```
package com.example.simplewebapp;

import com.example.simplewebapp.beans.Employee;
import com.example.simplewebapp.beans.ReviewBean;

import java.io.*;
import javax.servlet.ServletException;
import javax.servlet.http.*;
import javax.servlet.annotation.*;

@WebServlet(name = "reviewController", value = "/submitReview")
public class ReviewController extends HttpServlet {
    private String message;

    public void init() {
        message = "Hello World!";
    }

    public void doGet(HttpServletRequest request, HttpServletResponse response) throws
IOException, ServletException {


    }

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

        String userName = request.getParameter("userName");
        String userEmail = request.getParameter("userEmail");
        String reviewTitle = request.getParameter("reviewTitle");
        String reviewText = request.getParameter("reviewText");

        ReviewBean review = new ReviewBean();
        review.setUserName(userName);
        review.setUserEmail(userEmail);
        review.setReviewTitle(reviewTitle);
        review.setReviewText(reviewText);

        request.setAttribute("userName", review.getUserName());
        request.setAttribute("userEmail", review.getUserEmail());
        request.setAttribute("reviewTitle", review.getReviewTitle());
        request.setAttribute("reviewText", review.getReviewText());

        request.getRequestDispatcher("thank-you.jsp").forward(request, response);
    }

    public void destroy() {
    }
```

```
}
```

Index.jsp

```
<%@ page contentType="text/html; charset=UTF-8" pageEncoding="UTF-8" %>
<!DOCTYPE html>
<html>
<head>
  <style>
    body {
      font-family: Arial, sans-serif;
      background-color: #f2f2f2;
      margin: 0;
      padding: 0;
    }

    .header {
      background-color: #4CAF50;
      color: #ffffff;
      padding: 10px;
      text-align: center;
      font-size: 24px;
      border-radius: 5px 5px 0 0;
    }

    form {
      max-width: 400px;
      margin: 50px auto;
      padding: 20px;
      background-color: #ffffff;
      border-radius: 0 0 5px 5px;
      box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
    }

    form input[type="text"],
    form textarea {
      width: 100%;
      padding: 12px;
      margin: 8px 0;
      box-sizing: border-box;
      border: 1px solid #ccc;
      border-radius: 4px;
    }

    form textarea {
      resize: vertical;
    }

    form input[type="submit"] {
      background-color: #4CAF50;
      color: #ffffff;
      padding: 14px 20px;
```

```
            margin: 20px 0 0;
            border: none;
            border-radius: 4px;
            cursor: pointer;
            width: 100%;
            font-size: 16px;
        }

        form input[type="submit"]:hover {
            background-color: #45a049;
        }

        form label {
            font-weight: bold;
        }
    </style>
</head>
<body>
<div class="header">
    Divya Gyan College Review Form
</div>
<form action="submitReview" method="post">
    <label for="userName">Name:</label>
    <input type="text" id="userName" name="userName" required><br><br>

    <label for="userEmail">Email Id:</label>
    <input type="text" id="userEmail" name="userEmail" required><br><br>

    <label for="reviewTitle">Review Title:</label>
    <input type="text" id="reviewTitle" name="reviewTitle" required><br><br>

    <label for="reviewText">Review:</label>
    <textarea id="reviewText" name="reviewText" rows="4" required></textarea>

    <br><br>
    <input type="submit" value="Submit Review">
</form>
</body>
</html>
```

Thank-you.jsp

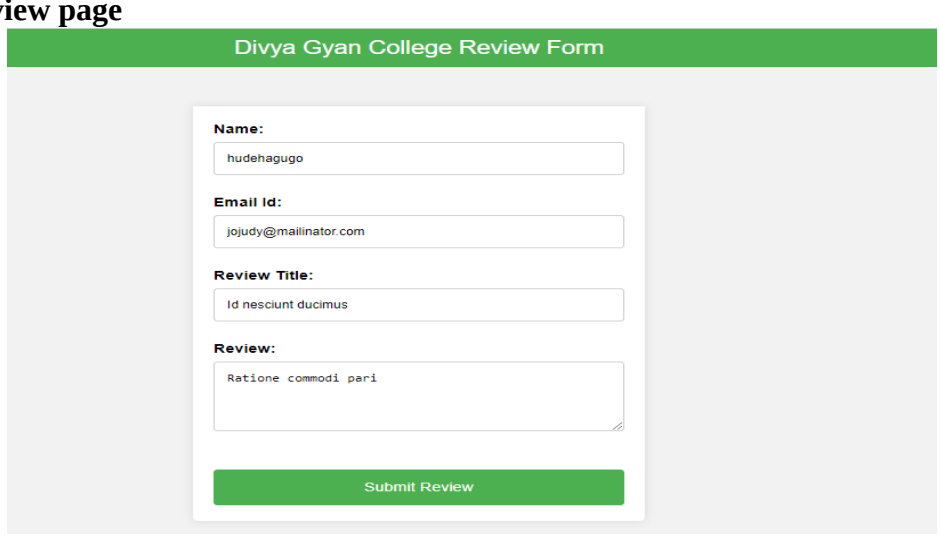
```
<%--
  Created by IntelliJ IDEA.
  User: bibash
  Date: 03/08/2023
  Time: 06:37 pm
  To change this template use File | Settings | File Templates.
--%>
<%@ page contentType="text/html;charset=UTF-8" language="java" %>
<!DOCTYPE html>
```

```
<html>
<head>
  <title>Thank You!</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      background-color: #f2f2f2;
      margin: 0;
      padding: 0;
    }

    .header {
      background-color: #4CAF50;
      color: #ffffff;
      padding: 10px;
      text-align: center;
      font-size: 24px;
      border-radius: 5px 5px 0 0;
    }

    .thank-you-container {
      max-width: 400px;
      margin: 50px auto;
      padding: 20px;
      background-color: #ffffff;
      border-radius: 0 0 5px 5px;
      box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
    }

    .thank-you-message {
      font-size: 18px;
      margin-bottom: 20px;
    }

    .thank-you-message b {
      font-weight: bold;
    }
  </style>
</head>
<body>
<div class="header">
  Divya Gyan College Review Form
</div>
<div class="thank-you-container">
  <h1>Thank You for Your Review!</h1>
  <p class="thank-you-message">
    Dear <b><%= request.getAttribute("userName") %></b>,<br>
    Thank you for your valuable review. We appreciate your feedback.<br>
    We have sent a confirmation email to <b><%= request.getAttribute("userEmail")
%></b>.<br>
    Your review title: <b><%= request.getAttribute("reviewTitle") %></b><br>
```

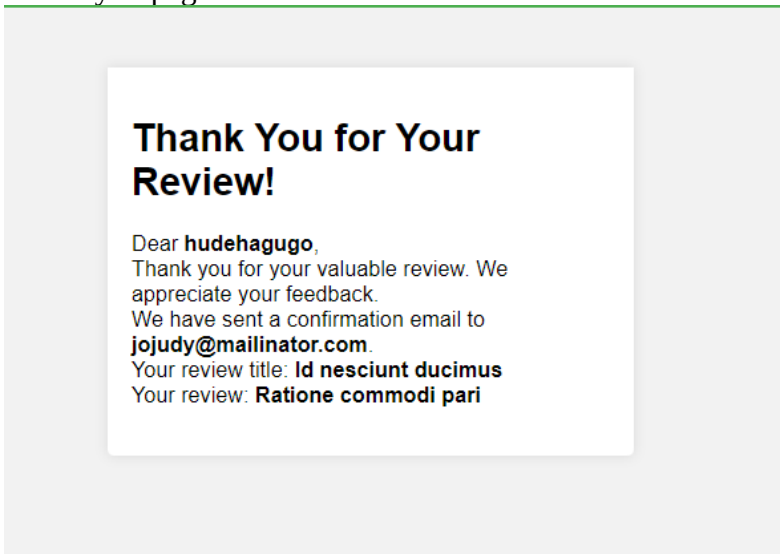
```
  Your review: <b><%= request.getAttribute("reviewText") %></b><br>
 </p>
 <!-- Add any additional content or formatting as needed -->
</div>
</body>
</html>
```

Output:

Review page



Thank you page



4. To create a distributed application using RMI.

RMI (Remote Method Invocation) in Java is a powerful distributed computing technology that facilitates communication between objects running in separate Java Virtual Machines (JVMs). It allows methods to be invoked on objects located in different JVMs, even if they are on different

physical machines. RMI plays a crucial role in building robust and scalable distributed applications, enabling seamless interaction between various components across a network.

Objective:

- To learn how to implement Java RM

Source Code:

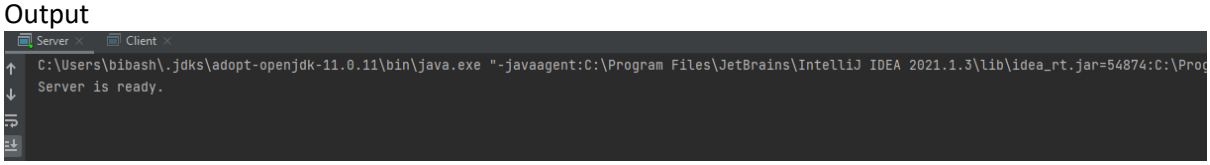
Server.java

```
import java.rmi.*;
public class FileServer {
   public static void main(String[] argv) {
      if(System.getSecurityManager() == null) {
         System.setSecurityManager(new RMISecurityManager());
      }
      try {
         FileInterface fi = new FileImpl("FileServer");
         Naming.rebind("//127.0.0.1/FileServer", fi);
      } catch(Exception e) {
         System.out.println("FileServer: "+e.getMessage());
         e.printStackTrace();
      }
   }
}
```

Client.java

```
import java.rmi.*;

public class FileServer {
   public static void main(String[] argv) {
      if(System.getSecurityManager() == null) {
         System.setSecurityManager(new RMISecurityManager());
      }
      try {
         FileInterface fi = new FileImpl("FileServer");
         Naming.rebind("//127.0.0.1/FileServer", fi);
      } catch(Exception e) {
         System.out.println("FileServer: "+e.getMessage());
         e.printStackTrace();
      }
   }
}
```

Output



Conclusion: Java RMI offers a straightforward and effective way to establish communication between distributed components in a Java application.