USN										

Internal Assessment Test 2 – May 2025

Sub:	Advanced Java - SET 2					Sub Code:	BIS402	Branch	ISE
Date:	23-05-2025	Duration:	90 min's	Max Marks:	50	Sem/Sec:	IV / A, B & C		OBE

Answer any FIVE questions

		MARKS	CO	RBT
1	**Develop an event-handling example using Button and TextField.**	10	CO3	L3

Answer:

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

public class ButtonTextFieldExample extends Frame implements ActionListener {

  TextField textField;
  Button button;

  // Constructor
  ButtonTextFieldExample() {
    // Set layout
    setLayout(null);

    // Create a text field
    textField = new TextField();
    textField.setBounds(60, 50, 170, 30);
    add(textField);

    // Create a button
    button = new Button("Click Me");
    button.setBounds(100, 100, 80, 30);
    add(button);

    // Add action listener to the button
    button.addActionListener(this);

    // Frame settings
    setTitle("Button & TextField Event Example");
    setSize(300, 200);
    setVisible(true);

    // Close button handling
    addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent e) {
        dispose(); // Close the frame
      }
    });
  }
  // Action performed when button is clicked
  public void actionPerformed(ActionEvent e) {
    String input = textField.getText();
    textField.setText("Hello, " + input + "!");
  }

  // Main method
  public static void main(String[] args) {
    new ButtonTextFieldExample();
  }
}
```

2	Describe the Model-View-Controller (MVC) architecture in Swing. **Answer:** **Model-Delegate architecture** Swing uses a modified version of MVC that combines the view and the controller into a single logical entity called the UI delegate. Swing's approach is called either the Model-Delegate architecture or the Separable Model architecture. Therefore, although Swing's component architecture is based on MVC, it does not use a classical implementation of it. Swing's pluggable look and feel is made possible by its Model-Delegate architecture. • Because the **view (look) and controller (feel)** are separate from the model, **the look and feel can be changed without affecting** how the component is used within a program. • it is possible to customize the model without affecting the way that the component appears on the screen or responds to user input. To support the Model-Delegate architecture, most Swing components contain two objects. **The first represents the model.** **The second represents the UI delegate.** Models are defined by interfaces. For example, the model for a button is defined by the ButtonModel interface. UI delegates are classes that inherit ComponentUI. For example, the UI delegate for a button is ButtonUI. Normally, your programs will not interact directly with the UI delegate.	10	CO3	L2
3	What is a servlet? Explain its lifecycle. **Answer:** **The Life Cycle of a Servlet** **Three methods are central to the life cycle of a servlet.These are init( ), service( ), and destroy( ).** • **They are implemented by every servlet and are invoked at specific times by the server.** • **Let us consider a typical user scenario to understand when these** • **methods are called.** **First, assume that a user enters a Uniform Resource Locator (URL) to a web browser. The browser then generates an HTTP request for this URL. This request is then sent to the appropriate server.** **Second, this HTTP request is received by the web server. The server maps this request to a particular servlet. The servlet is dynamically retrieved and loaded into the address space of the server.** **Third, the server invokes the init() method of the servlet. This method is invoked only when the servlet is first loaded into memory. It is possible to pass initialization parameters to the servlet so it may configure itself.** **Fourth, the server invokes the service( ) method of the servlet. This method is called to process the HTTP request. You will see that it is possible for the servlet to read data that has been provided in the HTTP request. It may also formulate an HTTP response for the client.** **The servlet remains in the server's address space and is available to process any other HTTP requests received from clients. The service( ) method is called for each HTTP request.** **Finally, the server may decide to unload the servlet from its memory. The algorithms by which this determination is made are specific to each server. The server calls the destroy () method to relinquish any resources such as file handles that are allocated for the servlet.** **Important data may be saved to a persistent store. The memory allocated for the servlet and its objects can then be garbage collected.**	10	CO4	L2
4	Compare servlets and CGI. In term of servlet explain the concept of cookies Answer:	10	CO4	L2

Servlets offer several advantages in comparison with CGI.

First, performance is significantly better.Servlets execute within the address space of a web server.

It is not necessary to create a separate process to handle each client request.

Second, servlets are platform-independent because they are written in Java.

Third, the Java security manager on the server enforces a set of restrictions to protect the resources on a server machine.

Finally, the full functionality of the Java class libraries is available to a servlet. It can communicate with applets, databases, or other software via the sockets and RMI mechanisms.

The Cookie Class

The Cookie class encapsulates a cookie. A cookie is stored on a clientand contains state information. Cookies are valuable for tracking user activities.

For example, assume that a user visits an online store. A cookie can save the user's name, address, and other information.

The user does not need to enter this data each time he or she visits the store.

A servlet can write a cookie to a user's machine via the addCookie( ) method of the HttpServletResponse interface. The data for that cookie is then included in the header of the HTTP response that is sent to the browser.
The names and values of cookies are stored on the user's machine. Some of the information that can be saved for each cookie includes the following:

• The name of the cookie

• The value of the cookie

• The expiration date of the cookie

• The domain and path of the cookie

The expiration date determines when this cookie is deleted from the user's machine. If an expiration date is not explicitly assigned to a cookie, it is deleted when the current browser session ends.
The domain and path of the cookie determine when it is included in the header of an HTTP request.

There is one constructor for Cookie. It has the signature shown
here:

Cookie(String name, String value)
let's develop a servlet that illustrates how to use cookies.
AddCookie.html

```
<!DOCTYPE html>

<html>

<body>

<center>

<form name="Form1" method="post"

action="http://localhost:8080/Cookie/AddCookieServlet">

<B>Enter a value for MyCookie:</B>

<input type=textbox name="data" size=25 value="">

<input type=submit value="Submit button">

</form>
```

```
</body>

</html>
```

AddCookieServlet.java

```
package Cookie;

import java.io.*;

import jakarta.servlet.*;

import jakarta.servlet.http.*;

public class AddCookieServlet extends HttpServlet {

public void doPost(HttpServletRequest request,

HttpServletResponse response)

throws ServletException, IOException {

// Get parameter from HTTP request.

String data = request.getParameter("data");

// Create cookie.

Cookie cookie = new Cookie("MyCookie", data);

// Add cookie to HTTP response.

response.addCookie(cookie);

// Write output to browser.

response.setContentType("text/html");

PrintWriter pw = response.getWriter();

pw.println("<B>MyCookie has been set to");

pw.println(data);

pw.close();

}

}
```

GetCookiesServlet.java

```
package Cookie;

import java.io.*;

import jakarta.servlet.*;

import jakarta.servlet.http.*;

public class GetCookiesServlet extends HttpServlet {

public void doGet(HttpServletRequest request,

HttpServletResponse response)

throws ServletException, IOException {
```

	// Get cookies from header of HTTP request.
	
	Cookie[] cookies = request.getCookies();
	
	// Display these cookies.
	
	response.setContentType("text/html");
	
	PrintWriter pw = response.getWriter();
	
	pw.println("<B>");
	
	for(int i = 0; i < cookies.length; i++) {
	
	String name = cookies[i].getName();
	
	String value = cookies[i].getValue();
	
	pw.println("name = " + name + "; value = " + value);
	
	}
	
	pw.close();
	
	}
	
	}			

5	Demonstrate a JAVA Program to insert data into Student DATA BASE and retrieve info based on queries (For example create, update, delete, search)
	
	Solution:
	import java.sql.*;
	import java.util.Scanner;
	public class StudentDBApp {
	static final String DB_URL = "jdbc:mysql://localhost:3306/student_db";
	static final String USER = "root";
	static final String PASS = "your_password"; // <-- Replace with your MySQL password
	public static void main(String[] args) {
	Scanner sc = new Scanner(System.in);
	try (Connection conn = DriverManager.getConnection(DB_URL, USER, PASS)) {
	Class.forName("com.mysql.cj.jdbc.Driver");
	while (true) {
	System.out.println("\n===== Student Database Menu =====");
	System.out.println("1. Insert Student");
	System.out.println("2. Update Student");
	System.out.println("3. Delete Student");
	
	System.out.println("4. Search Student by ID");
	System.out.println("5. Display All Students");
	System.out.println("6. Exit");
	System.out.print("Choose an option: ");
	int choice = sc.nextInt();
	switch (choice) {
	case 1:
	System.out.print("Enter ID: ");
	int id = sc.nextInt();
	sc.nextLine(); // clear buffer
	System.out.print("Enter Name: ");
	String name = sc.nextLine();
	System.out.print("Enter Age: ");
	int age = sc.nextInt();
	sc.nextLine();
	System.out.print("Enter Branch: ");
	String branch = sc.nextLine();
	String insertQuery = "INSERT INTO students (id, name, age, branch) VALUES
	(?, ?, ?, ?)";
	try (PreparedStatement pst = conn.prepareStatement(insertQuery)) {
	pst.setInt(1, id);
	pst.setString(2, name);
		10	CO5	L3

```
pst.setInt(3, age);
pst.setString(4, branch);
pst.executeUpdate();
System.out.println(&quot;Student inserted successfully!&quot;);
}
break;
case 2:
System.out.print(&quot;Enter ID of student to update: &quot;);
int updateId = sc.nextInt();
sc.nextLine();
System.out.print(&quot;Enter new name: &quot;);
String newName = sc.nextLine();
String updateQuery = &quot;UPDATE students SET name=? WHERE id=?&quot;;
try (PreparedStatement pst = conn.prepareStatement(updateQuery)) {
pst.setString(1, newName);
pst.setInt(2, updateId);
int rows = pst.executeUpdate();
System.out.println(rows &gt; 0 ? &quot;Update successful!&quot; : &quot;Student not found.&quot;);

}
break;
case 3:
System.out.print(&quot;Enter ID of student to delete: &quot;);
int deleteId = sc.nextInt();
String deleteQuery = &quot;DELETE FROM students WHERE id=?&quot;;
try (PreparedStatement pst = conn.prepareStatement(deleteQuery)) {
pst.setInt(1, deleteId);
int rows = pst.executeUpdate();
System.out.println(rows &gt; 0 ? &quot;Deletion successful!&quot; : &quot;Student not found.&quot;);
}
break;
case 4:
System.out.print(&quot;Enter ID to search: &quot;);
int searchId = sc.nextInt();
String searchQuery = &quot;SELECT * FROM students WHERE id=?&quot;;
try (PreparedStatement pst = conn.prepareStatement(searchQuery)) {
pst.setInt(1, searchId);
ResultSet rs = pst.executeQuery();
if (rs.next()) {
System.out.println(&quot;ID: &quot; + rs.getInt(&quot;id&quot;));
System.out.println(&quot;Name: &quot; + rs.getString(&quot;name&quot;));
System.out.println(&quot;Age: &quot; + rs.getInt(&quot;age&quot;));
System.out.println(&quot;Branch: &quot; + rs.getString(&quot;branch&quot;));
} else {
System.out.println(&quot;Student not found.&quot;);
}
}
break;
case 5:
String displayQuery = &quot;SELECT * FROM students&quot;;
try (Statement stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery(displayQuery)) {
while (rs.next()) {
System.out.println(&quot;ID: &quot; + rs.getInt(&quot;id&quot;) +
&quot;, Name: &quot; + rs.getString(&quot;name&quot;) +
&quot;, Age: &quot; + rs.getInt(&quot;age&quot;) +
&quot;, Branch: &quot; + rs.getString(&quot;branch&quot;));

}
}
break;
case 6:
System.out.println(&quot;Exiting...&quot;);
sc.close();
return;
default:
System.out.println(&quot;Invalid choice!&quot;);
}
}
} catch (Exception e) {
e.printStackTrace();
}
```

	} }			

6	A program to design the Login page and validate the USER_ID and PASSWORD using JSP and Database.
	
	Solution:
	login.jsp
	
	<%@ page language="java" contentType="text/html; charset=UTF-8" %>
	<!DOCTYPE html>
	<html>
	<head>
	<title>Login Page</title>
	</head>
	<body>
	 Login Page
	<form action="validate.jsp" method="post">
	User ID: <input type="text" name="user_id" required>
	
	Password: <input type="password" name="password" required>
	
	<input type="submit" value="Login">
	</form>
	</body>
	</html>
	
	validate.jsp
	
	<%@ page import="java.sql.*" %>
	<%
	String user_id = request.getParameter("user_id");
	String password = request.getParameter("password");
	
	Connection conn = null;
	PreparedStatement stmt = null;
	ResultSet rs = null;
	
	String dbURL = "jdbc:mysql://localhost:3306/login_db";
	String dbUser = "root";
	String dbPass = "your_password"; // Replace with your MySQL root password
	
	try {
	Class.forName("com.mysql.cj.jdbc.Driver");
	conn = DriverManager.getConnection(dbURL, dbUser, dbPass);
	
	String sql = "SELECT * FROM users WHERE user_id = ? AND password = ?";
	stmt = conn.prepareStatement(sql);
	stmt.setString(1, user_id);
	stmt.setString(2, password);
	
	rs = stmt.executeQuery();
	
	if(rs.next()) {
	out.println(" Login Successful. Welcome, " + user_id + "! ");
	} else {
	out.println(" Invalid User ID or Password. Please try again. ");
	}
	} catch(Exception e) {
	out.println("Error: " + e.getMessage());
	} finally {
		10	CO5	L3

	if(rs != null) rs.close();
	if(stmt != null) stmt.close();
	if(conn != null) conn.close();
	}
	%>			

Faculty Signature CCI Signature HOD Signature