LAB SHEET 12 – ANSWERS

JDBC + SWING

```
CREATE TABLE Employee (
    ID INT PRIMARY KEY AUTO_INCREMENT,
    Name VARCHAR(100),
    Age INT,
    Gender VARCHAR(10),
    DepartmentNo INT
);
```

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.sql.*;

public class EmployeeManagementApp {
    private Connection connection;
    private JFrame mainFrame;
    private JTextField nameField, idField, ageField, genderField, deptField;
```

```
private JTable table;

public static void main(String[] args) {
    EventQueue.invokeLater(() -> {
        try {
            EmployeeManagementApp app = new EmployeeManagementApp();
            app.initialize();
        } catch (Exception e) {
            e.printStackTrace();
        }
    });
}

private void initialize() {
    // Initialize the database connection (replace with your actual DB credentials)
    String dbURL = "jdbc:mysql://localhost:3306/your_database_name";
    String username = "your_username";
    String password = "your_password";

    try {
        connection = DriverManager.getConnection(dbURL, username, password);
    } catch (SQLException e) {
        e.printStackTrace();
        System.exit(1);
```

```
}

// Create the main JFrame
mainFrame = new JFrame("Employee Management");
mainFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

// Create components
nameField = new JTextField(20);
idField = new JTextField(5);
ageField = new JTextField(3);
genderField = new JTextField(10);
deptField = new JTextField(5);

JButton insertButton = new JButton("Insert");
insertButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        insertEmployee();
    }
});

JButton searchButton = new JButton("Search");
searchButton.addActionListener(new ActionListener() {
    @Override
```

```
    public void actionPerformed(ActionEvent e) {
        searchEmployee();
    }
});

JButton updateButton = new JButton("Update");
updateButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        updateEmployee();
    }
});

JButton deleteButton = new JButton("Delete");
deleteButton.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        deleteEmployee();
    }
});

// Create a table to display employee data
String[] columnNames = {"ID", "Name", "Age", "Gender", "Department No."};
Object[][] data = new Object[0][columnNames.length];
```

```
table = new JTable(data, columnNames);

// Add components to the main JFrame
JPanel inputPanel = new JPanel();
inputPanel.setLayout(new GridLayout(2, 5));
inputPanel.add(new JLabel("Name:"));
inputPanel.add(nameField);
inputPanel.add(new JLabel("ID:"));
inputPanel.add(idField);
inputPanel.add(new JLabel("Age:"));
inputPanel.add(ageField);
inputPanel.add(new JLabel("Gender:"));
inputPanel.add(genderField);
inputPanel.add(new JLabel("Department No.:"));
inputPanel.add(deptField);

JPanel buttonPanel = new JPanel();
buttonPanel.add(insertButton);
buttonPanel.add(searchButton);
buttonPanel.add(updateButton);
buttonPanel.add(deleteButton);

mainFrame.setLayout(new BorderLayout());
mainFrame.add(inputPanel, BorderLayout.NORTH);
```

```
        mainFrame.add(new JScrollPane(table), BorderLayout.CENTER);
        mainFrame.add(buttonPanel, BorderLayout.SOUTH);

        mainFrame.pack();
        mainFrame.setVisible(true);
    }

    private void insertEmployee() {
        try {
            String name = nameField.getText();
            int id = Integer.parseInt(idField.getText());
            int age = Integer.parseInt(ageField.getText());
            String gender = genderField.getText();
            int deptNo = Integer.parseInt(deptField.getText());

            String insertQuery = "INSERT INTO Employee (Name, ID, Age, Gender,
DepartmentNo) VALUES (?, ?, ?, ?, ?)";
            PreparedStatement statement =
connection.prepareStatement(insertQuery);
            statement.setString(1, name);
            statement.setInt(2, id);
            statement.setInt(3, age);
            statement.setString(4, gender);
            statement.setInt(5, deptNo);
```

```
            statement.executeUpdate();
            statement.close();
            refreshTable();
            clearFields();
        } catch (SQLException e) {
            e.printStackTrace();
            JOptionPane.showMessageDialog(mainFrame, "Error occurred while
inserting the employee.", "Error", JOptionPane.ERROR_MESSAGE);
        }
    }

    private void searchEmployee() {
        int id = Integer.parseInt(idField.getText());

        try {
            String searchQuery = "SELECT * FROM Employee WHERE ID = ?";
            PreparedStatement statement =
connection.prepareStatement(searchQuery);
            statement.setInt(1, id);

            ResultSet resultSet = statement.executeQuery();
            if (resultSet.next()) {
                nameField.setText(resultSet.getString("Name"));
                ageField.setText(String.valueOf(resultSet.getInt("Age")));
                genderField.setText(resultSet.getString("Gender"));
```

```
            deptField.setText(String.valueOf(resultSet.getInt("DepartmentNo")));
        } else {
            JOptionPane.showMessageDialog(mainFrame, "Employee not found.",
"Not Found", JOptionPane.INFORMATION_MESSAGE);
        }

        resultSet.close();
        statement.close();
    } catch (SQLException e) {
        e.printStackTrace();
        JOptionPane.showMessageDialog(mainFrame, "Error occurred while
searching for the employee.", "Error", JOptionPane.ERROR_MESSAGE);
    }
  }

  private void updateEmployee() {
    int id = Integer.parseInt(idField.getText());

    try {
        String name = nameField.getText();
        int age = Integer.parseInt(ageField.getText());
        String gender = genderField.getText();
        int deptNo = Integer.parseInt(deptField.getText());
```

```
        String updateQuery = "UPDATE Employee SET Name = ?, Age = ?, Gender = ?, DepartmentNo = ? WHERE ID = ?";
        PreparedStatement statement = connection.prepareStatement(updateQuery);
        statement.setString(1, name);
        statement.setInt(2, age);
        statement.setString(3, gender);
        statement.setInt(4, deptNo);
        statement.setInt(5, id);

        int rowsUpdated = statement.executeUpdate();
        if (rowsUpdated > 0) {
            JOptionPane.showMessageDialog(mainFrame, "Employee updated successfully.", "Success", JOptionPane.INFORMATION_MESSAGE);
            refreshTable();
        } else {
            JOptionPane.showMessageDialog(mainFrame, "Employee not found.", "Not Found", JOptionPane.INFORMATION_MESSAGE);
        }

        statement.close();
    } catch (SQLException e) {
        e.printStackTrace();
        JOptionPane.showMessageDialog(mainFrame, "Error occurred while updating the employee.", "Error", JOptionPane.ERROR_MESSAGE);
    }
```

```
    }

    private void deleteEmployee() {
        int id = Integer.parseInt(idField.getText ());

        try {
            String deleteQuery = "DELETE FROM Employee WHERE ID = ?";
            PreparedStatement statement =
connection.prepareStatement(deleteQuery);
            statement.setInt(1, id);

            int rowsDeleted = statement.executeUpdate();
            if (rowsDeleted > 0) {
                JOptionPane.showMessageDialog(mainFrame, "Employee deleted
successfully.", "Success", JOptionPane.INFORMATION_MESSAGE);
                refreshTable();
                clearFields();
            } else {
                JOptionPane.showMessageDialog(mainFrame, "Employee not found.",
"Not Found", JOptionPane.INFORMATION_MESSAGE);
            }

            statement.close();
        } catch (SQLException e) {
            e.printStackTrace();
```

```
            JOptionPane.showMessageDialog(mainFrame, "Error occurred while deleting the employee.", "Error", JOptionPane.ERROR_MESSAGE);
        }
    }

    private void refreshTable() {
        try {
            String selectQuery = "SELECT * FROM Employee";
            Statement statement = connection.createStatement();

            ResultSet resultSet = statement.executeQuery(selectQuery);
            ResultSetMetaData metaData = resultSet.getMetaData();
            int columnCount = metaData.getColumnCount();

            DefaultTableModel model = new DefaultTableModel();
            model.setColumnIdentifiers(new Object[]{"ID", "Name", "Age", "Gender", "Department No."});

            while (resultSet.next()) {
                Object[] rowData = new Object[columnCount];
                for (int i = 1; i <= columnCount; i++) {
                    rowData[i - 1] = resultSet.getObject(i);
                }
                model.addRow(rowData);
            }
```

```
            table.setModel(model);
            resultSet.close();
            statement.close();
        } catch (SQLException e) {
            e.printStackTrace();
            JOptionPane.showMessageDialog(mainFrame, "Error occurred while
refreshing the table.", "Error", JOptionPane.ERROR_MESSAGE);
        }
    }

    private void clearFields() {
        nameField.setText("");
        idField.setText("");
        ageField.setText("");
        genderField.setText("");
        deptField.setText("");
    }
}
```

Discussion Questions

Exceptions Handling:

During the process, exceptions are raised when unexpected or erroneous situations occur.

Developers use try-catch blocks to handle exceptions gracefully.

When an exception is thrown within the try block, the catch block(s) with compatible exception types are checked, and the appropriate one is executed.

If no matching catch block is found, the program terminates with an unhandled exception.

JDBC and Swing Connection Issues:

JDBC (Java Database Connectivity) issues can include connection failures, SQL syntax errors, and result set processing problems.

Common JDBC problems involve misconfigured database credentials or drivers, leading to connection errors.

Swing (Java GUI library) connection issues can occur due to improper event handling, causing unresponsive UI or concurrency problems.

Incorrect thread usage within Swing components might result in graphical glitches or crashes.

Using Threads:

Threads can be employed to enhance program performance and responsiveness.

In a GUI application like Swing, long-running tasks should run on separate threads (not on the UI thread) to avoid freezing the user interface.

For example, database queries or heavy computations should be delegated to worker threads to maintain a smooth user experience.

Java provides the Thread class and Executor framework to manage threads efficiently.

Proper synchronization mechanisms like locks or semaphores should be used to avoid thread-related issues like data races or deadlocks.