JDBC + SWING:

1. Create a table called "Employee" which will store employee: Name, ID, Age, Gender, Department no. (You may decide the data types ☺)

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;

public class EmployeeTableCreator {
  public static void main(String[] args) {
    String url = "jdbc:sqlite:/path/to/your/database.db"; // Replace this with the path to your SQLite database file
    String sql = "CREATE TABLE Employee (" +
        "ID INTEGER PRIMARY KEY," +
        "Name TEXT NOT NULL," +
        "Age INTEGER," +
        "Gender TEXT," +
        "DepartmentNo INTEGER" +
        ")";

    try (Connection connection = DriverManager.getConnection(url);
       Statement statement = connection.createStatement()) {

      statement.execute(sql);
      System.out.println("Employee table created successfully.");

    } catch (SQLException e) {
      System.err.println("Error creating Employee table: " + e.getMessage());
    }
  }
}
```

2. Then create a suitable swing based UI to enter required data to the Employee DB. Note that the user needs to insert all the five details to insert a new employee record to the database. (Insert)

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;

public class EmployeeInsertUI extends JFrame implements ActionListener {
  private JTextField nameField, idField, ageField, genderField, departmentNoField;
  private JButton insertButton;

  public EmployeeInsertUI() {
    setTitle("Employee Insert");
    setSize(400, 200);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    setLayout(new GridLayout(6, 2));

    JLabel nameLabel = new JLabel("Name:");
    nameField = new JTextField();
    add(nameLabel);
    add(nameField);

    JLabel idLabel = new JLabel("ID:");
    idField = new JTextField();
    add(idLabel);
    add(idField);

    JLabel ageLabel = new JLabel("Age:");
    ageField = new JTextField();
    add(ageLabel);
    add(ageField);

    JLabel genderLabel = new JLabel("Gender:");
    genderField = new JTextField();
    add(genderLabel);
    add(genderField);

    JLabel departmentNoLabel = new JLabel("Department No:");
    departmentNoField = new JTextField();
    add(departmentNoLabel);
    add(departmentNoField);

    insertButton = new JButton("Insert");
    insertButton.addActionListener(this);
    add(new JLabel());
    add(insertButton);

    setVisible(true);
  }

  @Override
  public void actionPerformed(ActionEvent e) {
    if (e.getSource() == insertButton) {
        insertEmployeeRecord();
    }
  }

  private void insertEmployeeRecord() {
    String url = "jdbc:sqlite:/path/to/your/database.db"; // Replace this with the path to your
SQLite database file

    String name = nameField.getText();
    int id = Integer.parseInt(idField.getText());
    int age = Integer.parseInt(ageField.getText());
    String gender = genderField.getText();
    int departmentNo = Integer.parseInt(departmentNoField.getText());

    try (Connection connection = DriverManager.getConnection(url);
         PreparedStatement preparedStatement = connection.prepareStatement(
                 "INSERT INTO Employee (ID, Name, Age, Gender, DepartmentNo) VALUES (?, ?, ?, ?,
?)")) {

        preparedStatement.setInt(1, id);
        preparedStatement.setString(2, name);
        preparedStatement.setInt(3, age);
        preparedStatement.setString(4, gender);
        preparedStatement.setInt(5, departmentNo);

        int rowsInserted = preparedStatement.executeUpdate();
        if (rowsInserted > 0) {
            System.out.println("Employee record inserted successfully.");
        } else {
            System.out.println("Failed to insert employee record.");
        }

    } catch (SQLException ex) {
        System.err.println("Error inserting employee record: " + ex.getMessage());
    }
  }

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

3. Insert 5 records to the table using the UI.

4. In the same UI create a another internal frame to perform search/update and delete operations.

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

public class EmployeeInsertUI extends JFrame implements ActionListener {
  //... (existing code for the employee insertion UI)

  private JInternalFrame searchUpdateDeleteFrame;
  private JTextField searchIdField, updateIdField, updateNameField, updateAgeField,
updateGenderField, updateDepartmentNoField;
  private JButton searchButton, updateButton, deleteButton;

  public EmployeeInsertUI() {
    //... (existing code for the employee insertion UI)

    // Create the internal frame for search, update, and delete operations
    searchUpdateDeleteFrame = new JInternalFrame("Search/Update/Delete Employee", true,
true, true, true);
    searchUpdateDeleteFrame.setLayout(new GridLayout(8, 2));
    searchUpdateDeleteFrame.setSize(400, 300);
    searchUpdateDeleteFrame.setVisible(true);

    JLabel searchIdLabel = new JLabel("Search Employee by ID:");
    searchIdField = new JTextField();
    searchButton = new JButton("Search");
    searchButton.addActionListener(this);
    searchUpdateDeleteFrame.add(searchIdLabel);
    searchUpdateDeleteFrame.add(searchIdField);
    searchUpdateDeleteFrame.add(searchButton);

    JLabel updateIdLabel = new JLabel("Update Employee ID:");
    updateIdField = new JTextField();
    JLabel updateNameLabel = new JLabel("Name:");
    updateNameField = new JTextField();
    JLabel updateAgeLabel = new JLabel("Age:");
    updateAgeField = new JTextField();
    JLabel updateGenderLabel = new JLabel("Gender:");
    updateGenderField = new JTextField();
    JLabel updateDepartmentNoLabel = new JLabel("Department No:");
    updateDepartmentNoField = new JTextField();
    updateButton = new JButton("Update");
    updateButton.addActionListener(this);
    deleteButton = new JButton("Delete");
    deleteButton.addActionListener(this);
```

```
        searchUpdateDeleteFrame.add(updateIdLabel);
        searchUpdateDeleteFrame.add(updateIdField);
        searchUpdateDeleteFrame.add(updateNameLabel);
        searchUpdateDeleteFrame.add(updateNameField);
        searchUpdateDeleteFrame.add(updateAgeLabel);
        searchUpdateDeleteFrame.add(updateAgeField);
        searchUpdateDeleteFrame.add(updateGenderLabel);
        searchUpdateDeleteFrame.add(updateGenderField);
        searchUpdateDeleteFrame.add(updateDepartmentNoLabel);
        searchUpdateDeleteFrame.add(updateDepartmentNoField);
        searchUpdateDeleteFrame.add(updateButton);
        searchUpdateDeleteFrame.add(deleteButton);

        add(searchUpdateDeleteFrame);
    }

    @Override
    public void actionPerformed(ActionEvent e) {
        // Handle the actions for Insert, Search, Update, and Delete buttons

        if (e.getSource() == insertButton) {
            insertEmployeeRecord();
        } else if (e.getSource() == searchButton) {
            searchEmployee();
        } else if (e.getSource() == updateButton) {
            updateEmployee();
        } else if (e.getSource() == deleteButton) {
            deleteEmployee();
        }
    }

    private void searchEmployee() {
        // Implement the logic to search an employee by ID and display the details in the respective
text fields.
        // You can use a similar approach as in the insertEmployeeRecord() method but with a SELECT
query.
    }

    private void updateEmployee() {
        // Implement the logic to update the employee details based on the provided ID.
        // You can use a similar approach as in the insertEmployeeRecord() method but with an
UPDATE query.
    }

    private void deleteEmployee() {
        // Implement the logic to delete an employee record based on the provided ID.
        // You can use a similar approach as in the insertEmployeeRecord() method but with a DELETE
query.
```

```
    }

    // ... (existing code for the main method)
}
```

I. Enable searching with employee ID, load and display the record in an table structure.

(Search)

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

public class EmployeeInsertUI extends JFrame implements ActionListener {
  //... (existing code)

  private JTable searchResultTable;

  public EmployeeInsertUI() {
    //... (existing code)

    // Create the internal frame for search, update, and delete operations
    searchUpdateDeleteFrame = new JInternalFrame("Search/Update/Delete
Employee", true, true, true, true);
    //... (existing code)

    searchResultTable = new JTable(new DefaultTableModel(new Object[]{"ID",
"Name", "Age", "Gender", "DepartmentNo"}, 0));
    JScrollPane scrollPane = new JScrollPane(searchResultTable);
    searchUpdateDeleteFrame.add(scrollPane);

    //... (existing code)
  }

  @Override
  public void actionPerformed(ActionEvent e) {
    // Handle the actions for Insert, Search, Update, and Delete buttons
    //... (existing code)

    if (e.getSource() == searchButton) {
      searchEmployee();
    } else if (e.getSource() == updateButton) {
      updateEmployee();
    } else if (e.getSource() == deleteButton) {
      deleteEmployee();
```

```
        }
    }

    private void searchEmployee() {
        int searchID;
        try {
            searchID = Integer.parseInt(searchIdField.getText());
        } catch (NumberFormatException ex) {
            System.err.println("Invalid Employee ID");
            return;
        }

        DefaultTableModel tableModel = (DefaultTableModel)
searchResultTable.getModel();
        tableModel.setRowCount(0); // Clear previous search results

        try {
            String url = "jdbc:sqlite:/path/to/your/database.db"; // Replace this with the
path to your SQLite database file

            Connection connection = DriverManager.getConnection(url);
            PreparedStatement preparedStatement = connection.prepareStatement(
                    "SELECT ID, Name, Age, Gender, DepartmentNo FROM Employee WHERE
ID = ?");
            preparedStatement.setInt(1, searchID);

            ResultSet resultSet = preparedStatement.executeQuery();
            while (resultSet.next()) {
                int id = resultSet.getInt("ID");
                String name = resultSet.getString("Name");
                int age = resultSet.getInt("Age");
                String gender = resultSet.getString("Gender");
                int departmentNo = resultSet.getInt("DepartmentNo");

                tableModel.addRow(new Object[]{id, name, age, gender, departmentNo});
            }

            resultSet.close();
            preparedStatement.close();
            connection.close();
        } catch (SQLException ex) {
            System.err.println("Error searching employee: " + ex.getMessage());
        }
    }

    //... (existing code)
}
```

II. Enable editing an existing record and save that to the DB. (Update)

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

public class EmployeeInsertUI extends JFrame implements ActionListener {
  //... (existing code)

  private JTable searchResultTable;
  private DefaultTableModel tableModel;

  //... (existing code)

  @Override
  public void actionPerformed(ActionEvent e) {
    //... (existing code)

    if (e.getSource() == updateButton) {
      updateEmployee();
    } else if (e.getSource() == deleteButton) {
      deleteEmployee();
    }
  }

  private void searchEmployee() {
    //... (existing code)
  }

  private void updateEmployee() {
    int updateID;
    try {
      updateID = Integer.parseInt(updateIdField.getText());
    } catch (NumberFormatException ex) {
      System.err.println("Invalid Employee ID");
      return;
    }

    String updateName = updateNameField.getText();
    int updateAge;
    try {
      updateAge = Integer.parseInt(updateAgeField.getText());
    } catch (NumberFormatException ex) {
      System.err.println("Invalid Age");
      return;
```

```
        }
        String updateGender = updateGenderField.getText();
        int updateDepartmentNo;
        try {
          updateDepartmentNo                                                    =
Integer.parseInt(updateDepartmentNoField.getText());
        } catch (NumberFormatException ex) {
          System.err.println("Invalid Department No");
          return;
        }

        try {
          String url = "jdbc:sqlite:/path/to/your/database.db"; // Replace this with the
path to your SQLite database file

          Connection connection = DriverManager.getConnection(url);
          PreparedStatement preparedStatement = connection.prepareStatement(
              "UPDATE Employee SET Name = ?, Age = ?, Gender = ?, DepartmentNo =
? WHERE ID = ?");
          preparedStatement.setString(1, updateName);
          preparedStatement.setInt(2, updateAge);
          preparedStatement.setString(3, updateGender);
          preparedStatement.setInt(4, updateDepartmentNo);
          preparedStatement.setInt(5, updateID);

          int rowsUpdated = preparedStatement.executeUpdate();
          if (rowsUpdated > 0) {
            System.out.println("Employee record updated successfully.");
            // Update the data in the JTable
            int selectedRow = searchResultTable.getSelectedRow();
            if (selectedRow != -1) {
              tableModel.setValueAt(updateName, selectedRow, 1);
              tableModel.setValueAt(updateAge, selectedRow, 2);
              tableModel.setValueAt(updateGender, selectedRow, 3);
              tableModel.setValueAt(updateDepartmentNo, selectedRow, 4);
            }
          } else {
            System.out.println("Failed to update employee record.");
          }

          preparedStatement.close();
          connection.close();
        } catch (SQLException ex) {
          System.err.println("Error updating employee record: " + ex.getMessage());
        }
      }

      //... (existing code)
```

```
}
```

III. Enable selected records to be deleted from the table. (Delete)

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

public class EmployeeInsertUI extends JFrame implements ActionListener {
   //... (existing code)

   private JTable searchResultTable;
   private DefaultTableModel tableModel;

   //... (existing code)

   @Override
   public void actionPerformed(ActionEvent e) {
      //... (existing code)

      if (e.getSource() == deleteButton) {
         deleteEmployee();
      }
   }

   private void searchEmployee() {
      //... (existing code)
   }

   private void updateEmployee() {
      //... (existing code)
   }

   private void deleteEmployee() {
      int selectedRow = searchResultTable.getSelectedRow();
      if (selectedRow == -1) {
         System.err.println("No employee record selected.");
         return;
      }

      int deleteID = (int) searchResultTable.getValueAt(selectedRow, 0);

      try {
         String url = "jdbc:sqlite:/path/to/your/database.db"; // Replace this with the
path to your SQLite database file
```

```
            Connection connection = DriverManager.getConnection(url);
            PreparedStatement preparedStatement = connection.prepareStatement(
                "DELETE FROM Employee WHERE ID = ?");
            preparedStatement.setInt(1, deleteID);

            int rowsDeleted = preparedStatement.executeUpdate();
            if (rowsDeleted > 0) {
                System.out.println("Employee record deleted successfully.");
                tableModel.removeRow(selectedRow); // Remove the selected row from
the JTable
            } else {
                System.out.println("Failed to delete employee record.");
            }

            preparedStatement.close();
            connection.close();
        } catch (SQLException ex) {
            System.err.println("Error deleting employee record: " + ex.getMessage());
        }
    }

    //... (existing code)
}
```

Discussion Questions:

1. Discuss how exceptions are handled during the process.

   In the provided code, various database-related operations are carried out using JDBC (Java Database Connectivity). During these operations, exceptions related to database connections, SQL queries, and data retrieval can occur.

   The code uses try-catch blocks to handle exceptions gracefully. Whenever an exception is encountered, it is caught in the corresponding catch block, and an error message is printed to the console using System.err.println() to provide information about the error.

   Proper exception handling ensures that the program does not crash abruptly and allows it to handle exceptional situations gracefully. It helps developers identify and diagnose issues easily during development or when the program is in use.

2. Discuss the main issues occurred during the JDBC and Swing connection process.

   Some of the main issues that can occur during the JDBC and Swing connection process include:
   Missing JDBC Driver: If the JDBC driver (e.g., SQLite JDBC driver) is not added to the project's classpath, the program won't be able to establish a database connection.

Incorrect Database URL: Providing an incorrect database URL can lead to connection failures, as the program won't be able to locate the database.
SQL Syntax Errors: Incorrect SQL queries can lead to SQLExceptions when executing statements. These errors may result from misspelled table/column names or incorrect query formation.

Thread Synchronization Issues: When Swing GUI components are accessed from multiple threads, it can lead to thread synchronization issues and result in unpredictable behavior or GUI freezes.
Memory Leaks: Improper resource management, such as failing to close connections, statements, or result sets, can lead to memory leaks and performance issues.

3. Discuss how you can use threads in this program.

in the provided program, threads can be used to perform long-running tasks (e.g., database queries) in the background without freezing the UI. Swing provides the Event Dispatch Thread (EDT) for handling UI-related tasks, and it's essential to keep the EDT responsive.

For example, when performing database operations, such as searching for an employee or executing complex queries, these operations can be performed in a separate thread, leaving the EDT free to handle user interactions.

To use threads, you can create a new thread using the Thread class or use higher-level constructs like SwingWorker for tasks that provide intermediate results during their execution.

By using threads properly, you can ensure that the UI remains responsive, and the user can interact with the application even while database operations are being performed in the background. Additionally, using threads can help prevent the "Not Responding" issues that may occur when long-running tasks are executed on the EDT. However, it's crucial to manage thread synchronization and handle concurrency issues properly to avoid race conditions and data inconsistencies.