V. Write a Java program to demonstrate the working of different collection classes. [Use package structure to store multiple classes].

To demonstrate the working of different collection classes in Java, we'll create a Java program that uses various collection classes like `ArrayList`, `HashSet`, `HashMap`, and `LinkedList`.

- ArrayList Example (`ListExample.java`): Demonstrates adding, accessing, checking, and removing elements from an `ArrayList`. It allows duplicates and maintains insertion order.
- HashSet Example (`SetExample.java`): Demonstrates adding, checking, and removing elements from a `HashSet`. It does not allow duplicates and does not maintain any specific order.
- HashMap Example (`MapExample.java`): Demonstrates adding, accessing, checking, and removing key-value pairs from a `HashMap`. It allows null keys and values and does not maintain any specific order.

Project Structure:

```
src/
└── collectionsdemo/
    ├── CollectionExample.java
    ├── ListExample.java
    ├── SetExample.java
    └── MapExample.java
```

1. `CollectionExample.java`

This is the main class that will call other classes to demonstrate the working of different collections.

```
package collectionsdemo;
public class CollectionExample {
public static void main(String[] args) {
ListExample.demoArrayList();
SetExample.demoHashSet();
MapExample.demoHashMap();
}
}
```

2. `ListExample.java`

This class demonstrates the usage of `ArrayList`.

```
package collectionsdemo;
import java.util.ArrayList;
import java.util.List;
public class ListExample {
public static void demoArrayList() {
System.out.println("ArrayList Example:");
List<String> arrayList = new ArrayList<>();
arrayList.add("Apple");
arrayList.add("Banana");
arrayList.add("Orange");
arrayList.add("Apple"); // Duplicate element
for (String fruit : arrayList) {
System.out.println(fruit);
}
System.out.println("Size of ArrayList: " + arrayList.size());
```

```
System.out.println("Is 'Banana' present? " + arrayList.contains("Banana"));
System.out.println("Removing 'Apple'...");
arrayList.remove("Apple");
System.out.println("ArrayList after removal:");
for (String fruit : arrayList) {
System.out.println(fruit);
}
System.out.println();
}
}
```

3. `SetExample.java`

This class demonstrates the usage of `HashSet`.

```
package collectionsdemo;
import java.util.HashSet;
import java.util.Set;
public class SetExample {
public static void demoHashSet() {
System.out.println("HashSet Example:");
Set<String> hashSet = new HashSet<>();
hashSet.add("Apple");
hashSet.add("Banana");
hashSet.add("Orange");
hashSet.add("Apple"); // Duplicate element, won't be added
for (String fruit : hashSet) {
System.out.println(fruit);
}
System.out.println("Size of HashSet: " + hashSet.size());
System.out.println("Is 'Banana' present? " + hashSet.contains("Banana"));
System.out.println("Removing 'Orange'...");
hashSet.remove("Orange");
System.out.println("HashSet after removal:");
for (String fruit : hashSet) {
System.out.println(fruit);
}
System.out.println();
}
}
```

4. `MapExample.java`

This class demonstrates the usage of `HashMap`.

```
package collectionsdemo;
import java.util.HashMap;
import java.util.Map;
public class MapExample {
public static void demoHashMap() {
System.out.println("HashMap Example:");
Map<Integer, String> hashMap = new HashMap<>();
```

```
hashMap.put(1, "Apple");
hashMap.put(2, "Banana");
hashMap.put(3, "Orange");
hashMap.put(1, "Grapes"); // Replaces the value for key 1
for (Map.Entry<Integer, String> entry : hashMap.entrySet()) {
System.out.println("Key: " + entry.getKey() + ", Value: " + entry.getValue());
}
System.out.println("Size of HashMap: " + hashMap.size());
System.out.println("Is key 2 present? " + hashMap.containsKey(2));
System.out.println("Removing key 3...");
hashMap.remove(3);
System.out.println("HashMap after removal:");
for (Map.Entry<Integer, String> entry : hashMap.entrySet()) {
System.out.println("Key: " + entry.getKey() + ", Value: " + entry.getValue());
}
System.out.println();
}
}
```

How to Run:

1. Compile the classes from the `src` directory:
   javac collectionsdemo/*.java
2. Run the `CollectionExample` class:
   java collectionsdemo.CollectionExample

Output:

```
ArrayList Example:
Apple
Banana
Orange
Apple
Size of ArrayList: 4
Is 'Banana' present? true
Removing 'Apple'...
ArrayList after removal:
Banana
Orange
Apple

HashSet
Example:
Apple
Banana
Orange
Size of HashSet: 3
Is 'Banana' present? true
Removing 'Orange'...
HashSet after removal:
Apple
Banana
```

HashMap Example:
Key: 1, Value: Grapes
Key: 2, Value: Banana
Key: 3, Value: Orange
Size of HashMap: 3
Is key 2 present? true
Removing key 3...
HashMap after removal:
Key: 1, Value: Grapes
Key: 2, Value: Banana

VI. Write a program to synchronize the threads acting on the same object. [Consider the example of any reservations like railway, bus, movie ticket booking, etc.]

To demonstrate thread synchronization in Java, let's consider a simple example of a movie ticket booking system where multiple threads are trying to book tickets simultaneously. We need to ensure that the ticket booking process is synchronized so that no two threads can book the same ticket at the same time.

Program: Synchronized Movie Ticket Booking-

Key Points:

- Synchronization: The `synchronized` keyword ensures that only one thread can execute the `bookTicket` method at a time.
- Race Conditions: Without synchronization, multiple threads might book the same tickets simultaneously, leading to inconsistent state and incorrect ticket counts.
- Thread Safety: The program ensures thread safety by synchronizing the critical section of code that modifies the shared resource (`availableTickets`).

The following classes has to be define-

1. TicketBooking Class:
   - This class manages the available tickets and provides a method `bookTicket` to book tickets.
   - The `bookTicket` method is marked as `synchronized`, which means only one thread can execute this method at a time. This ensures that if one thread is booking a ticket, others have to wait until it's done.
2. TicketBookingThread Class:
   - This class extends `Thread` and represents a user trying to book tickets.
   - It takes the `TicketBooking` object, the passenger's name, and the number of tickets to book as parameters.
   - In the `run` method, it calls the `bookTicket` method on the shared `TicketBooking` object.
3. SynchronizedTicketBooking (Main Class):
   - This class creates a `TicketBooking` object with 5 available tickets.
   - It then creates three threads (`t1`, `t2`, `t3`) to simulate three users trying to book tickets concurrently.
   - The threads are started, and they attempt to book tickets. Due to the `synchronized` keyword, the booking process will be thread-safe.

```
class TicketBooking {
private int availableTickets;
```

```
public TicketBooking(int availableTickets) {
this.availableTickets = availableTickets;
}
// Synchronized method to ensure only one thread can book a ticket at a time
public synchronized void bookTicket(String passengerName, int numberOfTickets) {
if (numberOfTickets <= availableTickets) {
System.out.println(passengerName + " booked " + numberOfTickets + " ticket(s).");
availableTickets -= numberOfTickets;
System.out.println("Tickets left: " + availableTickets);
} else {
System.out.println("Sorry, " + passengerName + ". Not enough tickets available.");
}
}
}
class TicketBookingThread extends Thread
{ private TicketBooking ticketBooking;
private String passengerName;
private int numberOfTickets;
public TicketBookingThread(TicketBooking ticketBooking, String passengerName,
int numberOfTickets) {
this.ticketBooking = ticketBooking;
this.passengerName = passengerName;
this.numberOfTickets = numberOfTickets;
}
@Override
public void run() {
ticketBooking.bookTicket(passengerName, numberOfTickets);
}
}
public class SynchronizedTicketBooking {
public static void main(String[] args) {
// Assume there are 5 tickets available initially
TicketBooking ticketBooking = new TicketBooking(5);
// Creating multiple threads to book tickets
TicketBookingThread t1 = new TicketBookingThread(ticketBooking, "Alice", 2);
TicketBookingThread t2 = new TicketBookingThread(ticketBooking, "Bob", 2);
TicketBookingThread t3 = new TicketBookingThread(ticketBooking, "Charlie", 2);
// Start the threads
t1.start();
t2.start();
t3.start();
}
}
```

Output:

```
Alice booked 2 ticket(s).
Tickets left: 3
Bob booked 2 ticket(s).
```

Tickets left: 1
Sorry, Charlie. Not enough tickets available.

VII. Write a program to perform CRUD operations on the student table in a database using JDBC.

To perform CRUD (Create, Read, Update, Delete) operations on a `student` table in a database using JDBC (Java Database Connectivity), you can follow the steps below. This example assumes you have a database set up with a `student` table.

Prerequisites:

1. JDBC Driver: Ensure you have the JDBC driver for your database (e.g., MySQL, PostgreSQL).
2. Database Setup:
   - Create a database (e.g., `school`).
   - Create a `student` table with columns like `id`, `name`, `age`, and `grade`.

Below is an example SQL script to create the **`student`** table:

```
CREATE DATABASE school;
USE school;
CREATE TABLE student (
id INT PRIMARY KEY AUTO_INCREMENT,
name VARCHAR(50),
age INT,
grade VARCHAR(5)
);
```

Java Program to Perform CRUD Operations-

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
public class StudentCRUD {
// Database URL, username, and password
static final String DB_URL = "jdbc:mysql://localhost:3306/school";
static final String USER = "root";
static final String PASS = "password";
// JDBC objects
private Connection conn = null;
private Statement stmt = null;
private PreparedStatement pstmt = null;
public StudentCRUD() {
try {
// 1. Open a connection
conn = DriverManager.getConnection(DB_URL, USER, PASS);
} catch (SQLException e) {
e.printStackTrace();
}
}
// Create a new student record
public void createStudent(String name, int age, String grade) {
String sql = "INSERT INTO student (name, age, grade) VALUES (?, ?, ?)";
```

```
try {
pstmt = conn.prepareStatement(sql);
pstmt.setString(1, name);
pstmt.setInt(2, age);
pstmt.setString(3, grade);
pstmt.executeUpdate();
System.out.println("Student created successfully!");
} catch (SQLException e) {
e.printStackTrace();
} finally {
closePreparedStatement();
}
}
// Read and display student records
public void readStudents() {
String sql = "SELECT * FROM student";
try {
stmt = conn.createStatement();
ResultSet rs = stmt.executeQuery(sql);
while (rs.next()) {
int id = rs.getInt("id");
String name = rs.getString("name");
int age = rs.getInt("age");
String grade = rs.getString("grade");
System.out.println("ID: " + id + ", Name: " + name + ", Age: " + age + ", Grade: " + grade);
}
} catch (SQLException e) {
e.printStackTrace();
} finally {
closeStatement();
}
}
// Update a student record
public void updateStudent(int id, String name, int age, String grade) {
String sql = "UPDATE student SET name = ?, age = ?, grade = ? WHERE id = ?";
try {
pstmt = conn.prepareStatement(sql);
pstmt.setString(1, name);
pstmt.setInt(2, age);
pstmt.setString(3, grade);
pstmt.setInt(4, id);
int rowsUpdated = pstmt.executeUpdate();
if (rowsUpdated > 0) {
System.out.println("Student updated successfully!");
} else {
System.out.println("Student with ID " + id + " not found.");
}
} catch (SQLException e) {
```

```
e.printStackTrace();
} finally {
closePreparedStatement();
}
}
// Delete a student record
public void deleteStudent(int id) {
String sql = "DELETE FROM student WHERE id = ?";
try {
pstmt = conn.prepareStatement(sql);
pstmt.setInt(1, id);
int rowsDeleted = pstmt.executeUpdate();
if (rowsDeleted > 0) {
System.out.println("Student deleted successfully!");
} else {
System.out.println("Student with ID " + id + " not found.");
}
} catch (SQLException e) {
e.printStackTrace();
} finally {
closePreparedStatement();
}
}
// Close PreparedStatement
private void closePreparedStatement() {
try {
if (pstmt != null) pstmt.close();
} catch (SQLException e) {
e.printStackTrace();
}
}
// Close Statement
private void closeStatement() {
try {
if (stmt != null) stmt.close();
} catch (SQLException e) {
e.printStackTrace();
}
}
// Close Connection
public void closeConnection() {
try {
if (conn != null) conn.close();
} catch (SQLException e) {
e.printStackTrace();
}
}
public static void main(String[] args) {
```

```
StudentCRUD studentCRUD = new StudentCRUD();
// Create student
studentCRUD.createStudent("RAMA", 24, "A");
studentCRUD.createStudent("LAKHAN", 22, "B");
// Read students
System.out.println("Reading
students...");
studentCRUD.readStudents();
// Update student
System.out.println("Updating student...");
studentCRUD.updateStudent(1, "RAMA", 22, "A+");
// Delete student
System.out.println("Deleting student...");
studentCRUD.deleteStudent(2);
// Read students again
System.out.println("Reading students after update and delete...");
studentCRUD.readStudents();
// Close connection
studentCRUD.closeConnection();
}
}
```

Note: Ensure MySQL JDBC Driver: Make sure the MySQL JDBC driver (`mysql-connector-java` .JAR file) is in your classpath.

Output:

```
Student created successfully!
Student created successfully!
Reading students...
ID: 1, Name: RAMA, Age: 24, Grade: A
ID: 2, Name: LAKHAN, Age: 22, Grade: B
Updating student...
Student updated successfully!
Deleting student...
Student deleted successfully!
Reading students after update and delete...
ID: 1, Name: RAMA, Age: 22, Grade: A+
```

VIII. Develop an applet and swing in Java that displays a simple message

i) An applet is a small Java program that runs in a web browser or an applet viewer. Below example displays a simple message using applet.

```
import java.applet.Applet;
import java.awt.Graphics;
public class SimpleApplet extends Applet {
@Override
public void paint(Graphics g) {
g.drawString("Hello World!", 50, 75);
```

}

}
After compile the above code, write the below code in text editor like notepad and save it as `SimpleApplet.html`

```
<html>
<body>
   <applet code="SimpleApplet.class" width="300" height="150">
   </applet>
</body>
</html>
```

- `paint` Method: The `paint` method is overridden to display a message. The `Graphics` object `g` is used to draw the string on the applet's window.
- HTML Embed: The applet is embedded in an HTML file using the `<applet>` tag (as shown in the comment). However, modern browsers no longer support Java applets, so you would typically run this using an applet viewer.

Running the Applet:

1. Compile the Java file:
   javac SimpleApplet.java
2. Run the applet using an applet viewer:
   appletviewer SimpleApplet.html

Output:

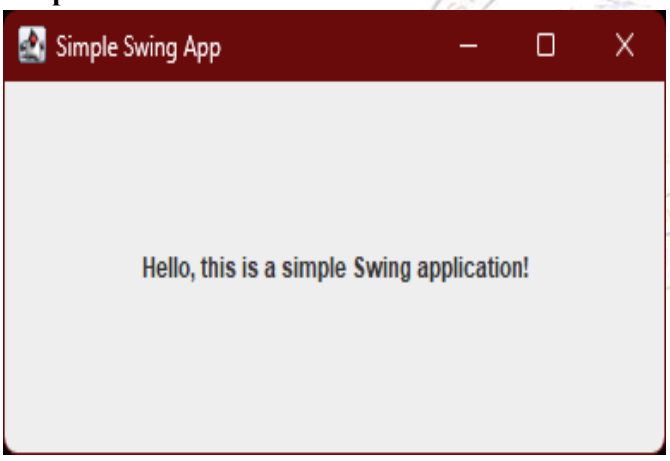
ii) Swing is a GUI toolkit in Java for building standalone applications. Swing is the preferred method for creating Java GUIs today since applets are largely obsolete due to modern web security concerns and lack of support in browsers. Below example displays a simple message using Swing.

```
import javax.swing.JFrame;
import javax.swing.JLabel;
import
javax.swing.SwingUtilities; public
class SimpleSwingApp {
public static void createAndShowGUI() {
// Create the frame
JFrame frame = new JFrame("Simple Swing App");
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
// Add a label with a message
JLabel label = new JLabel("Hello, this is a simple Swing application!", JLabel.CENTER);
frame.add(label);
// Set the frame size and make it visible
frame.setSize(400, 200);
frame.setVisible(true);
}
public static void main(String[] args) {
// Schedule a job for the event-dispatching thread:
// creating and showing this application's GUI.
SwingUtilities.invokeLater(() -> createAndShowGUI());
}
}
```

- `JFrame`: The main window where the components are placed.
- `JLabel`: A label that displays a simple message.
- `SwingUtilities.invokeLater`: Ensures that the GUI creation and updates happen on the Event Dispatch Thread (EDT), which is the standard practice for thread safety in Swing applications.
- Compile and Run the swing application as like a java program.

Output:



IX. Write a Java program that works as a simple calculator. Use a grid layout to arrange buttons for the digits and for the +, -,*, % operations. Add a text field to display the result. Handle any possible exceptions like divided by zero.

1. `JTextField display`:
   - The text field displays the current number, result, or error messages.
   - It is non-editable, meaning users can only interact with the calculator via buttons.
2. Buttons and `GridLayout`:
   - The buttons for digits (`0-9`) and operations (`+`, `-`, `*`, `/`, `C`, `=`) are arranged using a `GridLayout` with 4 rows and 4 columns.
   - The `C` button clears the display and resets the calculator.
3. Action Handling:
   - The program checks if the button pressed is a digit or an operator.

- If a digit is pressed, it is appended to the display.
- If an operator is pressed, the current number is stored as the first operand, and the operator is saved.
- When `=` is pressed, the calculation is performed based on the saved operator.

4. Exception Handling:
- The program handles division by zero by catching an `ArithmeticException` and displaying an error message.
- A general exception is also caught to handle any other unexpected errors.

5. Main Method:
- The main method uses `SwingUtilities.invokeLater` to ensure that the GUI is created and updated on the Event Dispatch Thread (EDT).

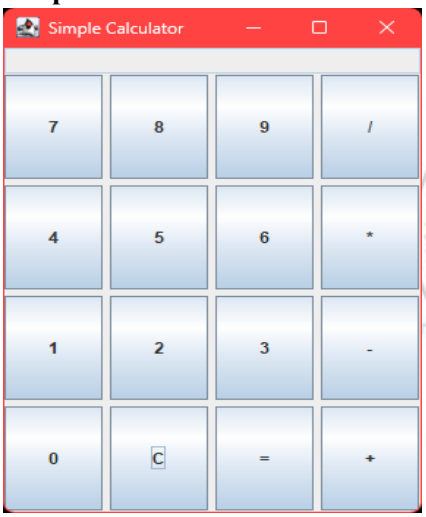
```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
public class SimpleCalculator extends JFrame implements ActionListener {
private JTextField display;
private String currentOperator; private double result, operand; public SimpleCalculator() {
// Create the display field
display = new JTextField();
display.setEditable(false);
display.setHorizontalAlignment(JTextField.RIGHT);
// Create the panel to hold buttons
JPanel panel = new JPanel();
panel.setLayout(new GridLayout(4, 4, 5, 5));
// Add buttons to the panel
String[] buttons = {
"7", "8", "9", "/",
"4", "5", "6", "*",
"1", "2", "3", "-",
"0", "C", "=", "+"
};
for (String text : buttons) {
JButton button = new JButton(text);
button.addActionListener(this);
panel.add(button);
}
// Set the layout of the frame
setLayout(new BorderLayout());
add(display, BorderLayout.NORTH);
add(panel, BorderLayout.CENTER);
// Frame settings
setTitle("Simple Calculator");
setSize(300, 400);
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

```
setLocationRelativeTo(null);
```

```
setVisible(true);
// Initialize variables
currentOperator = "";
result = 0;
operand = 0;
}
@Override
public void actionPerformed(ActionEvent e) {
String command = e.getActionCommand();
try {
if (command.charAt(0) >= '0' && command.charAt(0) <= '9') {
display.setText(display.getText() + command);
} else if (command.equals("C")) {
display.setText("");
result = 0;
operand = 0;
currentOperator = "";
} else if (command.equals("=")) {
calculate(Double.parseDouble(display.getText()));
display.setText(String.valueOf(result));
currentOperator = "";
} else {
if (!currentOperator.isEmpty()) {
calculate(Double.parseDouble(display.getText()));
display.setText(String.valueOf(result));
} else {
result = Double.parseDouble(display.getText());
}
currentOperator = command;
display.setText("");
}
} catch (ArithmeticException ex) {
display.setText("Error: Division by zero");
} catch (Exception ex) {
display.setText("Error");
}
}
private void calculate(double input) {
switch (currentOperator) {
case "+":
result += input;
break;
case "-":
result -= input;
break;
case "*":
result *= input;
break;
```

```
case "/":
if (input == 0) {
throw new ArithmeticException("Cannot divide by zero");
}
result /= input;
break;
}
}
public static void main(String[] args) {
SwingUtilities.invokeLater(SimpleCalculator::new);
}
}
```

Output:



X. Write a Java program that handles Keyboard and Mouse events and shows the event name at the center of the window when an event is fired. [Use Adapter classes]

1. `JFrame` and `JLabel`:
   - The program uses a `JFrame` to create the window and a `JLabel` to display the event name at the center of the window.
2. Adapter Classes:
   - To handle keyboard and mouse events in Java, you can use the `KeyAdapter` and `MouseAdapter` classes.
   - `KeyAdapter`: Handles keyboard events (`keyPressed`, `keyReleased`, `keyTyped`).
   - `MouseAdapter`: Handles basic mouse events (`mouseClicked`, `mousePressed`, `mouseReleased`, `mouseEntered`, `mouseExited`).
   - `MouseMotionAdapter`: Handles mouse motion events (`mouseMoved`, `mouseDragged`).
3. Event Handling:
   - When a key is pressed, released, or typed, the corresponding event name and key information are displayed in the label.
   - When the mouse is clicked, pressed, released, or moved, the event name is updated in the label.

4. setFocusable(true):
   - This ensures that the frame can receive keyboard events.
5. Main Method:
   - The main method uses `SwingUtilities.invokeLater` to ensure the GUI is created on the Event Dispatch Thread (EDT).

Java Program to Handling Keyboard and Mouse Events-

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class EventHandlingDemo extends JFrame {
private JLabel label;
public EventHandlingDemo() {
// Set up the frame
setTitle("Event Handling Demo");
setSize(400, 300);
setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
setLocationRelativeTo(null);
setLayout(new BorderLayout());
// Create a label to display the event name
label = new JLabel("", SwingConstants.CENTER);
label.setFont(new Font("Arial", Font.BOLD, 24));
add(label, BorderLayout.CENTER);
// Add key and mouse listeners using adapter classes
addKeyListener(new KeyAdapter() {
@Override
public void keyPressed(KeyEvent e) {
label.setText("Key Pressed: " + KeyEvent.getKeyText(e.getKeyCode()));
}
@Override
public void keyReleased(KeyEvent e) {
label.setText("Key Released: " + KeyEvent.getKeyText(e.getKeyCode()));
}
@Override
public void keyTyped(KeyEvent e) {
label.setText("Key Typed: " + e.getKeyChar());
}
});
addMouseListener(new MouseAdapter() {
@Override
public void mouseClicked(MouseEvent e) {
label.setText("Mouse Clicked");
}
@Override
public void mousePressed(MouseEvent e) {
label.setText("Mouse Pressed");
```

```
}
@Override
public void mouseReleased(MouseEvent e) {
label.setText("Mouse Released");
}
@Override
public void mouseEntered(MouseEvent e) {
label.setText("Mouse Entered");
}
@Override
public void mouseExited(MouseEvent e) {
label.setText("Mouse Exited");
}
});
addMouseMotionListener(new MouseMotionAdapter() {
@Override
public void mouseMoved(MouseEvent e) {
label.setText("Mouse Moved");
}
@Override
public void mouseDragged(MouseEvent e) {
label.setText("Mouse Dragged");
}
});
setFocusable(true); // To ensure the frame can receive key events
}
public static void main(String[] args) {
SwingUtilities.invokeLater(() -> {
EventHandlingDemo frame = new EventHandlingDemo();
frame.setVisible(true);
});
}
}
```

Output: