Set A

1. Define Swing and discuss about its advantage over AWT

Java Swing is a powerful and popular framework for creating graphical user interfaces (GUIs) in Java. It provides a comprehensive set of components and tools that allow developers to build interactive and visually appealing desktop applications. Swing is part of the Java Foundation Classes (JFC) and has been a fundamental part of Java's GUI programming for many years.

Swing offers a wide range of components, including buttons, labels, text fields, checkboxes, radio buttons, menus, scroll panes, and more. These components can be arranged and customized to create complex GUI layouts to suit various application requirements.

Here's a simple example that demonstrates the basic structure of a Swing application:

```
import javax.swing.*;

public class MySwingApp {
   public static void main(String[] args) {
      JFrame frame = new JFrame("My Swing App");
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

      // Add components to the frame
      JLabel label = new JLabel("Hello, Swing!");
      frame.getContentPane().add(label);

      // Set the size of the frame and make it visible
      frame.setSize(300, 200);
      frame.setVisible(true);
   }
}
```

Advantage of SWING over AWT

1. Swing components are light weight
2. It is based in MVC (Model View Contorller)
3. Swing has JTable and JTabbed pane.
4. Swing can have different feel and look.
5. Swing occupies less memory space.
6. Platform independency
7. Rich Component set
8. Enhanced Event handling
9. Double buffering and rendring
10. Customization

2. Why do we need top level container like JFrame to write Java Programs GUI ? How can we display two dimension objects in java.

In Java GUI programming, top-level containers like JFrame are used as the main window or frame for building graphical user interfaces. Here are a few reasons why top-level containers, such as JFrame, are essential in Java GUI programs:

1. Window Management: Top-level containers, like JFrame, provide a dedicated window that encapsulates the entire GUI application. They manage the application's appearance, size, title, and decorations, such as minimize, maximize, and close buttons. The top-level container acts as the main entry point for the GUI application and provides a frame for other GUI components to be added.
2. Layout Management: Top-level containers typically come with a layout manager that helps arrange and position the child components within the container. Layout managers handle tasks like automatic resizing, alignment, and responsiveness when the window is resized. They simplify the process of designing and organizing the GUI components in a structured and consistent manner.
3. Event Handling: Top-level containers can handle various events generated by user interactions or system events, such as button clicks, key presses, or window closing. They provide event listeners and callback methods to respond to these events effectively. Event handling is crucial for making the GUI interactive and responsive to user actions
4. Component Hierarchy: Top-level containers serve as the root or parent for other GUI components. They provide a container hierarchy where child components can be added, such as buttons, labels, text fields, and more. The hierarchical structure helps organize and manage the components within the GUI application, allowing for nesting and arranging components in a logical manner.
5. Accessibility: Top-level containers support accessibility features that aid users with disabilities in accessing and using the GUI application. They provide mechanisms for screen readers, keyboard navigation, and assistive technologies to interact with the application's user interface. Accessibility is an important aspect of GUI development to ensure inclusivity and compliance with accessibility standards.
6. Window Management Operations: Top-level containers offer methods to control and manipulate the window, such as minimizing, maximizing, or closing the window programmatically. These operations allow developers to control the behavior of the window and handle specific actions based on user interactions or application logic.

Overall, top-level containers like JFrame provide the fundamental structure and functionality required for building GUI applications in Java. They encapsulate the application's window, manage layout and event handling, and serve as the root for the component hierarchy. Top-level containers simplify the GUI development process and provide a consistent and organized framework for creating interactive and user-friendly Java applications.

```java
package kishan;

import java.awt.*;
import javax.swing.*;

public class Hello{

    // defined constructor
    public Hello() {

        // Creating a Frame
        JFrame frame = new JFrame();
        frame.setTitle("Mero Title");
        frame.setDefaultCloseOperation(frame.EXIT_ON_CLOSE);
        frame.setSize(400,500);
        frame.setLocationRelativeTo(null);// to center the frame to the screen

        //JPanel -  a GUI component that functions as a container to hold other components

        JPanel panel = new JPanel() {

            @Override
            protected void paintComponent(Graphics g) {
                super.paintComponent(g);

                g.setColor(Color.RED);
                g.fillRect(50,50,100,80);
            }

        };

        frame.setContentPane(panel);
        frame.setVisible(true);

    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(Hello::new);
    }
}
```

3. WAP to display images

```java
// WAP to display image
package kishan;

import javax.swing.*;

public class Hello{

    public Hello() {
        JFrame frame = new JFrame();
        frame.setTitle("Image Add");
        frame.setDefaultCloseOperation(frame.EXIT_ON_CLOSE);
        frame.setSize(1000,800);

        try {
            ImageIcon image = new ImageIcon(getClass().getResource("alien.png"));
            JLabel displayField = new JLabel(image);
            frame.add(displayField);

        }catch(Exception e) {
            System.out.println("image cannot be found");
        }
        frame.setVisible(true);
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(Hello::new);
        // Hello hello = new Hello();
    }

}
```

4. WAP for a given SQL Escape sequence.

5. Explain with code for executing a sql query "DELETE" using

7. Explain the significance of Java Bean Design

The Java Bean design pattern is a convention used in Java programming to create reusable and easily manageable components. Java Beans are Java classes that follow a specific set of conventions for properties, methods, and event handling. The key significance of Java Bean design is as follows:

Encapsulation: Java Beans promote encapsulation by providing private member variables with public getter and setter methods (properties). This allows for controlled access to the internal state of the object and ensures data integrity.

Reusability: Java Beans are designed to be reusable components. By following the Java Bean conventions, these components can be easily integrated into different applications or frameworks. They can be serialized, deserialized, and used in various contexts, facilitating code reuse.

Introspection: Java Beans support introspection, which allows tools and frameworks to examine and manipulate their properties and methods at runtime. This enables features like automatic GUI builders and IDE support, where components can be visually assembled and configured.

Customization and Extensibility: Java Beans can be customized and extended using properties, events, and methods. By providing well-defined interfaces and conventions, Java Beans can be easily integrated with other frameworks and extended to add new functionality.

Persistence: Java Beans can be made serializable, allowing their state to be persisted and transferred across network connections or stored in databases. This is especially useful in distributed systems and for saving application state.

Event Handling: Java Beans can support event handling through the use of event listeners and firing events. This allows components to generate events and be responsive to user actions or external stimuli.

```java
public class Hello implements java.io.Serializable{

    private String name;
    private int age;
    private String email;

    public Hello() {

    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name=name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age=age;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

}
```

LONG

WAP to find multiplication and divison using GUI Mouse EventhHandler

input 1

input 2

When the user presses the mouse perform division and when the user realseas the mouse perform multiplication

```
package kishan;

import java.awt.*;
import javax.swing.*;
import java.awt.event.*;


public class Hello implements MouseListener{

	JLabel label1;
	JTextField text1;
	JTextField text2;
	JButton button;
	JLabel resultLabel;
	Container frameC;

	public Hello() {

		JFrame frame = new JFrame();
		frame.getContentPane().setLayout(new FlowLayout());
		frame.setTitle("Evenet Listeners");
		frame.setDefaultCloseOperation(frame.EXIT_ON_CLOSE);
		frame.setSize(400,500);
		frame.setLocationRelativeTo(null);

		button = new JButton("click me");
		button.addActionListener(new ActionListener(){

			@Override
			public void actionPerformed(ActionEvent e) {
				// TODO Auto-generated method stub
		   JOptionPane.showMessageDialog(button, "kishan", "Button Clicked!",
0);
			}

		});

		label1 = new JLabel("Input 1:");
		text1 = new JTextField(10);
		text2 = new JTextField(10);
		resultLabel = new JLabel("");

		text1.addMouseListener(this);
		text2.addMouseListener(this);
```

```
        frameC = frame.getContentPane();
        frameC.add(label1);
        frame.getContentPane().add(text1);
        frame.getContentPane().add(new JLabel("Input 2"));
        frame.getContentPane().add(text2);

        frame.getContentPane().add(resultLabel);

        frame.getContentPane().add(button);
        frame.setVisible(true);

    }


    public static void main(String[] args) {
        Hello hello = new Hello();
    }


    @Override
    public void mouseClicked(MouseEvent e) {
        // TODO Auto-generated method stub

    }


    @Override
    public void mousePressed(MouseEvent e) {
        // TODO Auto-generated method stub
        performDivision();

    }


    @Override
    public void mouseReleased(MouseEvent e) {
        // TODO Auto-generated method stub
        performMultiplication();

    }


    @Override
    public void mouseEntered(MouseEvent e) {
        // TODO Auto-generated method stub

    }


    @Override
    public void mouseExited(MouseEvent e) {
        // TODO Auto-generated method stub

    }
```

```
	public void performDivision() {
		try {
			int num1 = Integer.parseInt(text1.getText());
			int num2 = Integer.parseInt(text2.getText());

			int ans  = num1 / num2;
			resultLabel.setText("Divison Result: " + ans);
		}catch(Exception e) {
			resultLabel.setText("Invalid Input");
		}
	}

	public void performMultiplication() {
		try {
			int num1 = Integer.parseInt(text1.getText());
			int num2 = Integer.parseInt(text2.getText());

			int ans = num1*num2;

			resultLabel.setText("Multiplication Result : "+ans);;

		}catch(Exception e) {
			resultLabel.setText("Invalid Input");
		}
	}


}
```

WAP using prepared statement to execute the insert query and also to retrive table and show it in the result set

1. Discuss about event Listener and adapter class. Give reason why adapter class is used

Event listeners and adapter classes are important concepts in Java for handling events in GUI applications. Let's discuss them and explore why adapter classes are used.

Event Listeners:

In Java, event listeners are interfaces that define callback methods to handle specific types of events. These interfaces provide a way to register and respond to events generated by GUI components, such as button clicks, mouse movements, or keyboard inputs.

Event listeners typically follow a naming convention, ending with the word "Listener" or "Adapter." Examples include ActionListener, MouseListener, KeyListener, etc. Each listener interface defines one or more callback methods that need to be implemented to handle the corresponding events.

To use an event listener, you need to create an instance of the listener interface and register it with the GUI component that generates the events. When an event occurs, the registered listener's callback method is invoked, allowing you to perform the desired actions in response to the event

Adapter Classes:

Adapter classes provide default implementations for all methods defined in an event listener interface. These adapter classes allow you to create listener objects by extending the adapter class and overriding only the methods you need, rather than implementing all methods of the listener interface.

The adapter class provides empty method implementations for all callback methods, allowing you to focus only on the events you are interested in handling. This simplifies the code and makes it more readable by removing the need to provide empty implementations for irrelevant event types.

For example, the MouseAdapter class is an adapter class that implements the MouseListener interface. It provides empty method implementations for all the methods of the MouseListener interface: mouseClicked, mousePressed, mouseReleased, mouseEntered, and mouseExited. By extending MouseAdapter, you can create a custom listener and override only the methods you need to handle the specific mouse events you are interested in.

Reasons for Using Adapter Classes:

Code Readability: Adapter classes improve code readability by allowing you to focus only on the events you need to handle. You can extend an adapter class and override specific methods instead of implementing all methods of an interface, reducing unnecessary code clutter.

Code Maintainability: Adapter classes make it easier to maintain and modify event handling code. If you need to add or change the event handling behavior in the future, you can simply override the relevant methods in your custom adapter class, without impacting other parts of the code.

Flexibility: Adapter classes provide flexibility in event handling. You can choose to handle only specific events and ignore others. This allows you to customize the behavior of your application based on the events that are relevant to your use case.

Backward Compatibility: Adapter classes also support backward compatibility. If new methods are added to an event listener interface in later versions of Java, the existing code that extends the adapter class will not break because the adapter class provides default empty implementations for all methods.

In summary, adapter classes are used in Java to simplify event handling code by providing default implementations for all methods in an event listener interface. They improve code readability, maintainability, flexibility, and support backward compatibility. Adapter classes allow you to focus on the events you need to handle and ignore the rest, resulting in cleaner and more concise code.

2. Explain JTextField and JtextArea components of Java Sing Libriary

JTextField:

The JTextField component is a text input field that allows the user to enter and edit a single line of text. It is typically used for accepting user input or displaying read-only text in a GUI application. Here are some key features and properties of JTextField:

Text Input: Users can type text directly into a JTextField component. The entered text can be accessed and manipulated programmatically.

Single Line: JTextField is designed for single-line input. It can accommodate a limited number of characters horizontally, and if the entered text exceeds the visible width, it provides scrolling functionality.

Editable: By default, JTextField is editable, allowing users to modify the text. However, you can set the editable property to false to make it read-only.

Event Handling: JTextField supports various events, such as action events triggered by pressing the Enter key or focus events when the field gains or loses focus. You can register event listeners to perform actions based on these events.

JTextArea:

The JTextArea component is a multi-line text area that allows the user to enter and edit multiple lines of text. It is suitable for handling larger amounts of text or displaying multi-line content. Here are some key features and properties of JTextArea:

Text Input: Users can enter and edit multiple lines of text in a JTextArea. It supports line breaks and allows the user to navigate through the text using the keyboard or mouse.

Multi-Line: JTextArea can display and handle multiple lines of text. It automatically wraps the text to the next line when it reaches the visible width. It also provides vertical scrolling if the text exceeds the visible height.

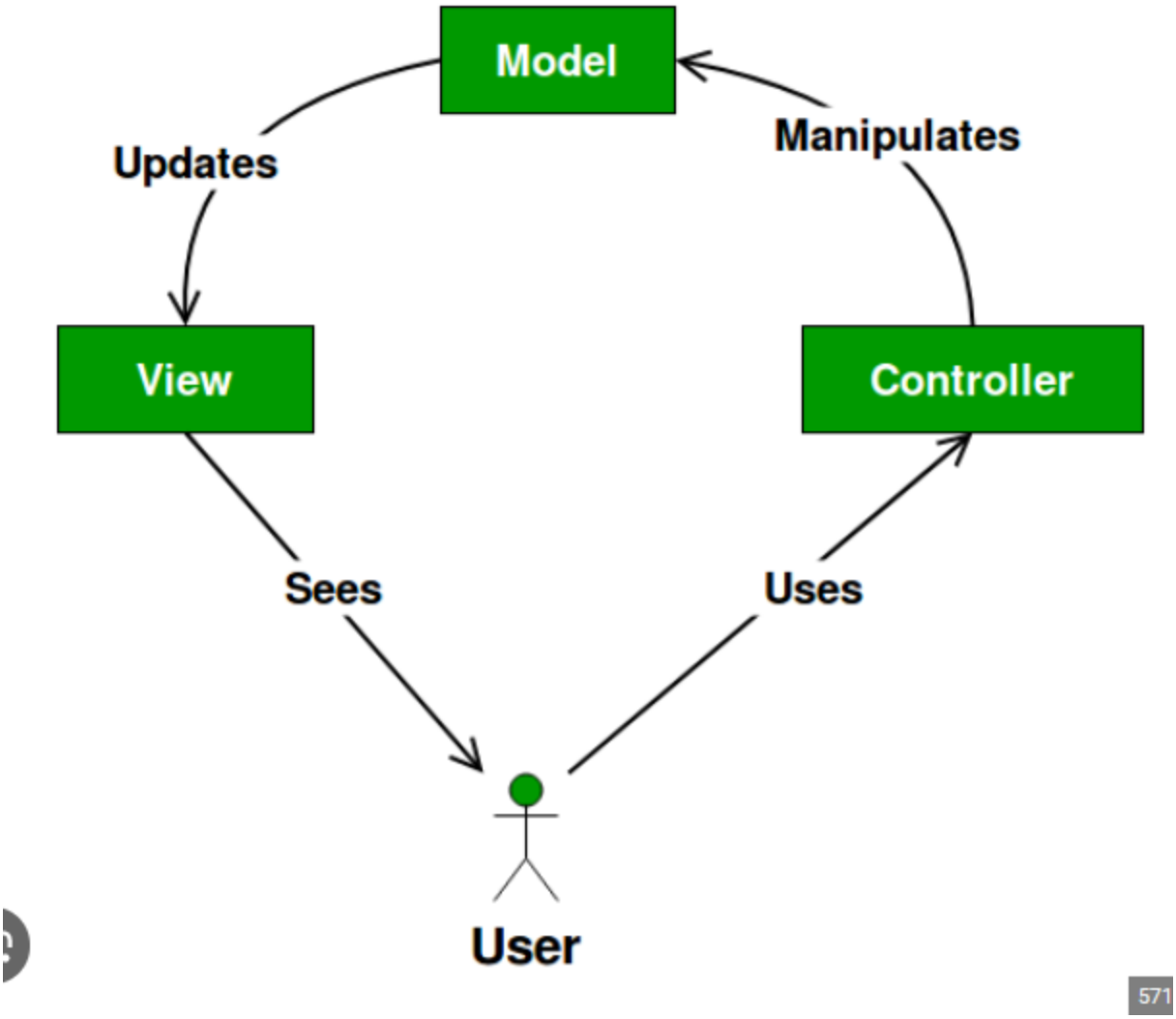
Editable: JTextArea is editable by default, enabling users to modify the text. You can set the editable property to false to make it read-only.

Formatting and Styling: JTextArea supports basic text formatting, such as setting fonts, colors, and alignments. You can also apply styles to portions of the text using HTML-like tags.

Event Handling: JTextArea supports events similar to JTextField, including action events and focus events. You can register event listeners to perform actions based on these events.

Both JTextField and JTextArea can be customized by setting properties such as font, foreground and background colors, borders, and more. They are part of the Swing library and provide powerful text input and display capabilities for creating interactive and user-friendly GUI applications in Java.

3. Write About MVC design pattern



4. Write a java program to display the following

JAVA

PY | C++ | JS

OOP

5. WAP to construct a rowjet operation for a given select query

```java
import javax.sql.rowset.JdbcRowSet;
import javax.sql.rowset.RowSetProvider;
import java.sql.SQLException;

public class RowSetOperationExample {
    public static void main(String[] args) {
        // Create a JdbcRowSet instance
        try (JdbcRowSet rowSet = RowSetProvider.newFactory().createJdbcRowSet
            // Set database connection details
            String url = "jdbc:mysql://localhost:3306/mydatabase";
            String username = "your_username";
            String password = "your_password";

            // Set query and execute it
            String query = "SELECT * FROM mytable";
            rowSet.setCommand(query);
            rowSet.setUrl(url);
            rowSet.setUsername(username);
            rowSet.setPassword(password);
            rowSet.execute();

            // Process the retrieved data
            while (rowSet.next()) {
                // Access columns using their names or indices
                int id = rowSet.getInt("id");
                String name = rowSet.getString("name");
                String email = rowSet.getString("email");

                // Perform desired operations with the retrieved data
                System.out.println("ID: " + id);
                System.out.println("Name: " + name);
                System.out.println("Email: " + email);
                System.out.println("-----------------");
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
```

Regenerate response

6. WAP to show indexed property and java beans

```
363 public class Hello {
364
365     private String[] names;
366
367     public Hello() {
368         names = new String[5];
369     }
370
371     public String getName(int index) {
372         return names[index];
373     }
374
375     public void setName(int index, String name) {
376         names[index] = name;
377     }
378
379     public static void main(String[] args) {
380         Hello bean = new Hello();
381
382         // Set values using indexed property
383         bean.setName(0, "John");
384         bean.setName(1, "Jane");
385         bean.setName(2, "Bob");
386         bean.setName(3, "Alice");
387         bean.setName(4, "Mike");
388
389         // Get values using indexed property
390         System.out.println("Names:");
391         for (int i = 0; i < 5; i++) {
392             String name = bean.getName(i);
393             System.out.println("Index " + i + ": " + name);
394         }
395     }
```

WAP to find factorial of a number using GUI for key event listener when the user enter any key perform

```java
package kishan;

import java.awt.*;
import javax.swing.*;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;

public class Hello extends JFrame implements KeyListener{

    private JLabel label;
    private JTextField textfield;

    public Hello() {

        setTitle("Hello");
        setSize(500,500);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLocationRelativeTo(null);
        setLayout(new FlowLayout());

        textfield = new JTextField(10);
        textfield.addKeyListener(this);
        add(textfield);

        label = new JLabel("result --");

        add(label);
        pack();
        setVisible(true);
    }

    public void calculateFactorial(int number) {

        int factorial = 1;

        for(int i = 2;i<=number; i++ ) {
            factorial *=i;
        }
        label.setText("factorila: "+ factorial);
    }
    @Override
    public void keyTyped(KeyEvent e) {
        // TODO Auto-generated method stub
    }
```

factorial

```java
    @Override
    public void keyPressed(KeyEvent e) {
        // TODO Auto-generated method stub
    }

    @Override
    public void keyReleased(KeyEvent e) {
        // TODO Auto-generated method stub
        if(e.getKeyCode() == KeyEvent.VK_ENTER) {
            String input = textfield.getText();
            try {
                int number = Integer.parseInt(input);
                calculateFactorial(number);
            }catch(Exception e1) {
                label.setText("invalid");
            }
        }
    }

    public static void main(String[] args) {
        Hello hello = new Hello();
    }
}
```

2 wap using scrollable and updatable result set and also update the row then reflect the change into the source database

```
import java.sql.*;

public class ScrollableResultSetExample {
    public static void main(String[] args) {
        try {
            // Set database connection details
            String url = "jdbc:mysql://localhost:3306/mydatabase";
            String username = "your_username";
            String password = "your_password";

            // Establish connection
            Connection connection = DriverManager.getConnection(url,
username, password);

            // Create scrollable and updatable ResultSet
            Statement statement =
connection.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE,
ResultSet.CONCUR_UPDATABLE);
            ResultSet resultSet = statement.executeQuery("SELECT *
FROM mytable");

            // Display initial data
            displayResultSet(resultSet);

            // Update a row
            resultSet.absolute(3);
            resultSet.updateString("name", "Updated Name");
            resultSet.updateRow();

            // Display updated data
            displayResultSet(resultSet);

            // Close resources
            resultSet.close();
            statement.close();
            connection.close();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
```

```
    private static void displayResultSet(ResultSet resultSet) throws
SQLException {
        System.out.println("Data:");
        while (resultSet.next()) {
            int id = resultSet.getInt("id");
            String name = resultSet.getString("name");
            String email = resultSet.getString("email");
            System.out.println("ID: " + id + ", Name: " + name + ",
Email: " + email);
        }
        System.out.println("------------------");
    }
}
```