1. Encapsulation and Modularity

- **Encapsulation**: The use of classes and access modifiers (`private`, `public`, `protected`) to control access to variables and methods. In the provided code, instance variables (`frame`, `panel`, `size`, `topping`) are encapsulated within the `PizzaOrderSystem` class.
- **Modularity**: Breaking down the code into smaller, manageable units (methods, classes) to promote reusability and maintainability. This is demonstrated in the division of functionality into methods like `createGUI()` and event listeners.

2. Event-Driven Programming

- **Event Listeners**: The use of interfaces like `ActionListener` to handle user actions (button clicks) asynchronously. This paradigm allows the program to respond dynamically to user input without blocking the main execution thread.

3. Layout Management

- **Layout Managers**: The use of `GridLayout` in `createGUI()` to arrange components (`JLabel`, `JButton`) in a grid. Understanding different layout managers (`FlowLayout`, `BorderLayout`, `GridBagLayout`, etc.) helps in designing responsive and visually appealing GUIs.

4. Error Handling and Exception Handling

- **Exception Handling**: Handling potential errors or exceptional situations that may arise during program execution (`NullPointerException`, `NumberFormatException`, etc.). Proper exception handling ensures robustness and reliability of the application.

5. Anonymous Classes and Lambdas

- **Anonymous Classes**: Used in event listeners (`new SizeButtonListener()`), they provide a way to implement interfaces or extend classes inline without explicitly creating a new class.
- **Lambdas**: In Java 8 and later, lambdas can be used to simplify event handling code, especially for single-method interfaces like `ActionListener`.

6. Polymorphism and Inheritance

- **Inheritance**: Extending classes (`JFrame`, `JPanel`) to customize their behavior or appearance. For example, creating a custom `JFrame` subclass to add specific features to the pizza ordering application.
- **Polymorphism**: Treating objects of different classes in a unified manner through inheritance and interfaces (`ActionListener`). This enhances flexibility and code reusability.

7. Resource Management

- **Resource Acquisition and Release**: Ensuring proper management of resources like `JFrame` (`frame`) and event listeners (`addActionListener`) to prevent memory leaks and ensure efficient use of system resources.

8. Concurrency

- **Thread Safety**: Ensuring that Swing components are accessed and modified only from the event dispatch thread (`SwingUtilities.invokeLater`) to avoid concurrency issues and ensure responsiveness.

Methods Used:

1. **Constructor `PizzaOrderSystem()`:**

```
java
Copy code
public PizzaOrderSystem() {
    createGUI();
}
```

   - Purpose: This is a constructor for the `PizzaOrderSystem` class. It initializes an instance of `PizzaOrderSystem` by calling the `createGUI()` method, which sets up the graphical user interface (GUI) components.

2. **Method `createGUI()`:**

```
java
Copy code
private void createGUI() {
    // JFrame setup
    frame = new JFrame("Pizza Order System");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setSize(300, 200);

    // JPanel setup
    panel = new JPanel();
    panel.setLayout(new GridLayout(5, 2));

    // Components setup: Labels and Buttons
    sizeLabel = new JLabel("Select Size:");
    panel.add(sizeLabel);

    smallButton = new JButton("Small");
    smallButton.addActionListener(new SizeButtonListener());
    panel.add(smallButton);

    mediumButton = new JButton("Medium");
    mediumButton.addActionListener(new SizeButtonListener());
    panel.add(mediumButton);

    largeButton = new JButton("Large");
    largeButton.addActionListener(new SizeButtonListener());
    panel.add(largeButton);

    toppingLabel = new JLabel("Select Topping:");
    panel.add(toppingLabel);
```

```
        veggieButton = new JButton("Veggie");
        veggieButton.addActionListener(new ToppingButtonListener());
        panel.add(veggieButton);

        meatButton = new JButton("Meat");
        meatButton.addActionListener(new ToppingButtonListener());
        panel.add(meatButton);

        orderButton = new JButton("Place Order");
        orderButton.addActionListener(new OrderButtonListener());
        panel.add(orderButton);

        orderLabel = new JLabel("");
        panel.add(orderLabel);

        // Add panel to frame and display everything
        frame.getContentPane().add(panel);
        frame.setVisible(true);
    }
```

- Purpose: This method sets up the graphical user interface (GUI) for the Pizza Order System.
- It initializes a `JFrame` (`frame`) and a `JPanel` (`panel`) with a `GridLayout`.
- Creates `JLabel` components (`sizeLabel`, `toppingLabel`, `orderLabel`) to display text.
- Creates `JButton` components (`smallButton`, `mediumButton`, `largeButton`, `veggieButton`, `meatButton`, `orderButton`) for user interaction.
- Adds `ActionListener` instances to buttons to handle user actions (e.g., selecting pizza size, topping, placing an order).

Loops:

There are no explicit loops (`for`, `while`, etc.) used in this code. The program flow is event-driven, responding to user interactions through event listeners attached to buttons (`ActionListener`).

Conditions:

1. **Inside `OrderButtonListener` (`OrderButtonListener` class):**

```
java
Copy code
private class OrderButtonListener implements ActionListener {
    @Override
    public void actionPerformed(ActionEvent e) {
        if (size != null && topping != null) {
            orderLabel.setText("You have ordered a " + size + " pizza
with " + topping + " topping.");
        } else {
            orderLabel.setText("Please select size and topping.");
        }
    }
}
```

- Purpose: This `ActionListener` is triggered when the "Place Order" button (`orderButton`) is clicked.
- It checks if both `size` and `topping` variables are not `null`.
- If both are selected (`size != null && topping != null`), it updates the `orderLabel` to display the selected pizza size and topping.
- If either or both are not selected, it updates the `orderLabel` to prompt the user to make selections (`"Please select size and topping."`).

Summary of Usage:

- **Methods**: Used for organizing and structuring code, defining behavior for GUI setup (`createGUI()`), and handling user actions (`ActionListener` implementations).
- **Loops**: Not used explicitly; the program relies on event-driven programming to respond to user actions.
- **Conditions**: Used to check whether the user has selected both a size and a topping before confirming the order.

This structure ensures that the program reacts dynamically to user input through GUI interactions, updating the display as users make selections and interact with the system.

Code:-

```
import javax.swing.*;

import java.awt.*;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;


public class PizzaOrderSystem {

   private JFrame frame;

   private JPanel panel;

   private JButton smallButton, mediumButton, largeButton, veggieButton, meatButton, orderButton;

   private JLabel sizeLabel, toppingLabel, orderLabel;

   private String size, topping;


   public PizzaOrderSystem() {

      createGUI();

   }


   private void createGUI() {
```

```
frame = new JFrame("Pizza Order System");
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setSize(300, 200);

panel = new JPanel();
panel.setLayout(new GridLayout(5, 2));

sizeLabel = new JLabel("Select Size:");
panel.add(sizeLabel);

smallButton = new JButton("Small");
smallButton.addActionListener(new SizeButtonListener());
panel.add(smallButton);

mediumButton = new JButton("Medium");
mediumButton.addActionListener(new SizeButtonListener());
panel.add(mediumButton);

largeButton = new JButton("Large");
largeButton.addActionListener(new SizeButtonListener());
panel.add(largeButton);

toppingLabel = new JLabel("Select Topping:");
panel.add(toppingLabel);

veggieButton = new JButton("Veggie");
veggieButton.addActionListener(new ToppingButtonListener());
panel.add(veggieButton);

meatButton = new JButton("Meat");
meatButton.addActionListener(new ToppingButtonListener());
```

```
    panel.add(meatButton);

    orderButton = new JButton("Place Order");
    orderButton.addActionListener(new OrderButtonListener());
    panel.add(orderButton);

    orderLabel = new JLabel("");
    panel.add(orderLabel);

    frame.getContentPane().add(panel);
    frame.setVisible(true);
}

private class SizeButtonListener implements ActionListener {
    @Override
    public void actionPerformed(ActionEvent e) {
        JButton button = (JButton) e.getSource();
        size = button.getText();
    }
}

private class ToppingButtonListener implements ActionListener {
    @Override
    public void actionPerformed(ActionEvent e) {
        JButton button = (JButton) e.getSource();
        topping = button.getText();
    }
}

private class OrderButtonListener implements ActionListener {
    @Override
```

```
        public void actionPerformed(ActionEvent e) {
            if (size != null && topping != null) {
                orderLabel.setText("You have ordered a " + size + " pizza with " + topping + " topping.");
            } else {
                orderLabel.setText("Please select size and topping.");
            }
        }
    }

    public static void main(String[] args) {
        new PizzaOrderSystem();
    }
}
```

Output:-