SHRI VILE PARLE KELAVANI MANDAL'S

SHRI BHAGUBHAI MAFATLAL POLYTECHNIC AND COLLEGE OF ENGINEERING

PT 2 – JAVA QUESTION BANK SOLUTION

CHAPTER NO:04
Exception Handling {10 MARKS}

Q.1 **Describe the working of the following with snippets (4m)**

a) try – Used to wrap code that may cause an exception.
b) throw – Used to explicitly throw an exception.
c) throws – Declares exceptions in method signature.
d) catch – Handles the exception.
e) finally – Always executes whether exception occurs or not.

Example (in Java):

```java
try {
    int result = 10 / 0;  // This causes ArithmeticException
} catch (ArithmeticException e) {
    System.out.println("Cannot divide by zero.");
} finally {
    System.out.println("Execution completed.");
}
```

Q.2 **Describe the class hierarchy of Exception. Also elaborate on caught and uncaught exceptions (6m)**

Exceptions are organized in a class hierarchy, typically something like this (using Java terminology).
This picture shows the Java Exception Hierarchy — how errors and exceptions are organized in Java.

```
java.lang.Object
    └─ java.lang.Throwable
        ├─ java.lang.Error                (Serious errors, JVM related)
        └─ java.lang.Exception
            ├─ java.lang.RuntimeException  (Unchecked exceptions)
            └─ Checked Exceptions          (Must be handled or declared)
```

- Throwable is the superclass for all errors and exceptions.
- Under Throwable, there are two branches: Error and Exception.
- Exception is further divided into Checked (compile-time) and Unchecked (runtime).

Caught Exception: Handled using try-catch.
Uncaught Exception: Not handled, program terminates.

Q.3 **Explain the process of creating a user-defined exception with an example program.(6m)**

To create a user-defined exception, extend the Exception class and use throw keyword.
Example:

```
class MyException extends Exception {
  MyException(String msg) { super(msg); }
}

public class Test {
  public static void main(String args[]) {
    try {
      throw new MyException("Custom Exception Occurred");
    } catch(MyException e) {
      System.out.println(e.getMessage());
    }
  }
}
```

Q.4

1. **Write a Java program to find out sqrt of a number, generate an exception for a negative number(4m)**

```
import java.util.*;
class SquareRootDemo {
  public static void main(String args[]) {
    Scanner sc = new Scanner(System.in);
    int num = sc.nextInt();
    try {
      if(num < 0) throw new Exception("Negative number not allowed");
      System.out.println("Square root: " + Math.sqrt(num));
    } catch(Exception e) {
      System.out.println(e.getMessage());
    }
  }
}
```

2. **WAP in Java to read Employee experience (years)(6m)**

   (lower limit = 0 upper limit = 40) and generate an exception if it is 41 or above

```
import java.util.*;
class EmployeeExp {
  public static void main(String args[]) {
    Scanner sc = new Scanner(System.in);
    int exp = sc.nextInt();
    try {
      if(exp > 40) throw new Exception("Invalid Experience");
      else System.out.println("Experience: " + exp);
```

```
    } catch(Exception e) {
      System.out.println(e.getMessage());
    }
  }
}
```

Q.5 **Explain checked and Unchecked exceptions in Java. (6m)**

- Exceptions that **must be either caught or declared** in the method using `throws`.
- Checked by the compiler during **compile time**.
- Usually represent **external problems** that your code can recover from.

Checked Exceptions: Occur at compile-time. Example: IOException, SQLException.

- Exceptions that are **not checked at compile time**.
- Usually caused by **programming mistakes** like logic errors or improper use of APIs.
- Subclasses of `RuntimeException`.

Unchecked Exceptions: Occur at runtime. Example: ArithmeticException, NullPointerException.

CHAPTER NO: 06
AWT and SWING Controls { 12 MARKS}

Q.6 **Draw the hierarchy of AWT classes. Describe any 2 in brief. (4m)**

```
java.lang.Object
   ↳ java.awt.Component
         ↳ java.awt.Container
               ↳ java.awt.Window
                     ↳ java.awt.Frame
                     ↳ java.awt.Dialog
         ↳ java.awt.Label
         ↳ java.awt.Button
         ↳ java.awt.TextComponent
               ↳ java.awt.TextField
               ↳ java.awt.TextArea
```

- **Frame**: A top-level window with a title bar.
- **Button**: A control that performs an action when clicked.

Class	Description
`Component`	Abstract superclass for all AWT GUI components.
`Container`	A component that can contain other components (e.g., panels, frames).
`Button`	Represents a push button.
`Label`	Displays a short string or text label.
`TextField`	Allows the user to enter a single line of text.
`TextArea`	Allows the user to enter multiple lines of text.
`Checkbox`	A checkbox that can be selected or deselected.
`Choice`	A drop-down list of choices.

Q.7 **Explain the event delegation model OR event handling mechanism with a diagram (4m)**

1. **Event Source** – The GUI component that generates an event (e.g., a Button).
2. **Event Object** – Contains information about the event (e.g., `ActionEvent`).
3. **Event Listener** – An interface that listens for events and defines how to handle them.

Delegation Event Model

Java uses the **Delegation Event Model** for event handling. In this model:

- A **source** (e.g., button) generates an **event** (e.g., button clicked).
- The event is sent (delegated) to an **event listener**.
- The listener **handles** the event using a method defined in the listener interface.

Think of it as:

Source → Event → Listener → Handler Method

Q.8 **Enlist all inbuilt packages (4m)**

- java.lang
- java.util
- java.io
- java.sql
- java.net
- java.awt
- javax.swing

Q.9 **Difference Between AWT and Swing. Give the different Swing controls. Also, write a simple program to demonstrate Swing controls. (6m)**

AWT	SWING
Heavyweight	Lightweight
limited controls	rich controls
OS dependent look	OS Independent look
Not as flexible as SWING	Flexible than AWT

AWT: Heavyweight, limited controls, OS dependent look.

Swing: Lightweight, rich controls, pluggable look and feel, flexible.

Common Swing controls: JButton, JLabel, JTextField, JCheckBox, JRadioButton.

Program:

```
import javax.swing.*;
public class SimpleSwing {
  public static void main(String args[]) {
    JFrame f = new JFrame("Demo");
    JButton b = new JButton("Click");
    b.setBounds(50,50,100,30);
    f.add(b);
    f.setSize(300,200);
    f.setLayout(null);
    f.setVisible(true);
  }
}
```

Q.10 **Describe the Adapter class in Java (4m)**

- Adapter class provides empty implementation of listener interfaces.
- We extend Adapter and override only required methods.
- Examples: MouseAdapter, KeyAdapter.

Q.11 **Explain the following Event listener with an example program (4m each)**

a)ActionListener b)KeyListener c)MouseListener

d)MouseMotionListener e)WindowListener

- **ActionListener**: Handles button clicks.
- **KeyListener**: Handles keyboard events.
- **MouseListener**: Handles mouse clicks.
- **MouseMotionListener**: Handles mouse movement.
- **WindowListener**: Handles window events.

Example (ActionListener):

```
import java.awt.*;
import java.awt.event.*;

class DemoAction implements ActionListener {
  public void actionPerformed(ActionEvent e){
    System.out.println("Button clicked");
  }
  public static void main(String args[]){
    Frame f = new Frame();
    Button b = new Button("Click");
    DemoAction d = new DemoAction();
    b.addActionListener(d);
    f.add(b);
    f.setSize(200,200);
    f.setVisible(true);
  }
```

	}
Q.12	**Enlist a different layout manager in AWT** **(6m)** • BorderLayout • FlowLayout • GridLayout • CardLayout • GridBagLayout • Null Layout

Q.13	**Develop a Java program where a user types text into a TextField and the program displays the last key pressed in a label.** **(4m)**
	import java.awt.*;
	import java.awt.event.*;
	
	class KeyDemo extends Frame implements KeyListener {
	Label l; TextField tf;
	KeyDemo(){
	l = new Label("Last Key:");
	tf = new TextField();
	tf.addKeyListener(this);
	add(tf,"North");
	add(l,"South");
	setSize(200,200);
	setVisible(true);
	}
	public void keyPressed(KeyEvent e){ l.setText("Last Key: "+e.getKeyChar()); }
	public void keyReleased(KeyEvent e){}
	public void keyTyped(KeyEvent e){}
	public static void main(String args[]){ new KeyDemo(); }
	}

Q.14	**Write a Java AWT application that responds to window minimize and restore actions by updating a status label.** **(4m)**
	import java.awt.*;
	import java.awt.event.*;
	
	class WindowDemo extends Frame implements WindowListener {
	Label l;
	WindowDemo(){
	l = new Label("Status");
	add(l);
	addWindowListener(this);
	setSize(200,200);
	setVisible(true);
	}
	public void windowIconified(WindowEvent e){ l.setText("Minimized"); }
	public void windowDeiconified(WindowEvent e){ l.setText("Restored"); }
	public void windowClosing(WindowEvent e){ dispose(); }

```
  public void windowOpened(WindowEvent e){} public void windowClosed(WindowEvent e){}
  public void windowActivated(WindowEvent e){} public void windowDeactivated(WindowEvent
e){}
  public static void main(String args[]){ new WindowDemo(); }
}
```

Q.15 **Create a program that asks for confirmation (Yes/No dialog) when the user tries to close the application window.(4m)**

```
import javax.swing.*;

class ConfirmClose {
  public static void main(String args[]){
    JFrame f = new JFrame("Confirm");
    f.setSize(200,200);
    f.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
    f.addWindowListener(new java.awt.event.WindowAdapter(){
      public void windowClosing(java.awt.event.WindowEvent e){
        int a = JOptionPane.showConfirmDialog(f,"Are you sure?");
        if(a==JOptionPane.YES_OPTION){ f.dispose(); }
      }
    });
    f.setVisible(true);
  }
}
```

Q.16 **Compare AWT and SWING min4 points(4m)**

AWT	SWING
Heavyweight	Lightweight
limited controls	rich controls
OS dependent look	OS Independent look
Not as flexible as SWING	Flexible than AWT

AWT: Heavyweight, limited controls, OS dependent look.
Swing: Lightweight, rich controls, pluggable look and feel, flexible.

Q.17 **WAP in Java to implement a simple arithmetic calculator (8m)**

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

class Calculator extends JFrame implements ActionListener {
  JTextField t1,t2,res; JButton add;
  Calculator(){
    t1 = new JTextField(); t2 = new JTextField(); res = new JTextField();
    add = new JButton("Add");
    t1.setBounds(50,50,100,30); t2.setBounds(50,100,100,30);
    add.setBounds(50,150,80,30); res.setBounds(50,200,100,30);
    add.addActionListener(this);
    add(t1); add(t2); add(add); add(res);
    setSize(300,300); setLayout(null); setVisible(true);
```

```
  }
  public void actionPerformed(ActionEvent e){
    int a = Integer.parseInt(t1.getText());
    int b = Integer.parseInt(t2.getText());
    res.setText(""+(a+b));
  }
  public static void main(String args[]){ new Calculator(); }
}
```

Q.18 **WAP to design a Login form in Java and display if the login is successful or not. (8m)**

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

class LoginForm extends JFrame implements ActionListener {
  JTextField user; JPasswordField pass; JButton login;
  LoginForm(){
    user = new JTextField(); pass = new JPasswordField(); login = new JButton("Login");
    user.setBounds(50,50,100,30); pass.setBounds(50,100,100,30);
    login.setBounds(50,150,80,30);
    add(user); add(pass); add(login);
    login.addActionListener(this);
    setSize(300,300); setLayout(null); setVisible(true);
  }
  public void actionPerformed(ActionEvent e){
    if(user.getText().equals("admin") && new String(pass.getPassword()).equals("123"))
      JOptionPane.showMessageDialog(this,"Login Successful");
    else JOptionPane.showMessageDialog(this,"Login Failed");
  }
  public static void main(String args[]){ new LoginForm(); }
}
```

Q.19 **Design a Java AWT application that detects mouse clicks and movements inside a frame and displays the coordinates on the screen. (6m)**

```
import java.awt.*;
import java.awt.event.*;

class MouseDemo extends Frame implements MouseListener, MouseMotionListener {
  Label l;
  MouseDemo(){
    l = new Label();
    add(l);
    addMouseListener(this);
    addMouseMotionListener(this);
    setSize(300,300); setVisible(true);
  }
  public void mouseClicked(MouseEvent e){ l.setText("Clicked: "+e.getX()+","+e.getY()); }
  public void mouseMoved(MouseEvent e){ l.setText("Moved: "+e.getX()+","+e.getY()); }
  public void mousePressed(MouseEvent e){} public void mouseReleased(MouseEvent e){}
  public void mouseEntered(MouseEvent e){} public void mouseExited(MouseEvent e){}
```

```
  public void mouseDragged(MouseEvent e){}
  public static void main(String args[]){ new MouseDemo(); }
}
```

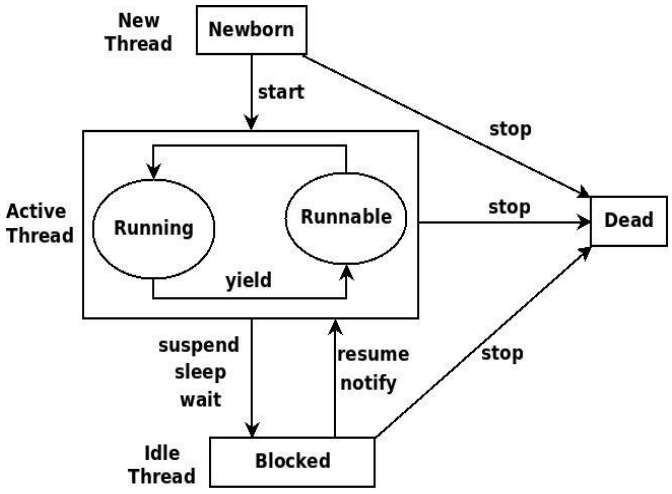
Q.20 **Create a Java program where the background color of the frame changes when the mouse enters and resets when the mouse exits the frame. (4m)**

```
import java.awt.*;
import java.awt.event.*;

class BgColorDemo extends Frame implements MouseListener {
  BgColorDemo(){
    addMouseListener(this);
    setSize(300,300); setVisible(true);
  }
  public void mouseEntered(MouseEvent e){ setBackground(Color.YELLOW); }
  public void mouseExited(MouseEvent e){ setBackground(Color.WHITE); }
  public void mouseClicked(MouseEvent e){} public void mousePressed(MouseEvent e){} public
void mouseReleased(MouseEvent e){}
  public static void main(String args[]){ new BgColorDemo(); }
}
```

CHAPTER NO:05
Multithreading {10 MARKS}

Q.21 **Describe the Thread Life cycle. (4m)**



- **This diagram shows the various states of a Java thread and how a thread transitions between these states during its life cycle.**

Thread States in Java

- **Newborn State** – When a thread object is created, it is in newborn state.
  - It is not yet running.

- We call `start()` to move it to Runnable.
- If we call `stop()`, it is killed before running.

- **Runnable State** – The thread is ready to run and waiting for CPU time.
  - All runnable threads are kept in a queue.
  - If multiple threads have same priority, they run in turns (Round-Robin).
  - We can use `yield()` to give chance to another thread of equal priority.
- **Running State** – The thread is actually executing its code.
  - Only one thread per CPU core can run at a time.
- **Blocked/Waiting State** – The thread is temporarily inactive.
  - Waiting for I/O, or waiting for another thread's signal.
  - Methods like `sleep()` and `wait()` put a thread in waiting.
- **Dead State** – After finishing its task, the thread enters dead state.
  - It cannot be restarted again.

Q.23 **Explain the life cycle of a thread in Java with a neat diagram. What are the different thread states? (6m)**

Refer above solution

Q.24 **Give the different ways to create a thread in Java? Explain with examples. (6m)**

Two main ways:

1. Extending the `Thread` Class

- Create a new class that extends `Thread`.
- Override the `run()` method with the code you want the thread to execute
- Create an instance of your class and call `start()` to begin the thread.

2. Implementing the `Runnable` Interface

- Create a class that implements `Runnable`.
- Implement the `run()` method.
- Create a `Thread` object by passing an instance of your class to the `Thread` constructor.
- Call `start()` on the `Thread` object.

Example (Extending Thread):

```
class MyThread extends Thread {
  public void run() {
    System.out.println("Thread is running");
  }
}
public class Demo {
  public static void main(String[] args) {
    MyThread t = new MyThread();
    t.start();
  }
}
```

Example (Runnable):

```
class MyRunnable implements Runnable {
  public void run() {
    System.out.println("Thread using Runnable");
```

	}
	}
	public class Demo2 {
	public static void main(String[] args) {
	Thread t = new Thread(new MyRunnable());
	t.start();
	}
	}

Q.25	**Explain implementing threads by extending the Thread class with an example.**
	class MyThread extends Thread {
	public void run() {
	for(int i=1;i<=5;i++) {
	System.out.println(i);
	}
	}
	}
	public class Test {
	public static void main(String args[]) {
	MyThread t = new MyThread();
	t.start();
	}
	}

Q.26	**Explain implementing threads by the Runnable interface with an example.**
	class MyRunnable implements Runnable {
	public void run() {
	for(int i=1;i<=5;i++) {
	System.out.println(i);
	}
	}
	}
	public class Test2 {
	public static void main(String args[]) {
	Thread t = new Thread(new MyRunnable());
	t.start();
	}
	}

Q.27	**How does thread priority affect thread execution? Does Java guarantee priority-based scheduling (6m)**

1. Thread Priorities

- Every thread in Java has a priority.
- Priorities range from `MIN_PRIORITY (1)` to `MAX_PRIORITY (10)`.
- Default priority is `NORM_PRIORITY (5)`.
- Higher priority threads get more CPU time (but this depends on the JVM and OS scheduler).
- Set priority using:

```
thread.setPriority(Thread.MAX_PRIORITY);
```

- Get priority using:

```
thread.getPriority();
```

• Thread priority is an integer value (1 to 10).
• Higher priority thread has more chance to run.
• But Java does NOT guarantee priority-based scheduling, it depends on JVM and OS

Q.28 **Explain the concept of synchronization in Java. How does the synchronized keyword work for methods and blocks?(6m)**

Synchronization ensures that only one thread accesses a shared resource at a time.
• synchronized method – Locks whole method.
• synchronized block – Locks only specific block of code.
This prevents race conditions.

2. Synchronization

- Synchronization controls access to shared resources (like variables or methods) to prevent **race conditions** and inconsistent data.
- When multiple threads access the same data, synchronization ensures only one thread accesses it at a time.

3. Using Synchronized Methods

- Mark a method with `synchronized` keyword to allow only one thread to execute it on the same object at a time.
- Example:

```
public synchronized void increment() {
    count++;
}
```

- This locks the object for that method until it finishes.

4. The synchronized Statement (Block)

- Instead of synchronizing the entire method, you can synchronize a block of code.
- Useful when only part of the method needs synchronization.
- Syntax:

```
public void increment() {
    synchronized(this) {
        count++;
    }
}
```

- `this` means the current object. You can also synchronize on other objects.

Q.29 **How do wait(), notify(), and notifyAll() work in Java? Explain with an example of the producer-consumer problem (4m)**

• wait() – Makes the thread wait until notified.
• notify() – Wakes up a single waiting thread.
• notifyAll() – Wakes up all waiting threads.
Example: Producer-consumer problem (refer Q.34)

Q.30 **Write a Java program where the main thread waits for two child threads to complete execution using the join() method. (6m)**

```
class MyThread extends Thread {
  public void run() {
    for(int i=1;i<=3;i++) {
      System.out.println(getName()+" - "+i);
    }
  }
}
public class TestJoin {
  public static void main(String args[]) throws Exception {
    MyThread t1 = new MyThread();
    MyThread t2 = new MyThread();
    t1.start(); t2.start();
    t1.join(); t2.join();
    System.out.println("Main thread ends");
  }
}
```

Q.31 **Explain the following methods of the thread class with an example:**
i)start() ii)run( ) iii)sleep() iv)yield() v)suspend() vi)resume() vii)join viii)isAlive()
i) start() – starts a new thread
ii) run() – defines the task of thread
iii) sleep() – makes thread pause for given ms

iv) yield() – gives chance to other threads
v) join() – waits for thread to finish
vi) isAlive() – checks if thread is alive
Methods like suspend() and resume() are deprecated.

Q.32 **Compare extending the thread and implementing Runnable, min 4 points (4m)**

Extending the thread	Implementing Runnable
Inherits Thread class	Implements Runnable interface
Cannot extend any other class	Can extend another class also
Simple for small tasks	Better for large applications
Less separation between task and thread	Clear separation between task and thread

Extending Thread:
- Inherits Thread class
- Cannot extend any other class
- Simple for small tasks

Implementing Runnable:
- Implements Runnable interface
- Can extend another class also
- Better for large applications

Q.33 **Explain inter-thread communication with an example.(4m)**

Inter-thread communication allows threads to talk using wait(), notify(), notifyAll().

In Java, **inter-thread communication** allows threads to communicate with each other. This is typically done using the `wait()`, `notify()`, and `notifyAll()` methods in the `Object` class.

- `wait()`: Causes the current thread to wait until another thread sends a signal (using `notify()` or `notifyAll()`).
- `notify()`: Wakes up one thread that is waiting on the object.
- `notifyAll()`: Wakes up all threads that are waiting on the object.

These methods can only be used inside a **synchronized block** or **synchronized method**, because they involve shared resources.

Example: Producer-consumer problem (refer Q.34)

Q.34 **Design a Java program using multithreading where one thread (Producer) generates a series of numbers and another thread (Consumer) consumes them. The producer should not produce if the buffer is full, and the consumer should not consume if the buffer is empty. Use wait() and notify() methods for inter-thread communication. (8m)**

```
class Buffer {
  int data; boolean available=false;
  synchronized void produce(int value) throws InterruptedException {
    while(available) wait();
    data = value; available=true;
    System.out.println("Produced: "+value);
    notify();
  }
```

```
  synchronized void consume() throws InterruptedException {
    while(!available) wait();
    System.out.println("Consumed: "+data);
    available=false; notify();
  }
}
class Producer extends Thread {
  Buffer b; Producer(Buffer b){this.b=b;}
  public void run(){ try{ for(int i=1;i<=5;i++) b.produce(i);}catch(Exception e){} }
}
class Consumer extends Thread {
  Buffer b; Consumer(Buffer b){this.b=b;}
  public void run(){ try{ for(int i=1;i<=5;i++) b.consume();}catch(Exception e){} }
}
public class TestPC {
  public static void main(String args[]){ Buffer b=new Buffer(); new Producer(b).start(); new
Consumer(b).start(); }
}
```

Q.35 (6m/8m)

Design and implement a Java application using **AWT (Abstract Window Toolkit)** that creates a **Student Registration Form** with the following components:

- **Components to include:**

1. **Labels** – For field names: Name, Gender, Hobbies, Country, Skills, Address
2. **TextFields** – For entering Name and Email
3. **Checkboxes** – For selecting Hobbies (Reading, Sports, Music)
4. **CheckboxGroup (Radio Buttons)** – For selecting Gender (Male, Female, Other)
5. **Choice (Dropdown)** – For selecting Country (e.g., India, USA, UK)
6. **List** – For selecting multiple Skills (e.g., Java, Python, C++)
7. **TextArea** – For entering full Address
8. **Buttons** – Submit and Reset

- **Functional Requirements:**

- On clicking the **Submit** button, display all entered/selected information using a **dialog box or console output**.
- On clicking **Reset**, clear all fields.
- Use proper **layout managers** (e.g., GridLayout, FlowLayout) for neat arrangement.

PROGRAM:

```
import java.awt.*;

import java.awt.event.*;
```

```
class StudentForm extends Frame implements ActionListener {
   TextField nameField, emailField;
   Checkbox cbReading, cbSports, cbMusic;
   CheckboxGroup genderGroup;
   Choice countryChoice;
   List skillList;
   TextArea addressArea;
   Button submitBtn, resetBtn;

   StudentForm() {
      setTitle("Student Registration Form");
      setSize(400, 500);
      setLayout(new GridLayout(10, 2, 5, 5));

      // Labels and TextFields
      add(new Label("Name:"));
      nameField = new TextField();
      add(nameField);

      add(new Label("Email:"));
      emailField = new TextField();
      add(emailField);

      // Gender (Radio Buttons)
      add(new Label("Gender:"));
      Panel genderPanel = new Panel(new FlowLayout());
      genderGroup = new CheckboxGroup();
      genderPanel.add(new Checkbox("Male", genderGroup, false));
      genderPanel.add(new Checkbox("Female", genderGroup, false));
      genderPanel.add(new Checkbox("Other", genderGroup, false));
      add(genderPanel);
```

```
// Hobbies (Checkboxes)
add(new Label("Hobbies:"));
Panel hobbiesPanel = new Panel(new FlowLayout());
cbReading = new Checkbox("Reading");
cbSports = new Checkbox("Sports");
cbMusic = new Checkbox("Music");
hobbiesPanel.add(cbReading);
hobbiesPanel.add(cbSports);
hobbiesPanel.add(cbMusic);
add(hobbiesPanel);

// Country (Choice Dropdown)
add(new Label("Country:"));
countryChoice = new Choice();
countryChoice.add("India");
countryChoice.add("USA");
countryChoice.add("UK");
add(countryChoice);

// Skills (List)
add(new Label("Skills:"));
skillList = new List(3, true);
skillList.add("Java");
skillList.add("Python");
skillList.add("C++");
add(skillList);

// Address (TextArea)
add(new Label("Address:"));
addressArea = new TextArea(3, 20);
```

```
    add(addressArea);

    // Buttons
    submitBtn = new Button("Submit");
    resetBtn = new Button("Reset");
    submitBtn.addActionListener(this);
    resetBtn.addActionListener(this);
    add(submitBtn);
    add(resetBtn);

    setVisible(true);
}

public void actionPerformed(ActionEvent e) {
    if (e.getSource() == submitBtn) {
        String name = nameField.getText();
        String email = emailField.getText();
        String gender = genderGroup.getSelectedCheckbox() != null ?
                genderGroup.getSelectedCheckbox().getLabel() : "Not Selected";
        String hobbies = "";
        if (cbReading.getState()) hobbies += "Reading ";
        if (cbSports.getState()) hobbies += "Sports ";
        if (cbMusic.getState()) hobbies += "Music ";

        String country = countryChoice.getSelectedItem();
        String skills = String.join(", ", skillList.getSelectedItems());
        String address = addressArea.getText();

        System.out.println("----- Student Details -----");
        System.out.println("Name: " + name);
        System.out.println("Email: " + email);
```

```
            System.out.println("Gender: " + gender);
            System.out.println("Hobbies: " + hobbies);
            System.out.println("Country: " + country);
            System.out.println("Skills: " + skills);
            System.out.println("Address: " + address);
        } else if (e.getSource() == resetBtn) {
            nameField.setText("");
            emailField.setText("");
            genderGroup.setSelectedCheckbox(null);
            cbReading.setState(false);
            cbSports.setState(false);
            cbMusic.setState(false);
            countryChoice.select(0);
            skillList.deselect(0); skillList.deselect(1); skillList.deselect(2);
            addressArea.setText("");
        }
    }

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