April - 2023

CS-19 : Programming With JAVA.

Prepared By : Lathiya Harshal

(A)

1. **JDK** – Java Development Kit
2. **JRE** – Java Runtime Environment
3. **OOP** – Object-Oriented Programming
4. **JVM** – Java Virtual Machine

(B)

(1) Class in Java:

A **class** in Java is a blueprint or template for creating objects. It defines properties (fields/variables) and behaviors (methods) that the created objects will have. For example, a `Car` class might have fields like `color` and `speed`, and methods like `drive()` and `brake()`.

(2) Operators in Java:

Operators in Java are special symbols used to perform operations on variables and values. Java provides various types of operators such as:

- **Arithmetic operators** (`+`, `-`, `*`, `/`)
- **Relational operators** (`==`, `!=`, `<`, `>`)
- **Logical operators** (`&&`, `||`, `!`)
- **Assignment operators** (`=`, `+=`, `-=`)
  These are used to manipulate data and control program logic.

(C)

(1) Explain Data Type in Java:

In Java, **data types** define the type of data that a variable can hold. Java is a strongly typed language, meaning every variable must be declared with a data type before use. Data types in Java are categorized into two main types:

A. Primitive Data Types:

These are the basic built-in data types:

- **byte** – 1 byte, stores whole numbers from -128 to 127.
- **short** – 2 bytes, range from -32,768 to 32,767.
- **int** – 4 bytes, commonly used for integers.
- **long** – 8 bytes, used for large integers.
- **float** – 4 bytes, used for decimal numbers.
- **double** – 8 bytes, used for precise decimal values.
- **char** – 2 bytes, stores a single character (Unicode).
- **boolean** – 1 bit, stores either `true` or `false`.

B. Non-Primitive (Reference) Data Types:

These refer to objects and include:

- **String**
- **Arrays**
- **Classes**
- **Interfaces**

Non-primitive types can be user-defined and are stored as references in memory

.

(2) Explain Features of Java:

Java is a powerful and versatile programming language with the following key features:

1. **Simple**:
   Java has a clean and easy-to-understand syntax. It removes complex features like pointers and operator overloading.

2. **Object-Oriented**:
   Java uses the object-oriented programming paradigm, which helps in organizing complex programs using objects and classes.

3. **Platform-Independent**:
   Java code is compiled into **bytecode** by the Java compiler. This bytecode runs on the **Java Virtual Machine (JVM)**, making Java programs run on any platform (Write Once, Run Anywhere).

4. **Secure**:
   Java has a strong security model, using the sandbox environment and bytecode verification to prevent malicious code execution.

5. **Robust**:
   Java emphasizes error handling (exception handling) and has a strong memory management system (Garbage Collection).

6. **Multithreaded**:
   Java supports multithreading, allowing multiple threads to run concurrently to improve performance.

7. **High Performance**:
   Although slower than native languages like C++, Java's performance is optimized with JIT (Just-In-Time) compiler.

8. **Distributed**:
   Java supports building distributed applications using networking capabilities via packages like `java.net`.

9. **Dynamic**:
   Java programs carry a large amount of run-time information that is used to resolve access to objects and classes dynamically.

(D)

(1) Explain OOP Concept in Java with Constructor Overloading

Java is an **Object-Oriented Programming (OOP)** language, which means it follows the principles of organizing software design around **objects** rather than functions and logic.

Main OOP Concepts in Java:

1. **Class** – A blueprint for objects.
2. **Object** – An instance of a class.
3. **Encapsulation** – Hiding internal details using private fields and providing public methods.
4. **Inheritance** – One class (child) can inherit properties and behaviors from another class (parent).
5. **Polymorphism** – The ability to take many forms (method overriding and overloading).

6. **Abstraction** – Hiding complex implementation and showing only the necessary details.

Constructor Overloading in Java:

Constructor Overloading is a form of **compile-time polymorphism** in Java. It allows a class to have more than one constructor with different parameter lists.

Example:

```
class Student {
    String name;
    int age;

    // Constructor 1
    Student() {
        name = "Unknown";
        age = 0;
    }

    // Constructor 2 (Overloaded)
    Student(String n, int a) {
        name = n;
        age = a;
    }

    void display() {
        System.out.println(name + " is " + age + " years old.");
    }
}

public class Main {
    public static void main(String[] args) {
        Student s1 = new Student();              // uses default constructor
        Student s2 = new Student("Mihir", 24);   // uses overloaded constructor

        s1.display();
        s2.display();
    }
}
```

(2) Explain in detail Java Token:

Java Tokens are the smallest building blocks of a Java program. When a program is compiled, it is broken down into these tokens. Java recognizes **five types of tokens**:

1. Keywords:

Reserved words in Java that have special meaning and cannot be used as identifiers.
Examples: `class`, `if`, `else`, `while`, `return`, `public`, `static`.

2. Identifiers:

Names used to identify classes, variables, methods, etc.
Rules:

- Must begin with a letter, `$`, or `_`.
- Cannot use keywords as identifiers.
- Cannot start with a digit.

3. Literals:

Constant values assigned to variables.
Types:

- Integer literals: `10`, `-50`
- Floating-point literals: `3.14`
- Character literals: `'A'`
- String literals: `"Hello"`
- Boolean literals: `true`, `false`

4. Operators:

Symbols used to perform operations.
Examples: `+`, `-`, `*`, `/`, `=`, `==`, `&&`, `||`

5. Separators (Punctuators):

Used to structure the program.
Examples:

- `;` (semicolon)
- `{}` (curly braces)
- `()` (parentheses)
- `[]` (square brackets)
- `,` (comma)

---

(2) (A)

(1) What is Public?
`public` is an **access modifier** in Java that allows a class, method, or variable to be **accessible from anywhere** in the program or even from other packages.

(2) What is Private?
`private` is an access modifier that restricts access to a class, method, or variable **only within the same class** where it is declared.

(3) What is Protected?
`protected` is an access modifier that allows access to the member **within the same package** and also to **subclasses in other packages**.

(4) What is Package?
A **package** in Java is a **namespace** that organizes related classes and interfaces. It helps in avoiding name conflicts and controls access with access modifiers. Example: `java.util` is a standard package.

(B)

(1) Explain Abstract Class:

An **abstract class** in Java is a class that **cannot be instantiated directly** and may contain **abstract methods** (methods without a body) as well as regular methods (with a body). It is used as a **base class** and meant to be **extended by subclasses** which provide implementations for the abstract methods.

Example:

```
abstract class Animal {

    abstract void sound(); // abstract method

    void sleep() {

        System.out.println("Sleeping...");

    }

}
```

---

(2) Explain Final Class:

A **final class** in Java is a class that **cannot be extended (inherited)** by any other class. This is useful when you want to **prevent modification** or **override behavior** of the class.

Example:

```
final class Vehicle {

    void display() {

        System.out.println("Vehicle class");

    }

}
```

Attempting to extend `Vehicle` will result in a compile-time error.

(c)

(1) Explain Hash Table with Example:

A **Hash Table** is a data structure that stores data in a **key-value** pair format. It uses a **hash function** to compute an index (also called a hash code) into an array of buckets or slots, from which the desired value can be found.

Key Features:

- Provides fast data access (`O(1)` average time complexity).
- Does **not allow duplicate keys**.
- Keys are hashed to determine where the value should be stored.

Java Implementation:

Java provides a class called `Hashtable` in the `java.util` package to implement a hash table.

Example:

```
import java.util.Hashtable;


public class HashTableExample {
    public static void main(String[] args) {
        Hashtable<Integer, String> ht = new Hashtable<>();


        ht.put(1, "Apple");
        ht.put(2, "Banana");
        ht.put(3, "Mango");


        System.out.println("Value at key 2: " + ht.get(2));
    }
}
```

Output:

Value at key 2: Banana

Note:

- `Hashtable` is **synchronized** (thread-safe).
- If the same key is used again, the old value is replaced.

---

(2) Explain Java API Package:

The **Java API (Application Programming Interface)** package is a **predefined library** in Java that provides **ready-made classes and interfaces** to simplify programming tasks.

Key Points:

- It includes packages like `java.lang`, `java.util`, `java.io`, etc.
- Java API allows developers to **reuse code** instead of writing everything from scratch.
- It covers functions for **data structures, file handling, networking, GUI, etc.**

Common Java API Packages:

Package	Description
`java.lang`	Contains fundamental classes (e.g., String, Math, Object)
`java.util`	Contains utility classes (e.g., ArrayList, HashMap, Date)

`java.io`	Classes for input/output operations
`java.net`	Classes for networking
`java.sql`	Classes for database access

Example:

```
import java.util.ArrayList;

public class JavaAPIExample {
    public static void main(String[] args) {
        ArrayList<String> list = new ArrayList<>();
        list.add("Java");
        list.add("Python");
        System.out.println(list);
    }
}
```

Output:

[Java, Python]

Conclusion:
Java API packages make development easier, faster, and more reliable by providing a wide range of built-in tools and functionalities.

(C)

(1) Explain Types of Inheritance in Java with Example:

Inheritance in Java is a mechanism where one class acquires properties and behaviors (fields and methods) of another class. It promotes code reusability and forms a relationship between classes.

Types of Inheritance in Java:

1. **Single Inheritance:**
    - **One subclass inherits from one superclass.**

```
class Animal
{
      void sound() { System.out.println("Animal makes sound");}
}
class Dog extends Animal {
    void bark() {
        System.out.println("Dog barks");
    }
```

```
}
```

2.
3. **Multilevel Inheritance:**
    - **A class is derived from a class which is also derived from another class.**

java
CopyEdit

```
class Animal {

    void eat() {

        System.out.println("Eating");

    }

}

class Dog extends Animal {

    void bark() {

        System.out.println("Barking");

    }

}

class Puppy extends Dog {

    void weep() {

        System.out.println("Weeping");

    }

}
```

4.
5. **Hierarchical Inheritance:**
    - **Multiple classes inherit from a single parent class.**

java
CopyEdit

```
class Animal {

    void sound() {

        System.out.println("Animal sound");

    }

}

class Dog extends Animal {

    void bark() {

        System.out.println("Dog barks");

    }

}

class Cat extends Animal {

    void meow() {

        System.out.println("Cat meows");
```

```
    }
}
```

6.

⚠️ **Note: Java does not support multiple inheritance using classes (to avoid ambiguity). However, it supports multiple inheritance through interfaces.**

---

(2) Explain Math Class with its Method:

The Math class in Java is part of the `java.lang` package and provides methods for performing mathematical operations like exponentiation, square root, trigonometry, rounding, etc.

Key Methods in Math Class:

Method	Description
`Math.abs(x)`	Returns absolute value
`Math.max(a, b)`	Returns maximum of two values
`Math.min(a, b)`	Returns minimum of two values
`Math.sqrt(x)`	Returns square root
`Math.pow(a, b)`	Returns a raised to the power b
`Math.round(x)`	Rounds a float/double to nearest int/long
`Math.random()`	Returns a random number between 0.0 and 1.0
`Math.sin(x)`	Returns sine of angle in radians
`Math.cos(x)`	Returns cosine of angle
`Math.floor(x)`	Rounds down to nearest integer
`Math.ceil(x)`	Rounds up to nearest integer

Example:

```
public class MathExample {
    public static void main(String[] args) {
        System.out.println("Square root of 16: " + Math.sqrt(16));
        System.out.println("Max of 10 and 20: " + Math.max(10, 20));
        System.out.println("Random number: " + Math.random());
    }
}
```

Output:

```
Square root of 16: 4.0
Max of 10 and 20: 20
Random number: 0.56789234 (varies each time)
```

Conclusion:
The `Math` class provides a wide range of static methods useful for mathematical calculations without creating objects.

---

(3) (A)

(1) Exceptions in Java arise in code sequence when an abnormal condition occurs during program execution, such as dividing by zero, accessing an invalid index, or opening a file that doesn't exist.

(2) `try` and `catch` keywords must be used to monitor for exceptions.

(3) `getPriority()` method of the `Thread` class is used to find out the priority given to a thread.

(4) `start()` method is used to start a thread.

(B)

(1) What is Daemon Thread:

A Daemon Thread in Java is a background service thread that runs continuously and performs tasks such as garbage collection or system monitoring. It does not prevent the JVM from exiting when all user threads finish.

- **It runs in the background and supports normal threads.**
- **Set using: `thread.setDaemon(true);` before calling `start()`.**
- **When all user (non-daemon) threads finish, daemon threads terminate automatically.**

(2) Explain Random Access File:

A Random Access File in Java allows both read and write operations at any position within the file. Unlike sequential file access, it allows moving the file pointer to a specific location to perform operations.

Java provides the `RandomAccessFile` class from the `java.io` package.

Key Methods:

- `seek(long pos)` **– Moves the file pointer.**
- `read()` **– Reads data.**
- `write()` **– Writes data.**

Example:

RandomAccessFile file = new RandomAccessFile("data.txt", "rw");

file.seek(10); // Move to 10th byte

file.writeBytes("Hello");

file.close();

It is useful for applications like databases or file systems where frequent updates at specific positions are needed.

(C)

(1) Explain BufferedReader and FileReader Class:

FileReader Class:

- `FileReader` **is a character-based stream class used to read data from files.**
- **It reads one character at a time and is suitable for reading text files.**
- **It is part of the** `java.io` **package.**

Example:

```
FileReader fr = new FileReader("example.txt");

int ch;

while ((ch = fr.read()) != -1) {

    System.out.print((char) ch);

}

fr.close();
```

BufferedReader Class:

- `BufferedReader` **is used to read text efficiently from character input streams.**
- **It wraps around** `FileReader` **and reads large chunks of data at once, making it faster.**
- **Also provides methods like** `readLine()` **to read a line of text at a time.**

Example:

```
FileReader fr = new FileReader("example.txt");

BufferedReader br = new BufferedReader(fr);

String line;

while ((line = br.readLine()) != null) {

    System.out.println(line);
```

```
}

br.close();

fr.close();
```

Difference:

- `FileReader` **reads character-by-character.**
- `BufferedReader` **reads line-by-line and is more efficient for large files.**

(2) Explain DataInputStream Class:

The `DataInputStream` **class in Java is used to read primitive Java data types (like int, float, double, etc.) from an input stream in a machine-independent way.**

It belongs to the `java.io` **package and is typically used when binary input is required.**

Constructor:

```
DataInputStream dis = new DataInputStream(new FileInputStream("data.bin"));
```

Common Methods:

Method	Description
`readInt()`	**Reads 4-byte integer**
`readDouble()`	**Reads 8-byte double**
`readBoolean()`	**Reads 1-byte boolean**
`readUTF()`	**Reads a string encoded in UTF-8**
`readChar()`	**Reads 2-byte character**

Example:

```
DataInputStream dis = new DataInputStream(new FileInputStream("data.bin"));

int number = dis.readInt();

double price = dis.readDouble();

String name = dis.readUTF();

dis.close();


System.out.println("Number: " + number);

System.out.println("Price: " + price);
```

```
System.out.println("Name: " + name);
```

Usage:

- **Used when reading data written using DataOutputStream.**
- **Ideal for reading binary files or structured data.**

Conclusion:

- **`BufferedReader` and `FileReader` are best for reading text files.**
- **`DataInputStream` is used for reading primitive data types in binary format.**

(D)

(1) Explain Thread Life Cycle with Example:

A Thread in Java goes through various stages in its life cycle. These stages are defined by the Thread Life Cycle, which represents the different states a thread can be in during its execution.

Thread Life Cycle States:

1.

New:
The thread is created using the `Thread` class but not yet started.

Thread t = new Thread();

2.

Runnable:
After calling `start()`, the thread is ready to run but waiting for CPU scheduling.

t.start();

3.

Running:
The thread is currently executing.

4.

Blocked/Waiting/Sleeping:
The thread is temporarily inactive and waiting for resources or time to pass (e.g., using `sleep()`, `wait()`).

5.

Terminated (Dead):
The thread has finished execution or was stopped.

Example:

```
class MyThread extends Thread {

    public void run() {

        System.out.println("Thread is running...");

    }


    public static void main(String[] args) {

        MyThread t1 = new MyThread();  // New
```

```
        t1.start();                 // Runnable → Running
    }
}
```

---

(2) Explain `try`, `catch`, `throw`, `throws`, `finally` with Example:

These are keywords used in Java for exception handling, allowing a program to manage runtime errors smoothly.

1. try block:

Contains code that might throw an exception.

```
try {
    int a = 5 / 0;
}
```

2. catch block:

Handles the exception thrown in the try block.

```
catch (ArithmeticException e) {
    System.out.println("Cannot divide by zero.");
}
```

3. throw keyword:

Used to manually throw an exception.

```
throw new ArithmeticException("Manual throw");
```

4. throws keyword:

Declares the exceptions that a method might throw.

```
void test() throws IOException {
    // some code
}
```

5. finally block:

Always executed whether an exception is thrown or not. Used for cleanup code.

```
finally {
    System.out.println("This block always executes.");
}
```

Full Example:

```
public class ExceptionExample {
    public static void main(String[] args) {
        try {
```

```
            int a = 10 / 0; // Exception
        } catch (ArithmeticException e) {
            System.out.println("Caught: " + e);
        } finally {
            System.out.println("Finally block executed.");
        }
    }
}
```

Output:

```
Caught: java.lang.ArithmeticException: / by zero
Finally block executed.
```

Conclusion:

- **The Thread Life Cycle defines the flow of execution of threads.**
- **Java exception handling with `try`, `catch`, `throw`, `throws`, and `finally` helps build robust applications by managing runtime errors gracefully.**

---

4 (a)

(1) What is Flow Layout?

FlowLayout arranges components left to right, and when space runs out, it moves to the next line. It is the default layout for Java applets and panels.

---

(2) What is No Layout?

No Layout (also called null layout) means there is no layout manager. You must manually set the position and size of each component using `setBounds()`.

---

(3) What is Card Layout?

CardLayout allows multiple components (like "cards") to share the same display space, with only one visible at a time. It's useful for implementing wizard-like interfaces or tab switching.

---

(4) What is Box Layout?

BoxLayout arranges components either vertically (Y_AXIS) or horizontally (X_AXIS) in a single line. It respects component size and alignment.

---

4 (b)

(1) Explain Border Layout:

BorderLayout divides the container into five regions:

- `NORTH`, `SOUTH`, `EAST`, `WEST`, and `CENTER`.

Each region can hold one component and gets resized accordingly.

Example:

```
setLayout(new BorderLayout());

add(new Button("North"), BorderLayout.NORTH);

add(new Button("Center"), BorderLayout.CENTER);
```

---

(2) Explain Applet Tag:

The `<applet>` tag is used in HTML to embed a Java applet in a webpage.

Syntax:

```
<applet code="MyApplet.class" width="300" height="200">

</applet>
```

- `code`: The name of the compiled applet class.
- `width` and `height`: Size of the applet display area.

⚠️ **Note: `<applet>` is now deprecated in modern browsers and replaced with Java Web Start.**

---

4 (c)

(1) Explain Graphics Class:

The Graphics class in Java is part of the `java.awt` package and is used to perform drawing operations (lines, shapes, text, etc.) on components.

Common Methods:

- `drawString(String str, int x, int y)`
- `drawLine(int x1, int y1, int x2, int y2)`
- `drawRect(int x, int y, int width, int height)`
- `setColor(Color c)`

Example:

```
public void paint(Graphics g) {
    g.drawString("Hello Graphics", 50, 50);
    g.drawLine(20, 20, 80, 80);
}
```

This code is typically written in an applet or component like `JPanel`.

---

(2) Explain SpringLayout with Example:

SpringLayout is a layout manager that allows you to position components relative to each other or to the container. It provides fine-grained control using springs (invisible, flexible constraints).

Example:

```
import javax.swing.*;

import java.awt.*;



public class SpringLayoutExample {

    public static void main(String[] args) {

        JFrame frame = new JFrame("Spring Layout");

        Container contentPane = frame.getContentPane();

        SpringLayout layout = new SpringLayout();

        contentPane.setLayout(layout);



        JButton b1 = new JButton("Button 1");

        contentPane.add(b1);



        layout.putConstraint(SpringLayout.WEST, b1, 50, SpringLayout.WEST, contentPane);

        layout.putConstraint(SpringLayout.NORTH, b1, 30, SpringLayout.NORTH, contentPane);



        frame.setSize(300, 200);

        frame.setVisible(true);

        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    }

}
```

---

4 (d)

(1) What is Applet? Explain Applet Life Cycle with Example:

An applet is a small Java program that runs inside a web browser or applet viewer. It is derived from `java.applet.Applet` or `javax.swing.JApplet`.

Applet Life Cycle Methods:

1. `init()` **– Called once when the applet is first loaded.**
2. `start()` **– Called each time the applet becomes active.**
3. `paint(Graphics g)` **– Used to draw content on the screen.**

4. `stop()` – Called when the applet is no longer active.
5. `destroy()` – Called when the applet is about to be removed.

Example:

```
import java.applet.*;
import java.awt.*;

/*
<applet code="MyApplet" width=300 height=200></applet>
*/
public class MyApplet extends Applet {
    public void init() {
        System.out.println("Applet initialized");
    }

    public void start() {
        System.out.println("Applet started");
    }

    public void paint(Graphics g) {
        g.drawString("Hello Applet", 50, 50);
    }

    public void stop() {
        System.out.println("Applet stopped");
    }

    public void destroy() {
        System.out.println("Applet destroyed");
    }
}
```

---

(2) Explain GridBagLayout with GridBagConstraints with Example:

GridBagLayout is a flexible layout manager that aligns components in a grid with varying cell sizes, controlled by GridBagConstraints.

Key Features:

- Each component can occupy multiple rows or columns.
- You can control alignment, padding, width, and height per component.

Example:

```
import java.awt.*;

import javax.swing.*;


public class GridBagExample {

    public static void main(String[] args) {

        JFrame frame = new JFrame("GridBagLayout Example");

        frame.setLayout(new GridBagLayout());

        GridBagConstraints gbc = new GridBagConstraints();


        JButton b1 = new JButton("Button 1");

        JButton b2 = new JButton("Button 2");


        gbc.gridx = 0;

        gbc.gridy = 0;

        frame.add(b1, gbc);  // Top-left


        gbc.gridx = 1;

        gbc.gridy = 0;

        frame.add(b2, gbc);  // Top-right


        frame.setSize(300, 200);

        frame.setVisible(true);

        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    }

}
```

Conclusion:

- **Applets are GUI programs with defined life cycles.**
- **Layout managers like GridBagLayout and SpringLayout offer advanced control over component positioning in Java GUIs.**

---

5 (a)

(1) What is JLabel?

`JLabel` is a Swing component used to display a short string or an image icon. It is a non-editable text element commonly used for labeling components like text fields.

---

(2) Full form of AWT is

Abstract Window Toolkit

---

(3) Full form of API is

Application Programming Interface

---

(4) _____ is a change in the state of an item?

Event is a change in the state of an item.

---

5 (b) Answer in brief (2 marks each):

(1) Explain JPasswordField:

`JPasswordField` **is a Swing component that allows users to enter a password securely. The text is not displayed directly, instead it shows masked characters (e.g., ●●●●).**

Example:

```
JPasswordField passField = new JPasswordField(20);
```

You can get the password as a character array:

```
char[] password = passField.getPassword();
```

---

(2) Explain JList:

`JList` **is a Swing component used to display a list of items to the user. Users can select one or multiple items depending on the mode set.**

Example:

```
String[] items = {"Apple", "Banana", "Mango"};

JList<String> list = new JList<>(items);
```

You can use `getSelectedValue()` to retrieve the selected item.

---

5 (c) Answer in detail (4 marks each):

(1) Difference between AWT and Swing:

Feature	AWT	Swing
GUI Type	Heavyweight (uses OS-native components)	Lightweight (pure Java components)
Look & Feel	OS dependent	Platform-independent & customizable

Components	**Fewer components**	**Rich set of advanced components**
Package	`java.awt`	`javax.swing`
Thread Safety	**Not entirely thread-safe**	**Mostly thread-safe**
Extensibility	**Limited customization**	**Highly customizable**

---

(2) Explain Menus:

Menus in Java (especially Swing) are used to create menu bars and options within a GUI application.

Main classes used:

- `JMenuBar`: **Holds menus.**
- `JMenu`: **Represents a drop-down menu.**
- `JMenuItem`: **Represents an item inside a menu.**

Example:

```
JMenuBar menuBar = new JMenuBar();

JMenu fileMenu = new JMenu("File");

JMenuItem openItem = new JMenuItem("Open");

fileMenu.add(openItem);

menuBar.add(fileMenu);

frame.setJMenuBar(menuBar);
```

You can also add action listeners to menu items to perform actions when clicked.

---

5 (d) Write a note on the following (5 marks each):

(1) Explain different types of Listener Interfaces with Example:

Listener interfaces are used in Java GUI to handle events like button clicks, mouse movement, etc.

Listener Interface	Purpose	Example Method
`ActionListener`	**Button click**	`actionPerformed()`
`ItemListener`	**Item selection (checkbox, list)**	`itemStateChanged()`

`MouseListener`	Mouse click/press	`mouseClicked()`
`KeyListener`	Keyboard key events	`keyPressed()`
`WindowListener`	Window events	`windowClosing()`

Example using ActionListener:

```
JButton btn = new JButton("Click");
btn.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        System.out.println("Button clicked!");
    }
});
```

---

(2) Explain different types of Adapter Classes with Example:

Adapter classes are abstract classes that provide empty implementations for listener interfaces with multiple methods. You can override only the method you need.

Adapter Class	Listener Interface
`MouseAdapter`	`MouseListener`
`KeyAdapter`	`KeyListener`
`WindowAdapter`	`WindowListener`

Example using MouseAdapter:

```
addMouseListener(new MouseAdapter() {
    public void mouseClicked(MouseEvent e) {
        System.out.println("Mouse clicked!");
    }
});
```

Benefits:

- **No need to implement all methods of the listener.**
- **Simplifies event handling code.**

Most Important Questions

What is Inheritance?

Explain Applet Life Cycle !

Explain Thread Life Cycle in detail / with Example.

Difference between AWT and Swing / AWT and Swing Components

Explain Event Delegation Model in detail