

Object Oriented Programming using Java

Notes

Unit 1: Introduction to Java and OOP

1. Java Programming Language

Detailed Definition: Java is a high-level, object-oriented programming language developed by Sun Microsystems (now owned by Oracle). It is designed to be platform-independent, meaning Java programs can run on any device that has a Java Virtual Machine (JVM) without needing recompilation.

Key Features:

- **Platform Independent:** "Write once, run anywhere" - works on Windows, Mac, Linux
- **Object-Oriented:** Everything in Java is an object
- **Simple:** Easy to learn with clean syntax
- **Secure:** Built-in security features prevent virus infection
- **Multithreaded:** Can perform multiple tasks simultaneously
- **Automatic Memory Management:** Garbage collector automatically frees unused memory

2. Object-Oriented Programming (OOP)

Detailed Definition: OOP is a programming paradigm that organizes software design around objects and data rather than functions and logic. Objects are instances of classes that contain both data (attributes) and methods (functions).

OOP vs Procedural Programming:

- **Procedural:** Focuses on procedures/functions that perform operations on data
- **Object-Oriented:** Focuses on objects that contain both data and methods

3. Java Basic Syntax

Data Types

Detailed Definition: Data types specify the different sizes and values that can be stored in variables.

Primitive Data Types:

- **byte:** 8-bit integer (-128 to 127)
- **short:** 16-bit integer (-32,768 to 32,767)
- **int:** 32-bit integer (-2^{31} to $2^{31}-1$)
- **long:** 64-bit integer
- **float:** 32-bit decimal numbers
- **double:** 64-bit decimal numbers
- **char:** 16-bit Unicode character
- **boolean:** true or false values

Variables

Detailed Definition: Variables are containers for storing data values in memory.

Types:

- **Local Variables:** Declared inside methods
- **Instance Variables:** Declared in class but outside methods
- **Static Variables:** Declared with static keyword

Operators

Detailed Definition: Symbols that perform operations on variables and values.

Types:

- **Arithmetic:** +, -, *, /, %
- **Assignment:** =, +=, -=, *=, /=
- **Comparison:** ==, !=, >, <, >=, <=
- **Logical:** &&, ||, !
- **Bitwise:** &, |, ^, ~

Control Flow

Detailed Definition: Statements that control the execution flow of a program.

Types:

- **Conditional:** if, else, switch
- **Looping:** for, while, do-while
- **Jump:** break, continue, return

Video Links:

- **English:** [Java Full Course for Beginners](#)
- **Hindi:** [Java Programming in Hindi](#)

Unit 2: Classes and Objects

1. Class

Detailed Definition: A class is a blueprint or template for creating objects. It defines the properties (attributes) and behaviors (methods) that objects of that class will have.

Example:

```

class Car {
    // Attributes (data)
    String color;
    String model;
    int speed;

    // Methods (behavior)
    void accelerate() {
        speed += 10;
    }

    void brake() {
        speed -= 10;
    }
}

```

2. Object

Detailed Definition: An object is an instance of a class. It is a real-world entity that has state (attributes) and behavior (methods).

Creating Objects:

```

Car myCar = new Car(); // myCar is an object of Car class
myCar.color = "Red";
myCar.model = "SUV";
myCar.accelerate();

```

3. Constructors

Detailed Definition: A special method that is automatically called when an object is created. It initializes the object's attributes.

Types:

- **Default Constructor:** No parameters
- **Parameterized Constructor:** Takes parameters
- **Copy Constructor:** Copies another object

Example:

```

class Car {
    String color;
    String model;

    // Default constructor
    Car() {
        color = "White";
        model = "Sedan";
    }

    // Parameterized constructor
    Car(String c, String m) {
        color = c;
        model = m;
    }
}

```

4. Methods

Detailed Definition: Methods are functions defined inside a class that describe the behaviors of objects.

Method Components:

- **Access Modifier:** public, private, protected
- **Return Type:** Data type or void
- **Method Name:** Identifier
- **Parameters:** Input values
- **Method Body:** Code to execute

5. Static Members

Detailed Definition: Static variables and methods belong to the class rather than to any specific object. They are shared by all objects of the class.

Example:

```
class Car {  
    static int carCount = 0; // Static variable  
  
    Car() {  
        carCount++; // Incremented for every car created  
    }  
  
    static void displayCount() { // Static method  
        System.out.println("Total cars: " + carCount);  
    }  
}
```

6. Access Modifiers

Detailed Definition: Keywords that set the accessibility (visibility) of classes, methods, and variables.

Types:

- **public:** Accessible from any other class
- **private:** Accessible only within its own class
- **protected:** Accessible within package and subclasses
- **default:** Accessible only within package

7. Encapsulation

Detailed Definition: The mechanism of wrapping data (variables) and code (methods) together as a single unit and restricting direct access to some of the object's components.

Implementation:

- Make variables private
- Provide public getter and setter methods

Example:

```
class BankAccount {  
    private double balance; // Private variable  
  
    // Public getter method  
    public double getBalance() {  
        return balance;  
    }  
  
    // Public setter method  
    public void deposit(double amount) {  
        if(amount > 0) {  
            balance += amount;  
        }  
    }  
}
```

8. UML Diagrams

Detailed Definition: Unified Modeling Language diagrams are standard visual representations of object-oriented systems.

Class Diagram Components:

- **Class Name:** Top compartment
- **Attributes:** Middle compartment
- **Methods:** Bottom compartment
- **Relationships:** Inheritance, association, dependency

Video Links:

- **English:** [Java Classes and Objects](#)
- **Hindi:** [Java OOP in Hindi](#)

Unit 3: Inheritance and Polymorphism

1. Inheritance

Detailed Definition: A mechanism where a new class (child/subclass) acquires the properties and behaviors of an existing class (parent/superclass).

Types of Inheritance:

- **Single Inheritance:** One class extends another class
- **Multiple Inheritance:** One class extends multiple classes (not directly supported in Java)
- **Multilevel Inheritance:** Chain of inheritance
- **Hierarchical Inheritance:** Multiple classes extend one class

Example:

```
// Parent class
class Vehicle {
    void start() {
        System.out.println("Vehicle started");
    }
}

// Child class
class Car extends Vehicle {
    void honk() {
        System.out.println("Car honking");
    }
}
```

2. super Keyword

Detailed Definition: A reference variable used to refer to the immediate parent class object.

Uses:

- Access parent class variables
- Call parent class methods
- Call parent class constructor

Example:

```
class Car extends Vehicle {
    String fuelType;

    Car(String fuel) {
        super(); // Call parent constructor
        this.fuelType = fuel;
    }
}
```

3. Method Overriding

Detailed Definition: When a subclass provides a specific implementation of a method that is already defined in its parent class.

Rules:

- Same method name and parameters
- Return type should be same or subtype
- Cannot override static, final, or private methods

Example:

```
class Vehicle {
    void start() {
        System.out.println("Vehicle starting");
    }
}

class Car extends Vehicle {
    @Override
    void start() {
        System.out.println("Car starting with key");
    }
}
```

4. Method Overloading

Detailed Definition: Having multiple methods with the same name but different parameters in the same class.

Rules:

- Same method name
- Different parameter lists
- Can have different return types

Example:

```
class Calculator {  
    int add(int a, int b) {  
        return a + b;  
    }  
  
    double add(double a, double b) {  
        return a + b;  
    }  
}
```

5. Abstract Classes

Detailed Definition: A class that cannot be instantiated and may contain abstract methods (without implementation) that must be implemented by subclasses.

Characteristics:

- Declared with abstract keyword
- Can have both abstract and concrete methods
- Cannot create objects
- Must be extended by subclasses

Example:

```
abstract class Animal {  
    abstract void sound(); // Abstract method  
  
    void sleep() {          // Concrete method  
        System.out.println("Sleeping");  
    }  
}  
  
class Dog extends Animal {  
    void sound() {  
        System.out.println("Bark");  
    }  
}
```

6. Interfaces

Detailed Definition: A completely abstract class that contains only abstract methods. It defines a contract that implementing classes must follow.

Characteristics:

- All methods are abstract (until Java 8)
- Can contain constants
- Supports multiple inheritance
- Implemented using implements keyword

Example:

```
interface Animal {
    void sound();
    void eat();
}

class Dog implements Animal {
    public void sound() {
        System.out.println("Bark");
    }

    public void eat() {
        System.out.println("Eating bones");
    }
}
```

Polymorphism

Detailed Definition: The ability of an object to take many forms. It allows performing a single action in different ways.

Types:

- **Compile-time Polymorphism:** Method overloading
- **Runtime Polymorphism:** Method overriding

8. Dynamic Method Dispatch

Detailed Definition: A mechanism where a call to an overridden method is resolved at runtime rather than compile time.

Example:

```
Animal myAnimal = new Dog(); // Parent reference, Child object
myAnimal.sound(); // Calls Dog's sound() method at runtime
```

Video Links:

- **English:** [Java Inheritance and Polymorphism](#)
- **Hindi:** [Java Inheritance in Hindi](#)

1. Packages

Detailed Definition: A namespace that organizes a set of related classes and interfaces. Physically, packages correspond to directories.

Benefits:

- Prevents naming conflicts
- Provides access protection
- Easier maintenance and organization

Types:

- **Built-in Packages:** java, javax, org
- **User-defined Packages:** Created by programmers

Example:

```
package com.mycompany.utils;

public class Calculator {
    public int add(int a, int b) {
        return a + b;
    }
}
```

2. Import Statements

Detailed Definition: Used to import classes and packages so they can be used without fully qualified names.

Types:

- **Specific Import:** import java.util.ArrayList;
- **Wildcard Import:** import java.util.*;

3. Exception Handling

Detailed Definition: A mechanism to handle runtime errors so that normal flow of the application can be maintained.

Exception Hierarchy:

- **Throwable:** Superclass of all errors and exceptions
- **Error:** Serious problems that applications should not try to catch
- **Exception:** Problems that can be handled

4. try-catch-finally Blocks

Detailed Definition: Keywords used to handle exceptions in Java.

try block: Contains code that might throw exception

catch block: Handles the exception

finally block: Always executes (for cleanup)

Example:

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

5. throw and throws Keywords

Detailed Definition:

- **throw:** Used to explicitly throw an exception
- **throws:** Used in method signature to declare exceptions

Example:

```
void checkAge(int age) throws InvalidAgeException {
    if(age < 18) {
        throw new InvalidAgeException("Age must be 18 or above");
    }
}
```

6. Checked vs Unchecked Exceptions

Detailed Definition:

- **Checked Exceptions:** Checked at compile-time (must be handled)
- **Unchecked Exceptions:** Checked at runtime (RuntimeException subclasses)

7. Custom Exceptions

Detailed Definition: User-defined exception classes created by extending Exception class.

Example:

```
class InvalidAgeException extends Exception {
    public InvalidAgeException(String message) {
        super(message);
    }
}
```

Video Links:

- **English:** [Java Exception Handling](#)
- **Hindi:** [Java Exception Handling in Hindi](#)

Unit 5: Arrays, Collections, and Generics

1. Arrays

Detailed Definition: A container object that holds a fixed number of values of a single type.

Types:

- **Single-dimensional:** One row of elements
- **Multi-dimensional:** Multiple rows and columns

Example:

```
// Single-dimensional array
int[] numbers = {1, 2, 3, 4, 5};

// Multi-dimensional array
int[][] matrix = {{1, 2, 3}, {4, 5, 6}};
```

2. Collections Framework

Detailed Definition: A unified architecture for representing and manipulating collections of objects.

Main Interfaces:

- **List:** Ordered collection, allows duplicates
- **Set:** Unordered collection, no duplicates
- **Map:** Key-value pairs, unique keys

3. ArrayList

Detailed Definition: A resizable array implementation of the List interface.

Characteristics:

- Dynamic size
- Allows duplicates
- Maintains insertion order
- Fast random access

Example:

```
ArrayList<String> list = new ArrayList<>();
list.add("Apple");
list.add("Banana");
list.add("Cherry");
```

4. LinkedList

Detailed Definition: A doubly-linked list implementation of the List and Deque interfaces.

Characteristics:

- Fast insertion and deletion
- Sequential access
- Implements both List and Queue

5. HashMap

Detailed Definition: A hash table based implementation of the Map interface.

Characteristics:

- Key-value pairs

- No duplicate keys
- No order guarantee
- Fast operations

Example:

```
HashMap<String, Integer> map = new HashMap<>();
map.put("Apple", 10);
map.put("Banana", 20);
map.get("Apple"); // Returns 10
```

6. Generics

Detailed Definition: A feature that allows type parameters when defining classes, interfaces, and methods.

Benefits:

- Type safety
- Eliminates type casting
- Enables generic algorithms

Example:

```
class Box<T> {
    private T content;

    public void setContent(T content) {
        this.content = content;
    }

    public T getContent() {
        return content;
    }
}
```

7. Iterators

Detailed Definition: An object that enables traversal through a collection and allows selective removal of elements.

Example:

```
ArrayList<String> list = new ArrayList<>();
// Add elements...

Iterator<String> iterator = list.iterator();
while(iterator.hasNext()) {
    String element = iterator.next();
    System.out.println(element);
}
```

Video Links:

- **English:** [Java Collections Framework](#)

- **Hindi:** [Java Collections in Hindi](#)
-

Unit 6: Multithreading and File I/O

1. Multithreading

Detailed Definition: The ability of a program to execute multiple threads concurrently, allowing parallel execution of tasks.

Benefits:

- Better CPU utilization
- Improved performance
- Responsive applications

2. Thread Lifecycle

Detailed Definition: The various states a thread can be in during its lifetime.

States:

- **New:** Created but not started
- **Runnable:** Ready to run or running
- **Blocked:** Waiting for monitor lock
- **Waiting:** Waiting indefinitely
- **Timed Waiting:** Waiting for specified time
- **Terminated:** Execution completed

3. Creating Threads

Two Ways:

1. **Extending Thread class**
2. **Implementing Runnable interface**

Example:

```
// Method 1: Extending Thread
class MyThread extends Thread {
    public void run() {
        System.out.println("Thread is running");
    }
}

// Method 2: Implementing Runnable
class MyRunnable implements Runnable {
    public void run() {
        System.out.println("Runnable is running");
    }
}
```

4. Synchronization

Detailed Definition: The capability to control the access of multiple threads to shared resources to prevent data inconsistency.

Methods:

- **Synchronized methods**
- **Synchronized blocks**

Example:

```
class Counter {  
    private int count = 0;  
  
    public synchronized void increment() {  
        count++;  
    }  
}
```

5. Thread Priority

Detailed Definition: An integer value (1-10) that indicates the relative priority of threads for scheduling.

Constants:

- MIN_PRIORITY: 1
- NORM_PRIORITY: 5
- MAX_PRIORITY: 10

6. Inter-thread Communication

Detailed Definition: A mechanism where threads can communicate with each other using wait(), notify(), and notifyAll() methods.

7. File I/O

Detailed Definition: Input and output operations performed on files using streams.

Stream Types:

- **Byte Streams:** For binary data (InputStream, OutputStream)
- **Character Streams:** For text data (Reader, Writer)

Example:

```
// Reading from file  
try (BufferedReader reader = new BufferedReader(new FileReader("file.txt"))) {  
    String line;  
    while ((line = reader.readLine()) != null) {  
        System.out.println(line);  
    }  
}
```

8. Serialization

Detailed Definition: The process of converting an object into a byte stream for storage or transmission.

Example:

```
class Student implements Serializable {  
    String name;  
    int age;  
}  
  
// Serialization  
ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream("student.ser"));  
out.writeObject(student);  
out.close();
```

Video Links:

- **English:** [Java Multithreading](#)
- **Hindi:** [Java File Handling in Hindi](#)

Unit 7: GUI Programming and Applications

1. GUI Programming

Detailed Definition: Creating Graphical User Interfaces that allow users to interact with programs through visual elements rather than text commands.

Java GUI Libraries:

- **AWT (Abstract Window Toolkit):** Original GUI toolkit
- **Swing:** More advanced, built on AWT

2. Swing Components

Detailed Definition: Lightweight, platform-independent components for building GUIs.

Common Components:

- **JFrame:** Main window
- **JButton:** Clickable button
- **JTextField:** Text input
- **JLabel:** Display text or image
- **JPanel:** Container for other components

Example:

```
import javax.swing.*;

public class SimpleGUI {
    public static void main(String[] args) {
        JFrame frame = new JFrame("My Application");
        JButton button = new JButton("Click Me");
        JLabel label = new JLabel("Hello World");

        frame.setLayout(new FlowLayout());
        frame.add(button);
        frame.add(label);
        frame.setSize(300, 200);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }
}
```

3. Event Handling

Detailed Definition: The mechanism that controls the program's response to user actions like clicking buttons, typing text, etc.

Event Handling Process:

1. User performs action
2. Event object is created
3. Event listener is notified
4. Listener processes the event

Example:

```
button.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        label.setText("Button Clicked!");
    }
});
```

4. Layout Managers

Detailed Definition: Objects that control the positioning and sizing of components in a container.

Types:

- **FlowLayout:** Components in left-to-right flow
- **BorderLayout:** North, South, East, West, Center
- **GridLayout:** Rectangular grid of equal-sized cells
- **GridBagLayout:** Flexible grid with constraints

5. Applets vs Applications

Detailed Definition:

- **Applications:** Standalone programs that run independently

- **Applets:** Small programs that run in web browsers (mostly deprecated)

6. Graphics and Multimedia

Detailed Definition: Drawing shapes, images, and working with audio/video in Java applications.

Graphics Methods:

- `drawLine()`, `drawRect()`, `drawOval()`
- `fillRect()`, `fillOval()`
- `drawImage()`, `drawString()`

Example:

```
public void paint(Graphics g) {  
    g.drawRect(50, 50, 100, 100);  
    g.fillOval(75, 75, 50, 50);  
    g.drawString("Hello Graphics", 60, 180);  
}
```

7. GUI Design Principles

Detailed Definition: Guidelines for creating effective and user-friendly interfaces.

Principles:

- **Consistency:** Similar operations work similarly
- **Feedback:** Users know what's happening
- **Simplicity:** Avoid unnecessary complexity
- **Forgiveness:** Allow undo operations
- **Accessibility:** Work for all users

Video Links:

- English: [Java GUI Tutorial](#)
- Hindi: [Java Swing in Hindi](#)

Important Java Concepts Summary

Key Features of Java:

1. Simple and Familiar
2. Object-Oriented
3. Platform Independent
4. Secure
5. Multithreaded
6. High Performance
7. Distributed

8. Dynamic

Memory Management in Java:

- **Stack Memory:** For method execution and local variables
- **Heap Memory:** For objects and instance variables
- **Garbage Collection:** Automatic memory management

Java Development Tools:

- **JDK (Java Development Kit):** For development
- **JRE (Java Runtime Environment):** For running applications
- **JVM (Java Virtual Machine):** Executes Java bytecode

Best Practices:

1. Follow naming conventions
2. Use proper access modifiers
3. Handle exceptions properly
4. Write reusable code
5. Document your code
6. Test thoroughly

Career Opportunities:

- Java Developer
- Web Application Developer
- Android Developer
- Enterprise Application Developer
- Software Engineer

Final English Video: [Java Complete Course](#)

Final Hindi Video: [Java Programming Complete](#)