CORE JAVA

Quick Learning Guide

Day 1: Java Basics (OOPs, Variables, Data Types, Access Modifiers)

1.1 Introduction to Java & Object-Oriented Programming (OOPs)

What is Java?

Java is a **high-level, object-oriented, platform-independent programming language**. It follows the **Write Once, Run Anywhere (WORA)** principle due to the **Java Virtual Machine (JVM)**.

Key Features of Java

✅ **Platform Independent** – Java code runs on any OS with a JVM.
✅ **Object-Oriented** – Everything in Java is treated as an object.
✅ **Robust & Secure** – Provides strong memory management and exception handling.

1.2 Object-Oriented Programming (OOPs) Principles

OOP Concept	Description	Example
Encapsulation	Wrapping data and methods together	private variables with getter/setter
Abstraction	Hiding implementation details	Abstract classes & interfaces
Inheritance	Reusing properties & methods of a class	extends & super keyword
Polymorphism	Ability to take many forms (method overloading/overriding)	same method name with different behavior

🔹 **Example: Encapsulation**

```
class Employee {
    private String name;

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

    public String getName() {
        return name;
    }
}
```

What can be asked in an evaluation or interview?

- ✅ Explain **Encapsulation with an example**.
- ✅ What is the **difference between Abstraction and Encapsulation**?
- ✅ What is **Method Overloading and Method Overriding**?

1.3 Variables, Data Types & Access Modifiers

Variables in Java

Variables are used to store data in memory.

Variable Type	Description	Example
Local	Declared inside a method/block	int num = 10;
Instance	Belongs to an object	private String name;
Static	Belongs to a class	static int count;

Data Types in Java

Type	Size	Example
byte	1 byte	byte a = 100;
int	4 bytes	int num = 5000;
double	8 bytes	double pi = 3.14;
boolean	1 bit	boolean isActive = true;

Access Modifiers in Java

Access Modifiers define **visibility** of variables and methods.

Modifier	Scope	Example
public	Accessible from everywhere	public int age;
private	Accessible within the same class	private String name;
protected	Accessible within package & subclasses	protected void show();
default	Accessible within the package	void display();

🔹 **Example: Access Modifiers**

```java
class Person {
    public String name = "John";  // Accessible everywhere
    private int age = 25;         // Accessible only within this class
    protected String city = "New York"; // Accessible within package & subclass
}
```

What can be asked in an evaluation or interview?

✅ Explain **different types of variables** in Java.
✅ What is the difference between **primitive and non-primitive data types**?
✅ Explain **Access Modifiers** with examples.

Day 2: Control Statements & String Manipulation

2.1 Conditional Statements (if-else, switch)

What are Conditional Statements?

Conditional statements allow the program to make decisions based on conditions.

- **Example: if-else Statement**

```
public class AgeCheck {
    public static void main(String[] args) {
        int age = 18;
        if (age >= 18) {
            System.out.println("You are eligible to vote.");
        } else {
            System.out.println("You are not eligible to vote.");
        }
    }
}
```

- **Example: switch Statement**

```
public class SwitchExample {
    public static void main(String[] args) {
        int day = 3;
        switch (day) {
            case 1 -> System.out.println("Monday");
            case 2 -> System.out.println("Tuesday");
            case 3 -> System.out.println("Wednesday");
            default -> System.out.println("Invalid day");
        }
    }
}
```

2.2 Loops (for, while, do-while)

Loops are used to execute a block of code multiple times.

Loops in Java

Loop	Description	Example
for loop	Repeats code **N times**	for(int i=0; i<5; i++)
while loop	Runs until a **condition is false**	while(condition)

Loop	Description	Example
do-while loop	Runs at least **once**	do { } while(condition);

- **Example: For Loop (Print Even Numbers)**

```
public class EvenNumbers {
    public static void main(String[] args) {
        for (int i = 2; i <= 10; i += 2) {
            System.out.print(i + " ");
        }
    }
}
```

2.3 String Manipulation

Method	Description	Example
charAt()	Returns character at a specific index	str.charAt(2)
substring()	Extracts part of a string	str.substring(1,4)
toUpperCase()	Converts string to uppercase	str.toUpperCase()
replace()	Replaces a character	str.replace('a','o')

- **Example: String Methods**

```
public class StringDemo {
    public static void main(String[] args) {
        String message = "Hello Java!";
        System.out.println("Uppercase: " + message.toUpperCase());
        System.out.println("Substring: " + message.substring(6));
        System.out.println("Replace: " + message.replace("Java", "World"));
    }
}
```

2.4 Arrays in Java

Why Use Arrays?

Arrays allow storing multiple values of **the same type** in **a single variable**.

```java
int[] numbers = {10, 20, 30, 40};
System.out.println(numbers[2]); // Output: 30
```

- **Multidimensional Array Example**

```java
int[][] matrix = {
    {1, 2, 3},
    {4, 5, 6}
};
System.out.println(matrix[1][2]); // Output: 6
```

What can be asked in an evaluation or interview?

- ✅ What is the difference between **while and do-while loops**?
- ✅ How does **switch-case** work internally?
- ✅ How do you declare and initialize an array?
- ✅ What is the difference between **Array and ArrayList**?
- ✅ Difference between **String, StringBuffer, and StringBuilder**?
- ✅ Why are **Strings immutable** in Java?

Day 3: Exception Handling & File Handling

3.1 Exception Handling in Java

What is an Exception?

An **exception** is an **unexpected event** that disrupts the normal flow of a program. It can occur due to **invalid input, network failures, file handling errors, division by zero**, etc.

Types of Exceptions in Java

Exception Type	Description	Example
Checked Exception	Checked at compile time (e.g., IOException, SQLException)	FileNotFoundException
Unchecked Exception	Occurs at runtime (e.g., NullPointerException, ArrayIndexOutOfBoundsException)	int arr[5] = arr[10];
Error	System-level issue (e.g., OutOfMemoryError, StackOverflowError)	Infinite recursion

---

Exception Handling using try-catch

- **Syntax of Exception Handling**

```
try {
   // Code that may throw an exception
} catch (ExceptionType e) {
   // Handling the exception
} finally {
   // Code that will execute no matter what
}
```

- **Example: Handling ArithmeticException**

```
public class ExceptionDemo {
    public static void main(String[] args) {
        try {
            int result = 10 / 0;  // This will throw an exception
        } catch (ArithmeticException e) {
            System.out.println("Error: Cannot divide by zero!");
        }
    }
}
```

Throw and Throws in Java

Keyword	Description	Example
throw	Used to **explicitly** throw an exception	throw new IllegalArgumentException("Invalid input!");
throws	Declares exceptions that may be thrown by a method	void myMethod() throws IOException {}

- **Example: Throwing a Custom Exception**

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

public class Test {
    public static void main(String[] args) {
        try {
            throw new CustomException("This is a custom exception");
        } catch (CustomException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

What can be asked in an evaluation or interview?

- ✅ What are **checked and unchecked exceptions**?
- ✅ Explain **try-catch-finally** with an example.
- ✅ What is the difference between **throw and throws**?
- ✅ Why should we use **custom exceptions**?

3.2 File Handling in Java

Why Do We Need File Handling?

In real-world applications, data is **read from and written to files** instead of working only with console input/output. Java provides the **java.io** and **java.nio** packages for file operations.

Reading a File using FileReader

```java
import java.io.FileReader;
import java.io.IOException;

public class ReadFile {
    public static void main(String[] args) {
        try (FileReader reader = new FileReader("test.txt")) {
            int ch;
            while ((ch = reader.read()) != -1) {
                System.out.print((char) ch);
            }
        } catch (IOException e) {
            System.out.println("File not found!");
        }
    }
}
```

Writing to a File using FileWriter

```java
import java.io.FileWriter;
import java.io.IOException;

public class WriteFile {
    public static void main(String[] args) {
        try (FileWriter writer = new FileWriter("output.txt")) {
            writer.write("Hello, this is a test file!");
            System.out.println("File written successfully!");
        } catch (IOException e) {
            System.out.println("Error writing file!");
        }
    }
}
```

What can be asked in an evaluation or interview?

- ✅ How do you read and write files in Java?
- ✅ What is the difference between **FileReader and BufferedReader**?
- ✅ What is **try-with-resources**, and how does it help in file handling?

Day 4: Collections Framework (List, Set, Map)

4.1 Introduction to Collections

Why Use Collections?

In Java, **arrays have fixed sizes** and do not provide built-in methods for data manipulation. The **Collections Framework** provides **dynamic, efficient data structures** like **ArrayList, HashSet, HashMap**, etc.

4.2 List (ArrayList, LinkedList)

Type	Features	Use Case
ArrayList	Fast random access, but slow insert/delete	Best for **search operations**
LinkedList	Fast insert/delete, but slow access	Best for **frequent additions/removals**

- **Example: ArrayList**

```
import java.util.ArrayList;

public class ListExample {
    public static void main(String[] args) {
        ArrayList<String> list = new ArrayList<>();
        list.add("Apple");
        list.add("Banana");
        list.add("Orange");

        for (String fruit : list) {
            System.out.println(fruit);
        }
    }
}
```

4.3 Set (HashSet, TreeSet)

Type	Features	Use Case
HashSet	Unordered, unique elements	Fast searches
TreeSet	Sorted, unique elements	Sorted data retrieval

- **Example: HashSet**

```java
import java.util.HashSet;

public class SetExample {
    public static void main(String[] args) {
        HashSet<Integer> numbers = new HashSet<>();
        numbers.add(10);
        numbers.add(20);
        numbers.add(10); // Duplicate ignored

        System.out.println(numbers);
    }
}
```

4.4 Map (HashMap, TreeMap)

Type	Features	Use Case
HashMap	Key-value pairs, fast lookups	Storing user sessions
TreeMap	Sorted key-value pairs	Sorted retrieval

- **Example: HashMap**

```java
import java.util.HashMap;

public class MapExample {
    public static void main(String[] args) {
        HashMap<Integer, String> map = new HashMap<>();
        map.put(1, "Alice");
        map.put(2, "Bob");
        map.put(3, "Charlie");

        System.out.println(map.get(2)); // Output: Bob
    }
}
```

What can be asked in an evaluation or interview?

- ✅ Difference between **ArrayList and LinkedList**
- ✅ Why are **HashSet elements unique**?
- ✅ What is the **time complexity of HashMap operations**?
- ✅ How does **TreeMap maintain order**?