Name= Ashwini Atole.

LMS UserName =AF04953400

Basics & OOPs Assignment Questions

# 1. Java Basics

**1. What is Java? Explain its features.**

**Java** is a **popular programming language** used to create software like apps, games, websites, and even smart devices.

Features:

* **Simple:**Java is easy to learn if you know basic programming. It has a clean and understandable syntax.
* **Object**-Oriented:Java treats everything as an object (like real-world items), making the code organized and reusable.
* **Secure**:Java has built-in features to protect data and avoid viruses or hacking.
* **Multithreaded:**Java can perform many tasks at the same time, like downloading while playing music
* **Dynamic:**Java handles errors well and avoids crashes by checking code during both compile-time and run-time.
* **Distributed:**Java supports networking easily – helpful in building apps that run on the internet or over networks.
* **High Performance:**Java is faster than many other languages because of its efficient code and Just-In-Time compiler.
* **Robust:**Java handles errors well and avoids crashes by checking code during both compile-time and run-time.

----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**2. Explain the Java program execution process.**

* Install JDK (Java Development Kit)
* Write the Java Program (.java file)
* Compile the Program using java
* Run the Program using java
* View the Output on the screen

---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

1. **Write a simple Java program to display 'Hello World'.**

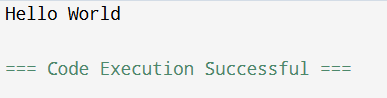
public class HelloWorld {

public static void main(String[] args) {

System.out.println("Hello World");

}

}



---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

1. **What are data types in Java? List and explain them.**

* Primitive Data Types

1. **Byte**

The byte data type is an 8-bit signed integer. It is used when you want to save memory in large arrays, especially in place of integers. It can store values from -128 to 127. It is mostly used in situations where memory savings are critical.

1. **short**  
   The short data type is a 16-bit signed integer. It can hold values from -32,768 to 32,767. It is larger than a byte but smaller than an int, and is used to save memory in applications where an int is not needed.
2. **int**  
   The int data type is a 32-bit signed integer and is the most commonly used data type for numeric values. It can store a wide range of values from -2,147,483,648 to 2,147,483,647. It is the default data type for integer values unless memory optimization is needed.
3. **long**  
   The long data type is a 64-bit signed integer. It is used when a larger range of values than int is required. It is ideal for big numbers like population counts, distance measurements, etc. When assigning a long value, an "L" is usually added to the end of the number.

**5. float**  
The float data type is a 32-bit floating-point number used to store decimal values. It is less precise than a double but consumes less memory. When using float, you should add an "f" or "F" at the end of the value.

**6. double**  
The double data type is a 64-bit floating-point number. It is used when high precision is needed for decimal values, like in scientific calculations. It is the default data type for decimal numbers in Java.

**7. char**  
The char data type is a 16-bit Unicode character. It is used to store a single character and must be enclosed in single quotes, like 'A' or '9'. It can also store symbols and letters from different languages.

**8. boolean**  
The boolean data type represents one bit of information and can only hold two values: true or false. It is commonly used for decision-making or conditional logic (e.g., if a condition is true or not).

* Non-Primitive (Reference) Data Types

1. **String**  
   A String is not a primitive type, but it is widely used to store sequences of characters or text. Strings in Java are objects and come with many built-in methods to manipulate text.

**2. Array**  
An Array is used to store multiple values of the same type in a single variable. For example, you can store a list of numbers or names. Arrays have a fixed size once they are created.

**3. Class**  
A Class is a user-defined data type in Java. It is a blueprint for creating objects and can contain fields (variables) and methods (functions). Classes are a core part of Java's object-oriented programming.

**4. Interface**  
An Interface in Java is a reference type, similar to a class, that contains only abstract methods (method declarations without body). It is used to achieve abstraction and multiple inheritance in Java.

**----------------------------------------------------------------------------------------------------------------------------------------------------------------**

**5. What is the difference between JDK, JRE, and JVM?**

| **Component** | **Stands for** | **What It Does** | **Includes** |
| --- | --- | --- | --- |
| **JVM** | Java Virtual Machine | Runs Java bytecode | Part of JRE |
| **JRE** | Java Runtime Environment | Runs Java programs | JVM + Libraries |
| **JDK** | Java Development Kit | Develops and runs Java programs | JRE + Compiler + Tools |

---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**6.What are variables in Java? Explain with examples.**

A **variable** in Java is a **name** that stores a **value** which can change during program execution. Think of it like a **box** where you keep data such as numbers, text, or characters.

Before using a variable, you must **declare its type** (like int, String, etc.) so the program knows what kind of data it will hold.

### ****Syntax of Variable Declaration:****

dataType variableName = value;

Integer Variable

int age = 20;

Here, int is the data type, age is the variable name, and 20 is the value stored.

* **String Variable**

String name = "Sanjana";

Here, String is used to store text.

* **Float Variable**

float price = 99.50f;

Here, f is added at the end because it's a float.

* **Boolean Variable**

boolean isJavaFun = true;

This stores a true/false value.

---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**7**.**What are the different types of operators in Java?**

* Arithmetic Operators

| **Operator** | **Meaning** | **Example** |
| --- | --- | --- |
| + | Addition | a + b |
| - | Subtraction | a - b |
| \* | Multiplication | a \* b |
| / | Division | a / b |
| % | Modulus (remainder) | a % b |

* Relational (Comparison) Operators

| **Operator** | **Meaning** | **Example** |
| --- | --- | --- |
| == | Equal to | a == b |
| != | Not equal to | a != b |
| > | Greater than | a > b |
| < | Less than | a < b |
| >= | Greater or equal | a >= b |
| <= | Less or equal | a <= b |

* Logical Operators

| **Operator** | **Meaning** | **Example** |
| --- | --- | --- |
| && | Logical AND | (a > 5 && b < 10) |
| ! | Logical NOT | !(a > 5) |

* Assignment Operators

| **Operator** | **Meaning** | **Example** |
| --- | --- | --- |
| = | Assign | x = 5 |
| += | Add and assign | x += 5 → x = x + 5 |
| -= | Subtract and assign | x -= 2 |
| \*= | Multiply and assign | x \*= 3 |
| /= | Divide and assign | x /= 2 |
| %= | Modulus and assign | x %= 2 |

------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**8. Explain control statements in Java (if, if-else, switch).**

### **if Statement**

The if statement runs a block of code **only if the condition is true**.

Syntax:

if (condition) {

// code to execute if condition is true

}

Example:

int age = 18;

if (age >= 18) {

System.out.println("You are eligible to vote.");

}

* if-else Statement

The if-else statement runs **one block if the condition is true**, otherwise it runs the **else block**.

Syntax:

if (condition) {

// code if condition is true

} else {

// code if condition is false

}

Example:

int age = 16;

if (age >= 18) {

System.out.println("You can vote.");

} else {

System.out.println("You are too young to vote.");

}

* Switch Statement

The switch statement is used to **select one of many options** based on the value of a variable.

Syntax:

switch (value) {

case option1:

// code

break;

case option2:

// code

break;

default:

// code if no match

}

Example:

int day = 2;

switch (day) {

case 1:

System.out.println("Sunday");

break;

case 2:

System.out.println("Monday");

break;

default:

System.out.println("Other Day");

}

--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**9.Write a Java program to find whether a number is even or odd.**

import java.util.Scanner;

public class EvenOddCheck {

public static void main(String[] args) {

Scanner input = new Scanner(System.in);

System.out.print("Enter a number: ");

int number = input.nextInt(); // taking number from user

if (number % 2 == 0) {

System.out.println(number + " is Even.");

} else {

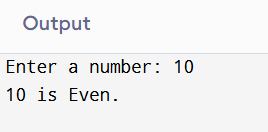
System.out.println(number + " is Odd.");

}

input.close();

}

}



------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**10.What is the difference between while and do-while loop?**

| **Feature** | **while Loop** | **do-while Loop** |
| --- | --- | --- |
| **Condition Check** | Checked before the loop runs | Checked after the loop runs |
| **Minimum**  **Executions** | May not run at all if condition is false | Runs at least once, even if condition is false |
| **Use Case** | When you want to check the condition **first** | When you want to run the code **at least once** |
| **Syntax** | while (condition) { ... } | do { ... } while (condition); |
| **Example Code** | java\nint i = 5;\nwhile (i < 5) {\n System.out.println("While: " + i);\n i++;\n} | java\nint i = 5;\ndo {\n System.out.println("Do-While: " + i);\n i++;\n} while (i < 5); |
| **Output** | *(No output — because the condition is false at the beginning)* | Do-While: 5 (Executes once even though condition is false) |

# Object-Oriented Programming (OOPs)

**1. What are the main principles of OOPs in Java? Explain each.**

### ****Encapsulation****

**Definition:**  
Encapsulation means **hiding the internal details** of a class and **only exposing necessary parts** through methods.

**Key Idea:**  
Keep variables **private**, and provide **public getter and setter methods** to access and update them.

**Example:**

java

CopyEdit

public class Student {

private String name; // hidden

public void setName(String n) {

name = n;

}

public String getName() {

return name;

}

}

* **Inheritance**

**Definition:**  
Inheritance means one class (child) can reuse the properties and behaviors of another class (parent).

**Key Idea:**

Helps in code reusability.

**Example:**

CopyEdit

class Animal {

void sound() {

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

}

}

class Dog extends Animal {

void bark() {

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

}

}

* **Polymorphism**

**Definition:**

Polymorphism means **one task can be done in multiple ways**.

**Types:**

* **Compile-time (Method Overloading)**
* **Run-time (Method Overriding)**

**Example:**

// Overloading (same method name, different parameters)

class Math {

int add(int a, int b) {

return a + b;

}

double add(double a, double b) {

return a + b;

}

}

* **Abstraction**

**Definition:**  
Abstraction means **hiding unnecessary details** and showing only the **important parts** to the user.

**Key Idea:**  
Achieved using **abstract classes** and **interfaces**.

**Example:**

abstract class Shape {

abstract void draw();

}

class Circle extends Shape {

void draw() {

System.out.println("Drawing Circle");

}

}

**2.What is a class and an object in Java? Give examples.**

### ****What is a Class in Java?****

A **class** is like a **blueprint** or **template**. It defines the **properties (variables)** and **behaviors (methods)** of objects, but it doesn’t do anything by itself until you create an object from it.

Example:

public class Car {

String color = "Red"; // property

void drive() { // method

System.out.println("Car is driving");

}

}

**What is an Object in Java?**

An **object** is a **real-world instance** of a class. It is created using the new keyword and allows you to access the variables and methods of the class.

Example:

public class Main {

public static void main(String[] args) {

Car myCar = new Car(); // Creating an object of class Car

System.out.println(myCar.color); // Accessing property

myCar.drive(); // Calling method

}

}

------------------------------------------------------------------------------------------------------------------------------------------------------------------

**3.Write a program using class and object to calculate area of a rectangle.**

import java.util.Scanner;

class Rectangle {

int length;

int width;

int calculateArea() {

return length \* width;

}

}

public class Main {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in);

Rectangle rect = new Rectangle(); // Creating object

System.out.print("Enter length: ");

rect.length = sc.nextInt(); // User input

System.out.print("Enter width: ");

rect.width = sc.nextInt(); // User input

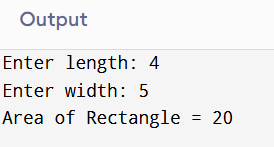
int area = rect.calculateArea(); // Calling method

System.out.println("Area of Rectangle = " + area);

sc.close();

}

}



------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**4.Explain inheritance with real-life example and Java code.**

Inheritance means one class (child/subclass) can use the properties and methods of another class (parent/superclass). It helps in code reusability.

Real-life Example:

* Think of a general class: Vehicle

All vehicles have speed and can move**.**

Then we have a specific class: Car

* A car is a type of vehicle, so it should inherit everything a vehicle has, and add its own features like music system.

Java Code Example:

// Parent class (Super class)

class Vehicle {

void move() {

System.out.println("Vehicle is moving");

}

}

// Child class (Sub class) that inherits from Vehicle

class Car extends Vehicle {

void musicSystem() {

System.out.println("Car has a music system");

}

}

// Main class to test

public class Main {

public static void main(String[] args) {

Car myCar = new Car(); // Create object of child class

myCar.move(); // Inherited method from Vehicle

myCar.musicSystem(); // Method of Car class

}

}

---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**5.What is polymorphism? Explain with compile-time and runtime examples.**

Polymorphism means "many forms".  
In Java, polymorphism allows one method or object to behave differently based on how it's used.

Compile-time Polymorphism (Method Overloading)

Example:

class Calculator {

int add(int a, int b) {

return a + b;

}

double add(double a, double b) {

return a + b;

}

}

public class Main {

public static void main(String[] args) {

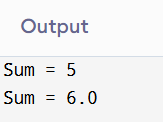
Calculator calc = new Calculator();

System.out.println("Sum = " + calc.add(2, 3)); // int version

System.out.println("Sum = " + calc.add(2.5, 3.5)); // double version

}

}



* Run-time Polymorphism (Method Overriding)

When a subclass provides a specific implementation of a method already defined in the parent class, and the method to run is decided at runtime.

Example:

class Animal {

void sound() {

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

}

}

class Dog extends Animal {

void sound() {

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

}

}

public class Main {

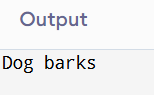
public static void main(String[] args) {

Animal a = new Dog(); // Parent class reference, child class object

a.sound(); // Calls Dog's version at runtime

}

}



**------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------**

1. **What is method overloading and method overriding? Show with examples**.

* Method Overloading

**Definition:**  
When **multiple methods** in the **same class** have the **same name** but **different parameters** (number, type, or order), it is called **method overloading**.

Example:

class Calculator {

// Method 1: Adds two integers

int add(int a, int b) {

return a + b;

}

// Method 2: Adds three integers

int add(int a, int b, int c) {

return a + b + c;

}

// Method 3: Adds two doubles

double add(double a, double b) {

return a + b;

}

}

public class Main {

public static void main(String[] args) {

Calculator calc = new Calculator();

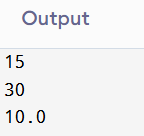
System.out.println(calc.add(5, 10)); // 15

System.out.println(calc.add(5, 10, 15)); // 30

System.out.println(calc.add(5.5, 4.5)); // 10.0

}

}



* **Method Overriding**

**Definition:**  
When a **subclass provides a specific implementation** of a method that is already **defined in the parent class**, it is called **method overriding**.

Example:

class Animal {

void sound() {

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

}

}

class Dog extends Animal {

void sound() {

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

}

}

public class Main {

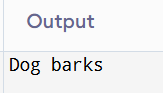
public static void main(String[] args) {

Animal a = new Dog(); // Polymorphism

a.sound(); // Output: Dog barks

}

}



------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------**6. What is encapsulation? Write a program demonstrating encapsulation.**

Encapsulation is the concept of hiding data (variables) inside a class and allowing access to it only through methods ,

Java Program Demonstrating Encapsulation

class Student {

// Step 1: private data

private String name;

private int age;

// Step 2: public setter method to set data

public void setName(String n) {

name = n;

}

public void setAge(int a) {

age = a;

}

// Step 3: public getter method to get data

public String getName() {

return name;

}

public int getAge() {

return age;

}

}

public class Main {

public static void main(String[] args) {

Student s = new Student();

s.setName("Amit"); // setting values

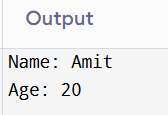
s.setAge(20);

System.out.println("Name: " + s.getName()); // getting values

System.out.println("Age: " + s.getAge());

}

}



-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

1. **What is abstraction in Java? How is it achieved?**

Abstraction is the process of hiding internal details and showing only the essential features to the user.

Abstraction in Java is achieved in two ways:

| **Method** | Description |
| --- | --- |
| **Abstract Class** | A class that has abstract methods (without body) and normal methods |
| **Interface** | A contract with method definitions only (no implementation) |

Using Abstract Class – Example:

abstract class Animal {

abstract void sound(); // abstract method

void sleep() {

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

}

}

class Dog extends Animal {

void sound() {

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

}

}

Using Interface – Example:

interface Vehicle {

void start(); // abstract method

}

class Car implements Vehicle {

public void start() {

System.out.println("Car starts with key");

}

}

------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**9. Explain the difference between abstract class and interface.**

| **Feature** | **Abstract Class** | **Interface** |
| --- | --- | --- |
| **Keyword** | abstract class | interface |
| **Method Types** | Can have abstract and concrete methods | Only abstract methods (till Java 7) |
| **Variables** | Can have variables with any access modifier | All variables are public static final |
| **Multiple Inheritance** | Not supported (only one abstract class) | Supported (a class can implement many) |
| **Constructors** | Can have constructors | Cannot have constructors |
| **Inheritance Type** | Class extends abstract class | Class implements interface |
| **Use Case** | When some **code is shared** across classes | When only **structure (methods)** is needed |

**----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------10. Create a Java program to demonstrate the use of interface.**

create an interface Vehicle, and two classes Car and Bike that implement this interface.

// Interface declaration

interface Vehicle {

void start(); // abstract method

void stop(); // abstract method

}

// Class Car implements Vehicle

class Car implements Vehicle {

public void start() {

System.out.println("Car is starting...");

}

public void stop() {

System.out.println("Car is stopping...");

}

}

// Class Bike implements Vehicle

class Bike implements Vehicle {

public void start() {

System.out.println("Bike is starting...");

}

public void stop() {

System.out.println("Bike is stopping...");

}

}

// Main class to test

public class Main {

public static void main(String[] args) {

Vehicle myCar = new Car();

Vehicle myBike = new Bike();

myCar.start(); // Output: Car is starting...

myCar.stop(); // Output: Car is stopping...

myBike.start(); // Output: Bike is starting...

myBike.stop(); // Output: Bike is stopping...

}

}

