

Name: Avishkar Bhapkar
LMS Code: AF04953327

Java Basics & OOPs Assignment Questions

1. Java Basics

1. What is Java? Explain its features.

At its core, Java is a set of instructions you give to a computer. What makes it special is this big idea: "Write Once, Run Anywhere."

Imagine you write a recipe. With Java, it's like writing that recipe in a universal language that any chef, no matter where they are or what kind of kitchen they have, can understand and follow. You don't need to rewrite the recipe for a gas stove vs. an electric one; it just works. That "universal understanding" for computers is handled by something called the Java Virtual Machine (JVM).

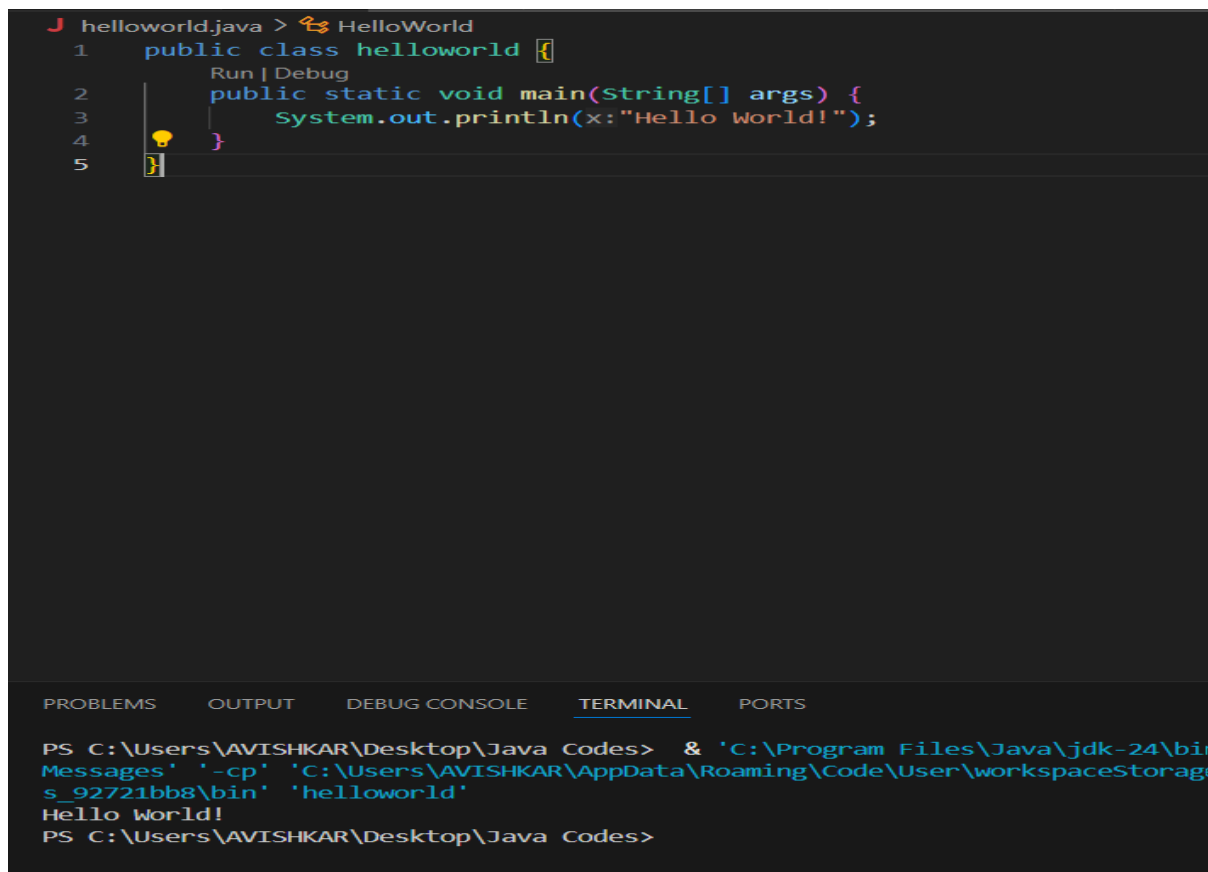
2. Explain the Java program execution process.

Write Code: You write .java (source code).

Compile: Java Compiler (javac) converts .java into .class (bytecode).

Execute: Java Virtual Machine (JVM) loads and runs the .class bytecode, converting it to machine code for your specific computer.

3. Write a simple Java program to display 'Hello World'.



```
helloworld.java > HelloWorld
1 public class helloworld {
2     public static void main(String[] args) {
3         System.out.println(x: "Hello World!");
4     }
5 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\AVISHKAR\Desktop\Java Codes> & 'C:\Program Files\Java\jdk-24\bin\
Messages' '-cp' 'C:\Users\AVISHKAR\AppData\Roaming\Code\User\workspaceStorage
s_92721bb8\bin' 'helloworld'
Hello World!
PS C:\Users\AVISHKAR\Desktop\Java Codes>
```

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

Data types define the type and size of data a variable can hold.

A. Primitive Data Types (store direct values):

- **Integers (whole numbers):** byte, short, int (most common), long (very large).
- **Floating-point (decimals):** float (less precise), double (most common, high precision).
- **Character:** char (single character).
- **Boolean:** boolean (true or false).

B. Non-Primitive (Reference) Data Types (store references to objects):

- **String:** Stores sequences of characters (text).
- **Arrays:** Stores collections of same-type elements.
- **Classes/Interfaces:** User-defined types.

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

- **JVM (Java Virtual Machine):** The **runtime engine** that executes Java bytecode. It's the "actual runner."
- **JRE (Java Runtime Environment):** Includes the **JVM + core libraries**. It's what you need to *run* Java applications.
- **JDK (Java Development Kit):** Includes **JRE + development tools** (like the compiler javac). It's what you need to *write, compile, and run* Java applications.

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

Variables are named memory locations that store data. You declare them with a type and a name.

```

J test1.java > test1 > main(String[])
1 public class test1 {
    Run | Debug
2     public static void main(String[] args) {
3
4
5         int age = 19; // Integer data type
6         // This is a comment explaining the variable
7
8
9         float temperature = 36.5f; // Float data type, 'f' is used to denote float literals
10        // This is a comment explaining the variable
11
12
13        double pi = 3.14159; // Double data type, more precision than float
14
15
16        char grade = 'B'; // Character data type, single character enclosed in single quotes
17
18
19        boolean isPassed = true; // Boolean data type, can be true or false
20
21
22        String name = "Avishkar"; // String data type, sequence of characters enclosed in double quotes
23
24
25        System.out.println("Name: " + name);
26        System.out.println("Age: " + age);
27
28    }
29 }

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

s_92721bb8\bin' 'test1'
Name: Avishkar
Age: 19
Temperature: 36.5
Value of Pi: 3.14159
Grade: B
Passed: true

```

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

Operators perform actions on values and variables.

Arithmetic: +, -, *, /, %, ++, -- (math operations)

Assignment: =, +=, -=, etc. (assign values)

Relational (Comparison): ==, !=, >, <, >=, <= (compare values, result is true/false)

Logical: && (AND), || (OR), ! (NOT) (combine/invert Boolean conditions)

Ternary (Conditional): condition? value1: value2 (shorthand for simple if-else)

Bitwise: &, |, ^, ~, <<, >>, >>> (operate on bits - advanced)

InstanceOf: Checks if an object is an instance of a class.

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

Control statements determine the flow of execution in your program based on conditions.

```
1 public class IfElse {
2     Run | Debug
3     public static void main(String[] args) {
4         // variable declaration
5         int a=45;
6         //to check a is even or odd
7         if(a % 2 == 0) { // if condition
8             // if condition is true
9             System.out.println(x:"a is even");
10        } else {
11            System.out.println(x:"a is odd");// print statement if condition is false
12        }
13    }
14 }
15
16
```

```
PS C:\Users\AVISHKAR\Desktop\Java Codes>
Messages' '-cp' 'C:\Users\AVISHKAR\AppData\Local\Temp\javac2_92721bb8\bin' 'IfElse'
a is odd
PS C:\Users\AVISHKAR\Desktop\Java Codes>
```

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

```

1 public class IfElse {
    Run | Debug
2     public static void main(String[] args) {
3         // variable declaration
4         int a=45;
5         //to check a is even or odd
6         if(a % 2 == 0) { // if condition
7             // if condition is true
8
9             System.out.println(x:"a is even");
10        } else {
11            System.out.println(x:"a is odd");// print statement if condition is false
12        }
13    }
14 }
15
16

```

```

PS C:\Users\AVISHKAR\Desktop\Java Codes>
Messages' '-cp' 'C:\Users\AVISHKAR\AppData
s_92721bb8\bin' 'IfElse'
a is odd
PS C:\Users\AVISHKAR\Desktop\Java Codes>

```

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

Both while and do-while loops repeat a block of code, but:

- **while loop:** Checks the condition *before* executing the loop body. It may not execute even once if the condition is initially false.
- **do-while loop:** Executes the loop body *at least once*, then checks the condition. It continues looping as long as the condition is true.

while (condition) { /* code */ } // Checks, then executes

do { /* code */ } while (condition); // Executes once, then checks

2. Object-Oriented Programming (OOPs)

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

OOPs (Object-Oriented Programming) principles are ways to structure code using "objects."

Encapsulation: Bundling data and methods together inside a class, and hiding internal details (like a phone hiding its circuits). Achieved with private variables and public methods.

Inheritance: A new class (child) gets properties/methods from an existing class (parent). Promotes code reuse ("is-a" relationship). Uses extends keyword.

Polymorphism: "Many forms." A single action (like a method call) can behave differently based on the object or context.

Abstraction: Showing only essential features and hiding complex implementation details (like driving a car without knowing its engine internals). Achieved with abstract classes and interfaces.

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

- **Class:** A blueprint or template for creating objects. It defines common properties and behaviors but doesn't exist physically.
 - **Example (Class):** `class Dog {String name; void bark () { /*...*/ } }`
- **Object:** A real-world instance of a class. It's a concrete entity created from the blueprint, occupying memory.
 - **Example (Object):** `Dog myDog = new Dog();` (myDog is an object of the Dog class)

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

```

1  class Rectangle {
2      double length;
3      double width;
4      double calculateArea() {
5          return length * width;
6      }
7  }
8
9  public class area {
10     Run | Debug
11     public static void main(String[] args) {
12         Rectangle rect1 = new Rectangle(); // Create object
13         rect1.length = 10.0;
14         rect1.width = 5.0;
15         System.out.println("Area: " + rect1.calculateArea());
16     }
17 }

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

PS C:\Users\AVISHKAR\Desktop\Java Codes> & 'C:\Program Files\Java\jdk-24\bin\java.exe'
Messages' '-cp' 'C:\Users\AVISHKAR\AppData\Roaming\Code\User\workspaceStorage\4ac1136b57
s_92721bb8\bin' 'area'
Area: 50.0
PS C:\Users\AVISHKAR\Desktop\Java Codes>

```

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

Inheritance: A mechanism where a new class (child/subclass) acquires properties and methods from an existing class (parent/superclass). Reuses code and creates an "is-a" relationship.


Real-life Example: A Car is a Vehicle. Car inherits speed and accelerate() from Vehicle, and adds numDoors.

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

Polymorphism: "Many forms." A single action can have different implementations depending on the context or object type.

Compile-time Polymorphism (Method Overloading):

- Multiple methods in the *same class* with the same name but different parameters.
- Compiler decides which method to call.
- **Example:**

```
J Mathop.java >  Mathop
1
2 class Mathop {
3     int add(int a, int b) { return a + b; }
4     double add(double a, double b) { return a + b; } // Overloaded
5
6     Run | Debug
7     public static void main(String[] args) {
8         Mathop calc = new Mathop();
9         System.out.println("Adding two integers: " + calc.add(a:7, b:3));
10        System.out.println("Adding two doubles: " + calc.add(a:4.5, b:2.1));
11    }
12 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\AVISHKAR\Desktop\Java Codes> & 'C:\Program Files\Java\jdk-24\bin\java.exe' '
Messages' '-cp' 'C:\Users\AVISHKAR\AppData\Roaming\Code\User\workspaceStorage\4ac1136b577
s_92721bb8\bin' 'Mathop'
Adding two integers: 10
Adding two doubles: 6.6
PS C:\Users\AVISHKAR\Desktop\Java Codes>
```


6. What is method overloading and method overriding? Show with examples.

Method Overloading:

- **Concept:** Multiple methods in the *same class* with the same name but different *parameter lists* (number, type, or order of arguments).
- **When decided:** Compile-time.

Example:

```
1  class Printer {
2      void print(int num) {
3          System.out.println("Printing integer: " + num);
4      }
5
6      // Overloaded method: same name, different parameter type
7      void print(String text) {
8          System.out.println("Printing string: " + text);
9      }
10
11     // Overloaded method: same name, different number of parameters
12     void print(int num1, int num2) {
13         System.out.println("Printing two integers: " + num1 + ", " + num2);
14     }
15 }
16
17 public class OverloadingDemo {
18     Run | Debug
19     public static void main(String[] args) {
20         Printer p = new Printer();
21         p.print(num:100);
22         p.print(text:"Hello Java!");
23         p.print(num1:5, num2:15);
24     }
25 }
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
S C:\Users\AVISHKAR\Desktop\Java Codes> & 'C:\Program Files\Java\jdk-24\bin\java.exe' '-
essages' '-cp' 'C:\Users\AVISHKAR\AppData\Roaming\Code\User\workspaceStorage\4ac1136b577e
_92721bb8\bin' 'OverloadingDemo'
rinting integer: 100
rinting string: Hello Java!
rinting two integers: 5, 15
```

7. What is encapsulation? Write a program demonstrating encapsulation.

```
1 package Encapsulation;
2
3 // Encapsulation Example: BankAccount
4
5 public class BankAccount {
6     // Private data members
7     private String accountHolder;
8     private double balance;
9
10    // Getter for accountHolder
11    public String getAccountHolder() {
12        return accountHolder;
13    }
14
15    // Setter for accountHolder
16    public void setAccountHolder(String accountHolder) {
17        this.accountHolder = accountHolder;
18    }
19
20    // Getter for balance
21    public double getBalance() {
22        return balance;
23    }
24
25    // Setter for balance with validation
26    public void setBalance(double balance) {
27        if (balance >= 0) {
28            this.balance = balance;
29        }
30    }
31 }
```

8. What is abstraction in Java? How is it achieved?

Abstraction: Hiding complex implementation details and showing only the essential features to the user. Focuses on *what* an object does, not *how* it does it.

How it's achieved:

1. **Abstract Classes:** Classes declared with the abstract keyword. They can have both regular methods and abstract methods (methods without a body, forcing subclasses to implement them). Cannot be instantiated directly.
2. **Interfaces:** Blueprints of a class. They define a set of methods that a class *must* implement. Contain only abstract methods (before Java 8, then default/static methods were added).

9. Explain the difference between abstract class and interface.

1. Feature	2. Abstract Class	3. Interface
4. Keyword	5. abstract class	6. interface

7. Methods	8. Can have abstract (no body) and concrete (with body) methods.	9. Before Java 8: All abstract. After Java 8: Can have default and static methods too.
10. Variables	11. Can have non-static, non-final (regular) variables.	12. All variables are public static final by default (constants).
13. Constructors	14. Can have constructors. (Used by subclasses with super()).	15. Cannot have constructors.
16. Inheritance	17. A class extends only one abstract class.	18. A class implements multiple interfaces. (implements keyword)
19. Purpose	20. Defines a common base for related classes, allowing some shared implementation and some mandatory unique implementations.	21. Defines a contract or a capability that classes can promise to fulfill.
22. Instantiation	23. Cannot be instantiated directly.	24. Cannot be instantiated directly.

**10. Create a Java program to demonstrate the use of interface.
(Use of AI tool Google Gemini was done in the below pgr)**

```

1 // 1. Define the Interface
2 interface Switchable {
3     // All methods in an interface are implicitly public and abstract (before Java 8)
4     // or public static/default (Java 8+).
5     // They define WHAT needs to be done, not HOW.
6     void turnOn();
7     void turnOff();
8     boolean isSwitchedOn(); // Check current state
9 }
10
11 // 2. Implement the Interface in different classes
12 class Light implements Switchable {
13     private boolean isOn = false;
14
15     @Override // Implementing the turnOn method from Switchable interface
16     public void turnOn() {
17         if (!isOn) {
18             isOn = true;
19             System.out.println(x:"Light is ON. Let there be light!");
20         } else {
21             System.out.println(x:"Light is already ON.");
22         }
23     }
24
25     @Override // Implementing the turnOff method
26     public void turnOff() {
27         if (isOn) {
28             isOn = false;
29             System.out.println(x:"Light is OFF. It's dark now.");
30         } else {
31             System.out.println(x:"Light is already OFF.");
32         }
33     }
34
35     @Override // Implementing the isSwitchedOn method
36     public boolean isSwitchedOn() {
37         return isOn;

```

```

    @Override // Implementing the isSwitchedOn method
    public boolean isSwitchedOn() {
        return isOn;
    }
}

class Fan implements Switchable {
    private boolean isRunning = false;
    private int speed = 0;

    @Override
    public void turnOn() {
        if (!isRunning) {
            isRunning = true;
            speed = 1; // Start at low speed
            System.out.println("Fan is ON at speed " + speed + ".");
        } else {
            System.out.println(x:"Fan is already ON.");
        }
    }

    @Override
    public void turnOff() {
        if (isRunning) {
            isRunning = false;
            speed = 0;
            System.out.println(x:"Fan is OFF.");
        } else {
            System.out.println(x:"Fan is already OFF.");
        }
    }
}

```

```

@Override
public boolean isSwitchedOn() {
    return isRunning;
}

// Fan-specific method (not from interface)
public void increaseSpeed() {
    if (isRunning && speed < 3) {
        speed++;
        System.out.println("Fan speed increased to " + speed + ".");
    } else if (isRunning) {
        System.out.println(x:"Fan is already at max speed.");
    } else {
        System.out.println(x:"Turn on the fan first!");
    }
}
}

// 3. Demonstrate the use of the interface
public class InterFaceDemo {
    Run | Debug
    public static void main(String[] args) {
        // We can declare references of the interface type
        Switchable device1 = new Light(); // Polymorphism: An object of Light (concrete type)
                                           // is referred to by an interface type Switchable.
        Switchable device2 = new Fan();    // An object of Fan (concrete type)
                                           // is referred to by an interface type Switchable.

        System.out.println(x:"--- Operating Light ---");
        device1.turnOn();
        System.out.println("Is Light on? " + device1.isSwitchedOn());
        device1.turnOff();
        System.out.println("Is Light on? " + device1.isSwitchedOn());
        device1.turnOn(); // Turn on again
    }
}

```

```

105
106 // We need to cast to Fan type to access Fan-specific methods
107 if (device2 instanceof Fan) { // Check if it's actually a Fan object
108     ((Fan) device2).increaseSpeed(); // Cast and call specific method
109     ((Fan) device2).increaseSpeed();
110     ((Fan) device2).increaseSpeed(); // Max speed
111 }
112
113 device2.turnOff();
114 System.out.println("Is Fan on? " + device2.isSwitchedOn());
115 }
116 }

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

cp' 'C:\Users\AVISHKAR\AppData\Roaming\Code\User\workspaceStorage\4ac1136b577e004e5806695bab95adae\redhat.java\jdt_w
in' 'InterFaceDemo'
--- Operating Light ---
Light is ON. Let there be light!
Is Light on? true
Light is OFF. It's dark now.
Is Light on? false
Light is ON. Let there be light!

--- Operating Fan ---
Fan is ON at speed 1.
Is Fan on? true
Fan speed increased to 2.
Fan speed increased to 3.
Fan is already at max speed.
Fan is OFF.
Is Fan on? false

```