Q.1 Very Short Answer Type Questions (2 marks each)

1. **What does the acronym OOP stand for in Java?**
   → OOP stands for Object-Oriented Programming, a programming paradigm based on the concept of objects, which can contain data and methods. Java uses OOP principles such as encapsulation, inheritance, polymorphism, and abstraction to build modular and reusable code.

2. **What is the purpose of this keyword in Java?**
   → The this keyword in Java is used to refer to the current class instance. It is commonly used to resolve naming conflicts between instance variables and parameters, invoke current class constructors, or pass the current object as a parameter to another method.

3. **Name two types of loops in Java.**
   → Two commonly used loops in Java are:
   - for loop – used when the number of iterations is known.
   - while loop – used when the number of iterations is not fixed and depends on a condition.

4. **What is the default value of a boolean variable in Java?**
   → The default value of a boolean variable in Java is false. This applies when a boolean is declared as a class or instance variable and not explicitly initialized.

5. **How do you declare an integer array in Java?**
   → An integer array can be declared as:
   int[] arr = new int[5];
   This creates an array named arr that can store 5 integer values. You can also initialize it directly, like: int[] arr = {1, 2, 3, 4, 5};.

6. **What does the final keyword do in Java?**
   → The final keyword in Java is used to restrict modification.
   - A final variable cannot be reassigned.
   - A final method cannot be overridden by subclasses.
   - A final class cannot be inherited.
     It is used to ensure consistency and prevent accidental changes.

7. **What is method overloading in Java?**
   → Method overloading is a feature in Java where multiple methods can have the same name but with different parameter lists (type, number, or order). It helps in increasing code readability and allows methods to perform similar functions with different inputs.

8. **What is the role of the super keyword in inheritance?**
   → The super keyword is used in Java to refer to the immediate parent class. It is mainly used to:
   - Call the parent class constructor using super()
   - Access parent class methods or variables that are hidden by the child class.

9. **What is the purpose of the try-catch block in exception handling?**
   → The try-catch block is used to handle exceptions (runtime errors) in Java. The code that might throw an exception is placed in the try block, and the catch block contains code to handle the exception gracefully without terminating the program abruptly.

10. **Define Package.**
    → A package in Java is a namespace that organizes a set of related classes and interfaces. It helps in avoiding name conflicts, access protection, and makes it easier to maintain and locate classes. Java provides built-in packages like java.util, java.io, and users can also create their own.

Q.2 Short Answer Type Questions (4 Marks)

11. What are the key principles of Object-Oriented Programming (OOP)?

-> OOP is a paradigm in programming that organizes software design around objects rather than functions. The key principles are:

1. Encapsulation: Bundling data and methods that operate on the data within a class. It helps in hiding the internal state of an object.
   Example: Private variables with public getters/setters.

1. Inheritance: Allows a class to acquire properties and behavior (methods) from another class. It promotes code reuse.
   Example: class Dog extends Animal {}

2. Polymorphism: Allows objects to be treated as instances of their parent class. There are two types: compile-time (method overloading) and runtime (method overriding).

3. Abstraction: Hides complexity by showing only the relevant details to the user. Achieved using abstract classes and interfaces.

12. What is the access specification? Explain with examples.

-> Access specifiers in Java control the visibility of classes, methods, and variables.
There are four main access specifiers:

Specifier	Visibility
Public:-	Accessible from anywhere
Private:-	Accessible only within the declared class
Protected:-	Accessible within the same package or subclasses
Default:-	Accessible only within the same package (no keyword needed)

Example:-

```
public class Person {
   private String name; // only within this class
   public void setName(String n) {
         name = n;
     } // accessible anywhere
}
```

13. Compare Object-Oriented Programming and Procedural-Oriented Programming.

Feature	OOP	POP
Focus	Focuses on objects and data	Focuses on functions and procedures
Modularity	Code is organized into classes	Code is organized into functions
Data security	Data is hidden using encapsulation	Data is shared and less secure
Reusability	Promotes reuse through inheritance	Less reusable
Examples	Java, Python, C++	C, Pascal, Fortran

14. What is type conversion and casting in Java, and how does it work?

Java supports two types of type conversion:

1. Implicit Type Conversion (Widening)
   Automatically done by the compiler when converting from a smaller to a larger data type.

Example:

```
int x = 10;

double y = x; // Implicitly converted
```

2. Explicit Type Casting (Narrowing)
   Done manually when converting from a larger to a smaller data type.

   Example:

```
double a = 10.5;

int b = (int) a; // Narrowing, loss of precision
```

Casting helps in memory management and operations involving multiple data types.

15. How do arrays work in Java, and how are they declared and initialized? Arrays in Java are used to store multiple values of the same type in a single variable.

- Declaration:

  int[] arr;

- Memory Allocation:

  arr = new int[5]; // 5 elements

- Initialization at declaration:

  int[] arr = {10, 20, 30, 40, 50};

Arrays have a fixed size and use zero-based indexing. You can access elements using indices like arr[0], arr[1], etc.

16. What is operator precedence in Java? Explain arithmetic operators with examples.

Operator precedence determines the order in which operators are evaluated in expressions. Arithmetic operators in Java:

Operator	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus (remainder)

Precedence Order (High to Low):

1. **()**
2. *** / %**
3. **+ -**

Example:

```
int result = 10 + 2 * 5; // 10 + 10 = 20 (Multiplication first)
```

You can change the order using parentheses:

```
int result = (10 + 2) * 5; // 12 * 5 = 60
```

17. What is the purpose of the this keyword in Java methods?

The this keyword refers to the current object of the class.

Uses:

- To distinguish between class fields and parameters with the same name.
- To call one constructor from another.
- To pass the current object as an argument.

Example:

```
class Student {
   int id;
   Student(int id) {
      this.id = id; // refers to instance variable
   }
}
```

18. What is the purpose of the final keyword in Java methods?

When a method is declared with the final keyword, it cannot be overridden by subclasses.

Purpose:

- To maintain the method's original behavior.
- To prevent accidental changes in the method logic.

Example:

```
class A {
   final void show() {
      System.out.println("This method cannot be overridden");
   }
}
class B extends A {
   // void show() {} // Compile-time error
}
```

19. Define Method. Explain Main method of Java.

A method is a block of code that performs a specific task and is called when needed.

Syntax:

```
returnType methodName(parameters) {
  // code }
```

Main method: The entry point of a Java program.

```
public static void main(String[] args) {
  System.out.println("Hello Java");
}
```

- public: accessible anywhere
- static: no object required to call
- void: does not return a value
- String[] args: accepts command-line arguments

20. Explain Abstract class with examples.

An abstract class is a class that cannot be instantiated and may contain abstract methods (methods without a body).

Purpose: To provide a base class for other classes to extend.

Example:

```
abstract class Animal {
  abstract void sound(); // abstract method
}
class Dog extends Animal {
  void sound() {
    System.out.println("Barks");
  }
}
```

Abstract classes can have both abstract and non-abstract methods.

21. What is recursion, and how can it be implemented in Java?

Recursion is a process in which a method calls itself to solve a problem. It is commonly used for tasks that can be broken into smaller, repetitive problems, such as calculating factorials or traversing data structures like trees.

Each recursive function has:

- Base case: Condition under which recursion stops.
- Recursive case: Function calls itself with modified input.

Example:

```
public class RecursionExample {
   static int factorial(int n) {
      if (n == 0) return 1;       // base case
      else return n * factorial(n - 1); // recursive case
   }
   public static void main(String[] args) {
      System.out.println(factorial(5)); // Output: 120
   }
}
```

This approach simplifies complex problems, but requires careful design to avoid infinite recursion or stack overflow errors.

22. What is the significance of the "super" keyword in inheritance?

In Java, the super keyword is used to refer to the immediate parent class of a subclass. It is commonly used in the following scenarios:

1. Calling Parent Class Constructor:

   super(); // calls the constructor of the parent class

2. Accessing Parent Class Methods:

   super.methodName(); // calls overridden method from parent class

3. Accessing Parent Class Variables:

   super.variableName; // accesses parent class variable

Example:

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

class Dog extends Animal {
   void sound() {
      super.sound(); // Calls Animal's sound()
      System.out.println("Dog barks");
   }
}
```

super helps resolve naming conflicts and reuse the parent class functionality efficiently.

23. What is method overloading in Java, and how does it differ from method overriding?

Method Overloading:

- Occurs within the same class.
- Methods have the same name but different parameters (type, number, or both).
- Achieved at compile-time (also known as compile-time polymorphism).

Example:

```
void show(int a) {}
void show(String s) {}
```

Method Overriding:

- Occurs between parent and child classes.
- Subclass redefines a method from the parent class with the same name and parameters.
- Achieved at runtime (runtime polymorphism).

Example:

```
class A {
   void display() {
      System.out.println("A's display");
   }
}
class B extends A {
   void display() {
      System.out.println("B's display");
   }
}
```

Key Difference:

- Overloading: Same class, compile-time.
- Overriding: Different classes (inheritance), runtime.

24. What is method overriding in Java, and how is it used?

Method Overriding allows a subclass to provide a specific implementation of a method that is already defined in its parent class. It enables runtime polymorphism and helps in achieving dynamic method dispatch.

Rules for Method Overriding:

- Method must have the same name, return type, and parameters.
- The overriding method must have equal or more accessible access modifiers.
- Only inherited methods can be overridden.

Example:

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

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

Usage:

```
Animal a = new Dog();
a.sound();  // Outputs: Dog barks
```

Method overriding provides flexibility in method behavior across class hierarchies.

25. What is the difference between the while loop and the do-while loop in Java?

Feature	While Loop	Do-while Loop
Condition Check	Checked before loop execution	Checked after loop execution
Minimum Execution	May execute 0 times	Executes at least once
Common Use Case	When condition must be true first	When code must run at least once
Syntax	while(condition) {}	do { } while(condition);

Example:-	int i = 5;
	while (i < 5) {
	System.out.println(i); // will not execute
	}
		int i = 5;
		do {
		System.out.println(i); // will execute once
		} while (i < 5);

✅ Q3. Long Answer Type Questions (6 Marks)

---

1. What are the key features of Java? Explain briefly.

Java is a high-level, object-oriented programming language developed by Sun Microsystems in 1995. It has become one of the most widely used programming languages due to its rich features. The key features are:

1. Platform Independence
    - Java programs are compiled into bytecode by the Java compiler.
    - Bytecode can be executed on any machine with a Java Virtual Machine (JVM).
    - This makes Java Write Once, Run Anywhere (WORA).
2. Object-Oriented
    - Java follows all the principles of OOP: Encapsulation, Inheritance, Polymorphism, and Abstraction.
    - Everything in Java is based on classes and objects, making code more modular and reusable.
3. Simple and Familiar
    - Java syntax is clean and easy to understand.
    - It is similar to C++ but removes complex features like pointers and operator overloading.
4. Robust
    - Java emphasizes early error checking, memory management, and exception handling.
    - It has a powerful runtime exception handling model that helps avoid program crashes.
5. Secure
    - Java programs run in a virtual machine (sandbox), which makes it secure from external threats.
    - It provides features like bytecode verification, access control, and no pointer manipulation.
6. Multithreaded
    - Java has built-in support for multithreading (handling multiple tasks simultaneously).

- It allows the creation of responsive and efficient programs.

7. Distributed
    - Java supports networking and distributed computing through APIs like RMI (Remote Method Invocation), Sockets, and CORBA.
8. High Performance
    - Though interpreted, Java uses a Just-In-Time (JIT) compiler that compiles bytecode into native machine code at runtime, improving speed.

2. What are the core concepts of Object-Oriented Programming (OOP)? Explain them briefly.

OOP (Object-Oriented Programming) is a programming paradigm that revolves around objects and classes. The four main pillars of OOP are:

1. Encapsulation

- It is the practice of wrapping the data (variables) and methods into a single unit called a class.
- It helps hide the internal implementation details from the user and allows access only through public methods.
- Example:

```
class Student {
   private int marks;
   public void setMarks(int m) { marks = m; }
   public int getMarks() { return marks; }
}
```

2. Abstraction

- Abstraction is hiding complex implementation and showing only the essential details to the user.
- Achieved using abstract classes and interfaces.
- Example:

```
abstract class Shape {
   abstract void draw();
}
```

3. Inheritance

- One class acquires the properties (fields) and methods of another.
- Promotes code reusability.
- Example:

```
class Animal {
    void eat() {
    }
}
class Dog extends Animal {
    void bark() {}
}
```

4. Polymorphism

- One method behaves differently based on the object or arguments.
- Compile-time polymorphism = Method Overloading
- Run-time polymorphism = Method Overriding

3. What is inheritance in Java? Describe the different types of inheritance.

Inheritance is a mechanism in Java by which one class (subclass) can inherit the properties and behaviors (fields and methods) of another class (superclass).

Advantages:

- Code reuse
- Method overriding
- Easier maintenance

Types of Inheritance in Java:

1. Single Inheritance
    - A class inherits from only one superclass.
    - Example:

```
class A { void msg() {} }
class B extends A { void display() {} }
```

2. Multilevel Inheritance
    - A class inherits from another class, which in turn inherits another class.
    - Example:

```
class A {}
class B extends A {}
class C extends B {}
```

3. Hierarchical Inheritance
    - Multiple subclasses inherit from a single superclass.
    - Example:

```
class Animal {}
class Dog extends Animal {}
class Cat extends Animal {}
```

4. Multiple Inheritance (using Interfaces)
    - Java does not support multiple inheritance with classes due to ambiguity.
    - Achieved using interfaces.
    - Example:

```
interface A {}
interface B {}
class C implements A, B {}
```

4. What is polymorphism in Java? Explain its different types.

Polymorphism means the ability of a single entity (method or object) to take multiple forms.

Types of Polymorphism:

1. Compile-Time Polymorphism (Static) – Method Overloading
    - Same method name with different parameters.
    - Decided during compilation.
    - Example:

```
class Display {
   void show(int a) {}
```

```
    void show(String b) {}
}
```

2. Run-Time Polymorphism (Dynamic) – Method Overriding
    - A subclass provides a specific implementation of a method already defined in its parent class.
    - Achieved through inheritance and dynamic binding.
    - Example:

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

Advantages of Polymorphism:

- Increases program flexibility
- Improves code maintainability
- Reduces duplication

5. What is looping in Java? Explain all the loop structures available in Java.

Looping is a fundamental programming concept that allows us to execute a block of code repeatedly based on a given condition. It reduces redundancy and enhances efficiency.

Java supports the following loop types:

1. for Loop

Best used when the number of iterations is known.

Syntax:

```
for(initialization; condition; increment/decrement) {
    // body of the loop
}
```

Example:

```
for (int i = 1; i <= 5; i++) {
   System.out.println(i);
}
```

2. while Loop

Executes a block of code as long as the condition is true. The condition is checked before the loop executes.

Syntax:

```
while (condition) {
   // loop body
}
```

Example:

```
int i = 1;
while (i <= 5) {
   System.out.println(i);
   i++;
}
```

3. do-while Loop

Executes the loop at least once, and then continues based on the condition. Condition is checked after loop execution.

Syntax:

```
do {
   // body
} while (condition);
```

Example:

```
int i = 1;
do {
   System.out.println(i);
   i++;
} while (i <= 5);
```

4. for-each Loop (Enhanced for loop)

Used for iterating over arrays or collections without using an index.

Syntax:

```
for (dataType var : array) {
   // use var
}
```

Example:

```
int[] arr = {1, 2, 3};
for (int num : arr) {
   System.out.println(num);
}
```

6. What is an exception in Java? Explain the try-catch block with an example.

An exception is an event that occurs during the execution of a program and disrupts the normal flow of instructions. Java provides a robust exception handling mechanism to handle runtime errors gracefully.

Types of Exceptions:

1. Checked Exceptions
   Handled during compile time.
   Example: IOException, SQLException.
2. Unchecked Exceptions
   Occur during runtime.
   Example: ArithmeticException, NullPointerException.

Exception Handling Keywords:

- try – Code that may throw an exception.
- catch – Handles the exception.
- finally – Executes after try-catch, regardless of exception.
- throw – Used to manually throw an exception.
- throws – Declares exceptions that a method might throw.

Example with try-catch:

```
public class Example {
   public static void main(String[] args) {
      try {
         int result = 10 / 0;
         System.out.println("Result: " + result);
      } catch (ArithmeticException e) {
         System.out.println("Error: Division by zero!");
      }
   }
}
```

Output:

vbnet

CopyEdit

Error: Division by zero!

This prevents program termination and provides a way to recover.

7. What are classes and objects in Java? Provide a suitable example.

Class:

A class is a blueprint for creating objects. It defines properties (fields) and behaviors (methods).

Object:

An object is a real-world instance of a class. It contains real data and can invoke the methods defined in the class.

Defining a Class:

```
class Car {
   String color;
   int speed;
   void drive() {
      System.out.println("Car is driving");
   }
}
```

Creating an Object:

```
public class TestCar {
   public static void main(String[] args) {
      Car myCar = new Car();  // object creation
      myCar.color = "Red";
      myCar.speed = 100;

      System.out.println("Color: " + myCar.color);
      myCar.drive(); // calling method
   }
}
```

Key Concepts:

- A class defines the structure.
- An object occupies memory and can perform actions.
- Multiple objects can be created from one class.

8. What is a data type in Java? Explain it with an example.

A data type defines the type of data a variable can store. Java is strongly typed, meaning each variable must be declared with a data type before use.

1. Primitive Data Types (8 total):

Data Type	Size	Description
byte	1 byte	Whole number (-128 to 127)
short	2 bytes	Whole number (-32K to 32K)
int	4 bytes	Whole number
long	8 bytes	Large whole number
float	4 bytes	Decimal number (single precision)
double	8 bytes	Decimal number (double precision)
char	2 bytes	Single character (Unicode)
boolean	1 bit	true or false

2. Non-Primitive Data Types:

- Arrays
- Strings
- Classes
- Interfaces

Example:

```
int age = 20;
float price = 99.99f;
char grade = 'A';
boolean isPassed = true;
String name = "Alice";
```

Each data type uses a different amount of memory and serves a different purpose.