

OOP Viva

🕒 Created	@January 19, 2025 6:38 AM
☑ Reviewed	<input type="checkbox"/>

1. What is Object-Oriented Programming (OOP)?

- **Answer:** OOP is a programming paradigm that organizes code into objects, which combine data (attributes) and behavior (methods). It uses principles like **encapsulation**, **inheritance**, **polymorphism**, and **abstraction**.
-

2. What are the main principles of OOP?

- **Answer:**
 1. **Encapsulation:** Bundling data and methods together while restricting access using access modifiers like `private`, `public`, and `protected`.
 2. **Inheritance:** Reusing and extending existing classes using `extends`.
 3. **Polymorphism:** Having multiple forms, like method overloading (compile-time) and method overriding (runtime).
 4. **Abstraction:** Hiding complex details and showing only essential features using abstract classes or interfaces.
-

3. What is a Class and an Object?

- **Answer:**
 - A **class** is a blueprint for creating objects. It defines properties and methods.
 - An **object** is an instance of a class. It represents a specific entity with data and behavior.
-

4. What is the difference between Method Overloading and Method Overriding?

- **Answer:**
 - **Method Overloading:** Multiple methods in a class with the same name but different parameters (compile-time polymorphism).
 - **Method Overriding:** A subclass provides a specific implementation of a method from its parent class (runtime polymorphism).
-

5. What is the difference between an Abstract Class and an Interface?

- **Answer:**
 - **Abstract Class:**
 - Can have both abstract (no body) and concrete methods.
 - Can have instance variables.
 - Use `extends` to inherit.
 - **Interface:**
 - Only has abstract methods (before Java 8).
 - Can have static and default methods (from Java 8).
 - Use `implements` to inherit.
-

6. What are Access Modifiers in Java?

- **Answer:**
 - **Private:** Accessible only within the class.
 - **Default:** Accessible within the same package.
 - **Protected:** Accessible within the same package and subclasses.
 - **Public:** Accessible everywhere.
-

7. What is the difference between `this` and `super` keywords?

- **Answer:**

- `this` : Refers to the current object.
 - `super` : Refers to the parent class and is used to access its methods or constructors.
-

8. Explain Constructor and its types.

- **Answer:**
 - A **constructor** is a special method used to initialize an object. It has the same name as the class.
 - Types:
 1. **Default Constructor:** No arguments.
 2. **Parameterized Constructor:** Takes arguments to initialize object attributes.
-

9. What is a Static Method?

- **Answer:** A static method belongs to the class, not an object. It can be called without creating an object using `ClassName.methodName()`.
-

10. What is the difference between Composition and Inheritance?

- **Answer:**
 - **Composition:** "Has-a" relationship. A class contains another class as a field.
 - **Inheritance:** "Is-a" relationship. A subclass inherits the properties and behavior of its parent class.
-

11. What is Garbage Collection in Java?

- **Answer:** Garbage collection is the process of automatically freeing memory by removing objects that are no longer in use. It's managed by the JVM.
-

12. What are Wrapper Classes?

- **Answer:** Wrapper classes are used to convert primitive data types into objects (e.g., `int` to `Integer`, `double` to `Double`).
-

13. What is an Exception? How is it handled?

- **Answer:** An exception is an event that disrupts the normal flow of a program. It is handled using:
 - `try` block: Code that might throw an exception.
 - `catch` block: Handles the exception.
 - `finally` block: Executes code regardless of exceptions.
-

14. What is the `final` keyword in Java?

- **Answer:**
 - **Final variable:** Its value cannot be changed.
 - **Final method:** Cannot be overridden.
 - **Final class:** Cannot be extended.
-

15. What is the difference between `Array` and `ArrayList` ?

- **Answer:**
 - **Array:** Fixed size, supports primitives.
 - **ArrayList:** Dynamic size, only supports objects.
-

Would you like help with specific topics or additional questions?

40

You said:

can you give some small arrays , arraylist and for and a while as well as a small thread to code

ChatGPT said:

ChatGPT

Sure! Here are small examples of code snippets for **arrays**, **ArrayList**, **for loop**, **while loop**, and **threading** in Java:

1. Array Example

```
java
CopyEdit
public class ArrayExample {
    public static void main(String[] args) {
        // Declare and initialize an array
        int[] numbers = {10, 20, 30, 40, 50};

        // Print all elements using a for loop
        for (int i = 0; i < numbers.length; i++) {
            System.out.println("Element at index " + i + ": "
+ numbers[i]);
        }
    }
}
```

2. ArrayList Example

```
java
CopyEdit
import java.util.ArrayList;

public class ArrayListExample {
    public static void main(String[] args) {
        // Create an ArrayList
        ArrayList<String> fruits = new ArrayList<>();

        // Add elements to the ArrayList
        fruits.add("Apple");
    }
}
```

```
        fruits.add("Banana");
        fruits.add("Cherry");

        // Print all elements using an enhanced for loop
        for (String fruit : fruits) {
            System.out.println(fruit);
        }
    }
}
```

3. For Loop Example

```
java
CopyEdit
public class ForLoopExample {
    public static void main(String[] args) {
        // Simple for loop
        for (int i = 1; i <= 5; i++) {
            System.out.println("Count: " + i);
        }
    }
}
```

4. While Loop Example

```
java
CopyEdit
public class WhileLoopExample {
    public static void main(String[] args) {
        int i = 1;

        // While loop
```

```

        while (i <= 5) {
            System.out.println("Count: " + i);
            i++;
        }
    }
}

```

5. Thread Example

```

java
CopyEdit
// Thread using the Runnable interface
class MyThread implements Runnable {
    public void run() {
        for (int i = 1; i <= 5; i++) {
            System.out.println("Thread is running: " + i);
            try {
                Thread.sleep(500); // Pause for 500ms
            } catch (InterruptedException e) {
                System.out.println(e.getMessage());
            }
        }
    }
}

public class ThreadExample {
    public static void main(String[] args) {
        // Create and start a thread
        Thread thread = new Thread(new MyThread());
        thread.start();
    }
}

```