#Encapsulation:

Encapsulation is often introduced first because it forms the foundation of OOP principles. It emphasizes the bundling of data (attributes) and methods (behaviors) into a single unit (class), allowing for data hiding and controlled access.

Understanding encapsulation helps learners grasp the importance of data abstraction and how to create classes with well-defined interfaces.

#Inheritance:

Inheritance allows classes to inherit properties and behaviors from other classes, promoting code reuse and establishing "is-a" relationships between classes.

Understanding inheritance builds on encapsulation by showing how classes can be organized hierarchically, with more specialized classes inheriting from more general classes.

#Polymorphism:

Polymorphism can be introduced after encapsulation and inheritance because it builds on these concepts. Polymorphism allows objects of different classes to be treated as objects of a common superclass, enabling flexibility and extensibility in code.

Understanding polymorphism helps learners see how different classes can share common interfaces and be used interchangeably, enhancing code readability and maintainability.

**1. What is Object-Oriented Programming (OOP) and how does Java support it?**

**Answer:** Object-Oriented Programming is a programming paradigm based on the concept of "objects", which can contain data, in the form of fields (often known as attributes or properties), and code, in the form of procedures (often known as methods). Java supports OOP through the use of classes, objects, inheritance, polymorphism, abstraction, and encapsulation.

**Example:** In Java, you can create a Car class (template) and then create multiple objects (instances) of that class, each representing a specific car with its own state and behavior.

java

class Car {

String model;

int year;

void display() {

System.out.println(model + " " + year);

}

}

**2. Explain the concept of a class and an object in Java with an example.**

**Answer:** In Java, a class is a blueprint from which individual objects are created. It defines the data and the methods that the objects will use. An object is an instance of a class, containing actual values instead of the variable placeholders.

**Example:**

java

class Bicycle {

int gear;

int speed;

Bicycle(int gear, int speed) {

this.gear = gear;

this.speed = speed;

}

void applyBrake(int decrement) {

speed -= decrement;

}

void speedUp(int increment) {

speed += increment;

}

}

public class Main {

public static void main(String[] args) {

Bicycle myBike = new Bicycle(3, 30);

myBike.applyBrake(10);

System.out.println("Current Speed: " + myBike.speed);

}

}

**3. What is inheritance in Java? Give an example.**

**Answer:** Inheritance in Java is a mechanism where a new class is derived from an existing class. The new class inherits all the fields and methods of the existing class, and can add its own fields and methods.

**Example:**

java

class Vehicle {

int maxSpeed;

Vehicle(int maxSpeed) {

this.maxSpeed = maxSpeed;

}

}

class Car extends Vehicle {

int doors;

Car(int maxSpeed, int doors) {

super(maxSpeed);

this.doors = doors;

}

}

**4. Describe method overloading and method overriding in Java.**

**Answer:** Method overloading occurs when two or more methods in the same class have the same name but different parameters (type or number). Method overriding means defining a method in a child class that already exists in the parent class with the same signature (name and parameters).

**Example of Overloading:**

java

class Display {

void show(int n) {

System.out.println(n);

}

void show(String s) {

System.out.println(s);

}

}

**Example of Overriding:**

java

class Animal {

void sound() {

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

}

}

class Dog extends Animal {

@Override

void sound() {

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

}

}

**5. What is polymorphism in Java? Provide an example.**

**Answer:** Polymorphism in Java is the ability of an object to take on many forms. It mainly occurs in two ways: compile-time polymorphism (method overloading) and runtime polymorphism (method overriding).

**Example of Runtime Polymorphism:**

java

class Animal {

void sound() {

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

}

}

class Cat extends Animal {

void sound() {

System.out.println("Cat meows");

}

}

public class TestPolymorphism {

public static void main(String[] args) {

Animal myAnimal = new Cat(); // Cat object is treated as an Animal

myAnimal.sound();

}

}

**6. Explain encapsulation in Java with an example.**

**Answer:** Encapsulation is the technique of wrapping the data (variables) and code acting on the data (methods) together as a single unit. In encapsulation, the variables of a class are hidden from other classes and can be accessed only through the methods of their current class.

**Example:**

java

class Account {

private double balance;

public void deposit(double amount) {

if (amount > 0) {

balance += amount;

}

}

public double getBalance() {

return balance;

}

}

**7. What is an abstract class in Java? Provide an example.**

**Answer:** An abstract class in Java is a class that cannot be instantiated and is designed to be subclassed. It can contain abstract methods (without a body) as well as concrete methods (with a body).

**Example:**

java

abstract class Animal {

abstract void sound();

void sleep() {

System.out.println("Zzz");

}

}

class Dog extends Animal {

void sound() {

System.out.println("Woof");

}

}

**8. Describe the concept of an interface in Java and give an example.**

**Answer:** An interface in Java is a reference type that can contain only constants, method signatures, default methods, static methods, and nested types. Interfaces cannot contain instance fields. The methods in interfaces are abstract by default.

**Example:**

java

interface Animal {

void sound();

}

class Cat implements Animal {

public void sound() {

System.out.println("Meow");

}

}

**9. What is the use of the super keyword in Java?**

**Answer:** The super keyword in Java is a reference variable that is used to refer to the immediate parent class object. It is used to access methods of the parent class, invoke the parent class constructor, and access variables of the parent class.

**Example:**

java

class Parent {

void display() {

System.out.println("Parent display()");

}

}

class Child extends Parent {

void display() {

super.display(); // Calls the display method of Parent class

System.out.println("Child display()");

}

}

**10. Explain the concept of packages in Java.**

**Answer:** In Java, a package is a namespace that organizes a set of related classes and interfaces. Conceptually, you can think of packages as being similar to different folders on your computer. Packages are used to avoid name conflicts and to control access, making code modules more manageable.

**Example:**

java

package animals;

public class Dog {

public void bark() {

System.out.println("Woof");

}

}

**11. How does Java implement multiple inheritances?**

**Answer:** Java does not support multiple inheritances with classes to avoid complexity and ambiguity. However, it supports multiple inheritances through interfaces. A class can implement multiple interfaces, which allows it to inherit the abstract methods of multiple interfaces.

**Example:**

java

interface WaterAnimal {

void swim();

}

interface LandAnimal {

void walk();

}

class Frog implements WaterAnimal, LandAnimal {

public void swim() {

System.out.println("Frog swims");

}

public void walk() {

System.out.println("Frog walks");

}

}

**12. What is the final keyword in Java, and how can it be used?**

**Answer:** The final keyword in Java is used to apply restrictions on classes, methods, and variables. A final class cannot be subclassed, a final method cannot be overridden, and a final variable’s value cannot be modified once it is initialized.

**Example:**

java

final class Immutable {

final int VALUE = 10;

final void showMessage() {

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

}

}

**13. Explain exception handling in Java.**

**Answer:** Exception handling in Java is a powerful mechanism that handles runtime errors to maintain normal application flow. An exception is an event that disrupts the normal flow of the program. Java uses a try-catch block to handle exceptions.

**Example:**

java

try {

int data = 50 / 0; // May throw exception

} catch (ArithmeticException e) {

System.out.println(e);

}

**14. What is garbage collection in Java?**

**Answer:** Garbage collection in Java is the process by which the JVM reclaims the runtime unused memory automatically. The garbage collector automatically deletes objects to which no active pointer exists.

**Example:** There's no explicit way to force garbage collection; it's handled by the JVM. However, calling System.gc() suggests that the JVM expedites the process.

java

public class TestGarbage1 {

public void finalize() {

System.out.println("object is garbage collected");

}

public static void main(String args[]) {

TestGarbage1 s1 = new TestGarbage1();

TestGarbage1 s2 = new TestGarbage1();

s1 = null;

s2 = null;

System.gc();

}

}

**15. Explain the use of the static keyword in Java.**

**Answer:** The static keyword in Java is used to indicate that a particular field or method belongs to a class, rather than instances of it. This means that a static member can be accessed directly by the class name and doesn’t need an object to access it.

**Example:**

java

class MathUtility {

static int add(int a, int b) {

return a + b;

}

}

public class Test {

public static void main(String[] args) {

int sum = MathUtility.add(5, 10); // No object is required

System.out.println("Sum is: " + sum);

}

}

**16. What is a constructor in Java? Provide an example.**

**Answer:** A constructor in Java is a block of code similar to a method that's called when an instance of an object is created. Unlike methods, constructors have no return type, not even void, and are named after the class.

**Example:**

java

class Book {

String title;

// Constructor

Book(String title) {

this.title = title;

}

void displayTitle() {

System.out.println("Title: " + title);

}

}

public class Main {

public static void main(String[] args) {

Book book = new Book("Java Basics");

book.displayTitle();

}

}

**17. How do you achieve data hiding and encapsulation in Java? Explain with an example.**

**Answer:** Data hiding and encapsulation are achieved in Java by making the class variables private and providing public getter and setter methods to modify and view the variables' values.

**Example:**

java

class Employee {

private int salary;

public void setSalary(int salary) {

if (salary > 0) {

this.salary = salary;

}

}

public int getSalary() {

return salary;

}

}

**18. What is the difference between == and equals() in Java?**

**Answer:** In Java, == is a reference comparison operator that compares if two references point to the same object. equals() is a method for content comparison that determines if two objects have the same value.

**Example:**

java

String s1 = new String("Java");

String s2 = new String("Java");

System.out.println(s1 == s2); // false because reference is different

System.out.println(s1.equals(s2)); // true because content is same

**19. What is the use of the this keyword in Java?**

**Answer:** The this keyword in Java is a reference variable that refers to the current object. It is used to refer to the current object's instance variables, constructors, and methods.

**Example:**

java

class Student {

int rollNo;

String name;

Student(int rollNo, String name) {

this.rollNo = rollNo;

this.name = name;

}

}

**20. Explain the difference between static and non-static methods in Java.**

**Answer:** Static methods belong to the class rather than any specific instance of the class and can be called without creating an object of the class. Non-static methods (instance methods) belong to an instance of the class and require an object of the class to be created before they can be called.

**Example:**

java

class MathOperations {

static int add(int a, int b) {

return a + b;

}

int subtract(int a, int b) {

return a - b;

}

}

public class Test {

public static void main(String[] args) {

int sum = MathOperations.add(10, 20); // static method call

MathOperations obj = new MathOperations();

int difference = obj.subtract(20, 10); // non-static method call

}

}

**21. How does the finally block work in exception handling in Java?**

**Answer:** The finally block in Java is used to execute a set of statements following a try block that occurs regardless of whether an exception was thrown or caught. It is typically used to close resources like files or databases.

**Example:**

java

try {

// risky code

} catch (Exception e) {

// handle exception

} finally {

// cleanup code, e.g., close file

}

**22. What is the difference between abstract classes and interfaces in Java?**

**Answer:** Abstract classes can have both abstract (without body) and non-abstract methods (with body), provide a base for subclasses, and can have constructors. Interfaces can only have abstract methods (until Java 8) and static or default methods (from Java 8 onwards), and cannot have constructors. Interfaces support multiple inheritances, whereas abstract classes do not.

**Example:**

java

abstract class Animal {

abstract void eat();

void sleep() {

System.out.println("Animal sleeps");

}

}

interface Movable {

void move();

}