Polymorphism, Life of an object in Java and OOPs concept - fundamentals of any object-oriented language.

**1. Polymorphism**

Polymorphism in Java is the ability of an object to take on many forms. It is a fundamental concept of object-oriented programming (OOP) that allows us to write more flexible and reusable code.

There are two main types of polymorphism in Java:

**Compile-time polymorphism:** This type of polymorphism is resolved at compile-time by the compiler. It is achieved by function overloading.

public class Polymorphism {

// First addition function

public static int add (int a, int b) {

return a + b;

}

// Second addition function

public static double add (double a, double b) {

return a + b;

}

public static void main (String args []) {

System.out.println(add (2, 3) );

System.out.println(add (2.0, 3.0) );

}

}

**Runtime polymorphism:** This type of polymorphism is resolved at runtime. Method overriding is the runtime polymorphism having the same method with same parameters but different classes.

class Test {

public void method () {

System.out.println("Method 1");

}

}

// Defining a child class

public class Test2 extends Test {

// Overriding the parent method

public void method () {

System.out.println("Method 2");

}

public static void main(String args[])

{

Test test= new Test2 ();

test.method ();

}

}

**2. Life of an Object in Java:**

**Creation & Initialization:**

- An object is created from a class template using the `new` keyword.

- Memory is allocated for the object, and its constructor is called for initialization.

**Usage:**

- The object is actively used to perform tasks, access data, or call methods.

- It does its purpose within the program.

**Destruction (Garbage Collection):**

- When no references to the object exist, it becomes unreachable.

- The Java garbage collector identifies unreachable objects and reclaims their memory.

- This process ensures efficient memory management by removing objects that are no longer needed.

public class ObjectLifeCycleExample {

public static void main(String[] args) {

// Creation and Initialization

Person person = new Person("John", 25);

// Usage

System.out.println("Name: " + person.getName());

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

}

}

class Person {

private String name;

private int age;

public Person(String name, int age) {

this.name = name;

this.age = age;

}

public String getName() {

return name;

}

public int getAge() {

return age;

}

}

**Fundamentals of OOPS of any programming language**

Certainly! Object-Oriented Programming (OOP) is a fundamental programming paradigm that aims to model real-world entities as objects, providing a structured way to organize and manage code. The key OOP concepts are as follows:

**1. Classes and Objects:**

Classes are blueprints or templates for creating objects. They define the structure (attributes/properties) and behavior (methods/functions) that objects of the class will have.

Objects are instances of classes. They represent real-world entities and have specific attributes and behaviors as defined by the class.

**2.** **Encapsulation:**

- Encapsulation is the concept of bundling data (attributes) and methods (functions) that operate on that data into a single unit, known as a class. It hides the internal details and protects the integrity of the data.

- Access modifiers (public, private, protected) control the visibility and accessibility of class members.

**3. Abstraction:**

- Abstraction focuses on the essential features of an object while hiding the unnecessary details. It simplifies complex reality by modeling classes based on their core characteristics and behaviors.

- Abstract classes and interfaces are used to define contracts that concrete classes must adhere to.

**4. Inheritance:**

- Inheritance is a mechanism that allows a new class (subclass or derived class) to inherit attributes and behaviors from an existing class (superclass or base class).

- It promotes code reusability and the creation of class hierarchies.

**5. Polymorphism:**

- Polymorphism allows objects of different classes to be treated as objects of a common superclass. It enables a single interface to represent general behavior while allowing different classes to provide their specific implementations.

- Polymorphism is achieved through method overriding and method overloading.

class Animal {

private String name;

public Animal(String name) {

this.name = name;

}

public void speak() {

System.out.println(name + " makes a sound");

}

}

class Dog extends Animal {

public Dog(String name) {

super(name);

}

public void speak() {

System.out.println(name + " barks");

}

}

abstract class Shape {

abstract double calculateArea();

}

class Circle extends Shape {

double radius;

Circle(double radius) {

this.radius = radius;

}

@Override

double calculateArea() {

return Math.PI \* radius \* radius;

}

}

class Vehicle {

void start() {

System.out.println("The vehicle is started.");

}

}

class Car extends Vehicle {

void accelerate() {

System.out.println("The car is accelerating.");

}

}

public class OOPExample {

public static void main(String[] args) {

**// Classes and Objects**

Dog myDog = new Dog("Buddy");

myDog.speak();

**// Encapsulation**

Circle circle = new Circle(5.0);

System.out.println("Circle area: " + circle.calculateArea());

**// Abstraction**

Vehicle myCar = new Car();

myCar.start();

((Car) myCar).accelerate(); // Casting is needed to access the accelerate method

**// Inheritance**

Animal myAnimal = new Dog("Rex");

myAnimal.speak();

**// Polymorphism**

myAnimal = new Animal("Unknown");

myAnimal.speak();

}

}