**The four main principles of Object-Oriented Programming (OOP) are:**

**1. Encapsulation**

Encapsulation is the concept of bundling data and methods that operate on that data within a single unit, called a class or object. This helps to hide the internal implementation details and expose only the necessary information to the outside world.

Example:

public class BankAccount {

private double balance;

public void deposit(double amount) {

balance += amount;

}

public void withdraw(double amount) {

if (balance >= amount) {

balance -= amount;

}

}

public double getBalance() {

return balance;

}

}

**2. Abstraction**

Abstraction is the concept of showing only the necessary information to the outside world while hiding the internal implementation details. This helps to reduce complexity and improve modularity.

Example:

public abstract class Shape {

public abstract double area();

}

public class Circle extends Shape {

private double radius;

public Circle(double radius) {

this.radius = radius;

}

@Override

public double area() {

return Math.PI \* radius \* radius;

}

}

**3. Inheritance**

Inheritance is the concept of creating a new class based on an existing class. The new class inherits the properties and behavior of the existing class and can also add new properties and behavior or override the existing ones.

Example:

public class Animal {

public void sound() {

System.out.println("The animal makes a sound.");

}

}

public class Dog extends Animal {

@Override

public void sound() {

System.out.println("The dog barks.");

}

public void fetch() {

System.out.println("The dog fetches.");

}

}

**4. Polymorphism**

Polymorphism is the concept of having multiple forms of a single entity, such as a method or an object. This can be achieved through method overloading or method overriding.

Example:

public class Calculator {

public int add(int a, int b) {

return a + b}

public double add(double a, double b) {

return a + b;

}

}

**2. How does inheritance work in OOP? Write a sample program to demonstrate single and multilevel inheritance.**

Inheritance in Object-Oriented Programming (OOP) is a mechanism where a new class, called the subclass or derived class, inherits the properties and behavior of an existing class, called the superclass or base class. The subclass can also add new properties and behavior or override the ones inherited from the superclass.

Single Inheritance

Single inheritance is a type of inheritance where a subclass inherits from a single superclass.

Example:

public class Animal {

public void sound() {

System.out.println("The animal makes a sound.");

}

}

public class Dog extends Animal {

public void bark() {

System.out.println("The dog barks.");

}

}

public class Main {

public static void main(String[] args) {

Dog myDog = new Dog();

myDog.sound();

myDog.bark();.

}

}

**Multilevel Inheritance**

Multilevel inheritance is a type of inheritance where a subclass inherits from a superclass that itself inherits from another superclass.

Example:

public class Animal {

public void sound() {

System.out.println("The animal makes a sound.");

}

}

public class Mammal extends Animal {

public void walk() {

System.out.println("The mammal walks.");

}

}

public class Dog extends Mammal {

public void bark() {

System.out.println("The dog barks.");

}

}

public class Main {

public static void main(String[] args) {

Dog myDog = new Dog();

myDog.sound();

myDog.walk();

myDog.bark();

}

}

**3. What is the difference between method overloading and method overriding? Provide code examples.**

**Method overloading and method overriding are two fundamental concepts in object-oriented programming (OOP).**

**Method Overloading**

Method overloading is a technique where multiple methods with the same name can be defined, but with different parameter lists. This allows objects of a class to behave differently based on the input parameters.

Example:

public class Calculator {

public int add(int a, int b) {

return a + b;

}

public double add(double a, double b) {

return a + b;

}

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

return a + b + c;

}

public static void main(String[] args) {

Calculator calculator = new Calculator();

System.out.println(calculator.add(1, 2));

System.out.println(calculator.add(1.5, 2.5));

System.out.println(calculator.add(1, 2, 3));

}

}

**Method Overriding**

Method overriding is a technique where a subclass provides a different implementation of a method that is already defined in its superclass. This allows the subclass to specialize the behavior of the method.

Example:

public class Animal {

public void sound() {

System.out.println("The animal makes a sound.");

}

}

public class Dog extends Animal {

@Override

public void sound() {

System.out.println("The dog barks.");

}

public static void main(String[] args) {

Animal animal = new Animal();

animal.sound();

Dog dog = new Dog();

dog.sound();

Animal polymorphicDog = new Dog();

polymorphicDog.sound();

}

}

**4. What is encapsulation and how does it help in software development? Show how it is implemented in code.**

**Encapsulation is a fundamental concept in object-oriented programming (OOP) that binds together the data and the methods that manipulate that data, keeping both safe from outside interference and misuse.**

Benefits of Encapsulation

1. Data Hiding: Encapsulation helps to hide the internal implementation details of an object from the outside world, making it harder for other parts of the program to modify or depend on the internal workings of the object.

2. Code Reusability: Encapsulation promotes code reusability by allowing objects to be used in different contexts without worrying about the internal implementation details.

3. Improved Security: Encapsulation helps to protect the internal state of an object from unauthorized access or modification.

4. Easier Maintenance: Encapsulation makes it easier to modify or extend the internal implementation of an object without affecting other parts of the program.

**Implementing Encapsulation in Code**

**Here's an example of encapsulation in Java:**

public class BankAccount {

private double balance;

public BankAccount(double initialBalance) {

this.balance = initialBalance;

}

public void deposit(double amount) {

if (amount > 0) {

balance += amount;

}

}

public void withdraw(double amount) {

if (amount > 0 && balance >= amount) {

balance -= amount;

}

}

public double getBalance() {

return balance;

}

}

Using the Encapsulated Class

Here's an example of how to use the BankAccount class:

public class Main {

public static void main(String[] args) {

BankAccount account = new BankAccount(1000.0);

account.deposit(500.0);

account.withdraw(200.0);

System.out.println("Current balance: " + account.getBalance());

}

}