Question 1: What are the four main principles of Object oriented programming?

Ans:- Encapsulation involves bundling data (attributes) and methods (functions) that operate on the data into a single unit, typically a class. It restricts direct access to some of an object's components, which is a means of preventing unintended interference and misuse of the data.

This is achieved through access modifiers like private, protected, and public.

In a class representing a bank account, the balance can be kept private, and access to it can be provided through public methods like deposit() and withdraw().

2. Abstraction

Abstraction means hiding complex implementation details and showing only the essential features of the object. It helps in reducing programming complexity and effort. By using abstraction, a programmer can focus on interactions at a higher level without needing to understand all the underlying details.

Example: When you use a List interface in Java, you can perform operations like add() or remove() without knowing the underlying implementation (e.g., ArrayList or LinkedList).

3. Inheritance

Inheritance is a mechanism where a new class (child class) inherits properties and behaviors (methods) from an existing class (parent class). It promotes code reusability and establishes a hierarchical relationship between classes.

Example: If you have a general class Vehicle, you can create more specific classes like Car and Bike that inherit from Vehicle and add their unique features.

4. Polymorphism

Polymorphism allows objects to be treated as instances of their parent class rather than their actual class. The two types of polymorphism are compile-time (method overloading) and runtime (method overriding). It enables a single interface to control access to a variety of underlying data types.

Example: A function draw() can be defined in a base class Shape, and each subclass like Circle or Rectangle can have its own implementation of draw(). When you call draw() on a Shape reference, the appropriate subclass method is invoked.

Question 2: How does inheritance work in oop? Write a sample program to demonstrate single and multilevel inheritance.

Answer: Inheritance is a fundamental concept in object-oriented programming (OOP) that allows a class (called a child or derived class) to inherit attributes and methods from another class (called a parent or base class). This promotes code reuse and establishes a hierarchical relationship between class.

1. Single Inheritance

```
class Animal {
  void eat() {
    System.out.println("This animal eats food.");
  }
}
class Dog extends Animal {
  void bark() {
    System.out.println("The dog barks.");
  }
}
public class Main {
  public static void main(String[] args) {
    Dog d = new Dog();
    d.eat();
    d.bark();
  }
}
```

2. Multilevel Inheritance

```
class Animal {
  void eat() {
    System.out.println("This animal eats food.");
class Dog extends Animal {
  void bark() {
    System.out.println("The dog barks.");
  }
}
class Puppy extends Dog {
  void weep() {
    System.out.println("The puppy weeps.");
  }
}
public class Main {
  public static void main(String[] args) {
    Puppy p = new Puppy();
    p.eat();
    p.bark();
    p.weep();
  }
```

Question 3: What are the differences between method overloading and method overriding? Provide code examples.

Answer:-

- **Features Method Overloading Method Overriding**
- **Defination-- same method name same method name**
- **With different Pararameter. And parameter in**
- **Subclass.**
- **polymorphism --- compile -time runtime**

- **Inheritance --- NO Yes**
- **Parameter list must differ must be the same**
- **Return Type can be same or different must be same or convenient**
- **Access Modifiers can vary should not be more restrictive**
- **Static methods can be overloaded cannot be overridden**

Method overloading

```
class Calculator {
   int add(int a, int b) {
      return a + b;

   }
  double add(double a, double b) {
    return a + b;
       }

       int add(int a, int b, int c) {
          return a + b + c;

        }
    }
   public class Main {
       public static void main(String[] args) {
          Calculator calc = new Calculator();
          System.out.println(calc.add(2, 3));
    System.out.println(calc.add(2.5, 3.5));
          System.out.println(calc.add(1, 2, 3));
     }
}
```

2 Method overriding

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

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

public class Main {
    public static void main(String[] args) {
        Animal obj = new Dog();
        obj.sound();    // Output: Dog barks
    }
}
```

Question 4:-- what is encapsulation and how does it help in software development? Show how it is implemented in class.

Ans-- Encapsulation is a fundamental concept in object-oriented programming (OOP) that involves bundling data (attributes) and the methods (functions) that operate on that data into a single unit, typically a class. It restricts direct access to some of an object's components, which is a means of preventing unintended interference and misuse of the data. This is achieved by making the data private and providing public methods to access and modify that data.

Data Hiding: Encapsulation hides the internal state of objects from the outside world, exposing only what is necessary through a public interface. This prevents external code from making unintended changes.

Improved Security: By controlling access to data, encapsulation helps protect an object's integrity by preventing unintended or harmful interactions.

Modularity: Encapsulation promotes modularity by keeping related data and behavior together, making the codebase easier to manage and understand.

Ease of Maintenance: Changes to encapsulated code can be made with minimal impact on other parts of the program, as long as the public interface remains consistent.

Flexibility and Scalability: Encapsulation allows for more flexible and scalable code, as objects can be modified or extended with minimal impact on other parts of the system.

Implementation Example in Javapublic class BankAccount {

```
// Private data members
private String accountHolderName;
private double balance;

// Constructor
public BankAccount(String name, double initialBalance) {
    this.accountHolderName = name;
    this.balance = initialBalance;
}

// Public getter for accountHolderName
public String getAccountHolderName() {
    return accountHolderName;
}

// Public setter for accountHolderName
```

```
    public void setAccountHolderName(String name) {
        this.accountHolderName = name;
    }

    // Public method to deposit money
    public void deposit(double amount) {
        if (amount > 0) {
            balance += amount;
        }
    }

    // Public method to withdraw money
    public void withdraw(double amount) {
        if (amount > 0 && amount <= balance) {
            balance -= amount;
        }
    }

    // Public getter for balance
    public double getBalance() {
        return balance;
    }
}
```