Java Object Oriented Principles

Java is an object-oriented programming language, meaning it is based on the principles of encapsulation, inheritance, polymorphism, and abstraction.

1. Encapsulation: This refers to the bundling of data with the methods that operate on that data. It restricts direct access to some of the object's components, which is an important principle for security and data integrity.

2. Inheritance: Inheritance allows a class to inherit properties and behaviour from another class, promoting code reusability and the creation of hierarchical relationships between classes.

3. Polymorphism: Polymorphism refers to the ability of different classes to be treated as instances of the same class through a common interface. This allows for flexibility in method implementation and is a key feature of Java's dynamic method dispatch.

4. Abstraction: Abstraction is the concept of simplifying complex systems by modelling classes appropriate to the problem, and working at the most relevant level of detail. In Java, this is often achieved through abstract classes and interfaces.

Encapsulation

Encapsulation in Java is the process of bundling data (attributes) and methods (functions) that operate on the data into a single unit known as a class. This concept is aimed at restricting direct access to some of the object's components, maintaining the integrity of the data.

Example

```
public class Employee {
    private String name;
    private double salary;
    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public double getSalary() {
        return salary;
    }

    public void setSalary(double salary) {
        if (salary > 0) {
            this.salary = salary;
        }
    }
}
```

Inheritance

Inheritance is a fundamental concept in object-oriented programming that allows a class (subclass or derived class) to inherit attributes and methods from another class (superclass or base class). This promotes code reusability and establishes a hierarchical relationship between classes.

There are several types of inheritance:

1. Single Inheritance: In Java, a class can inherit from only one superclass. This is known as single inheritance.

2. Multilevel Inheritance: This occurs when a class is derived from a class that is also derived from another class.

3. Hierarchical Inheritance: In this type of inheritance, more than one derived class inherits from a single base class.

4. Multiple Inheritance: Java does not support multiple inheritance where a class can inherit from more than one class directly.

5. Hybrid Inheritance: This is a combination of two or more types of inheritance. Java supports hybrid inheritance through interfaces, where a class can implement multiple interfaces.

Example

```
// Single Inheritance
class Vehicle {
    void drive() {
        System.out.println("Driving vehicle");
    }
}

class Car extends Vehicle {
    void display() {
        System.out.println("This is a car");
    }
}
```

```
// Multilevel Inheritance
class Animal {
    void eat() {
        System.out.println("Eating");
    }
}

class Dog extends Animal {
    void bark() {
        System.out.println("Barking");
    }
}

class Bulldog extends Dog {
    void guard() {
        System.out.println("Guarding");
    }
}

// Hierarchical Inheritance
class Shape {
    void draw() {
        System.out.println("Drawing shape");
    }
}

class Circle extends Shape {
    void display() {
        System.out.println("Displaying circle");
    }
}

class Square extends Shape {
    void print() {
        System.out.println("Printing square");
    }
}
```

super keyword

The `super` keyword in Java is used to access and call the superclass's methods, constructors, and variables from within a subclass, allowing for explicit invocation of superclass behavior, handling method overriding, and resolving variable name clashes between super and subclass.

Example

```
// Parent class
class Animal {
    String name = "Animal";

    // Constructor of the parent class
    Animal() {
        System.out.println("Animal Constructor Called");
    }

    // Method in the parent class
    void sound() {
        System.out.println("Animal makes a sound");
    }
}

// Child class
class Dog extends Animal {
    String name = "Dog";

    // Constructor of the child class
    Dog() {
        super(); // Calls the parent class constructor
        System.out.println("Dog Constructor Called");
    }

    // Overriding the method in the parent class
    @Override
    void sound() {
        super.sound(); // Calls the parent class method
        System.out.println("Dog barks");
    }

    void printName() {
        System.out.println("Child class name: " + name);
        System.out.println("Parent class name: " + super.name); // Refers to the
parent class variable
    }
}

public class Main {
    public static void main(String[] args) {
        Dog dog = new Dog();
        dog.sound();
        dog.printName();
    }
}
```

super() method

The `super()` method in Java is used to explicitly call a constructor of the superclass from a subclass constructor. It must be the first statement in a subclass constructor if no constructor is explicitly called, ensuring that superclass initialization is performed before any subclass-specific operations. This helps to establish the inheritance hierarchy and initialize superclass state in subclass instances.

Example

```
class Animal {
    Animal() {
        System.out.println("Animal constructor called");
    }
}

class Dog extends Animal {
    Dog() {
        super(); // Calling the constructor of the superclass (Animal)
        System.out.println("Dog constructor called");
    }
}

public class Main {
    public static void main(String[] args) {
        Dog myDog = new Dog();
    }
}
```