**Inheritance**

Inheritance in Java is a fundamental concept in object-oriented programming (OOP) that allows a new class to inherit the properties and behaviors (fields and methods) of an existing class.

The new class, known as the subclass (or derived class or child class), inherits from the existing class, which is referred to as the superclass (or base class or parent class).

This mechanism promotes code reusability, modularity, and hierarchical classification.

**Key Points of Inheritance in Java:**

1. **Syntax**:
   * The keyword **extends** is used to define a subclass.

class Superclass {

// fields and methods

}

class Subclass extends Superclass {

// additional fields and methods

}

1. **Types of Inheritance**:
   * **Single Inheritance**: A subclass inherits from one superclass.
   * **Multilevel Inheritance**: A subclass inherits from another subclass, forming a chain.
   * **Hierarchical Inheritance**: Multiple subclasses inherit from one superclass.
   * **Hybrid Inheritance**: A combination of two or more types of inheritance. However, Java does not support multiple inheritance directly (a class cannot extend more than one class), but it can be achieved through interfaces.
2. **Access Control**:
   * Members of a superclass can be **private**, **protected**, **default** (package-private), or **public**, which controls their visibility in the subclass.
   * **private** members are not inherited.
   * **protected** and **public** members are inherited and can be accessed in the subclass.
   * **default** members are accessible in the subclass if both classes are in the same package.
3. **Overriding Methods**:
   * A subclass can override a method of its superclass to provide a specific implementation. The **@Override** annotation is often used to indicate that a method is being overridden.

class Superclass {

void display() {

System.out.println("Display in Superclass");

}

}

class Subclass extends Superclass {

@Override

void display() {

System.out.println("Display in Subclass");

}

}

1. **Constructors and Inheritance**:
   * Constructors are not inherited, but a subclass can call the constructor of the superclass using the **super** keyword.

class Superclass {

Superclass() {

System.out.println("Superclass constructor");

}

}

class Subclass extends Superclass {

Subclass() {

super(); // calls the superclass constructor

System.out.println("Subclass constructor");

}

}

1. **The super Keyword**:
   * The **super** keyword is used to refer to the immediate superclass object. It can be used to access superclass methods and constructors.

class Superclass {

void display() {

System.out.println("Display in Superclass");

}

}

class Subclass extends Superclass {

void display() { super.display(); // calls the superclass method

System.out.println("Display in Subclass");

}

}

1. **IS-A Relationship**:
   * Inheritance represents an "IS-A" relationship.
   * For example, if **Dog** extends **Animal**, then a **Dog** is an **Animal**.

**Example:**

Here's a practical example to illustrate 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 myDog = new Dog();

myDog.eat(); // Inherited method

myDog.bark(); // Subclass method

}

}

In this example, the **Dog** class inherits the **eat** method from the **Animal** class and also defines its own method **bark**.

**Benefits of Inheritance:**

* **Code Reusability**: Common code is shared among classes, reducing redundancy.
* **Method Overriding**: Subclasses can modify the behavior of superclass methods.
* **Polymorphism**: Objects of different classes can be treated as objects of a common superclass, facilitating flexible and dynamic code.

Inheritance is a powerful feature in Java that enables efficient and effective object-oriented programming.