In Java, **inheritance** is a mechanism where one class acquires the properties (fields) and behaviors (methods) of another class. The class that is inherited from is called the **superclass** (or parent class), and the class that inherits is called the **subclass** (or child class).

Java supports different types of inheritance:

1. Single Inheritance:

In this type, a class inherits from only one superclass.

Example:

java

Copy code

```
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 dog = new Dog();
        dog.eat();  // Inherited method from Animal class
        dog.bark(); // Method of Dog class
    }
}
```

Output:

Copy code

This animal eats food.

The dog barks.

2. Multilevel Inheritance:

In this type, a class is derived from a class which is also derived from another class.

Example:

java

Copy code

```
class Animal {
   void eat() {
      System.out.println("This animal eats food.");
   }
}

class Mammal extends Animal {
   void walk() {
      System.out.println("This mammal walks.");
   }
}

class Dog extends Mammal {
   void bark() {
      System.out.println("The dog barks.");
   }
}

public class Main {
```

```
    public static void main(String[] args) {
        Dog dog = new Dog();
        dog.eat();  // Inherited from Animal class
        dog.walk(); // Inherited from Mammal class
        dog.bark(); // Method of Dog class
    }
}
```

Output:

Copy code

This animal eats food.

This mammal walks.

The dog barks.

3. Hierarchical Inheritance:

In this type, multiple classes inherit from the same superclass.

Example:

java

Copy code

```
class Animal {
    void eat() {
        System.out.println("This animal eats food.");
    }
}

class Dog extends Animal {
    void bark() {
```

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

class Cat extends Animal {
    void meow() {
        System.out.println("The cat meows.");
    }
}

public class Main {
    public static void main(String[] args) {
        Dog dog = new Dog();
        Cat cat = new Cat();

        dog.eat();  // Inherited method from Animal class
        dog.bark(); // Method of Dog class

        cat.eat();  // Inherited method from Animal class
        cat.meow(); // Method of Cat class
    }
}
```

Output:

bash

Copy code

This animal eats food.

The dog barks.

This animal eats food.

The cat meows.

4. Multiple Inheritance (Not Supported by Classes):

Java does not support multiple inheritance with classes (i.e., a class cannot inherit from more than one class directly) to avoid ambiguity. However, Java allows multiple inheritance through interfaces.

Multiple inheritance is needed when a class requires features from multiple sources. While it adds complexity, it enhances code reuse, flexibility, and separation of concerns. In Java, this is efficiently handled through interfaces, allowing a class to implement multiple interfaces without ambiguity, making the system more robust and modular.

The Diamond Problem (Illustrating Multiple Inheritance Issue)

Suppose you have two classes, Addition and Multiplication, both of which have a method calculate() that returns a number. Now, if you want to create a Calculator class that inherits from both Addition and Multiplication, this creates ambiguity (i.e., the diamond problem) because Calculator doesn't know which version of calculate() to inherit.

Hypothetical Example (if multiple inheritance was allowed with classes):

```
class Addition {
    int calculate(int a, int b) {
        return a + b;
    }
}


class Multiplication {
    int calculate(int a, int b) {
        return a * b;
    }
}
```

```
// If Java supported multiple inheritance for classes, it would look like this:
class Calculator extends Addition, Multiplication {
   // Ambiguity: which `calculate` method should be inherited?
}


public class Main {
   public static void main(String[] args) {
      Calculator calc = new Calculator();
      int result = calc.calculate(5, 3);  // Should this be 8 (Addition) or 15 (Multiplication)?
      System.out.println("Result: " + result);
   }
}
```

Problem:

- **The Calculator class has two parent classes (Addition and Multiplication) with a method named calculate().**
- **Java would not know whether to inherit the calculate() method from Addition (which performs addition) or Multiplication (which performs multiplication).**
- **This creates ambiguity because the result could either be 8 (from Addition) or 15 (from Multiplication).**

Java's Solution: Interfaces

Since Java does not support multiple inheritance with classes, we use interfaces to resolve this issue.

Using Interfaces to Avoid Ambiguity:

java

Copy code

```
interface Addition {
   int add(int a, int b);
}


interface Multiplication {
```

```
    int multiply(int a, int b);
}

class Calculator implements Addition, Multiplication {
    public int add(int a, int b) {
        return a + b;
    }

    public int multiply(int a, int b) {
        return a * b;
    }
}

public class Main {
    public static void main(String[] args) {
        Calculator calc = new Calculator();
        int sum = calc.add(5, 3);           // Calls add method
        int product = calc.multiply(5, 3);   // Calls multiply method
        System.out.println("Sum: " + sum);       // Output: 8
        System.out.println("Product: " + product); // Output: 15
    }
}
```

Example Using Interfaces:

```
interface Animal {
    void eat();
}
```

```
interface Mammal {
    void walk();
}

class Dog implements Animal, Mammal {
    public void eat() {
        System.out.println("The dog eats.");
    }

    public void walk() {
        System.out.println("The dog walks.");
    }

    void bark() {
        System.out.println("The dog barks.");
    }
}

public class Main {
    public static void main(String[] args) {
        Dog dog = new Dog();
        dog.eat();  // From Animal interface
        dog.walk(); // From Mammal interface
        dog.bark(); // Method of Dog class
    }
}
```

Output:

Copy code

The dog eats.

The dog walks.

The dog barks.

In summary, Java supports **single inheritance**, **multilevel inheritance**, and **hierarchical inheritance** with classes. **Multiple inheritance** is achieved through interfaces.

5. Hybrid Inheritance:

Hybrid inheritance is a combination of different types of inheritance, such as a mix of single and multiple inheritance. Although Java does not directly support multiple inheritance for classes, it can still be achieved through interfaces.

Example:

java

Copy code

```
interface Animal {
    void eat();
}


class Mammal {
    void walk() {
        System.out.println("This mammal walks.");
    }
}


class Dog extends Mammal implements Animal {
    public void eat() {
        System.out.println("The dog eats.");
    }


    void bark() {
```

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

public class Main {
    public static void main(String[] args) {
        Dog dog = new Dog();
        dog.eat();  // From Animal interface
        dog.walk(); // From Mammal class
        dog.bark(); // Method of Dog class
    }
}
```