Inheritance is a fundamental concept in **Object-Oriented Programming (OOP)** where a new class, called the **subclass** (or **child class**), inherits fields and methods from an existing class, called the **superclass** (or **parent class**).

**Key Concepts of Inheritance in Java**

* **Reusability:** Inheritance allows you to reuse code from an existing class without having to rewrite it.
* **The extends Keyword:** Java uses the keyword extends to indicate that a class is inheriting from another class.
* **'Is-a' Relationship:** Inheritance models an "**is-a**" relationship. For example, a "Dog **is-a** Animal," and a "Car **is-a** Vehicle."

**Java Inheritance Example**

Let's illustrate with an example where a Dog class inherits from an Animal class.

**1. The Superclass (Parent Class)**

The Animal class contains properties and behaviors common to all animals.

Java

// Superclass (Parent Class)

class Animal {

String name;

public Animal(String name) {

this.name = name;

}

public void eat() {

System.out.println(name + " is eating.");

}

public void sleep() {

System.out.println(name + " is sleeping.");

}

}

**2. The Subclass (Child Class)**

The Dog class **extends** Animal, inheriting the name field, and the eat() and sleep() methods. It also adds its own unique behavior, bark().

Java

// Subclass (Child Class)

class Dog extends Animal {

// The Dog class implicitly has the 'name' field, 'eat()' and 'sleep()' methods.

public Dog(String name) {

// Calls the constructor of the superclass (Animal)

super(name);

}

// New method specific to Dog

public void bark() {

System.out.println(name + " is barking.");

}

}

* The **super(name)** call is crucial; it invokes the constructor of the superclass (Animal) to initialize the inherited name field.

**Demonstration**

Here's how you can use these classes:

Java

public class InheritanceDemo {

public static void main(String[] args) {

// Create an object of the subclass (Dog)

Dog myDog = new Dog("Buddy");

// 1. Access inherited field:

System.out.println("My dog's name is: " + myDog.name); // Output: My dog's name is: Buddy

// 2. Call inherited methods from Animal:

myDog.eat(); // Output: Buddy is eating.

myDog.sleep(); // Output: Buddy is sleeping.

// 3. Call its own unique method:

myDog.bark(); // Output: Buddy is barking.

}

}

**Summary of the Example**

| **Feature** | **Animal (Superclass)** | **Dog (Subclass)** | **Description** |
| --- | --- | --- | --- |
| **Field** | name | Inherited | The dog has a name because it's an animal. |
| **Method** | eat() | Inherited | The dog can eat. |
| **Method** | sleep() | Inherited | The dog can sleep. |
| **Method** | N/A | bark() | The dog can bark, which is a unique behavior. |

**Important Notes on Inheritance**

* **Constructor Inheritance:** Constructors are **NOT** inherited. A subclass must call a superclass constructor (often using super()) to initialize the inherited parts of the object.
* **Multiple Inheritance:** Java **does not** support multiple inheritance of classes (a class cannot extend two classes). It supports multiple inheritance of **type** through **interfaces**.
* **final Keyword:** If a class is declared as final, it cannot be inherited. If a method is declared as final, it cannot be overridden by subclasses.