

# Java Inheritance

Inheritance is one of the four core principles of Object-Oriented Programming (OOP), along with Encapsulation, Polymorphism, and Abstraction. In Java, inheritance allows a new class (subclass/child class) to inherit properties and behaviors (fields and methods) from an existing class (superclass/parent class).

## Key Concepts:

1. **Superclass (Parent class):** The class whose properties and methods are inherited by another class.
2. **Subclass (Child class):** The class that inherits from another class. A subclass can add its own properties and methods, as well as override or extend the behavior of methods from the superclass.

## Types of Inheritance in Java

Java supports single inheritance, meaning a subclass can inherit from only one superclass. However, it can still implement multiple interfaces.

1. **Single Inheritance:** A class inherits from one superclass.
  - Java does not support multiple inheritance with classes (i.e., one class cannot directly inherit from multiple classes), though it supports multiple inheritance through interfaces.

Example:

```
class Animal {
    void eat() {
        System.out.println("Animal is eating");
    }
}

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

public class Main {
```

```
public static void main(String[] args) {  
    Dog dog = new Dog();  
    dog.eat(); // Inherited from Animal  
    dog.bark(); // Defined in Dog  
}  
}
```

Output:

csharp

Animal is eating

Dog is barking

2. **Multilevel Inheritance:** A subclass can also serve as a superclass for another subclass.
- A class can inherit from another subclass, which itself inherits from a superclass.

Example:

```
class Animal {  
    void eat() {  
        System.out.println("Animal is eating");  
    }  
}
```

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

```
class Puppy extends Dog {  
    void play() {  
        System.out.println("Puppy is playing");  
    }  
}
```

```
public class Main {  
    public static void main(String[] args) {
```

```
Puppy puppy = new Puppy();
puppy.eat(); // Inherited from Animal
puppy.bark(); // Inherited from Dog
puppy.play(); // Defined in Puppy
    }
}
```

**3. Hierarchical Inheritance:** Multiple classes can inherit from a single superclass.

Example:

```
class Animal {
    void eat() {
        System.out.println("Animal is eating");
    }
}

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

class Cat extends Animal {
    void meow() {
        System.out.println("Cat is meowing");
    }
}

public class Main {
    public static void main(String[] args) {
        Dog dog = new Dog();
        dog.eat(); // Inherited from Animal
        dog.bark(); // Defined in Dog

        Cat cat = new Cat();
        cat.eat(); // Inherited from Animal
        cat.meow(); // Defined in Cat
    }
}
```

```
}  
}
```

## Key Features of Inheritance

1. **Reusability:** Code from the parent class can be reused in the child class.
2. **Method Overriding:** A subclass can modify or "override" the behavior of a method defined in the parent class.
3. **Access to Parent Class Members:** A subclass can access the public and protected members of the parent class, but not private members.

## Method Overriding

Method overriding is when a subclass provides a specific implementation for a method that is already defined in its superclass. The method signature (name, return type, parameters) must be the same.

### Syntax:

```
@Override  
return_type method_name(parameters) {  
    // new implementation  
}
```

Example of 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");
    }
}

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

public class Main {
    public static void main(String[] args) {
        Animal animal = new Animal();
        animal.sound(); // Output: Animal makes a sound

        Dog dog = new Dog();
        dog.sound(); // Output: Dog barks

        Cat cat = new Cat();
        cat.sound(); // Output: Cat meows
    }
}
```

## Important Notes on Method Overriding:

- **@Override** annotation is optional but helps the compiler check for errors (e.g., wrong method signature).
- The overridden method in the subclass should have the same access level or be more permissive than the method in the superclass.

## Constructor Inheritance in Java

- **Constructors are not inherited** by the child class. However, the child class can call the constructor of the parent class using `super()`.
- If the parent class has a no-argument constructor, the subclass can call it implicitly.
- If the parent class has a parameterized constructor, the subclass must explicitly call it using `super()`.

Example:

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

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

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

Output:

```
Animal constructor
Dog constructor
```

## Polymorphism and Inheritance

Polymorphism in Java is closely tied to inheritance. When a subclass object is referred to by a superclass reference, the method invoked is based on the actual object type (i.e., dynamic method dispatch).

Example of polymorphism with inheritance:

```
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 myAnimal = new Animal();
        Animal myDog = new Dog();

        myAnimal.sound(); // Output: Animal makes a sound
        myDog.sound();    // Output: Dog barks
    }
}
```

## Access Modifiers and Inheritance

Access modifiers determine the visibility of members (fields and methods) across classes:

- **Public:** Accessible from anywhere.
- **Protected:** Accessible within the same package or by subclasses.
- **Default (no modifier):** Accessible within the same package.
- **Private:** Not accessible outside the class.

Example:

```
class Animal {
    public String name; // Can be accessed from anywhere
    protected int age; // Can be accessed by subclasses or
    within the same package
    private String breed; // Only accessible within the Animal
    class
}
```

```
class Dog extends Animal {
    void printDetails() {
        System.out.println("Name: " + name); // Public field,
        accessible
        System.out.println("Age: " + age);    // Protected field,
        accessible
        // System.out.println("Breed: " + breed); // Error: breed
        is private
    }
}

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