Introduction to Inheritance in Java

Lecture 07

Inheritance in Java

Definition:

Inheritance is a mechanism in Java where one class (child class) inherits properties (fields) and behaviors (methods) from another class (parent class).

Key Concept:

The child class (or subclass) can reuse methods of the parent class and also add its own methods or properties.

Inheritance in Java

- **Inheritance is probably the most powerful feature of object-oriented programming**

- **Inheritance is the process of creating new classes, called derived classes, from existing or base classes.**

- **The derived class inherits all the capabilities of the base class but can add additions and modifications of its own.**

Inheritance in Java

- Inheritance is an essential part of OOP. A class can be created once and it can be reused again and again to create many sub classes.

- Inheritance saves a lot of time, money and effort to write the same classes again. Reusing existing classes allows the program to work only on new classes

- In some languages the base class is called the superclass and the derived class is called the subclass.

- Some writers also refer to the base class as the parent and the derived class as the child.

Example of Inheritance

Base class	Derived classes
Student	Graduate student, Undergraduate student
Employee	Faculty, Staff
Loan	Car loan, Business loan

Example of Inheritance

Parent Class: Vehicle

- Properties: wheels, engine, fuel
- Method: move()

Child Class: Car (inherits from Vehicle)

- Inherits: wheels, engine, fuel, move()
- Additional property: air_conditioner
- Additional method: play_music()

Example of Inheritance

Parent Class: Appliance

- Properties: power_source, brand
- Method: turn_on(), turn_off()

Child Class: Washing Machine (inherits from Appliance)

- Inherits: power_source, brand, turn_on(), turn_off()
- Additional method: wash_clothes()

Example of Inheritance

Parent Class: Person

- Properties: name, age
- Method: speak()

Child Class: Student (inherits from Person)

- Inherits: name, age, speak()
- Additional property: grade
- Additional method: study()

Example of Inheritance

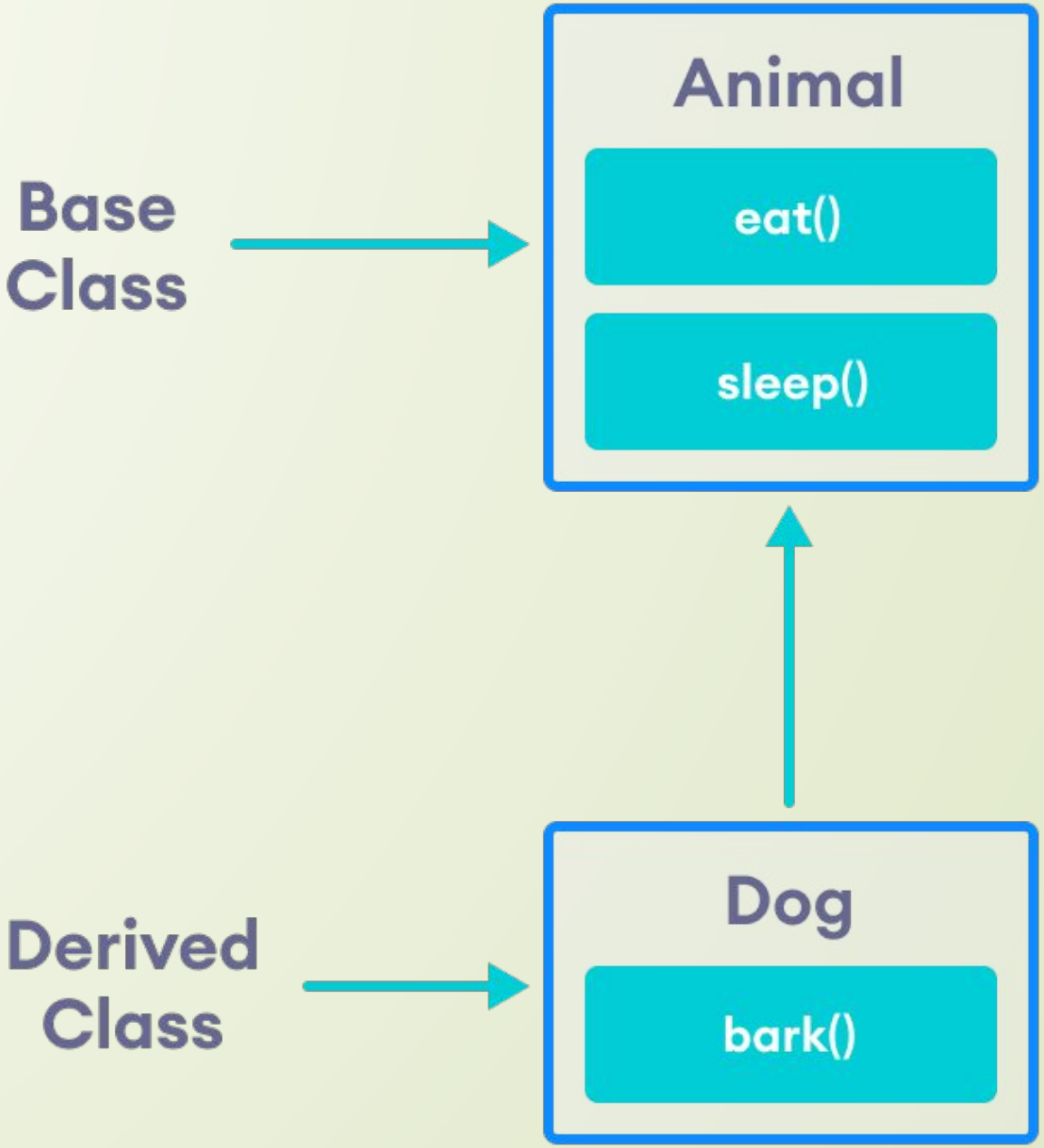
Parent Class: Employee

- Properties: name, salary
- Method: work()

Child Class: Manager (inherits from Employee)

- Inherits: name, salary, work()
- Additional method: manage_team()

Example of Inheritance



Key Benefits of Inheritance

Reusability:

Code written in one class can be reused in other classes.

Extensibility:

Child classes can add new functionality to parent classes.

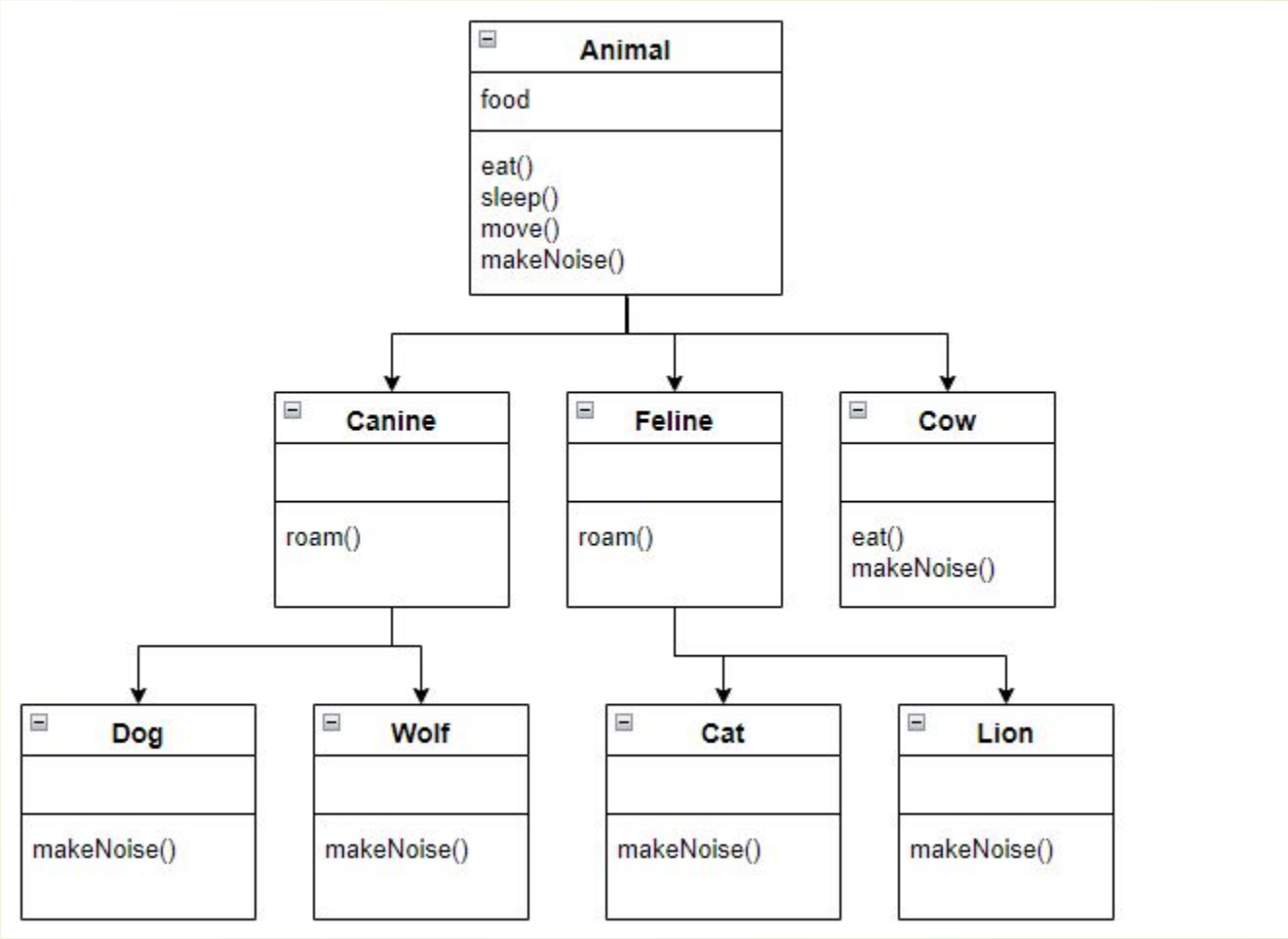
Maintainability:

Simplifies code management by avoiding redundancy.

Hierarchy Representation:

Provides a clear and natural hierarchy between classes.

Key Benefits of Inheritance



Types of Inheritance in Java

1. **Single Inheritance:**

A child class inherits from one parent class.

```
class Parent {
}
class Child extends Parent {
 }
```

Types of Inheritance in Java

2. Multilevel Inheritance:

A class is derived from a child class, which is itself derived from another parent class.

```
class Grandparent {
}
class Parent extends Grandparent {
 }
class Child extends Parent {
 }
```

Types of Inheritance in Java

3. Hierarchical Inheritance

One parent class is inherited by multiple child classes.

```
class Parent {
}
class Child1 extends Parent {
 }
class Child2 extends Parent {
 }
```

Note: **Java does not support multiple inheritance** directly (a class inheriting from multiple classes) due to ambiguity problems.

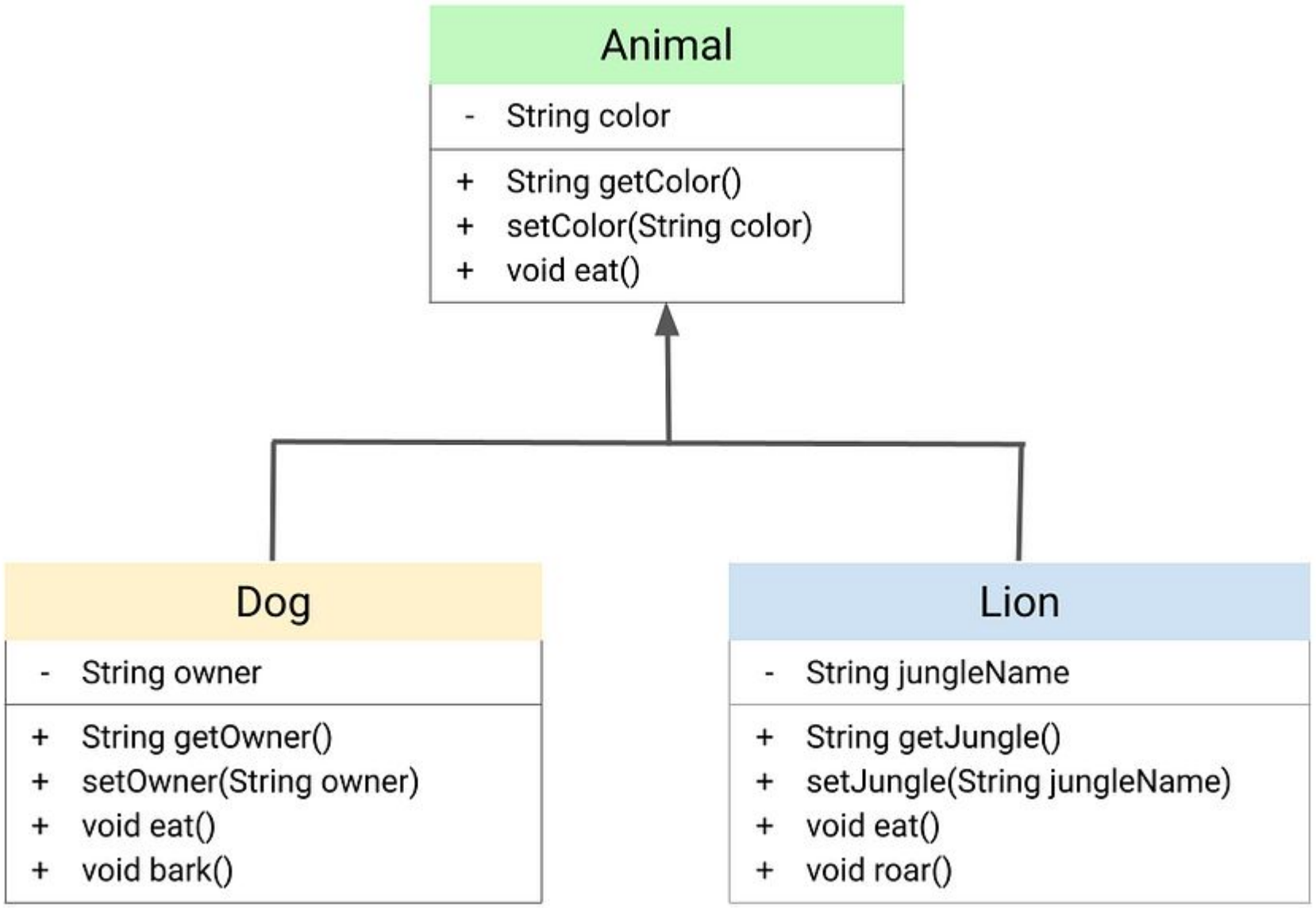
Practical Example of Inheritance (Single Inheritance)

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

// Child class
class Dog extends Animal {
    void bark() {
        System.out.println("The dog barks.");
    }
}
```

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

Practical Example of Inheritance (Single Inheritance)



Method Overriding in Inheritance

The child class can override the parent class's methods to provide a more specific implementation.

```
class Animal {
    void sound() {
        System.out.println("Animal makes a sound.");
    }
}

class Dog extends Animal {
    @Override
    void sound() {
        System.out.println("Dog barks.");
    }
}
```

Access Modifiers in Inheritance

Public:

Members are accessible everywhere.

Protected:

Members are accessible within the package and subclasses.

Private:

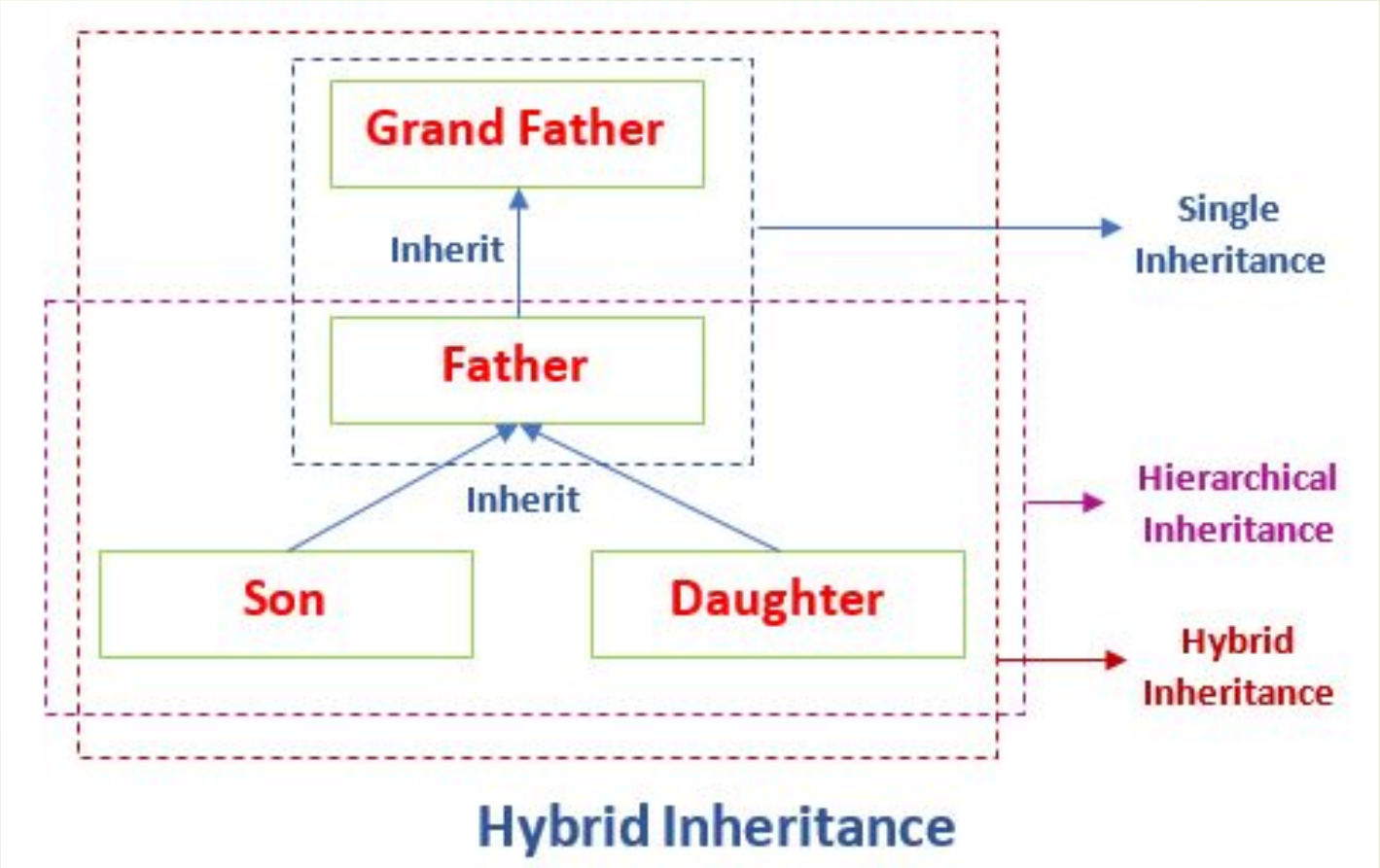
Members are not accessible in the subclass.

Multilevel Inheritance Example

```
class Grandparent {
    void message() {
        System.out.println("I am the grandparent.");
    }
}
class Parent extends Grandparent {
    void message() {
        System.out.println("I am the parent.");
    }
}
class Child extends Parent {
    void message() {
        System.out.println("I am the child.");
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        Child child = new Child();
        child.message(); // Outputs: I am the child.
    }
}
```

Examples


Grand Father
Inherit
Father
Inherit
Son
Daughter
Single Inheritance
Hierarchical Inheritance
Hybrid Inheritance
Hybrid Inheritance

Hierarchical Inheritance Example

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

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

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

Conclusion

- **Inheritance promotes code reuse and better maintainability.**
- **Multiple types of inheritance are supported in Java, except for multiple inheritance (directly).**
- **Overriding methods provides a way for the child class to implement its own version of the parent's behavior.**