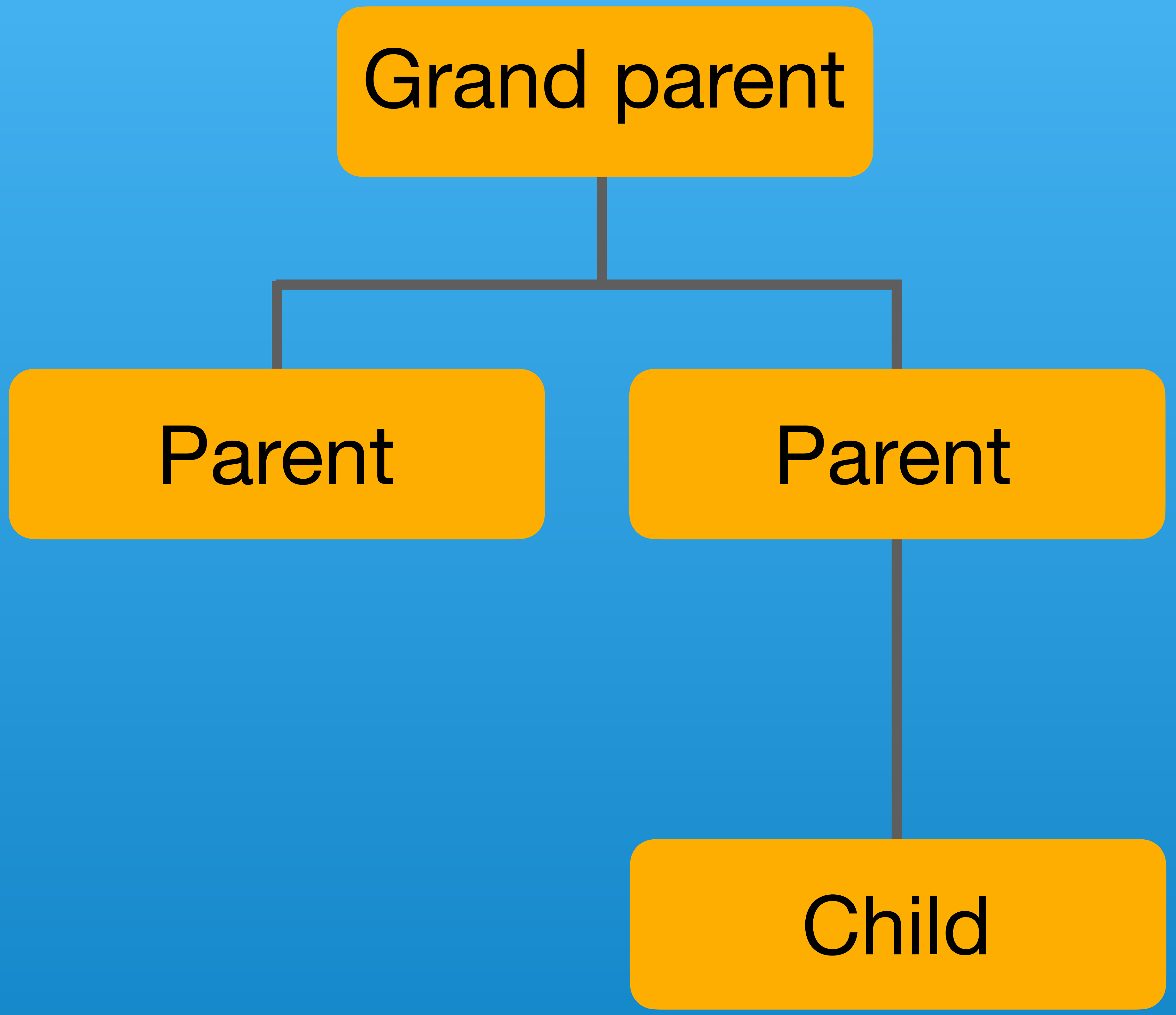
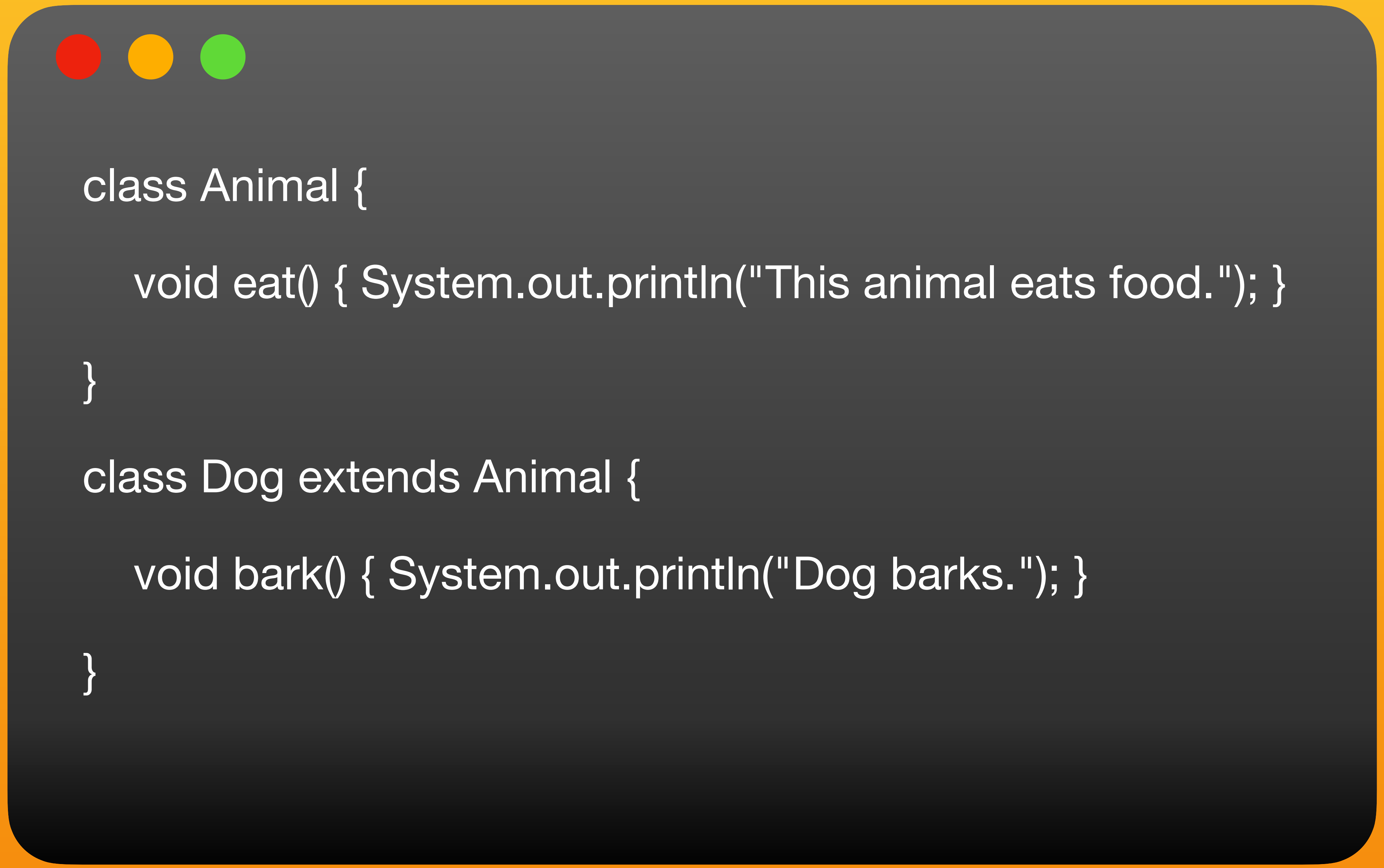
Inheritance and Method Overriding in Java



Lecture Overview

- •Introduction to Inheritance
- •Types of Inheritance in Java
- •Method Overriding
- •Practical Examples and Code Walkthrough



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

What is Inheritance?

- Inheritance allows one class to inherit properties and methods from another class.
- Promotes code reuse and enables the creation of hierarchical relationships.
- Syntax: `class Subclass extends Superclass`

Benefits of Inheritance

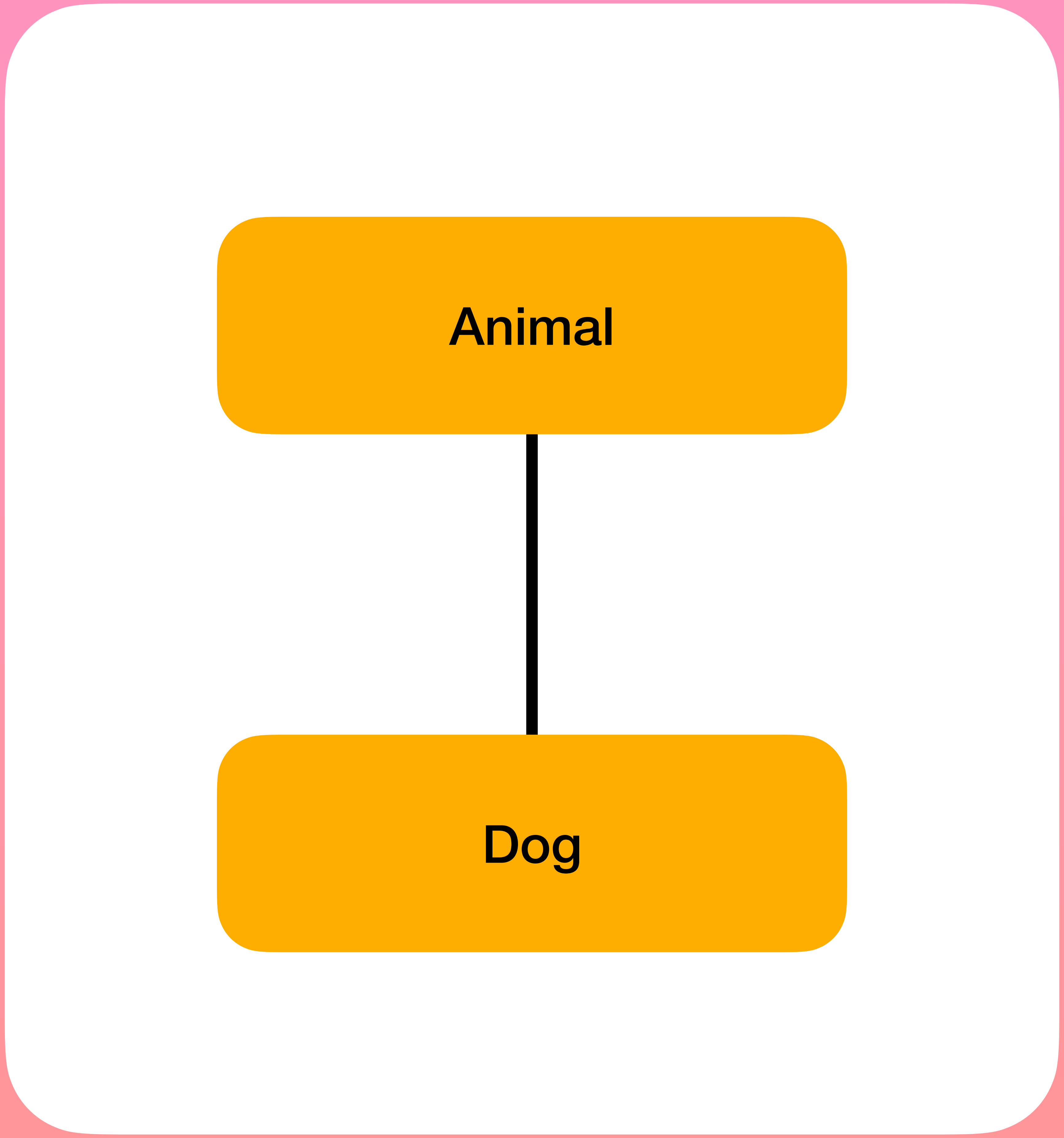
1. Reusability of code.
2. Method overriding allows flexibility.
3. Improved maintainability with hierarchical relationships.
4. Enables polymorphism (dynamic method dispatch).

Inheritance in Java

Single Inheritance

- Definition: When one class inherits from another class.

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

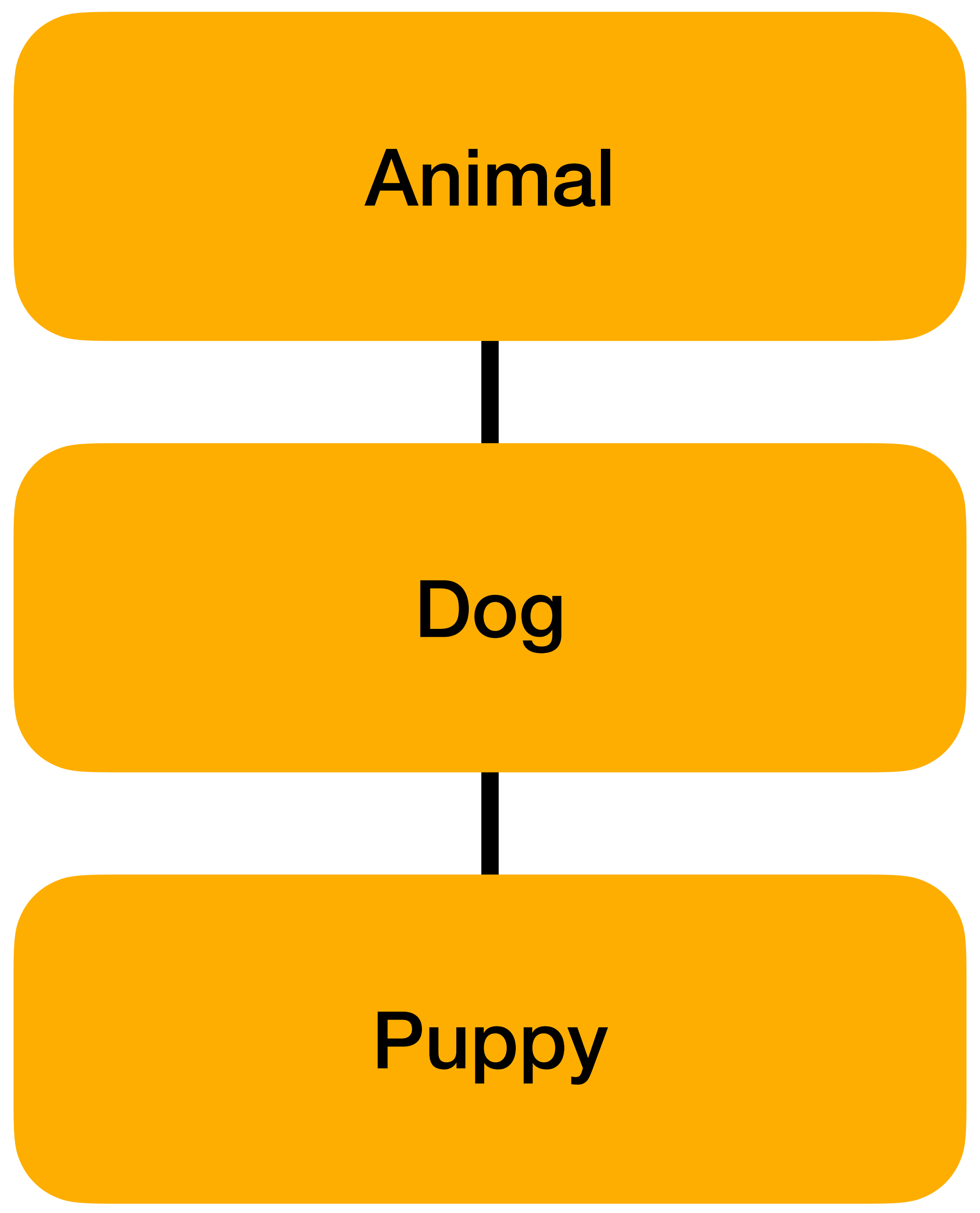


Inheritance in Java

Multilevel Inheritance

- Definition: A chain of inheritance, where a class inherits from a class that is already a subclass.

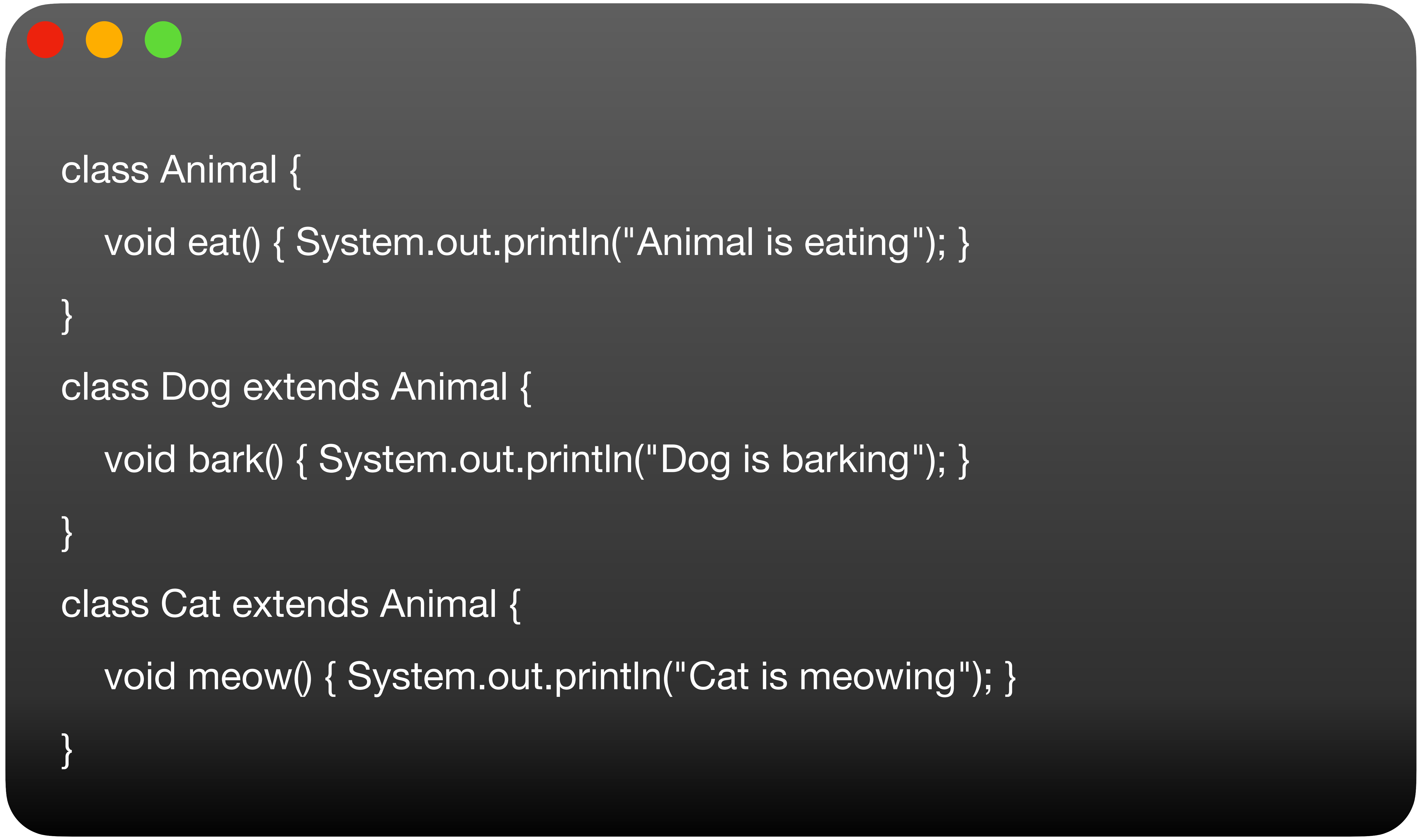
```
class Animal {
    void eat() { System.out.println("Animal is eating"); }
}
class Dog extends Animal {
    void bark() { System.out.println("Dog is barking"); }
}
class Puppy extends Dog {
    void weep() { System.out.println("Puppy is weeping"); }
}
```



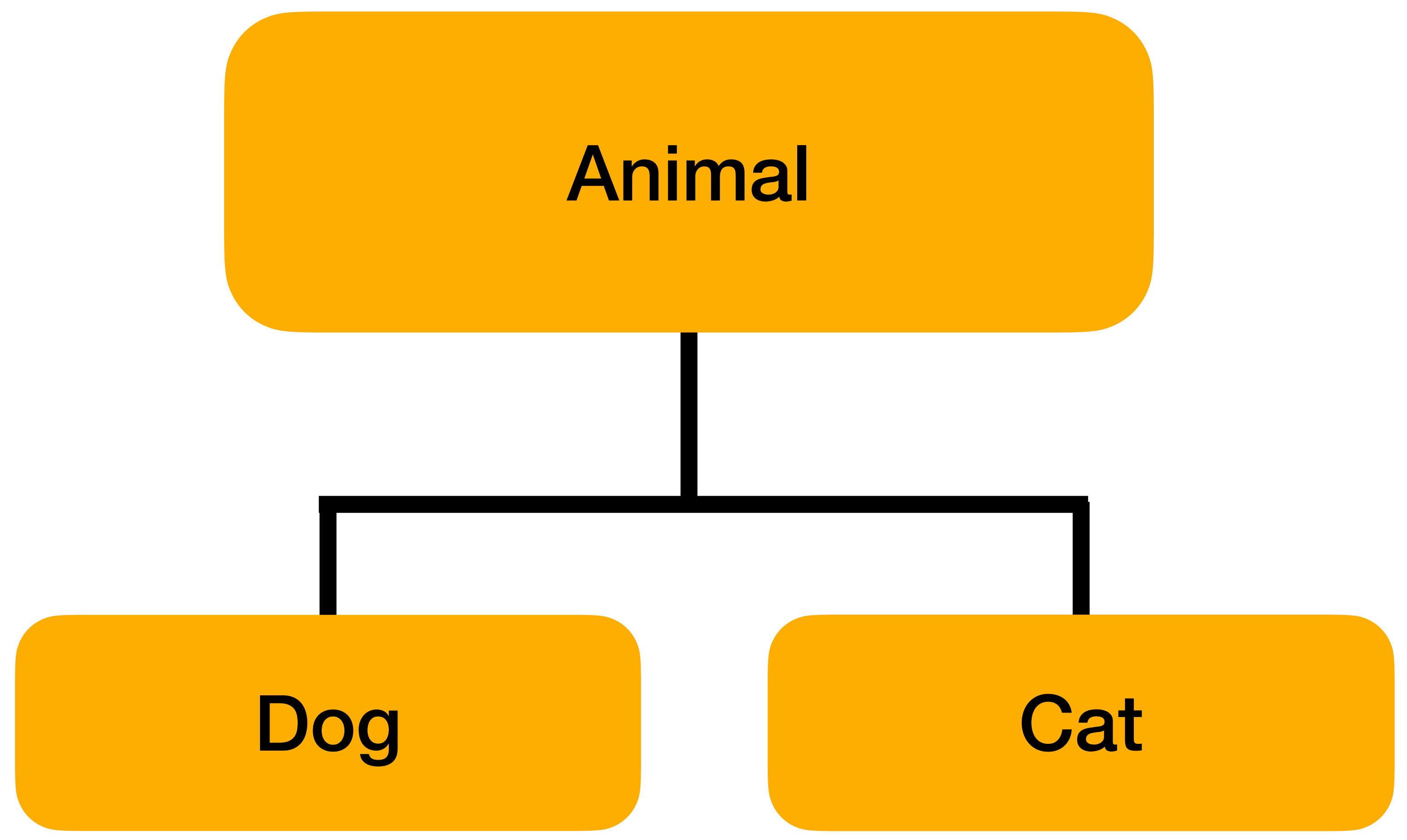
Inheritance in Java

Hierarchical Inheritance

- Definition: When multiple classes inherit from a single superclass.



```
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"); }
}
```

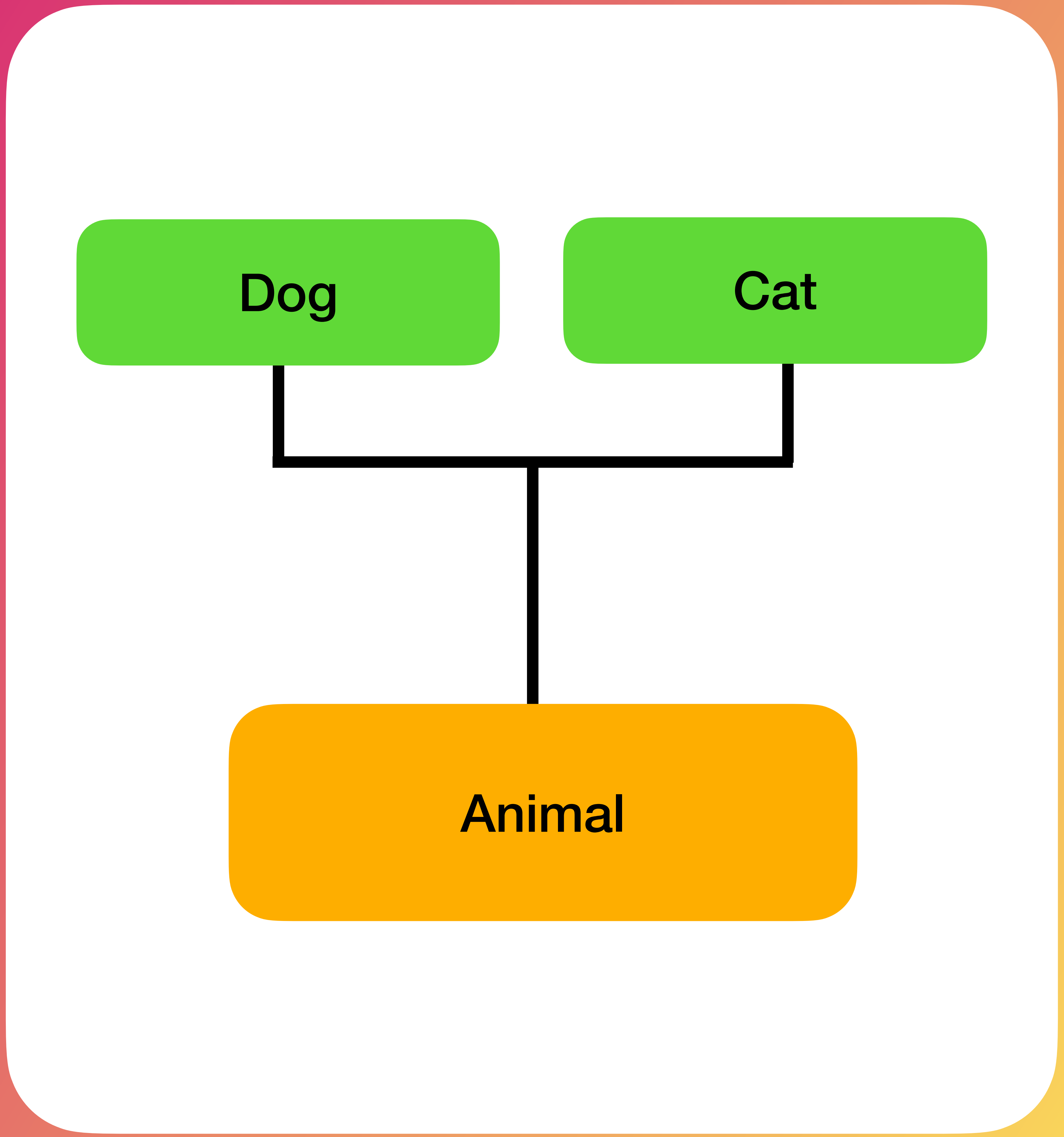


Inheritance in Java

Multiple Inheritance

- Java doesn’t support multiple inheritance with classes to avoid ambiguity.
- Interface-based multiple inheritance:

```
interface CanFly {
    void fly();
}
interface CanSwim {
    void swim();
}
class Bird implements CanFly, CanSwim {
    public void fly() { System.out.println("Bird is flying"); }
    public void swim() { System.out.println("Bird is swimming"); }
}
```



What is Method Overriding?

- When a subclass provides a specific implementation of a method that is already defined in its superclass.
- Signature of the overridden method must match exactly.

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

Rules for Method Overriding

1. The method must have the same name as in the superclass.
2. The return type must be the same (or covariant).
3. The method must not have a lower visibility than the overridden method (e.g., cannot override a `public` method with `private`).
4. Cannot override methods marked as `final` or `static`.

Using `super` Keyword

- `super` is used to call a method or constructor from the superclass.

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

Best Practices in Inheritance and Method Overriding

- Use inheritance only when it logically models an "is-a" relationship.
- Avoid deep inheritance hierarchies, as they can make the code hard to maintain.
- Always use `@Override` annotation to avoid mistakes in overriding.
- Use interfaces and abstract classes to model shared behaviors

Conclusion

- Summarize the importance of inheritance and method overriding in building scalable Java applications.
- Recap types of inheritance and key rules for overriding methods.

Thank You