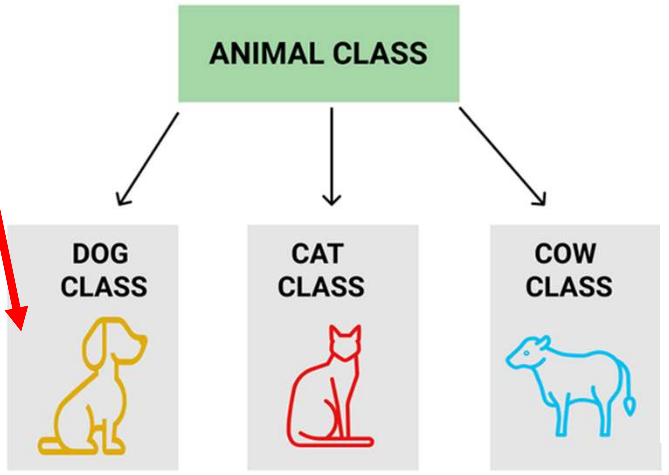
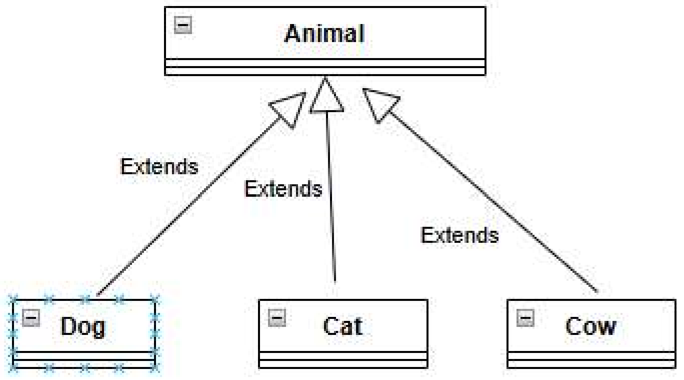
Java Programming Chapter 8

Java Inheritance

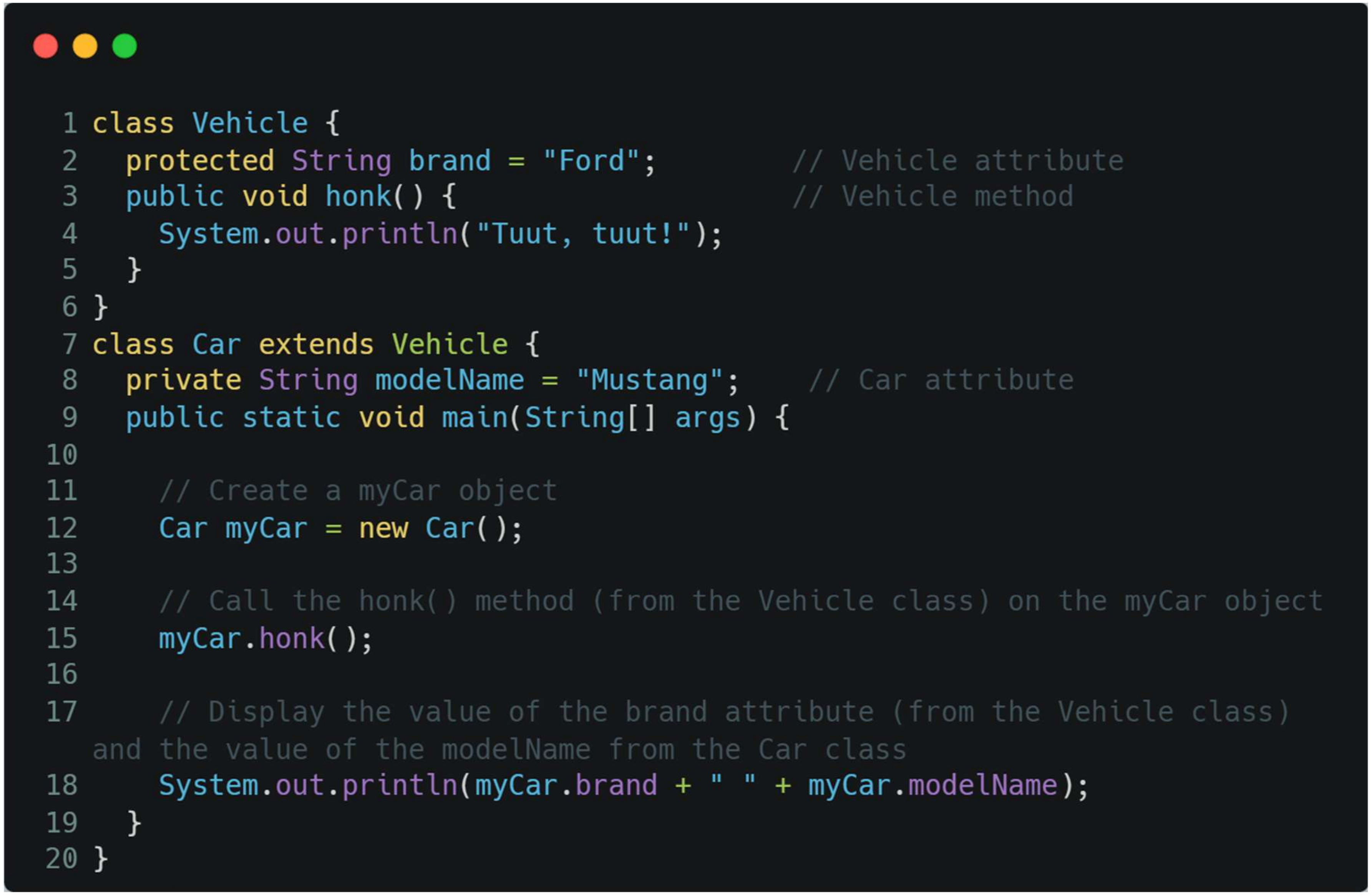
Java Inheritance

- Java Inheritance (Subclass and Superclass)
  - it is possible to inherit attributes and methods from one class to another.
  - We group the "inheritance concept" into two categories
    - **subclass** (child) - the class that inherits from another class
    - **superclass** (parent) - the class being inherited from





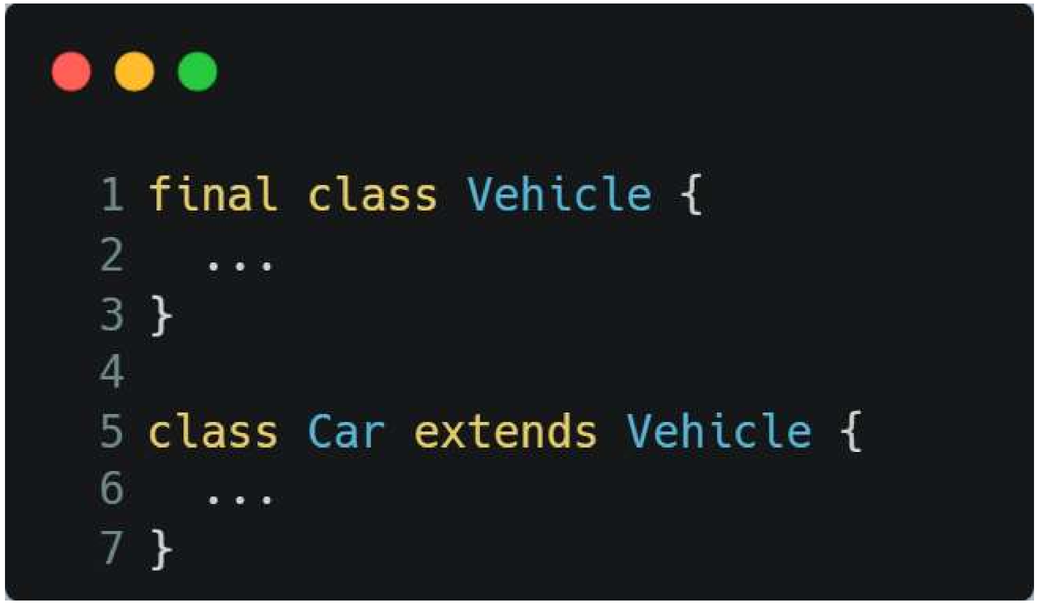
Java Inheritance



```java
class Vehicle {
  protected String brand = "Ford";        // Vehicle attribute
  public void honk() {                    // Vehicle method
    System.out.println("Tuut, tuut!");
  }
}
class Car extends Vehicle {
  private String modelName = "Mustang";    // Car attribute
  public static void main(String[] args) {

    // Create a myCar object
    Car myCar = new Car();

    // Call the honk() method (from the Vehicle class) on the myCar object
    myCar.honk();

    // Display the value of the brand attribute (from the Vehicle class) and the value of the modelName from the Car class
    System.out.println(myCar.brand + " " + myCar.modelName);
  }
}
```

Java Inheritance

- The final Keyword
  - If you don't want other classes to inherit from a class, use the final keyword:

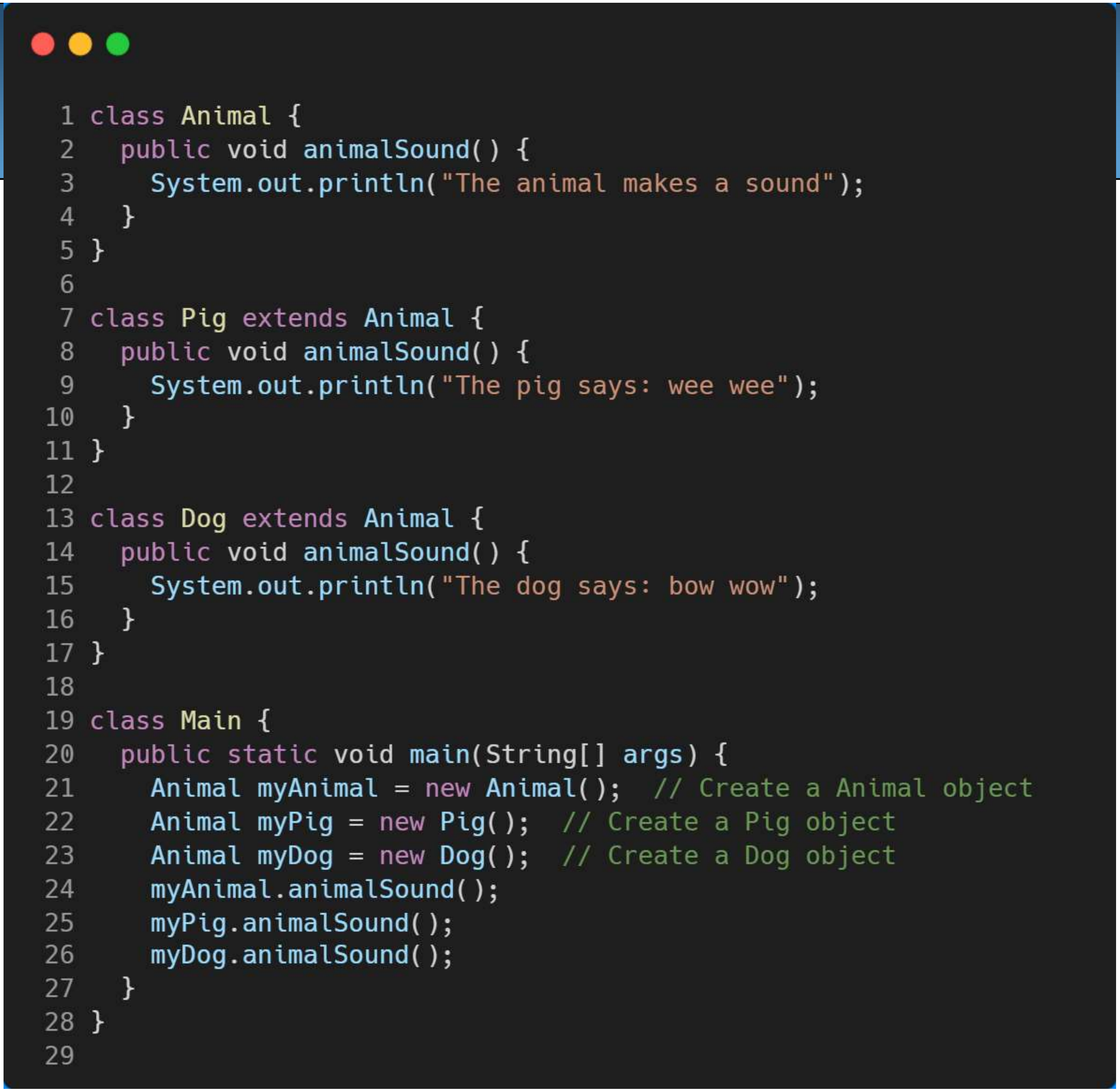


```java
final class Vehicle {
  ...
}

class Car extends Vehicle {
  ...
}
```

Polymorphism

- Polymorphism
  - means "many forms", and it occurs when we have many classes that are related to each other by inheritance.
  - **Polymorphism** uses those methods to perform different tasks. This allows us to perform a single action in different ways.

  - For example, think of a superclass called Animal that has a method called animalSound().
  - Subclasses of Animals could be Pigs, Cats, Dogs, Birds - And they also have their own implementation of an animal sound (the pig oinks, and the cat meows, etc.):



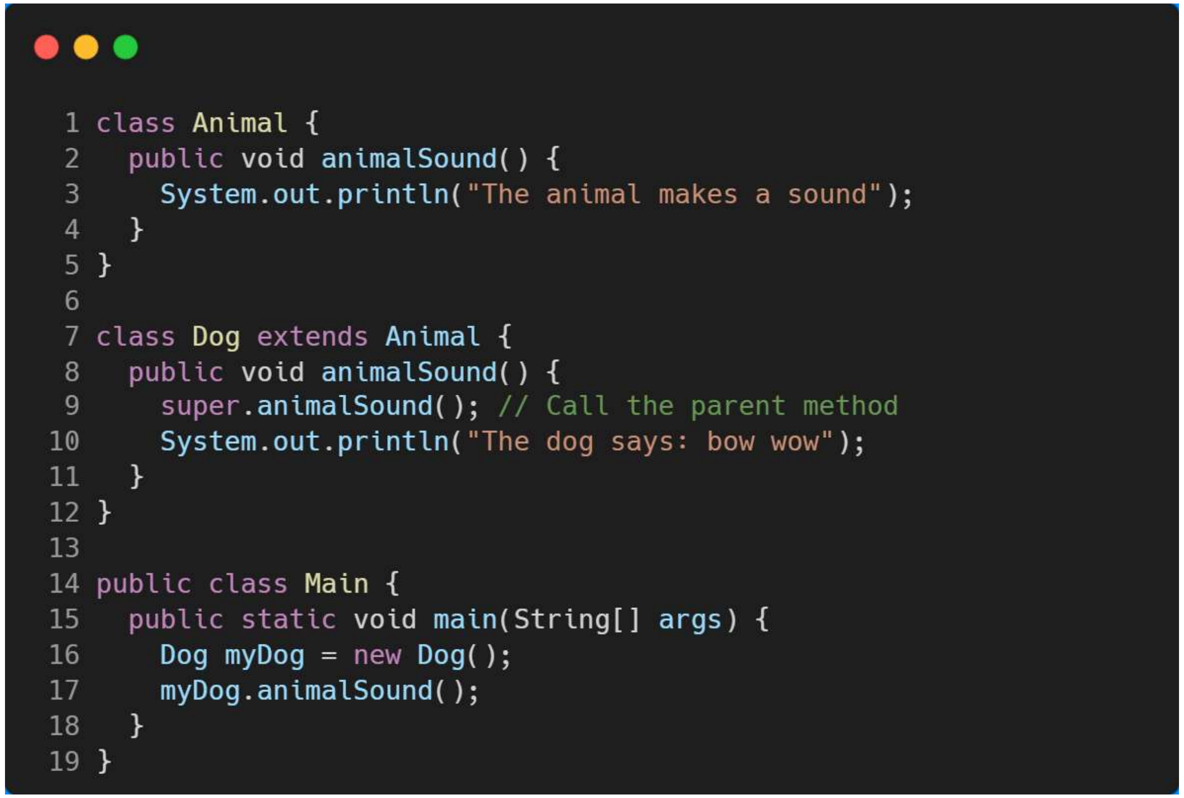
```java
class Animal {
  public void animalSound() {
    System.out.println("The animal makes a sound");
  }
}

class Pig extends Animal {
  public void animalSound() {
    System.out.println("The pig says: wee wee");
  }
}

class Dog extends Animal {
  public void animalSound() {
    System.out.println("The dog says: bow wow");
  }
}

class Main {
  public static void main(String[] args) {
    Animal myAnimal = new Animal();  // Create a Animal object
    Animal myPig = new Pig();  // Create a Pig object
    Animal myDog = new Dog();  // Create a Dog object
    myAnimal.animalSound();
    myPig.animalSound();
    myDog.animalSound();
  }
}

```

Java super

- super Keyword
  - In Java, the super keyword is used to refer to the parent class of a subclass.
  - The most common use of the super keyword is to eliminate the confusion between super classes and subclasses that have methods with the same name.
  - It can be used in two main ways:
    - To access attributes and methods from the parent class
    - To call the parent class constructor

super Keyword

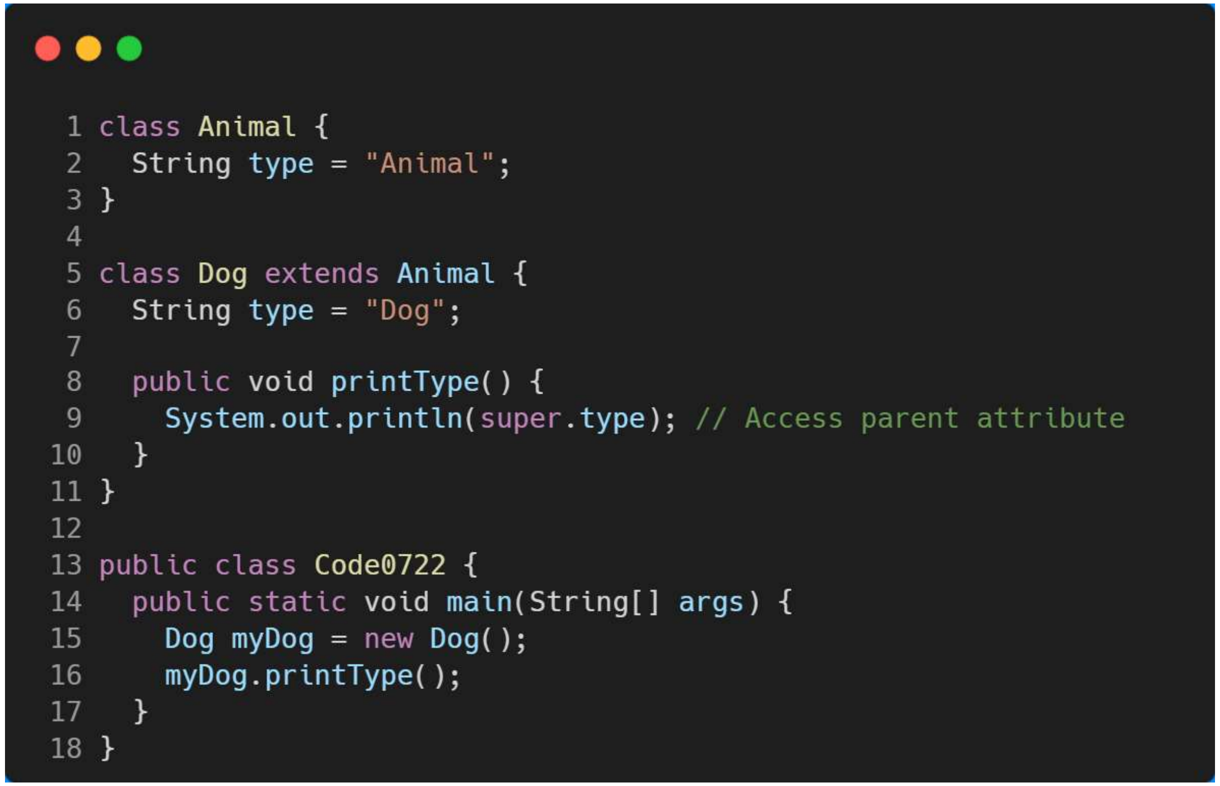
- Access Parent Methods
  - If a subclass has a method with the same name as one in its parent class, you can use super to call the parent version:



```java
class Animal {
  public void animalSound() {
    System.out.println("The animal makes a sound");
  }
}

class Dog extends Animal {
  public void animalSound() {
    super.animalSound(); // Call the parent method
    System.out.println("The dog says: bow wow");
  }
}

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

super Keyword

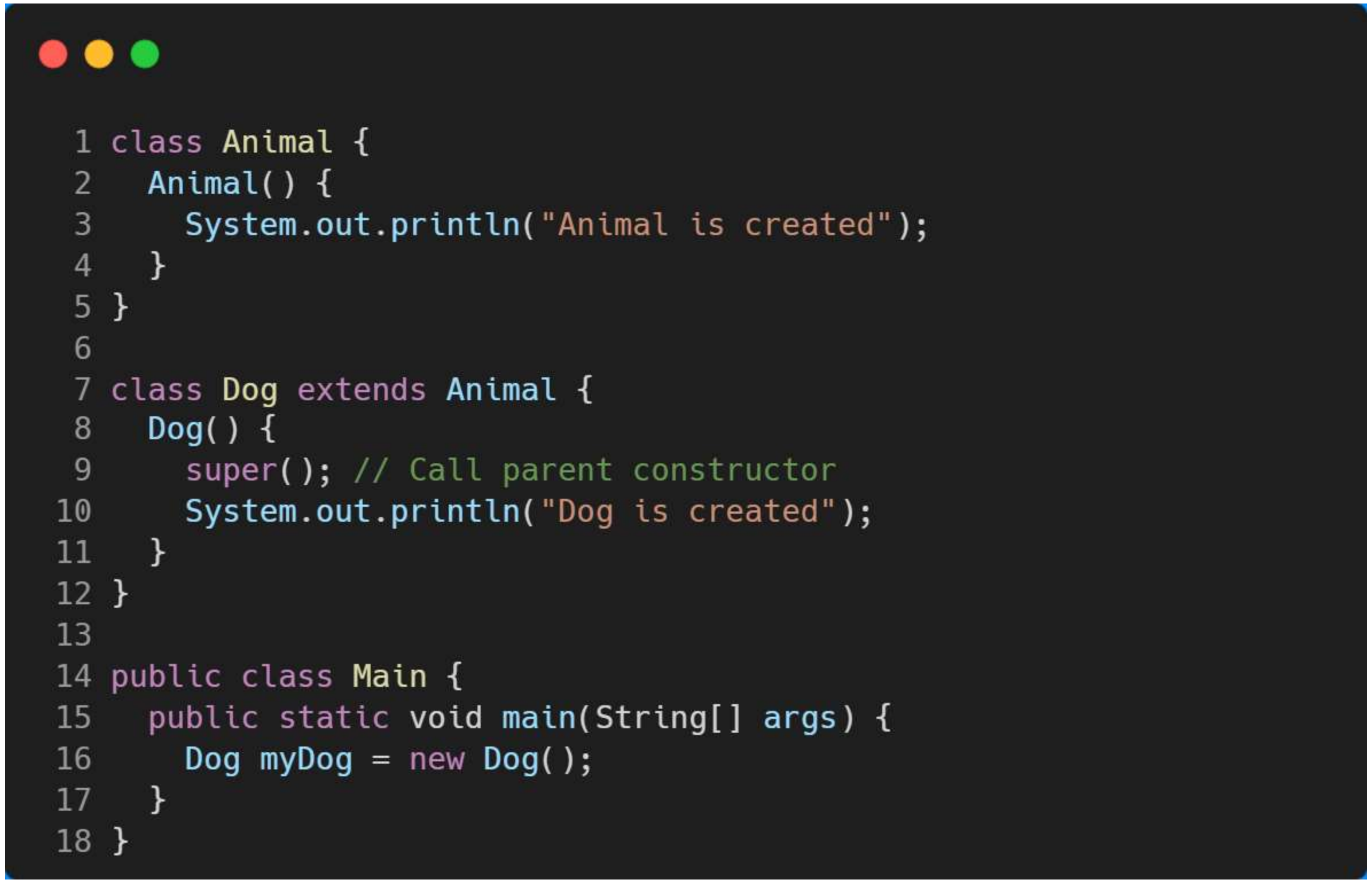
- Access Parent Attributes
  - You can also use super to access an attribute from the parent class if they have an attribute with the same name:



```java
class Animal {
  String type = "Animal";
}

class Dog extends Animal {
  String type = "Dog";

  public void printType() {
    System.out.println(super.type); // Access parent attribute
  }
}

public class Code0722 {
  public static void main(String[] args) {
    Dog myDog = new Dog();
    myDog.printType();
  }
}
```

super Keyword

- Call Parent Constructor
  - Use super() to call the constructor of the parent class. This is especially useful for reusing initialization code.



```java
class Animal {
  Animal() {
    System.out.println("Animal is created");
  }
}

class Dog extends Animal {
  Dog() {
    super(); // Call parent constructor
    System.out.println("Dog is created");
  }
}

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