CS 102

Object Oriented Programming

Inheritance

Reyyan Yeniterzi

reyyan.yeniterzi@ozyegin.edu.tr

Some slides are adapted from the originals available at http://www-cs-faculty.stanford.edu/~eroberts/books/ArtAndScienceOfJava/

Announcements

- Midterm 1 is next week on Tuesday.
- Sample questions and solutions will be uploaded to LMS.

Animal Class

- Assume that we have an animal class
  - Attributes
    - Name
    - Color

Animal Class


```java
public class Animal {
    private String name;
    private String color;

    public void setName(String name) {
        this.name = name;
    }

    public void setColor(String color) {
        this.color = color;
    }

    public String getName() {
        return name;
    }

    public String getColor() {
        return color;
    }

    public String toString() {
        return "Hi, my name is " + name +". I'm " + color + ".";
    }
}
```

Animal Class

- Assume that we have an animal class
  - Attributes
    - Name
    - Color
  - Behaviors
    - Speak

Animal Class

- Assume that we have an animal class
  - Attributes
    - Name
    - Color
  - Behaviors
    - Speak
    - Do they all speak the same way?

Animal Class

- Assume that we have an animal class
  - Attributes
    - Name
    - Color
  - Behaviors
    - Speak
    - Do they all speak the same way?
      - Dogs bark
      - Cats meow
      - Cows moo
      - Ducks quack
      - ....

Animal Class

- Assume that we have an animal class
  - Attributes
    - Name
    - Color
  - Behaviors
    - Speak
    - Do they all speak the same way?
      - Dogs bark
      - Cats meow
      - Cows moo
      - Ducks quack
      - ....

How do we implement them?

Different type of animals

- We will talk about two bad solutions:
  - First way: Inside the same class
  - Second way: Writing a different class

First Way – Inside the same class

- Need to hold an additional attribute:
  - Type of the animal

```
public class Animal {
    private String name;
    private String color;
    private String type;

    public Animal (String name, String type) {
        this.name = name;
        this.type = type;
    }
```

First Way – Inside the same class

- Need to hold an additional attribute:
  - Type of the animal

- Main:

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

    Animal cat = new Animal("Serafettin", "cat");
    Animal dog = new Animal("Scooby", "dog");
    Animal cow = new Animal("Sarı Kız", "cow");
```

First Way – Inside the same class

- Speak function
  - Check the type of the animal and speak accordingly

```
public String speak() {
    if (type.compareTo("dog") == 0)
        return "Woof!";
    else if (type.compareTo("cat") == 0)
        return "Miyauv!";
    else if (type.compareTo("cow") == 0)
        return "Mooo!";
    else
        return "Some Noise";
}
```

First Way – Inside the same class

- Main:

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

    Animal cat = new Animal("Serafettin", "cat");
    Animal dog = new Animal("Scooby", "dog");
    Animal cow = new Animal("Sarı Kız", "cow");

    System.out.println(cat.speak());
    System.out.println(dog.speak());
    System.out.println(cow.speak());
}
```

```
Miyauv!
Woof!
Mooo!
```

First Way – Problem 1

- Many dog types, all bark differently
  - Loudness
  - Pace
  - …
- Do we need to define another dog type variable?

```
private String dogType;
```

Source: https://www.popchartlab.com/products/the-diagram-of-dogs

First Way – Problem 1

- Many dog types, all bark differently
  - Loudness
  - Pace
  - …

```
private String dogType;
```

```
public String speak() {
    if (type.compareTo("dog") == 0) {
        if (dogType.compareTo("kangal dog") == 0)
            return "Loud Woof!";
        if (dogType.compareTo("chow dog") == 0)
            return "Cute Woof!";
    }
    else if (type.compareTo("cat") == 0)
        return "Miyauv!";
    else if (type.compareTo("cow") == 0)
        return "Mooo!";
    else
        return "Some Noise";
}
```

Source: https://www.popchartlab.com/products/the-diagram-of-dogs

First Way – Problem 1

- Many dog types, all bark differently
  - Loudness
  - Pace
  - …

```
private String dogType;
```

```
public String speak() {
    if (type.compareTo("dog") == 0) {
        if (dogType.compareTo("kangal dog") == 0)
            return "Loud Woof!";
        if (dogType.compareTo("chow dog") == 0)
            return "Cute Woof!";
    }
    else if (type.compareTo("cat") == 0)
        return "Miyauv!";
    else if (type.compareTo("cow") == 0)
        return "Mooo!";
    else
        return "Some Noise";
}
```

Ugly Code ☹

Source: https://www.popchartlab.com/products/the-diagram-of-dogs

First Way – Problem 2

- Jumping is behavior of some animals

```
public void jump() {
    System.out.println(this.name + "jumped!");
}
```

First Way – Problem 2

- Jumping is behavior of some animals

```
public void jump() {
    System.out.println(this.name + "jumped!");
}
```

- cat, dog can jump but not the fish…

First Way – Problem 2

- Jumping is behavior of some animals

```
public void jump() {
    System.out.println(this.name + "jumped!");
}
```

- cat, dog can jump but not the fish…
- In main, we should not call jump for fish, but right now we can as follows:

```
Animal fish = new Animal("Nemo", "fish");
fish.jump();
```

Different type of animals

- We will talk about two bad solutions:
  - First way: Inside the same class ✗
  - Second way: Writing a different class

Second way: Class for each type

- We will have individual classes for each animal.

Second way: Class for each type

Cat

```java
public class Cat {
    private String name;
    private String color;

    public Cat (String name) {
        this.name = name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public void setColor(String color) {
        this.color = color;
    }
    public String getName() {
        return name;
    }
    public String getColor() {
        return color;
    }
    public String speak() {
        return "Miyauv";
    }
}
```

Dog

```java
public class Dog {
    private String name;
    private String color;

    public Dog (String name) {
        this.name = name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public void setColor(String color) {
        this.color = color;
    }
    public String getName() {
        return name;
    }
    public String getColor() {
        return color;
    }
    public String speak() {
        return "Woof";
    }
}
```

Second way: Class for each type


```java
public class Cat {
    private String name;
    private String color;

    public Cat (String name) {
        this.name = name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public void setColor(String color) {
        this.color = color;
    }
    public String getName() {
        return name;
    }
    public String getColor() {
        return color;
    }
    public String speak() {
        return "Miyauv";
    }
}
```

```java
public class Dog {
    private String name;
    private String color;

    public Dog (String name) {
        this.name = name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public void setColor(String color) {
        this.color = color;
    }
    public String getName() {
        return name;
    }
    public String getColor() {
        return color;
    }
    public String speak() {
        return "Woof";
    }
}
```

Second way: Class for each type


```java
public class Cat {
    private String name;
    private String color;

    public
        th
    }
    public void setName(String name) {
        this.name = name;
    }
    public void setColor(String color) {
        this.color = color;
    }
    public String getName() {
        return name;
    }
    public String getColor() {
        return color;
    }
    public String speak() {
        return "Miyauv";
    }
}
```

```java
public class Dog {
    private String name;
    private String color;

    public void setName(String name) {
        this.name = name;
    }
    public void setColor(String color) {
        this.color = color;
    }
    public String getName() {
        return name;
    }
    public String getColor() {
        return color;
    }
    public String speak() {
        return "Woof";
    }
}
```

Code repetition ☹.

It is hard to keep the common code consistent

Different type of animals

- We will talk about two bad solutions:
  - First way: Inside the same class ✗
  - Second way: Writing a different class ✗

Different type of animals

- We will talk about two bad solutions:
  - First way: Inside the same class ✗
  - Second way: Writing a different class ✗

- A good solution is to use inheritance ✓
  - Keep the common attributes and functionalities in one class
  - Split only the different attributes and functionalities in different classes.

Inheritance

- A class can inherit some of its attibutes and behaviors from another class.

Inheritance

- A class can inherit some of its attibutes and behaviors from another class.
- A **derived** class inherits from the **base** class.
- A **sub** class inherits from the **super** class.

Inheritance

- A class can inherit some of its attibutes and behaviors from another class.
- A **derived** class **inherits from** the **base** class.
- A **sub** class **extends** the **super** class.

Inheritance

- Keep the common attributes and functionalities in one class
  - Animal class
    - name and color
    - setter and getter functions
- Split only the different attributes and functionalities in different classes.
  - Cat, dog, cow ... classes
    - speak, jump function

Animal Class


```java
public class Animal {

    private String name;
    private String color;

    public void setName(String name) {
        this.name = name;
    }
    public void setColor(String color) {
        this.color = color;
    }
    public String getName() {
        return name;
    }
    public String getColor() {
        return color;
    }
    public String toString() {
        return "Hi, my name is " + name +". I'm " + color + ".";
    }
}
```

Cat and Dog Classes


```
public class Cat extends Animal {
    public Cat(String name) {
        setName(name);
        setColor("gray");
    }
}
```

```
public class Dog extends Animal {
    public Dog(String name) {
        setName(name);
        setColor("gray");
    }
}
```

Cat and Dog Classes

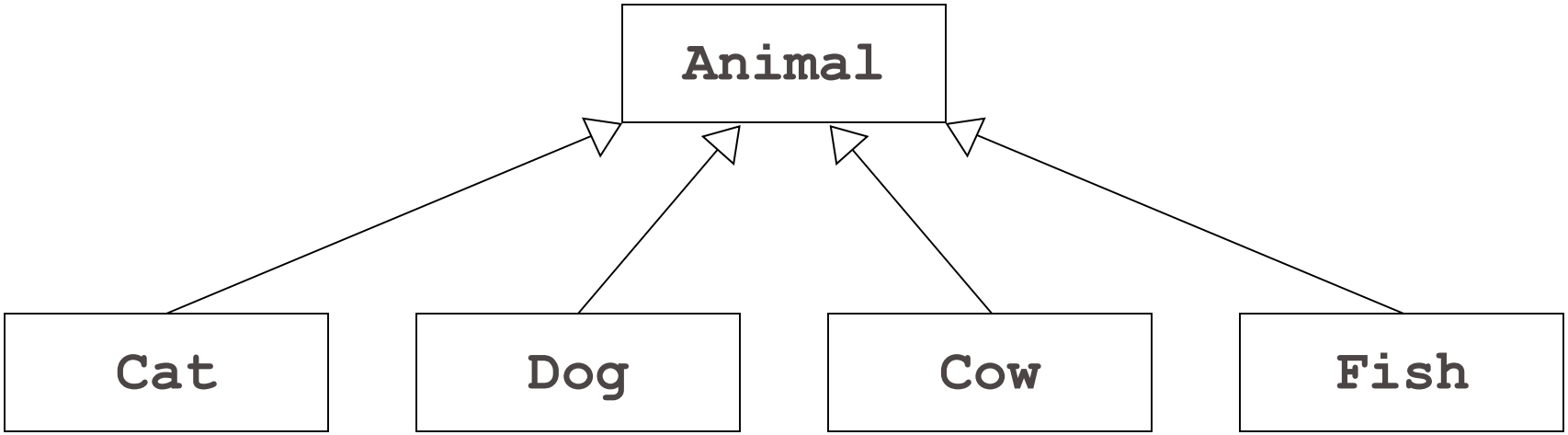

```
public class Cat extends Animal {
    public Cat(String name) {
        setName(name);
        setColor("gray");
    }
}
```

inherits from

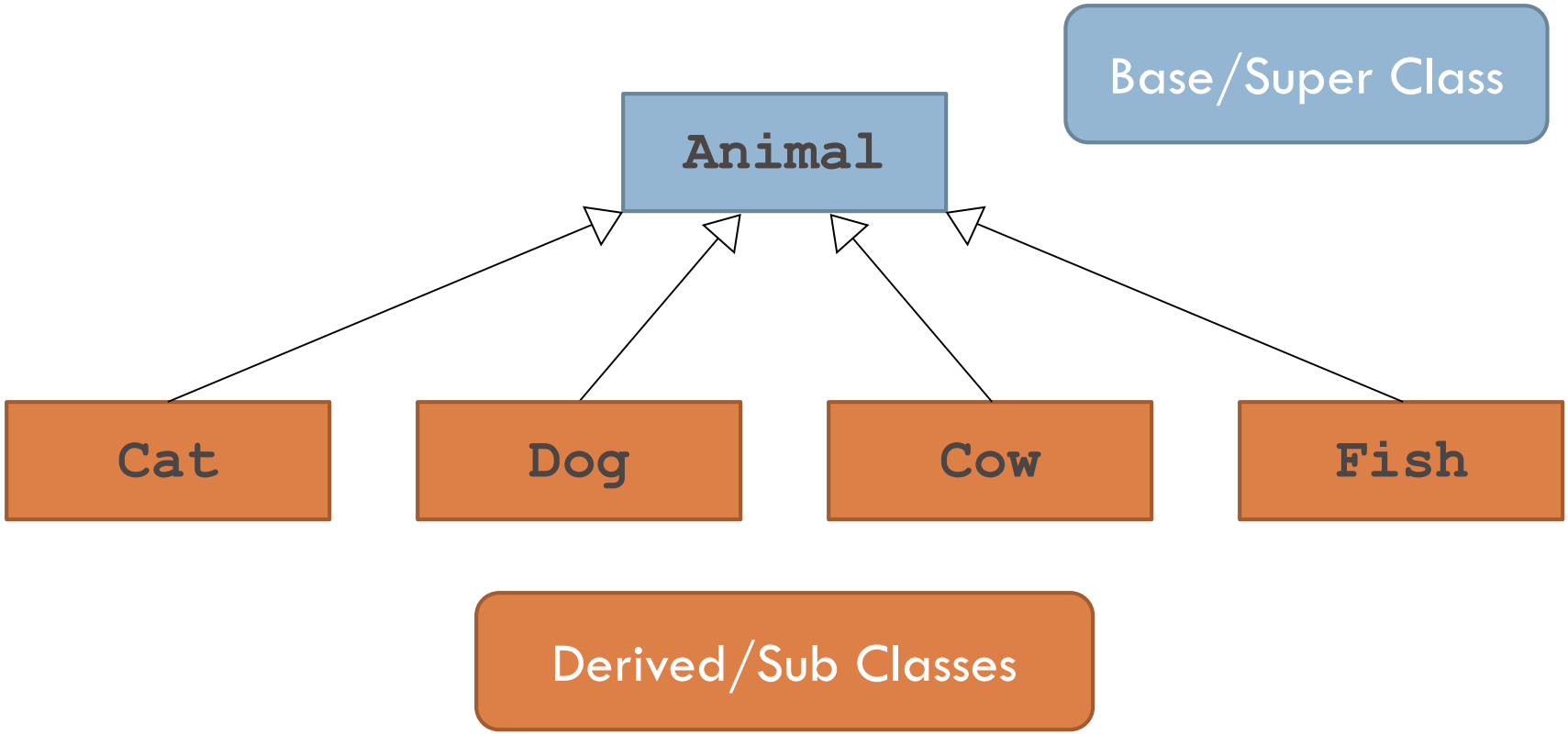
```
public class Dog extends Animal {
    public Dog(String name) {
        setName(name);
        setColor("gray");
    }
}
```

Class Hierarchy

Classes in Java form **hierarchies.**

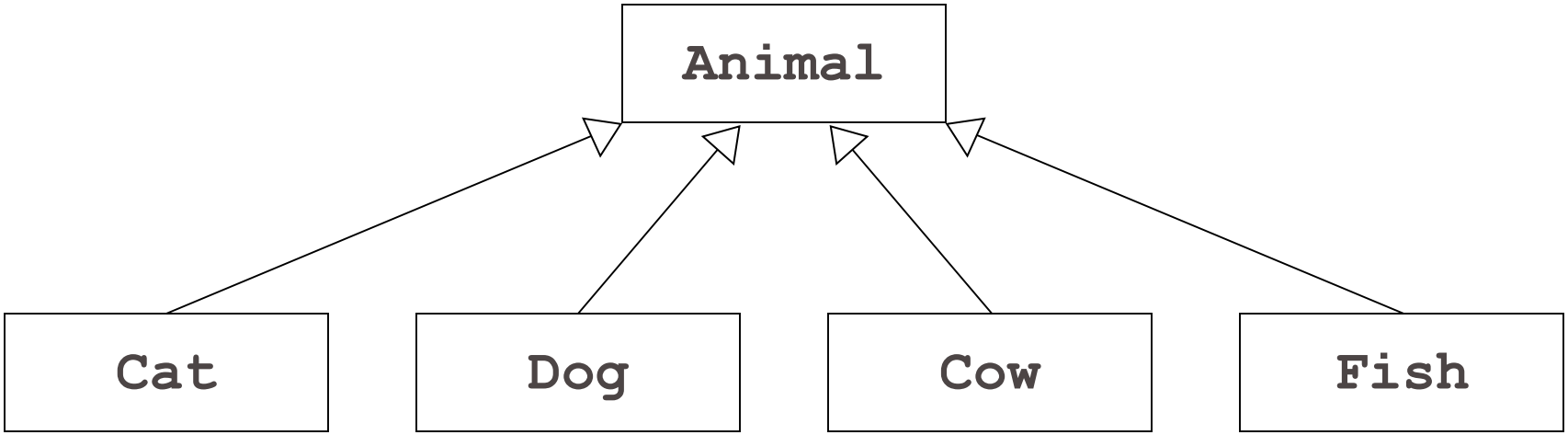


Class Hierarchy



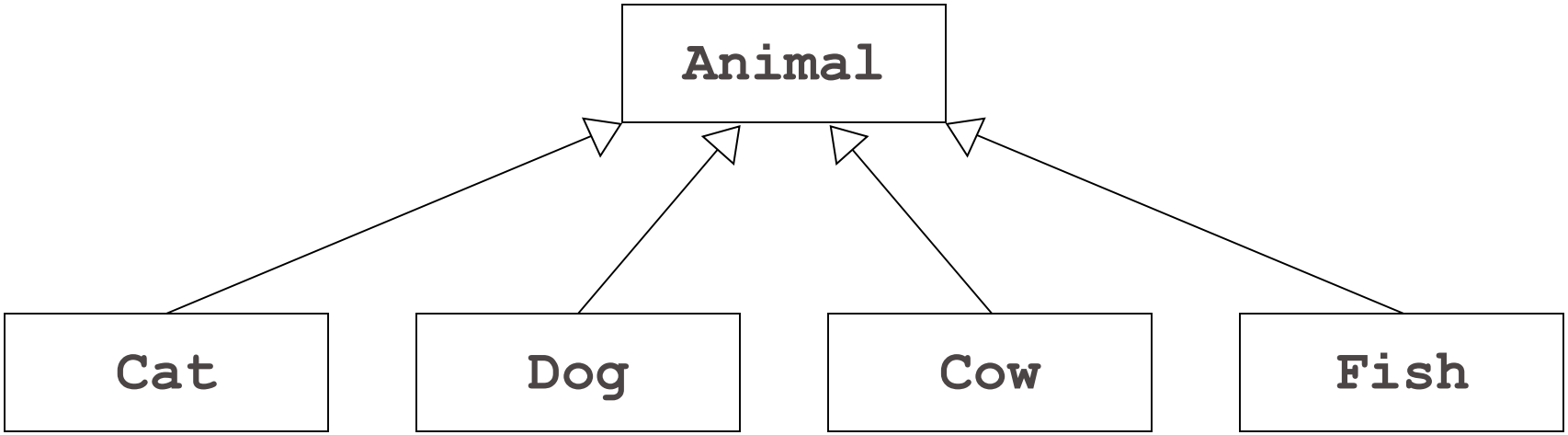
Class Hierarchy

Animal class represent all animal objects.



The four subclasses correspond to particular animal type object.

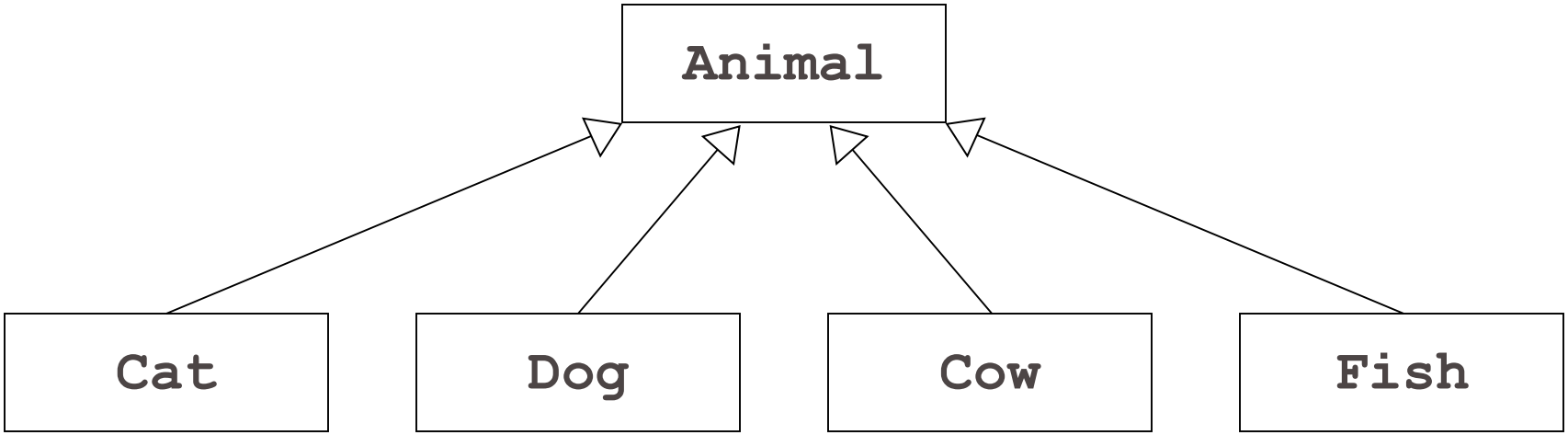
Class Hierarchy



This class diagram shows that Cat, Dog, Cow and Fish is also an Animal but the inverse is NOT TRUE. Any animal is not a Cat, any Animal is not a Dog, etc.

Class Hierarchy

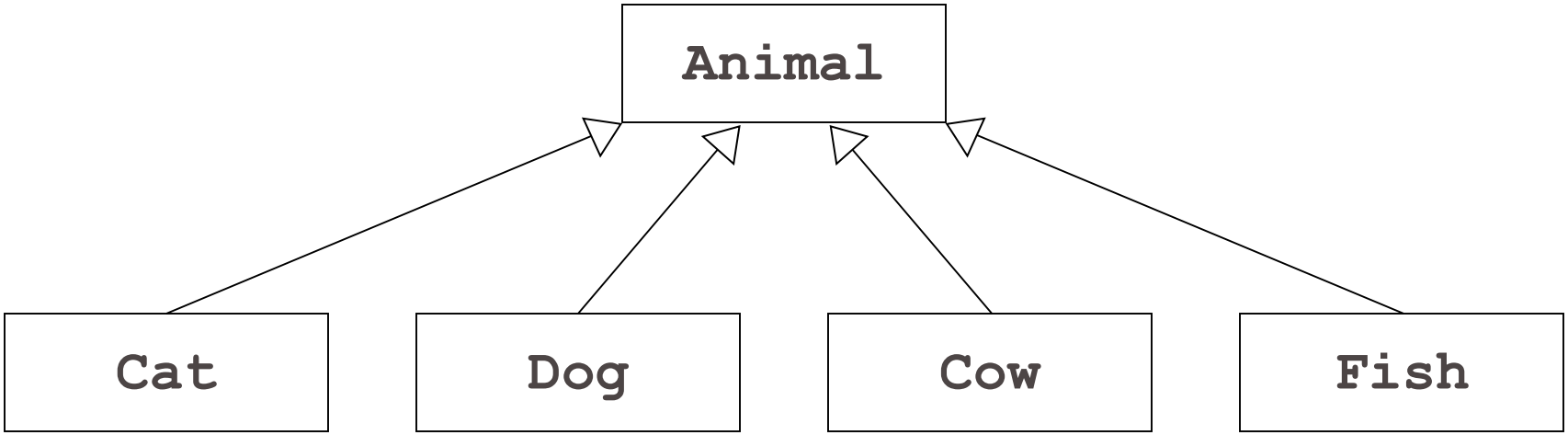
Can the Animal class be also a derived class?



When you define a new class in Java, that class automatically **inherits** the behavior of its superclass.

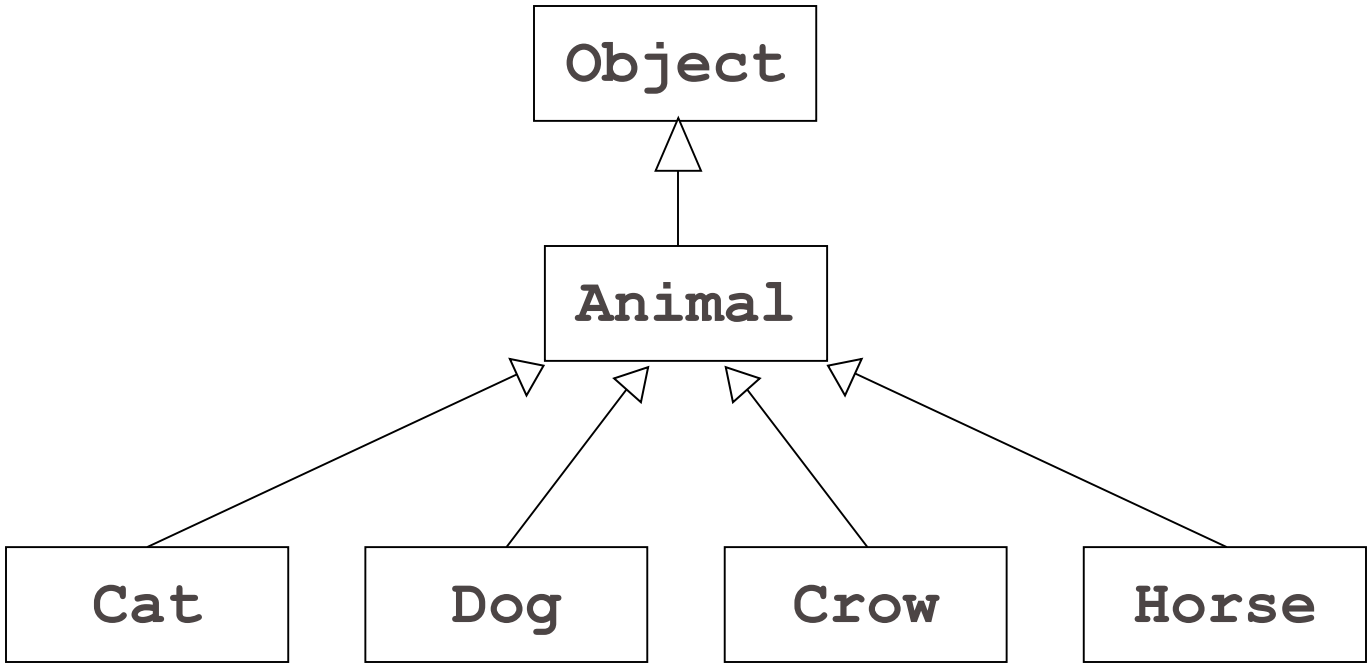
Class Hierarchy

Can the Animal class be also a derived class?



If no superclass is defined, by default, the class will inherit from the **Object** class.

Class Hierarchy



Extending from Object Class


```
public class Animal { ...
```

- is the same as

```
public class Animal extends Object { ...
```

Extending from Object Class

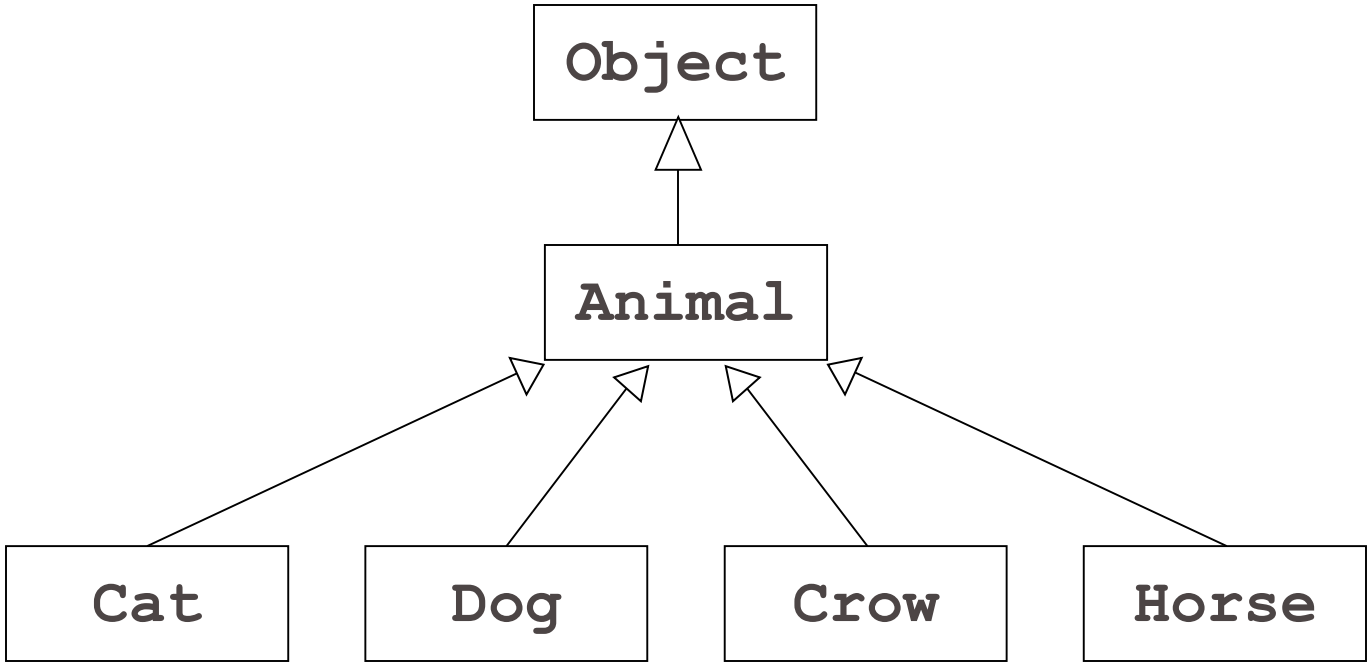

```
public class Animal { ...
```

- is the same as

```
public class Animal extends Object { ...
```

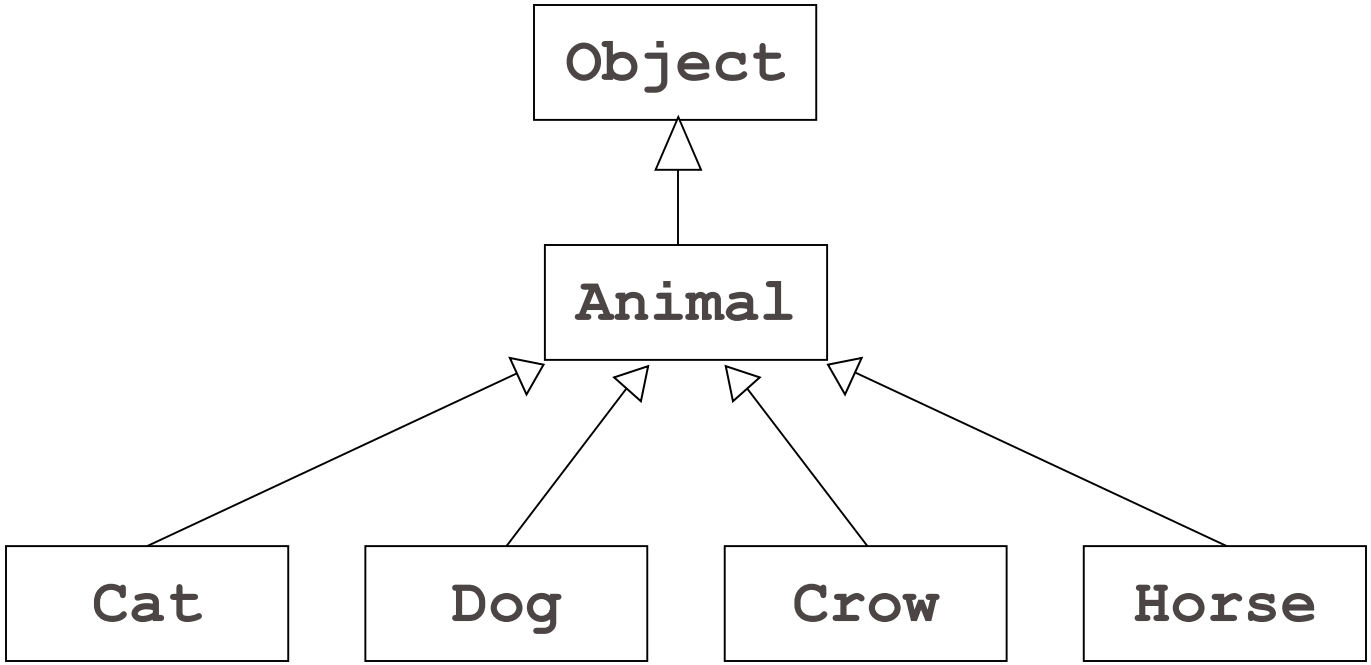
- The **extends** clause on the header line specifies the name of the superclass. If the **extends** clause is missing, the new class becomes a direct subclass of **Object**, which is the root of Java's class hierarchy.

Class Hierarchy



- Except for the class named **Object** that stands at the top of the hierarchy, every class in Java is a **subclass** of some other class.

Class Hierarchy



- A class can have many subclasses, but each class has only one superclass.

Lets get back to our Animal class


```java
public class Animal {

    private String name;
    private String color;

    public void setName(String name) {
        this.name = name;
    }
    public void setColor(String color) {
        this.color = color;
    }
    public String getName() {
        return name;
    }
    public String getColor() {
        return color;
    }
    public String toString() {
        return "Hi, my name is " + name +". I'm " + color + ".";
    }
}
```

```java
public class Cat extends Animal {
    public Cat(String name) {
        setName(name);
        setColor("gray");
    }
}
```

```java
public class Dog extends Animal {
    public Dog(String name) {
        setName(name);
        setColor("gray");
    }
}
```

Derived Classes


```
public class Cat extends Animal {
    public Cat(String name) {
        setName(name);
        setColor("gray");
    }
}
```

- Instead of calling the set methods can we just modify the name and color directly?

Derived Classes


```
public class Cat extends Animal {
    public Cat(String name) {
        setName(name);
        setColor("gray");
    }
}
```

- What is the problem?

```
public class Cat extends Animal {
    public Cat(String name) {
        this.name = name;
        this.color = "gray";
    }
}
```

Derived Classes


```
public class Cat extends Animal {
    public Cat(String name) {
        this.name = name;
        this.
    }
}
```

The field Animal.name is not visible

2 quick fixes available:

- Change visibility of 'name' to 'protected'
- Create field 'name' in type 'Cat'

Press 'F2' for focus

- The name and color is private therefore cannot be accessed from the derived/sub class Cat.
- Therefore we need to use the getters and setters method to reach these private class instances.

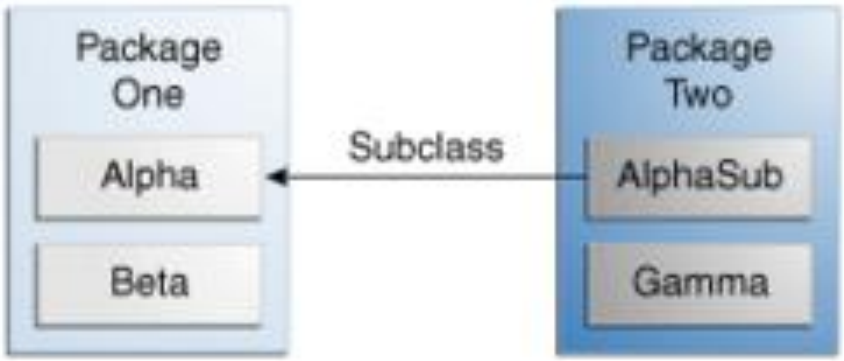
What is inherited?

- All class instances and functions of the base/super class are inherited.
- But not all of them are visible from the sub class

What is inherited?

- All class instances and functions of the base/super class are inherited.
- But not all of them are visible from the sub class
  - Public and protected ones are visible
  - Default and private ones are NOT visible
    - These can be accessed only through getter and setter functions.

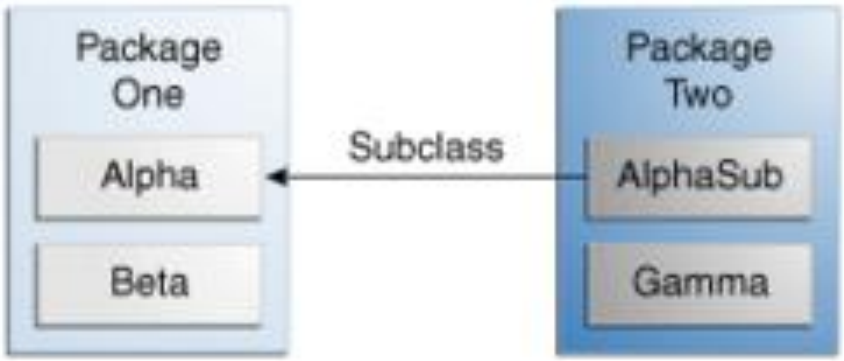
Remember Visibility



Alpha	Beta	AlphaSub	Gamma
public			
protected			
default			
private			

Source: https://docs.oracle.com/javase/tutorial/java/javaOO/accesscontrol.html

Remember Visibility



Alpha	Beta	AlphaSub	Gamma
public	Y	Y	Y
protected	Y	Y	N
default	Y	N	N
private	N	N	N

Source: https://docs.oracle.com/javase/tutorial/java/javaOO/accesscontrol.html

Lets get back to our Animal class


```java
public class Animal {

    private String name;
    private String color;

    public void setName(String name) {
        this.name = name;
    }
    public void setColor(String color) {
        this.color = color;
    }
    public String getName() {
        return name;
    }
    public String getColor() {
        return color;
    }
    public String toString() {
        return "Hi, my name is " + name +". I'm " + color + ".";
    }
}
```

```java
public class Cat extends Animal {
    public Cat(String name) {
        setName(name);
        setColor("gray");
    }
}
```

```java
public class Dog extends Animal {
    public Dog(String name) {
        setName(name);
        setColor("gray");
    }
}
```

Constructors

- What happens inside this constructor?

```
public class Cat extends Animal {
    public Cat(String name) {
        setName(name);
        setColor("gray");
    }
}
```

- Initially the default constructor of the super class is called by compiler.
- Whenever you create an object of an extended class, Java must call some constructor for the superclass object to ensure that its structure is correctly initialized.

Constructors

- Below two are the same!

```
public class Cat extends Animal {
    public Cat(String name) {
        setName(name);
        setColor("gray");
    }
}
```

```
public class Cat extends Animal {
    public Cat(String name) {
        super();
        setName(name);
        setColor("gray");
    }
}
```

`super();`

- Similar to this();

- this(arg); // same class constructor call
- super(arg); // super class constructor call

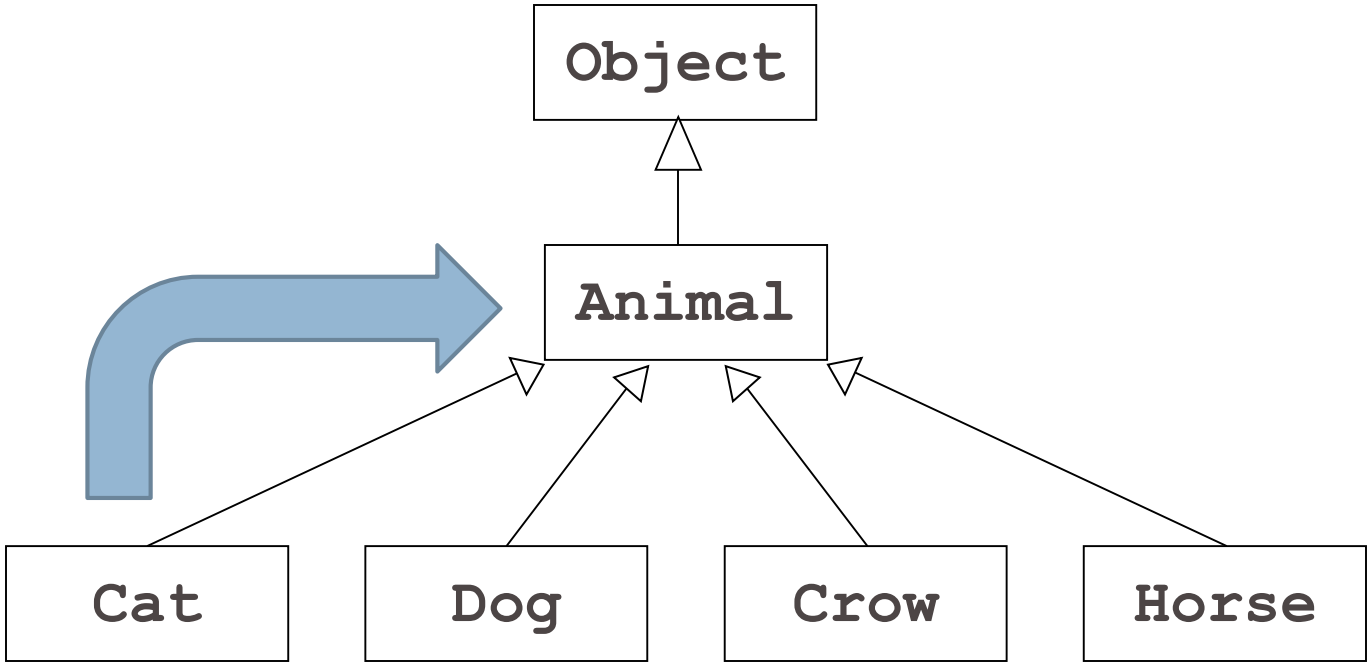
`super();`

- Similar to this();

- this(arg); // same class constructor call
- super(arg); // super class constructor call

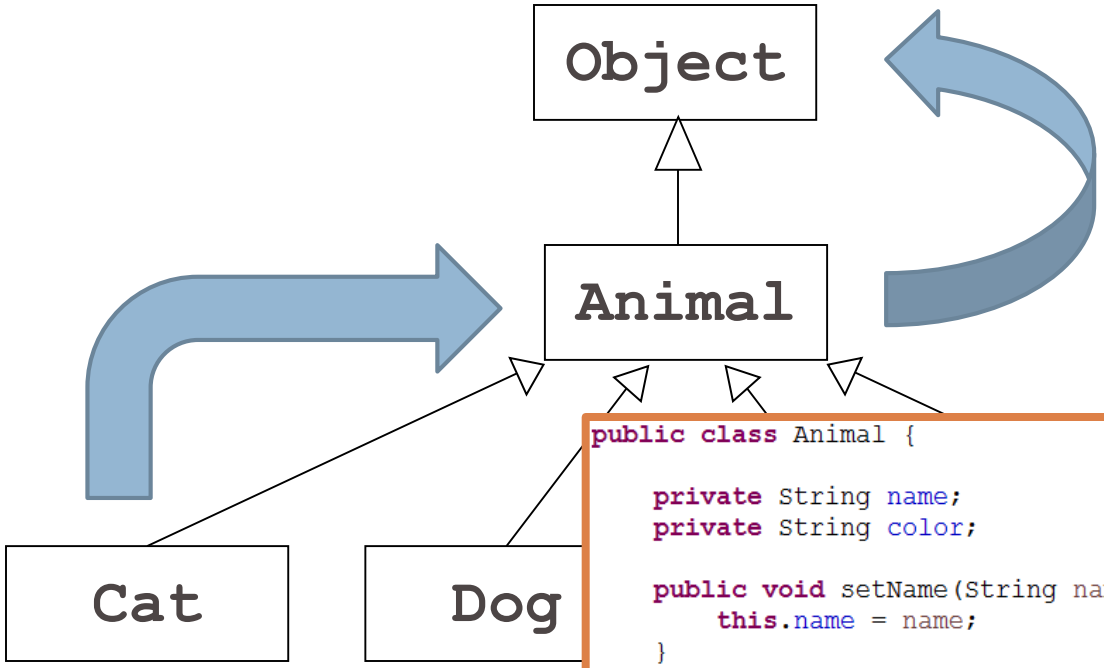
- Both of these need to be used in the first line of the constructor.
- If not, then default super() will be included by compiler

Constructor Call




```
public class Cat extends Animal {
    public Cat(String name) {
        super();
        setName(name);
        setColor("gray");
    }
}
```

Constructor Call

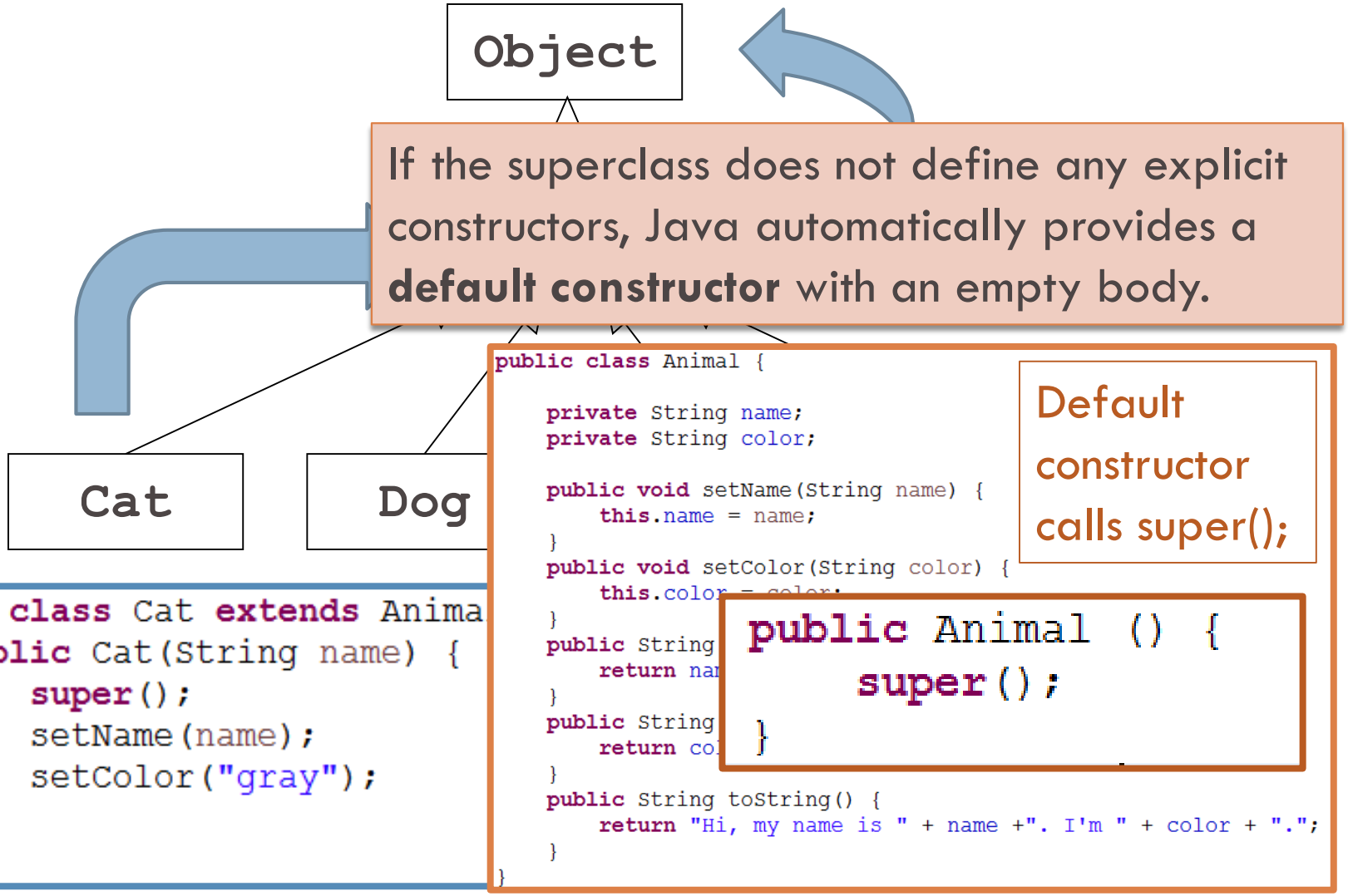



```
public class Cat extends Anima
    public Cat(String name) {
        super();
        setName(name);
        setColor("gray");
    }
}
```

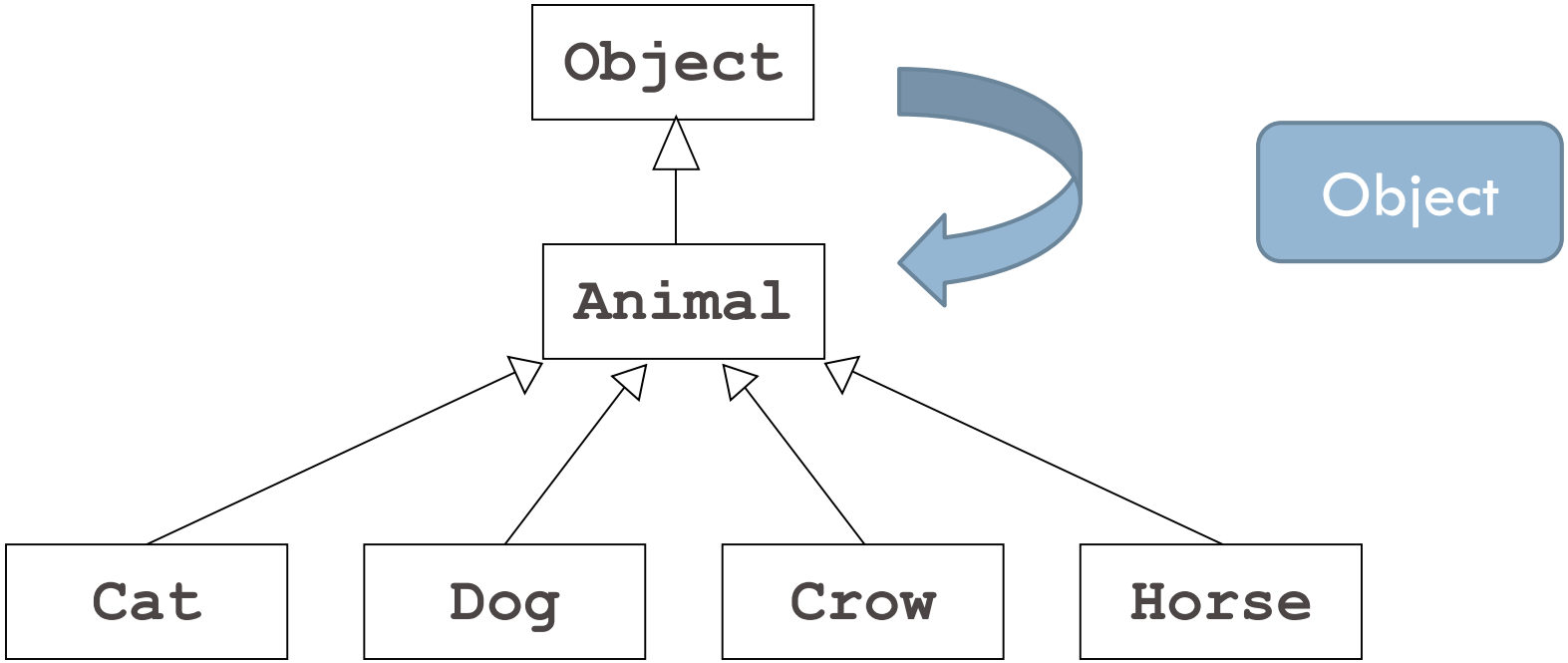
```
public class Animal {

    private String name;
    private String color;

    public void setName(String name) {
        this.name = name;
    }
    public void setColor(String color) {
        this.color = color;
    }
    public String getName() {
        return name;
    }
    public String getColor() {
        return color;
    }
    public String toString() {
        return "Hi, my name is " + name +". I'm " + color + ".";
    }
}
```

Default constructor calls super();

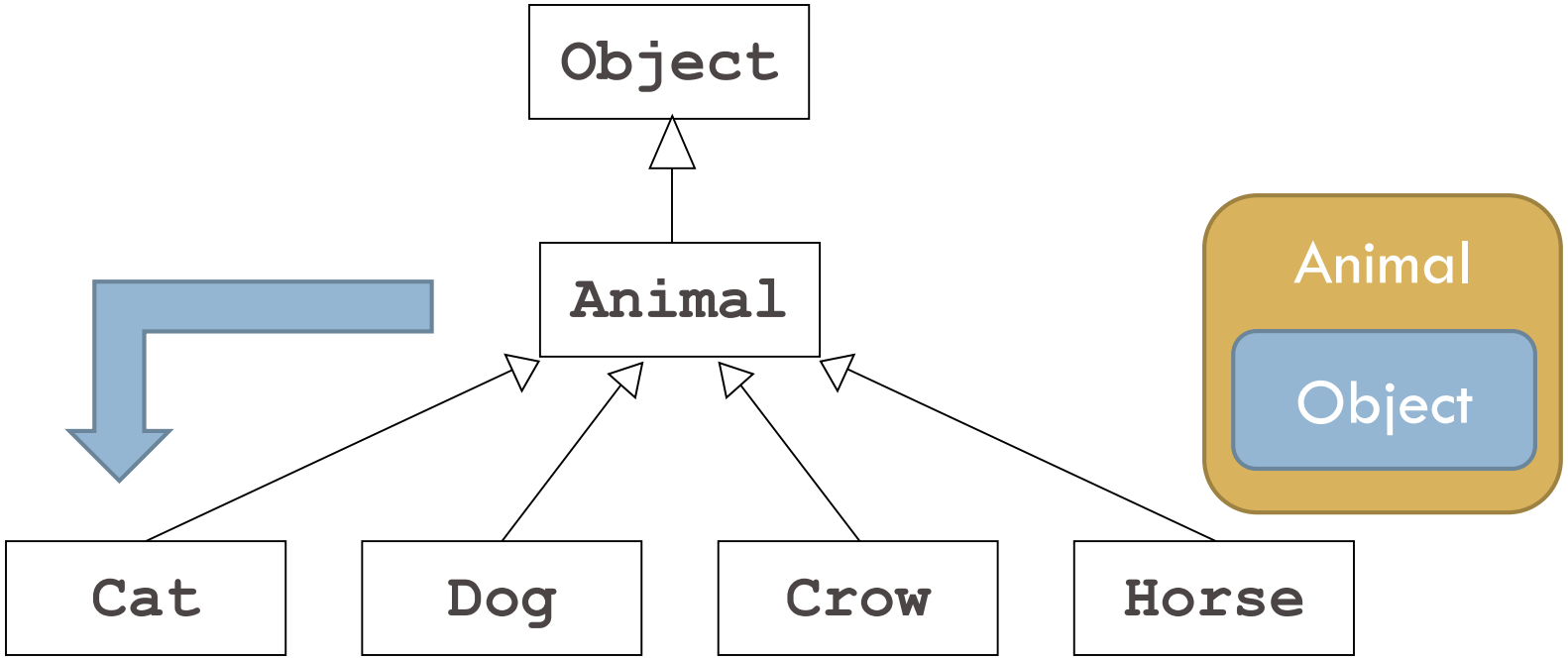
Constructor Call



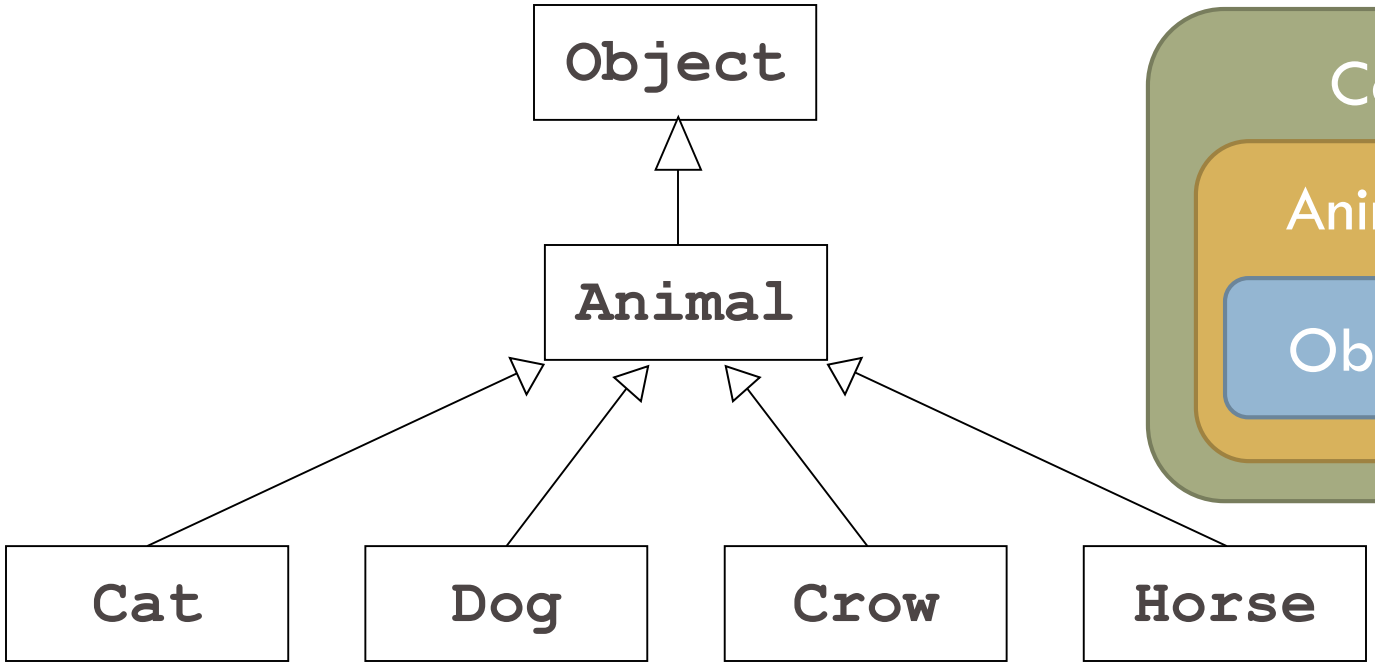
Constructor Call

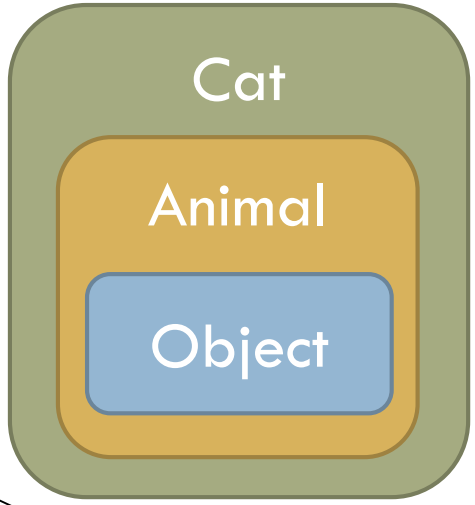


Constructor Call



Constructor Call





Explicit Animal Constructor

- Lets have an explicit Animal constructor

```
public class Animal {

    private String name;
    private String color;

    public Animal (String name) {
        this.name = name;
    }
```

Explicit Animal Constructor

- Lets have an explicit Animal constructor

```
public class Animal {

    private String name;
    private String color;

    public Animal (String name) {
        this.name = name;
    }
```

```
public class Cat extends Animal {
    public Cat(String name) {
        setName(name);
        setColor("gray");
    }
}
```

Explicit Animal Constructor

- Lets have an explicit Animal constructor

```
public class Animal {

    private String name;
    private String color;

    public Animal (String name) {
        this.name = name;
    }
```

```
public class Cat extends Animal {
    public Cat(String name) {
        set
        set
    }
}
```

Implicit super constructor Animal() is undefined. Must explicitly invoke another constructor

Press 'F2' for focus

```java
public class Animal {

    private String name;
    private String color;

    public Animal (String name) {
        this.name = name;
    }
```

```java
public class Cat extends Animal {
    public Cat(String name) {
        setName(name);
        setColor("gray");
    }
}
```

```java
public class Cat extends Animal {
    public Cat(String name) {
        super(name);
        setColor("gray");
    }
}
```

Constructor Calls

- Java therefore invokes the superclass constructor in one of the following ways:
  - Classes that begin with an explicit call to **this** invoke one of the other constructors for this class, delegating responsibility to that constructor for making sure that the superclass constructor gets called.
  - Classes that begin with a call to **super** invoke the constructor in the superclass that matches the argument list provided.
  - Classes that begin with no call to either **super** or **this** invoke the default superclass constructor with no arguments.

Lets implement some functions

- Animals speak differently, so speak function needs to be implemented differently.

Animal Class

- In animal class we don't have a speak() function

```
public class Animal {

    private String name;
    private String color;

    public void setName(String name) {
        this.name = name;
    }
    public void setColor(String color) {
        this.color = color;
    }
    public String getName() {
        return name;
    }
    public String getColor() {
        return color;
    }
    public String toString() {
        return "Hi, my name is " + name +". I'm " + color + ".";
    }
}
```

Cat Class

- Speak function implemented within the Cat class.

```
public class Cat extends Animal {
    public Cat(String name) {
        super(name);
        setColor("gray");
    }
    public String speak() {
        return "Miyauv";
    }
}
```

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

    Cat cat = new Cat("Serafettin");
    cat.speak();
}
```

toString method

- Can we call the toString method from a cat object?

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

    Cat cat = new Cat("Serafettin");
    cat.speak();
    System.out.println(cat);
}
```

toString method

- Can we call the toString method from a cat object?

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

    Cat cat = new Cat("Serafettin");
    cat.speak();
    System.out.println(cat);
}
```

- Yes, I can. What will be the output?

toString method

- Can we call the toString method from a cat object?

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

    Cat cat = new Cat("Serafettin");
    cat.speak();
    System.out.println(cat);
}
```

- Yes, I can. What will be the output?

```
Hi, my name is Serafettin. I'm gray.
```

toString method

- Can the Cat class has its own toString function?

toString method

- Can the Cat class has its own toString function?

```
public class Cat extends Animal {
    public Cat(String name) {
        super(name);
        setColor("gray");
    }
    public String speak() {
        return "Miyauv";
    }
    public String toString() {
        return "I am a cat and I Miyauv.";
    }
}
```

Overriding (Overwriting)

- Yes, it can. It is called function **overwriting**.
- A **subclass** may redefine a method that is defined by a **superclass**. In this case, it is said that the subclass **overrides** the method.

Overriding (Overwriting)

- When one class extends another, the subclass is allowed to **override** method definitions in its superclass. Whenever you invoke that method on an instance of the extended class, Java chooses the new version of the method provided by that class and not the original version provided by the superclass.

Overriding (Overwriting)

- The decision about which version of a method to use is always made on the basis of what the object in fact *is* and not on what it happens to be declared as at that point in the code.
- Will be covered more in the upcoming weeks!

What will be the output?


```java
public class Cat extends Animal {
    public Cat(String name) {
        super(name);
        setColor("gray");
    }
    public String speak() {
        return "Miyauv";
    }
    public String toString() {
        return "I am a cat and I Miyauv.";
    }
}
```

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

    Cat cat = new Cat("Serafettin");
    cat.speak();
    System.out.println(cat);
}
```

What will be the output?


```java
public class Cat extends Animal {
    public Cat(String name) {
        super(name);
        setColor("gray");
    }
    public String speak() {
        return "Miyauv";
    }
    public String toString() {
        return "I am a cat and I Miyauv.";
    }
}
```

```
I am a cat and I Miyauv.
```

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

    Cat cat = new Cat("Serafettin");
    cat.speak();
    System.out.println(cat);
}
```

toString method

- Cat is still an animal. How can I print animal representation as well as the cat representation.

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

      Cat cat = new Cat("Serafettin");
      cat.speak();
      System.out.println(cat);
}
```

- How can I print out the following from the above main?

```
Hi, my name is Serafettin. I'm gray.
I am a cat and I Miyauv.
```

toString method


```java
public class Cat extends Animal {
    public Cat(String name) {
        super(name);
        setColor("gray");
    }
    public String speak() {
        return "Miyauv";
    }
    public String toString() {
        return super.toString() + "\n" +
                "I am a cat and I Miyauv.";
    }
}
```

```
Hi, my name is Serafettin. I'm gray.
I am a cat and I Miyauv.
```

- If you need to invoke the original version of a method, you can do so by using the keyword **super** as a receiver. For example, if you needed to call the original version of an **init** method as specified by the superclass, you could call

```
super.init();
```

Overloading vs. Overriding

- What is the difference between these two?

Overloading vs. Overriding

- What is the difference between these two?
- Overloading
  - Same class has the same function name but with different parameters.
- Overwriting
  - Subclass has the same function signature (name and parameters) with the superclass

toString method in Animal Class

- Is toString an overriding function or not?

```
public class Animal {

    private String name;
    private String color;

    public void setName(String name) {
        this.name = name;
    }
    public void setColor(String color) {
        this.color = color;
    }
    public String getName() {
        return name;
    }
    public String getColor() {
        return color;
    }
    public String toString() {
        return "Hi, my name is " + name +". I'm " + color + ".";
    }
}
```

toString method in Animal Class

- Is toString an overriding function or not?
  - Yes it overrides the toString method of the Object class

```
public class Animal {

    private String name;
    private String color;

    public void setName(String name) {
        this.name = name;
    }
    public void setColor(String color) {
        this.color = color;
    }
    public String getName() {
        return name;
    }
    public String getColor() {
        return color;
    }
    public String toString() {
        return "Hi, my name is " + name +". I'm " + color + ".";
    }
}
```

- We will continue with Polymorphism after Midterm 1

Any Questions ?