COMP 250
INTRODUCTION TO COMPUTER SCIENCE

Lecture 6 – OOD2 Inheritance

Giulia Alberini, Fall 2018

MCGILL DRAGON BOATZ

INFO session

Sept. 19th @7:00pm

McConnell 204

FROM LAST CLASS

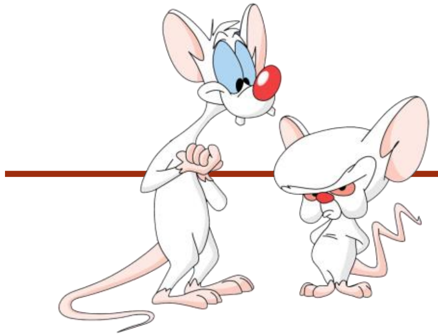
- Packages
- Modifiers

QUESTIONS FROM LAST CLASS

- Aliasing is not allowed in Java + I can create two classes with the same name in different packages. → How can I use 2 classes with the same name?
  - You cannot import both of them. You can import one and use the fully qualified name for the other.

- Suppose I have a class `Dog` inside the package `animals`. And supposed Dog is declared to be package private (no modifier). Is Dog visible from within the package `animals.domestic`? No! Dog is visible only within the package `animals`.

WHAT ARE WE GOING TO DO TODAY?



- Inheritance
  - Subclasses
  - Overloading VS Overriding
  - Constructors
  - Keyword: `super`

INHERITANCE

THE DOG CLASS

- Throughout the next few lectures I'll often refer to a `Dog` class.

```
public class Dog {
   private String name;
   private Person owner;

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

EXAMPLES USING DOG

```
public class Dog {
   private String name;
   private Person owner;

   public Dog(String aName) {
      this.name = aName;
   }

   public static void main(String[] args) {
      Dog myDog = new Dog(“Snoopy”);
      System.out.println(myDog);
   }
}
```

- **What prints?**
  - Dog@4aeda9d5

EXAMPLES USING DOG

```
public class Dog {
    private String name;
    private Person owner;

    public Dog(String aName) {
        this.name = aName;
    }

    public static void main(String[] args) {
        Dog myDog = new Dog(“Snoopy”);
        String s = myDog.toString();
        System.out.println(s);
    }
}
```

- **What prints?**
    - Dog@4aeda9d5

EXAMPLES USING DOG

```
public class Dog {
   private String name;
   private Person owner;

   public Dog(String aName) {
      this.name = aName;
   }

   public static void main(String[] args) {
      Dog myDog = new Dog(“Snoopy”);
      Dog aDog = myDog;
      System.out.println(myDog.equals(aDog));
   }
}
```

- What prints?
  - ➢ `true`

EXAMPLES USING DOG

```
public class Dog {
   private String name;
   private Person owner;

   public Dog(String aName) {
      this.name = aName;
   }

   public static void main(String[] args) {
      Dog myDog = new Dog(“Snoopy”);
      Dog aDog = new Dog(“Snoopy”);
      System.out.println(myDog.equals(aDog));
   }
}
```

- **What prints?**
  - `false`

`toString()` AND `equals()`

We have not defined these methods in the `Dog` class…

- Where do they come from?
- Why can we use them?
- Can we change what they do?

INHERITANCE

- In java, classes can be ***derived*** from other classes.

- A class that is derived from another class is called a ***subclass***.

- The class from which the subclass is derived is called a ***superclass***.

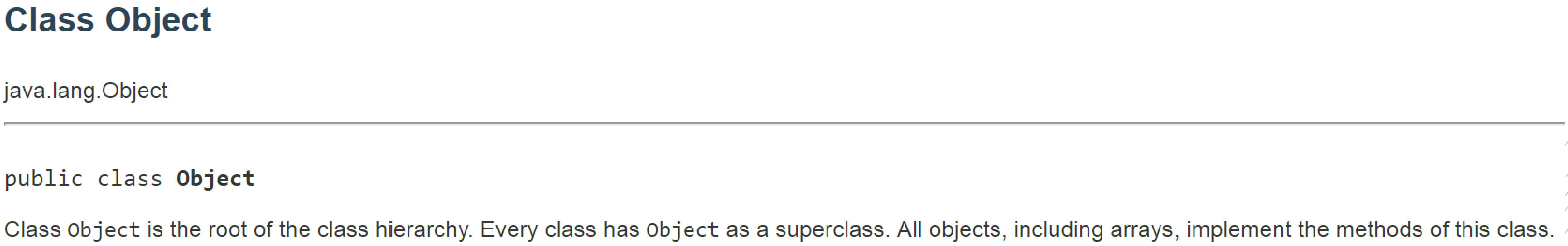
- A subclass ***inherits*** all `public` (or `protected`) fields and methods from its superclass. Constructors are the only thing that a subclass does not inherit.

BASIC IDEA

Suppose that you want to create a new class and that there is already a class that includes some of the code you want. Then instead of implementing this code, you can derive your new class from the existing one. By doing this, you can reuse the code from the existing class without having to write it and debug it again.

THE `Object` CLASS

- `Object` is the only class in java without a superclass. All other classes have one and only one direct superclass.
- In the absence of any other specific superclass, every class is implicitly a subclass of `Object`.



Class Object

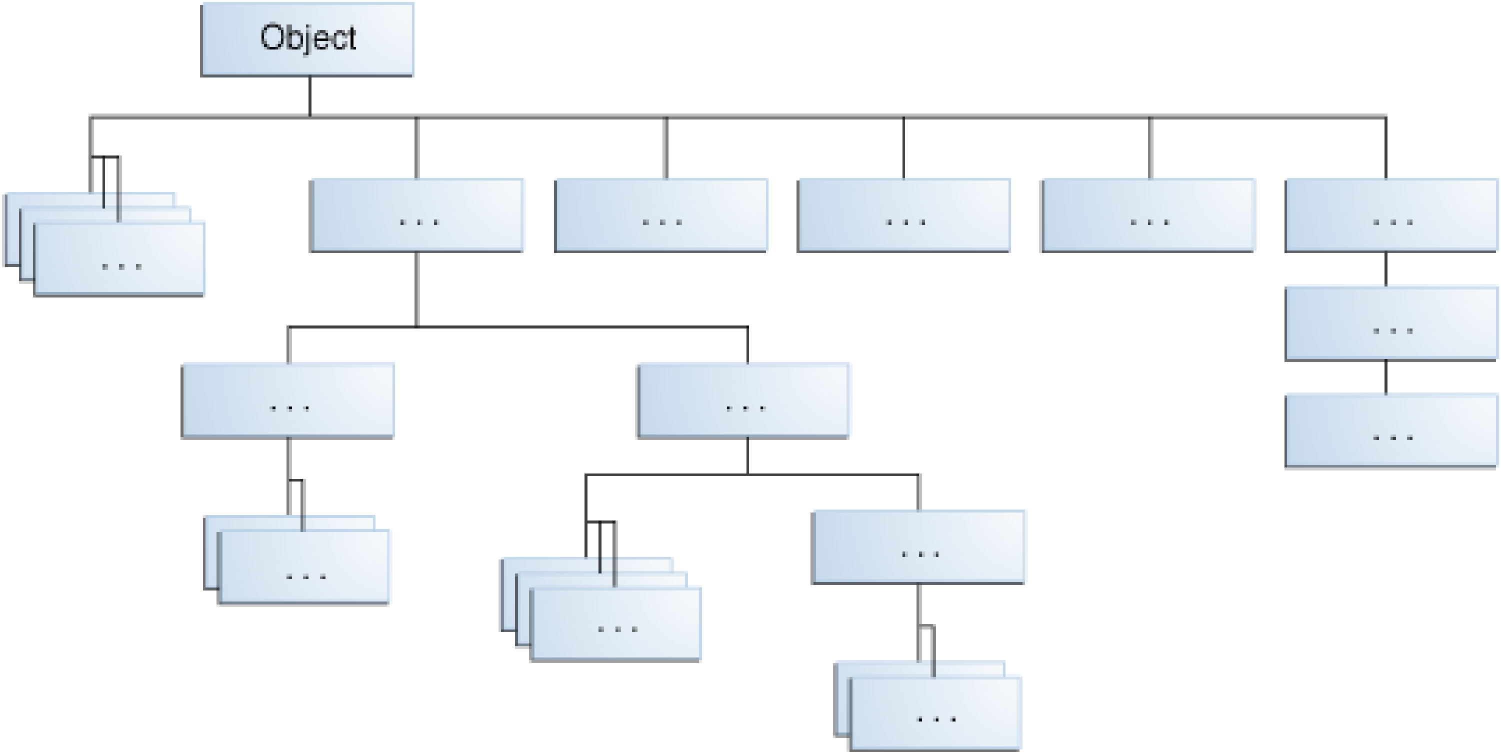
java.lang.Object

```
public class Object
```

Class `Object` is the root of the class hierarchy. Every class has `Object` as a superclass. All objects, including arrays, implement the methods of this class.

https://docs.oracle.com/javase/7/docs/api/java/lang/Object.html

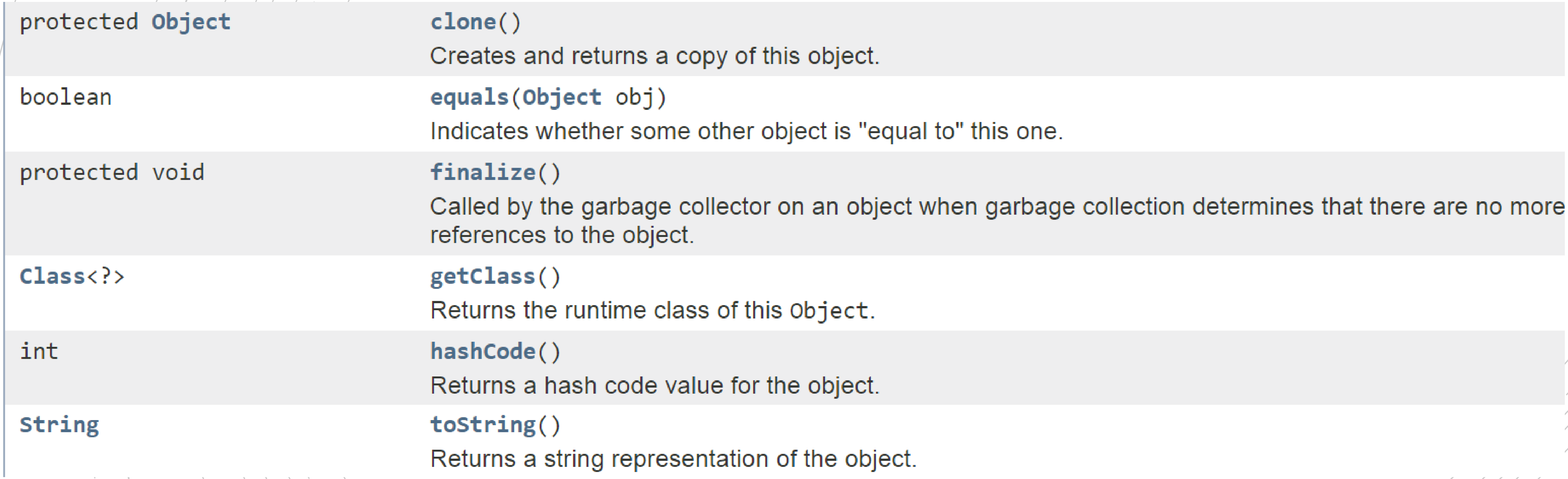
JAVA CLASS HIERARCHY



`Object` **defines and implement methods common to all classes, including the ones you have been writing.**

METHODS FROM Object

This is where `equals` **and** `toString` **come from!!**



protected Object	clone() Creates and returns a copy of this object.
boolean	equals(Object obj) Indicates whether some other object is "equal to" this one.
protected void	finalize() Called by the garbage collector on an object when garbage collection determines that there are no more references to the object.
Class<?>	getClass() Returns the runtime class of this Object.
int	hashCode() Returns a hash code value for the object.
String	toString() Returns a string representation of the object.

https://docs.oracle.com/javase/7/docs/api/java/lang/Object.html

AN EXAMPLE

Suppose we want to write a program with 3 classes: Animal, Dog, and Beagle.

All dogs are animals.

All beagles are dogs.

relationships between classes

Animals have a birthdate.

Dogs bark.

Beagles chase rabbits.

class definitions

AN EXAMPLE

Suppose the class `Animal` **is implemented as follows:**

```
public class Animal {
   private Date birth;

   public void eat(){
      System.out.println(“Nom, nom, nom.”);
   }

   ⋮
}
```

AN EXAMPLE

Then, we can declare a class `Dog` that is a subclass of `Animal` as follow:

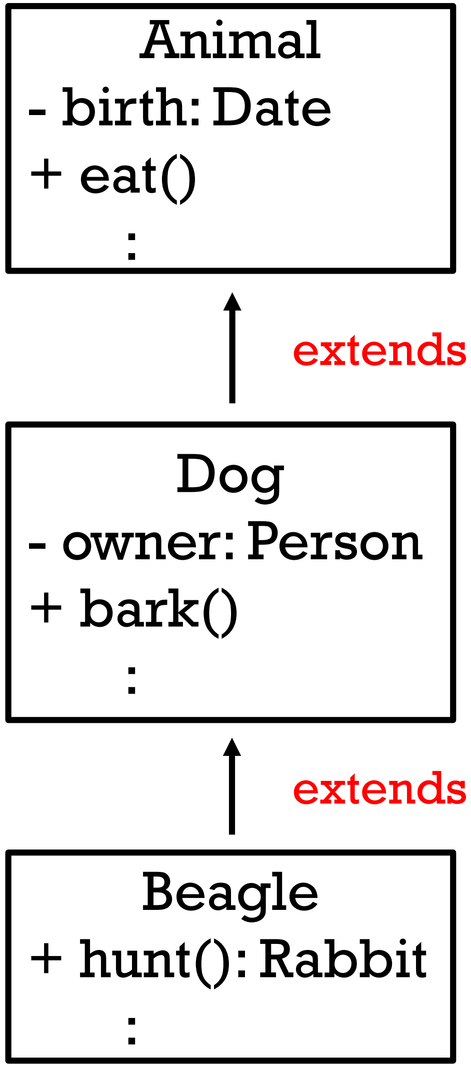
```
public class Dog extends Animal {
   private Person owner;

   public void bark(){
      System.out.println(“Woof!”);
   }

}
```

- `Dog` inherits the method `eat` from `Animal`. It does not inherit the field `birth` because it is `private`. `Dog` also adds the field `owner` and the method `bark`.

A BIGGER PICTURE



Animal
- birth: Date
+ eat()
:

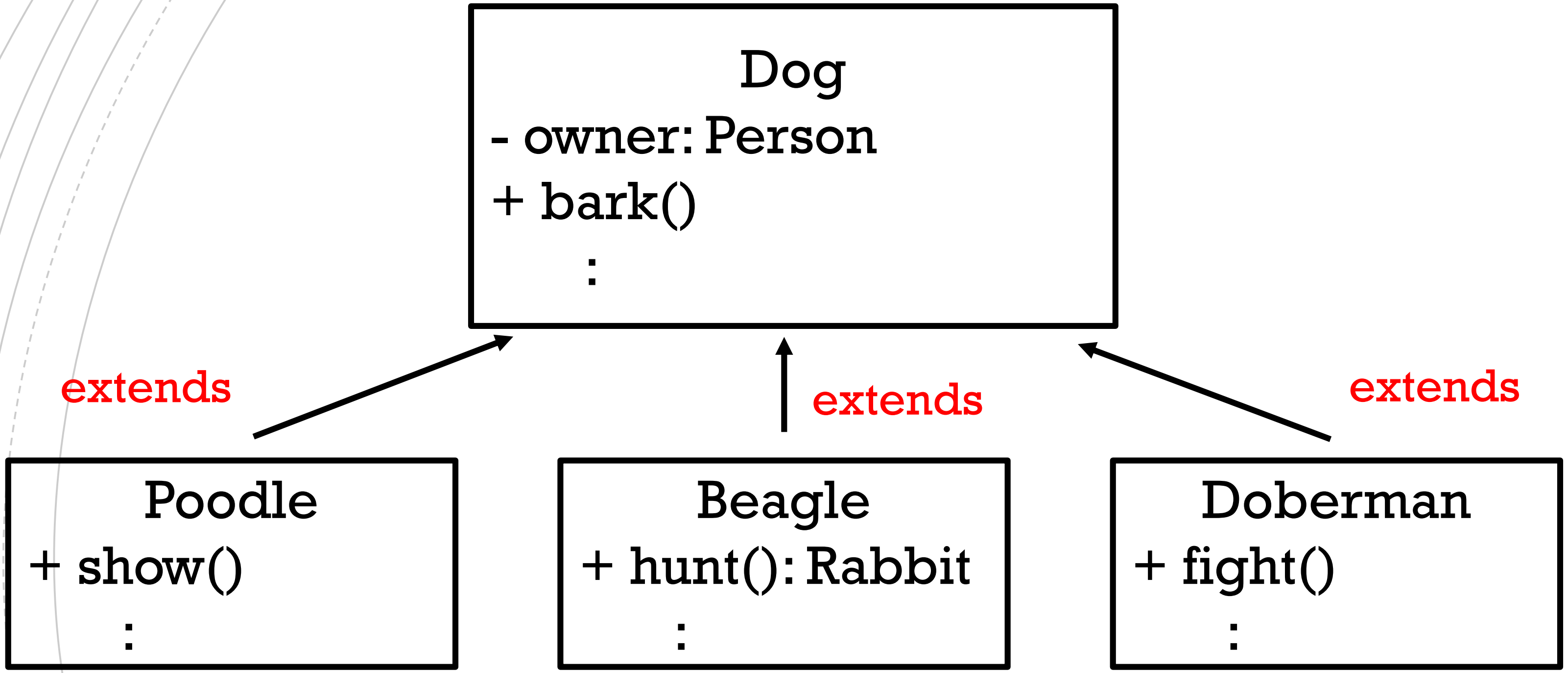
extends

Dog
- owner: Person
+ bark()
:

extends

Beagle
+ hunt(): Rabbit
:

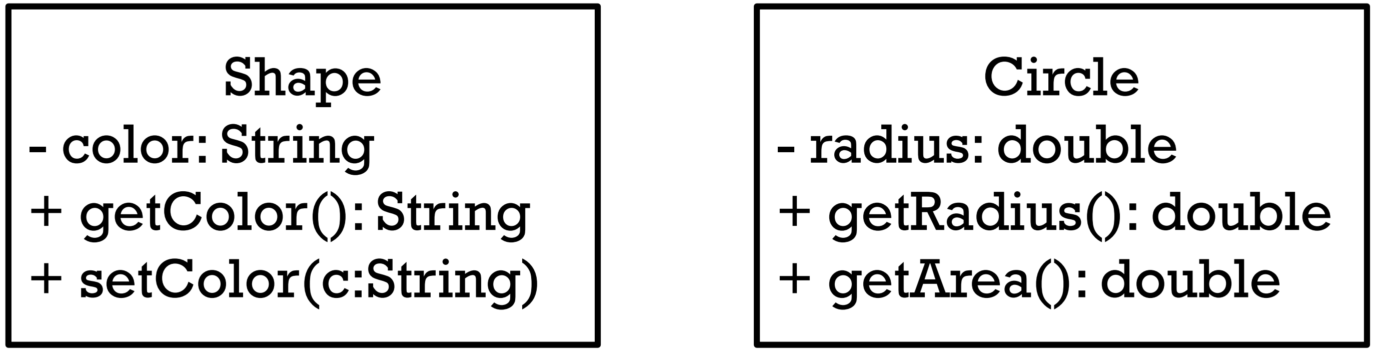
AS MANY SUBCLASSES AS WE NEED



`Poodle`, `Beagle`, and `Doberman` are all a *subclasses* of `Dog`. `Dog` is their *superclass.*

TRY IT!

Let’s take a moment to create the `Shape` and `Circle` class and play around with methods and fields.



Shape
- color: String
+ getColor(): String
+ setColor(c:String)

Circle
- radius: double
+ getRadius(): double
+ getArea(): double

WHAT CAN YOU DO IN A SUBCLASS?

A subclass inherits all the non-private fields and methods of its superclass. In the subclass you can use the inherited members as is, replace them, or hide them. You can also add new members.

- Fields:
  - The inherited fields can be used as any other field.
  - What if in the subclass you declare a field with the same name as the one in the superclass? Then you **hide** the inherited attribute.
    (<u>you should NOT do this</u>)
  - You can declare new field.

WHAT CAN YOU DO IN A SUBCLASS?

method signature = method name + list of parameters.

- Methods:
  - The inherited methods can be used as they are.
  - If you write a non-static method with the same signature (and same return type) as the one from the superclass, you are **overriding** the method.
  - If you write a static method with the same signature (and same return type) as the one from the superclass, you are **hiding** the method.
  - You can declare new methods in the subclass.

OVERLOADING VS OVERRIDING

OVERLOADING

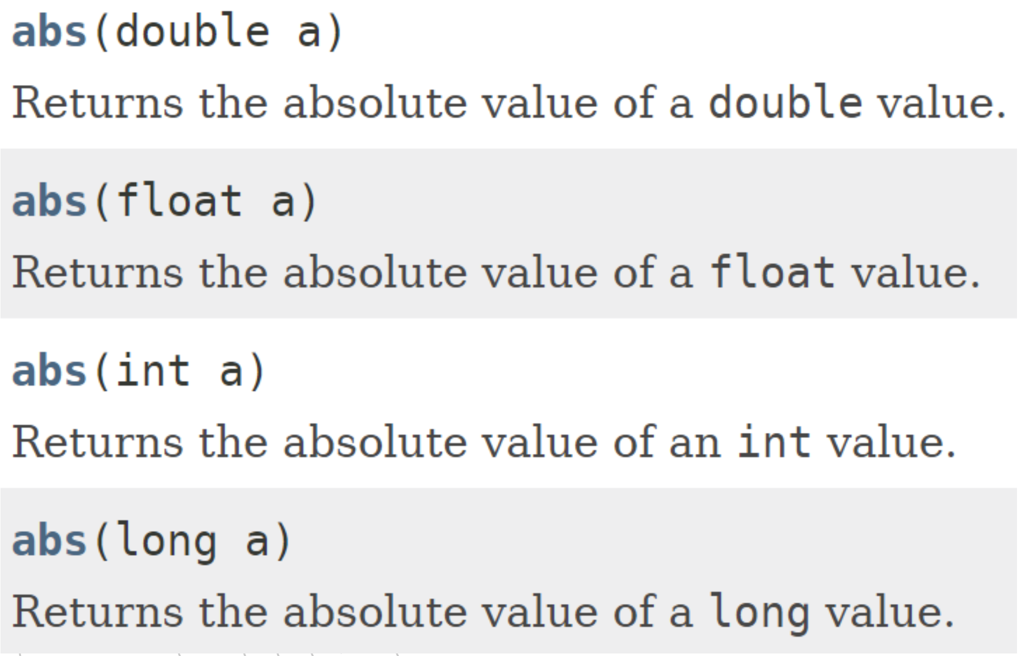
- Two or more methods in the same class with *same name* but *different parameters*. (i.e. different signature)

OVERRIDING

- Two (instance) methods with *same signature and return type*, one in the parent class, one in the child class.

EXAMPLES – OVERLOADING

The method `abs` from `Math` is overloaded



abs(double a)
Returns the absolute value of a `double` value.

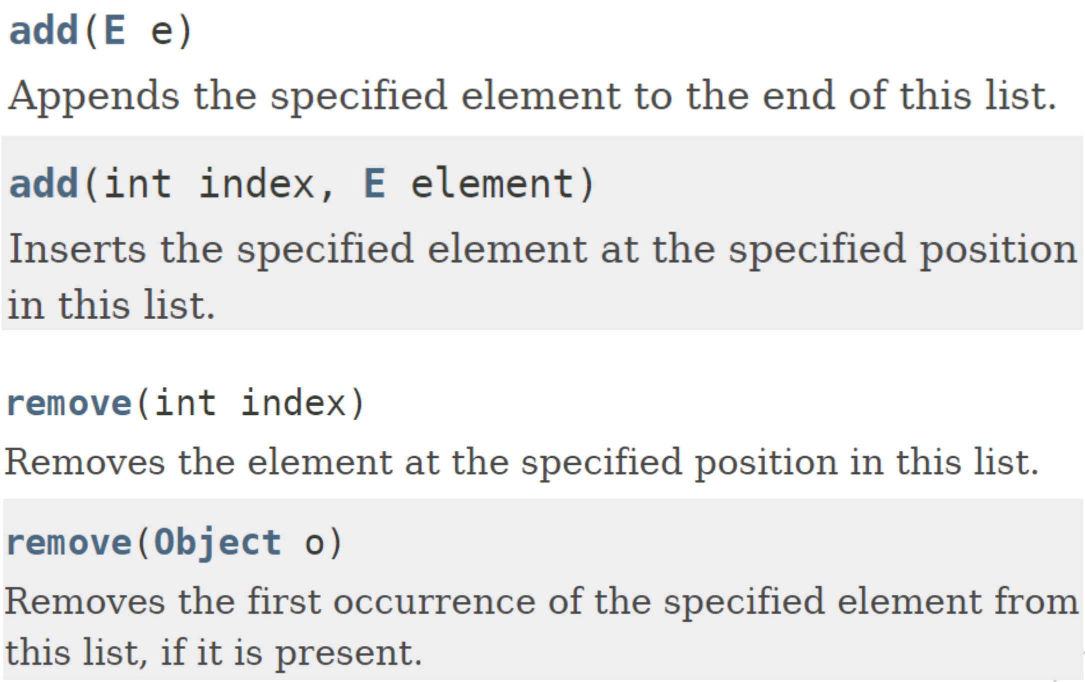
abs(float a)
Returns the absolute value of a `float` value.

abs(int a)
Returns the absolute value of an `int` value.

abs(long a)
Returns the absolute value of a `long` value.

https://docs.oracle.com/javase/8/docs/api/java/lang/Math.html

The methods `add` and `remove` from `ArrayList<E>` are overloaded.



add(E e)
Appends the specified element to the end of this list.

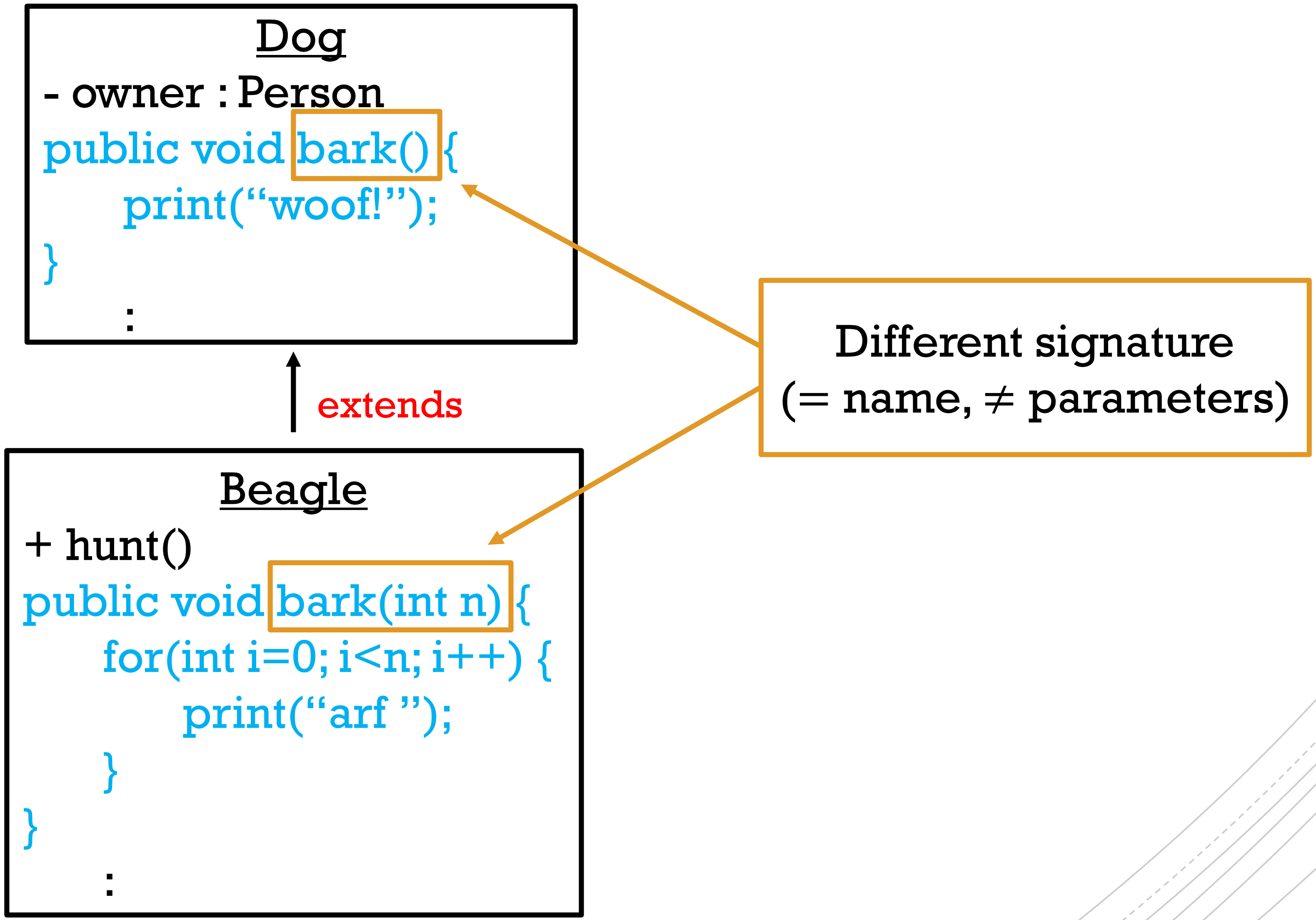
add(int index, E element)
Inserts the specified element at the specified position in this list.

remove(int index)
Removes the element at the specified position in this list.

remove(Object o)
Removes the first occurrence of the specified element from this list, if it is present.

https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html

EXAMPLES – OVERLOADING



Dog

```
- owner : Person
public void bark() {
      print("woof!");
}
      :
```

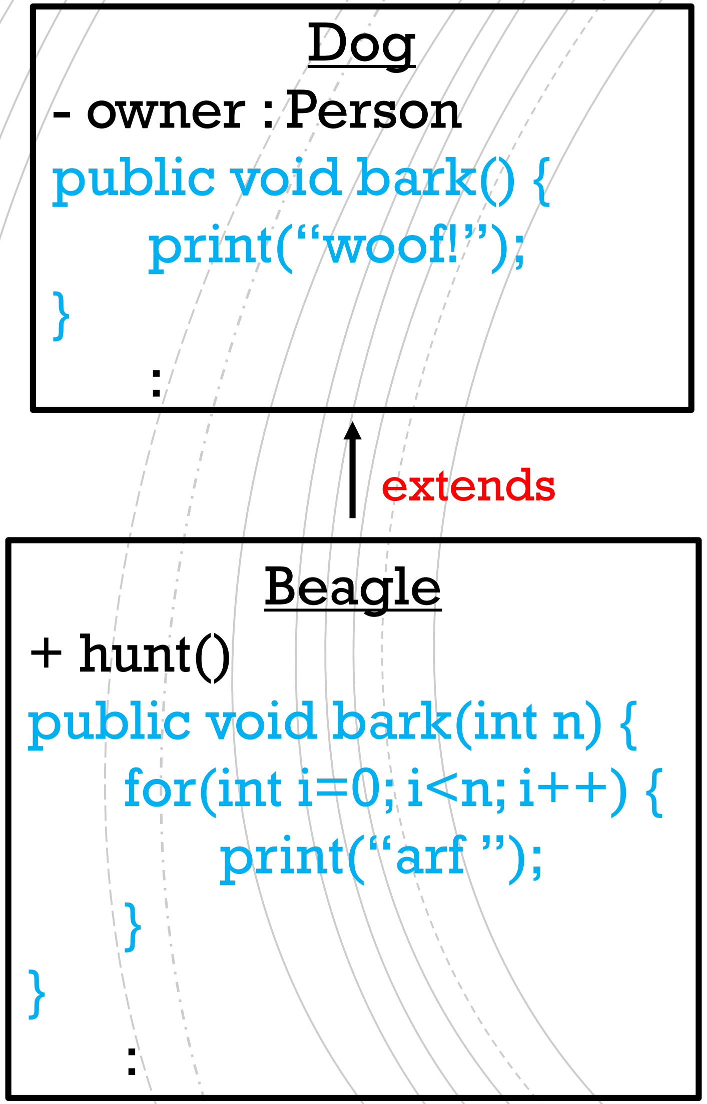
extends

Beagle

```
+ hunt()
public void bark(int n) {
      for(int i=0; i<n; i++) {
            print("arf ");
      }
}
      :
```

Different signature
(= name, ≠ parameters)

EXAMPLES – OVERLOADING



Dog

- owner : Person

```
public void bark() {
      print("woof!");
}
      :
```

extends

Beagle

+ hunt()

```
public void bark(int n) {
      for(int i=0; i<n; i++) {
            print("arf ");
      }
}
      :
```

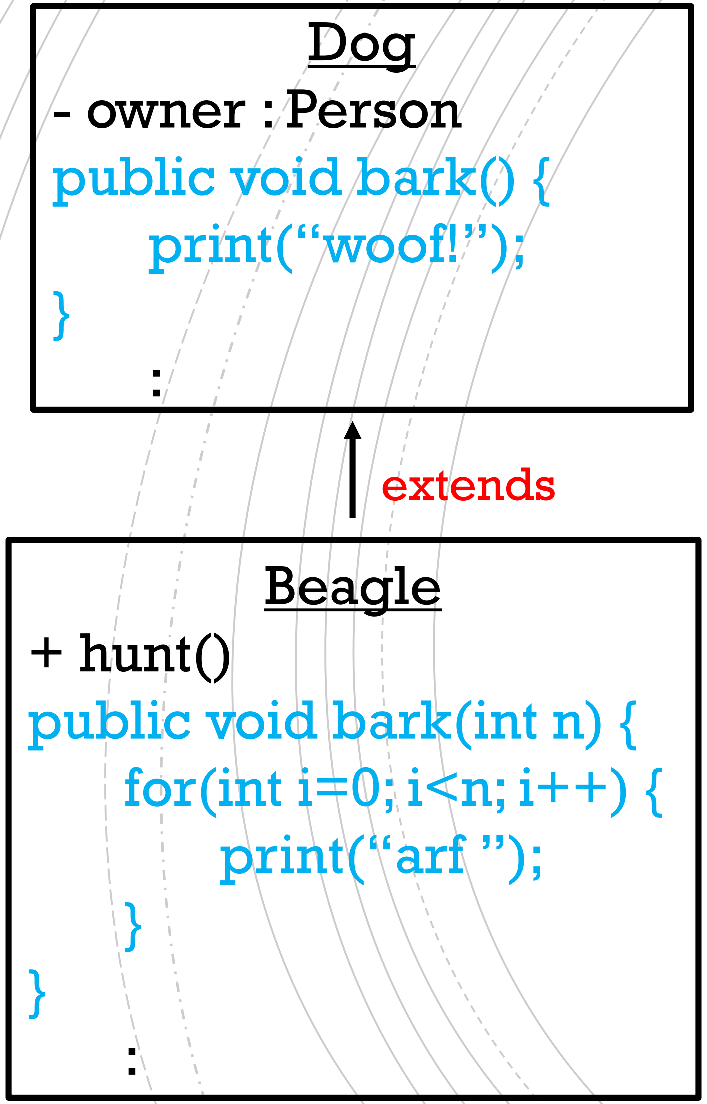
```
public class Test {
   public static void main(String[] args) {
      Beagle snoopy = new Beagle();
      snoopy.bark();
   }
}
```

What prints?

➢ `woof!`

The method defined in the `Dog` class executes!

EXAMPLES – OVERLOADING



Dog

- owner : Person

```
public void bark() {
      print("woof!");
}
      :
```

extends

Beagle

+ hunt()

```
public void bark(int n) {
      for(int i=0; i<n; i++) {
            print("arf ");
      }
}
      :
```

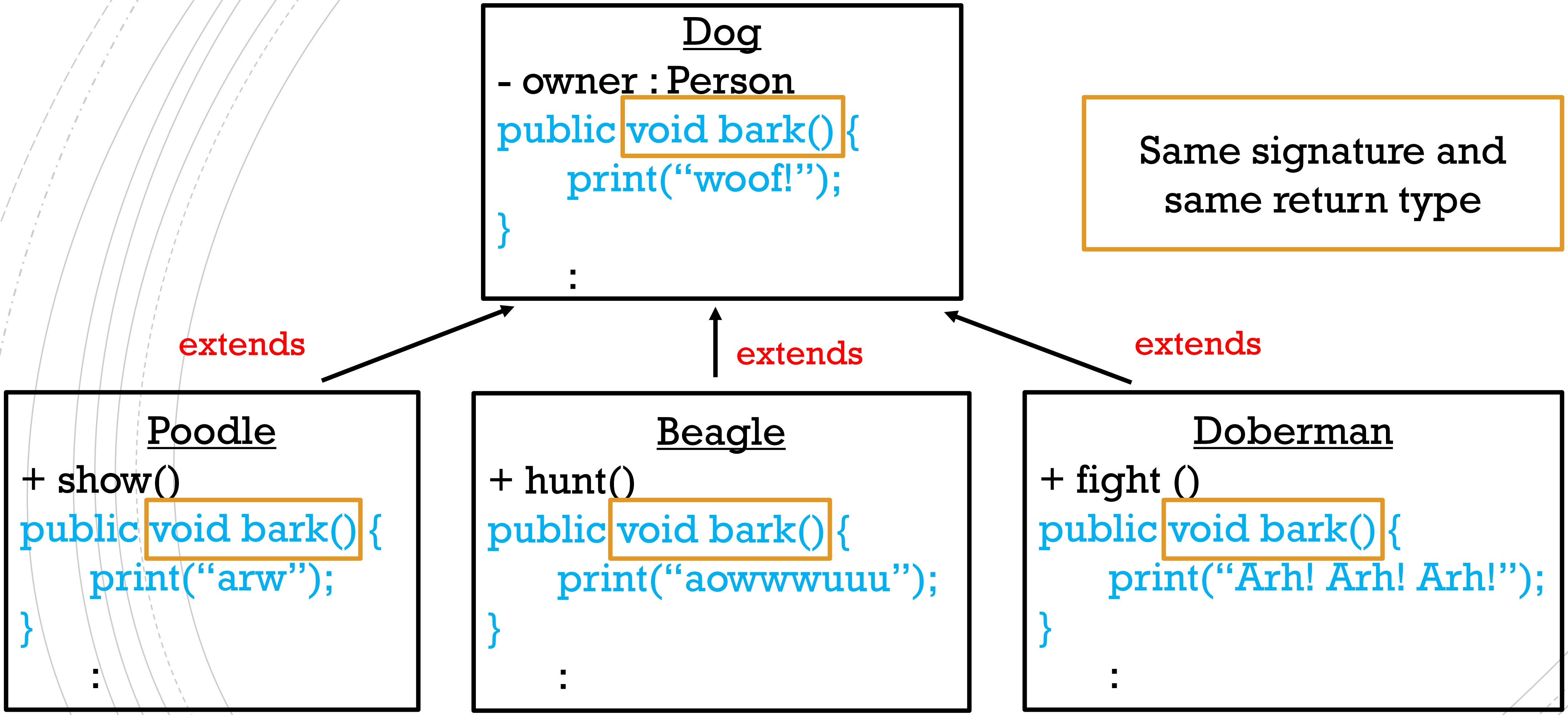
```
public class Test {
   public static void main(String[] args) {
      Beagle snoopy = new Beagle();
      snoopy.bark(3);
   }
}
```

What prints?

➢ `arf arf arf`

The method defined in the `Beagle` **class executes!**

EXAMPLES – OVERRIDING



Dog

```
- owner : Person
public void bark() {
      print("woof!");
}
      :
```

extends

Poodle

```
+ show()
public void bark() {
      print("arw");
}
      :
```

extends

Beagle

```
+ hunt()
public void bark() {
      print("aowwwuuu");
}
      :
```

extends

Doberman

```
+ fight ()
public void bark() {
      print("Arh! Arh! Arh!");
}
      :
```

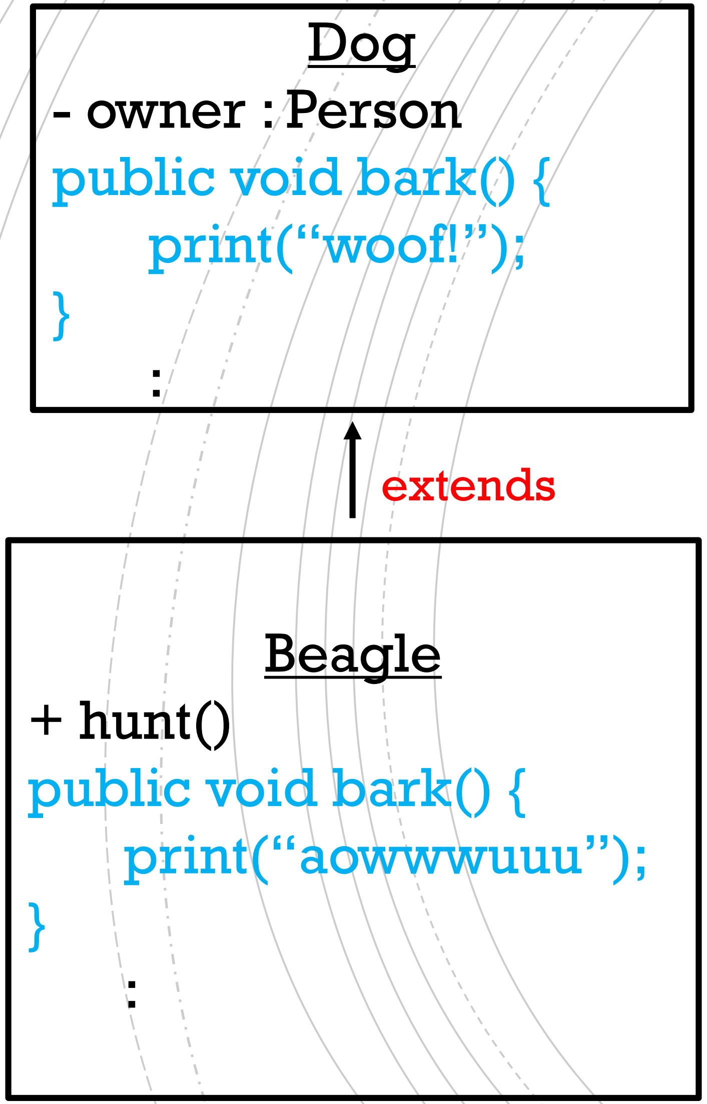
Same signature and same return type

https://www.youtube.com/watch?v=_wqK15EtCMo

https://www.youtube.com/watch?v=esjec0JWEXU

https://www.youtube.com/watch?v=s5Y-Gyt57Dw

EXAMPLES – OVERRIDING



Dog

- owner : Person

```
public void bark() {
      print("woof!");
}
      :
```

extends

Beagle

+ hunt()

```
public void bark() {
      print("aowwwuuu");
}
      :
```

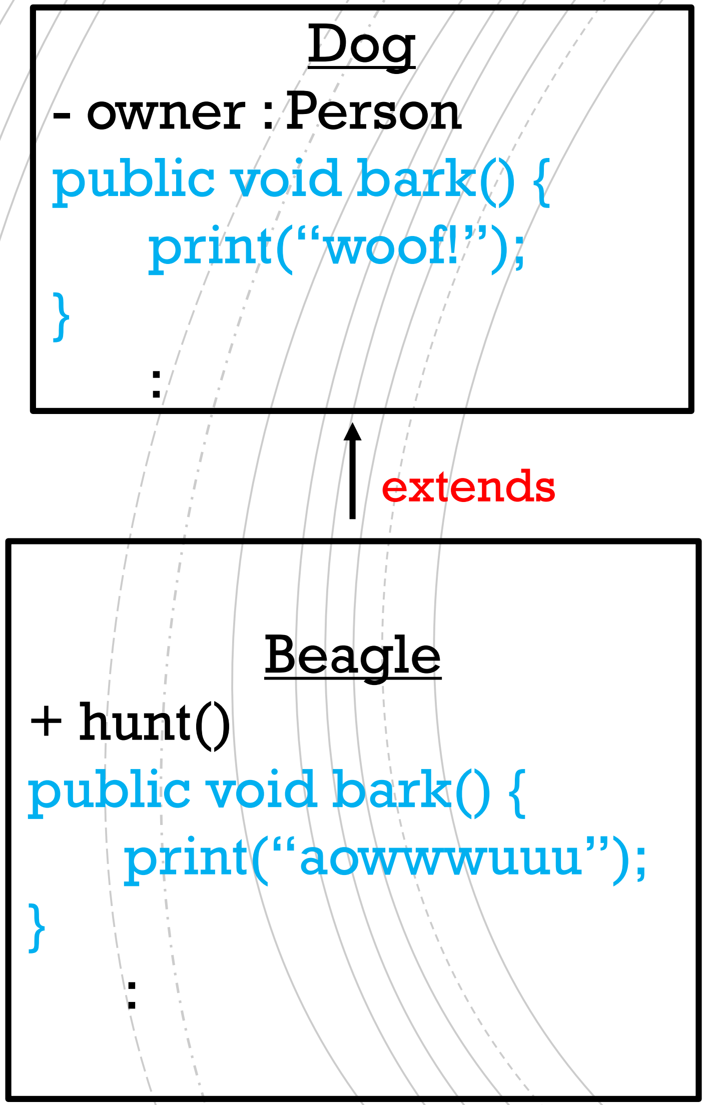
```
public class Test {
   public static void main(String[] args) {
      Beagle snoopy = new Beagle();
      snoopy.bark();
   }
}
```

What prints?

- ➢ `aowwwuuu`

The method defined in the `Beagle` **class executes!**

EXAMPLES – OVERRIDING



Dog

- owner : Person

```
public void bark() {
    print("woof!");
}
    :
```

extends

Beagle

+ hunt()

```
public void bark() {
    print("aowwwuuu");
}
    :
```

```
public class Test {
   public static void main(String[] args) {
      Dog snoopy = new Dog();
      snoopy.bark();
   }
}
```

What prints?

- `woof!`

The method defined in the `Dog` class executes!

NEXT FEW CLASSES!

Dog

- owner : Person

```
public void bark() {
    print("woof!");
}
    :
```

↑ extends

Beagle

+ hunt()

```
public void bark() {
    print("aowwwuuu");
}
    :
```

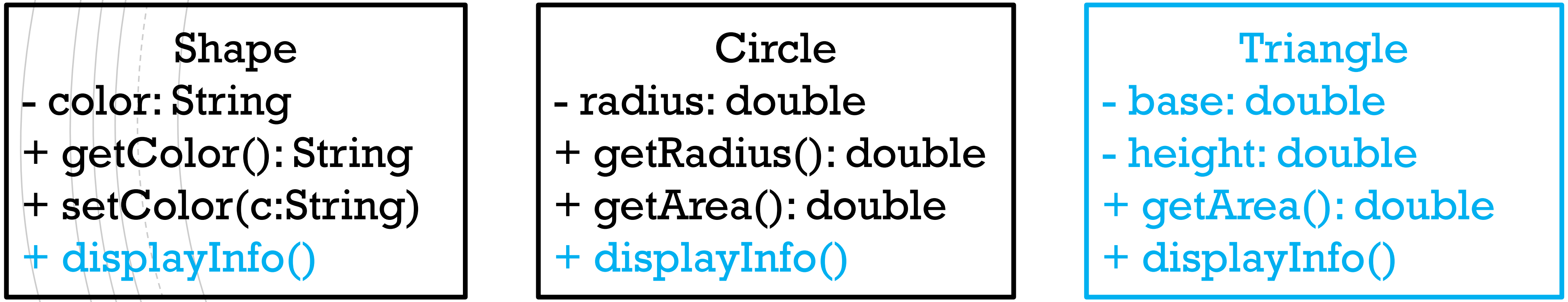
```
public class Test {
   public static void main(String[] args) {
      Dog snoopy = new Beagle();
      snoopy.bark();
   }
}
```

Is this allowed??

If so, which `bark()` will execute???

TRY IT!

To the two previous classes, let’s add a class `Triangle` and a `void` method `displayInfo()` to all three classes.



Shape
- color: String
+ getColor(): String
+ setColor(c:String)
+ displayInfo()

Circle
- radius: double
+ getRadius(): double
+ getArea(): double
+ displayInfo()

Triangle
- base: double
- height: double
+ getArea(): double
+ displayInfo()

WHAT ABOUT CONSTRUCTORS?

Remember that if you don’t write a constructor, the default constructor for a class looks as follows

```
public ClassName() {

}
```

It is a constructor with no-argument and with an empty body.

Important: as soon as you write your own constructor, you no longer have access to the default constructor.

WHAT ABOUT CONSTRUCTORS?

- Constructors are not inherited! Each class has its own.
  You can write constructors for the subclass.

- In the implementation of these constructors you *can* invoke one of the constructors from the superclass.

- If your constructor doesn’t specifically invoke a superclass constructor, then java automatically inserts a call to the no-argument constructor of the superclass.
  NOTE: if the superclass does not have a no-argument constructor, we will get a compile-time error.

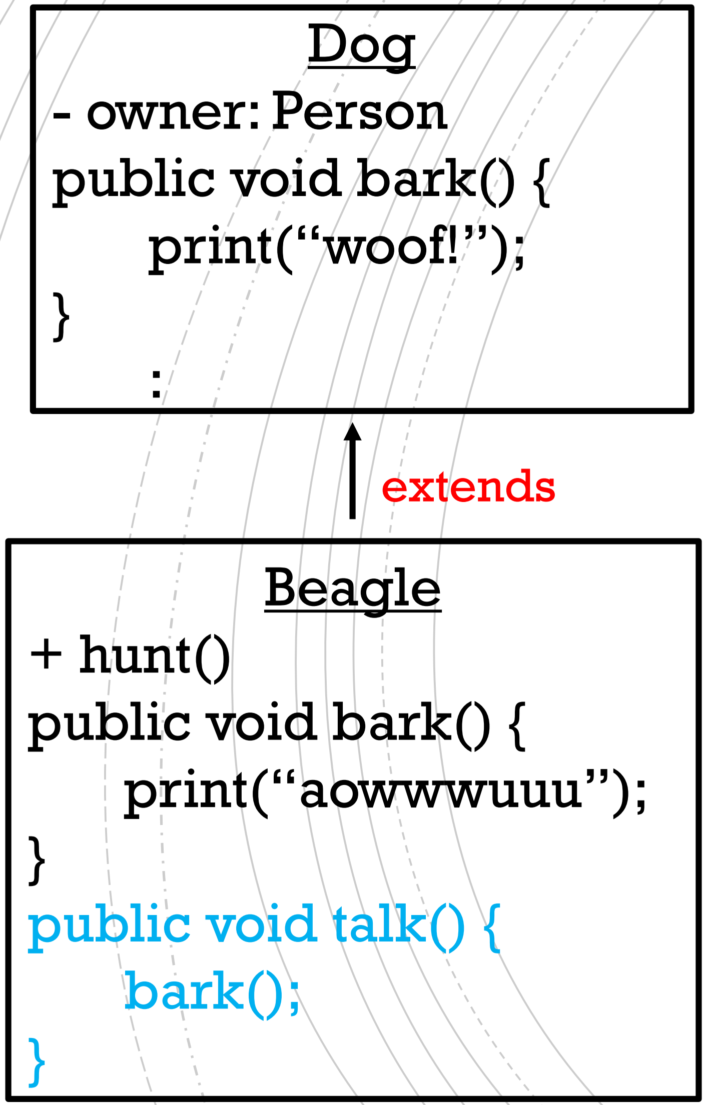
- `Object` has a no-argument constructor, this is why we never received a compile-time error when implementing the constructors for our classes.

KEYWORD `super`

There are 2 uses for the keyword `super`:

1. To access members of the superclass. To do so, we can use `super` in a similar way to `this`.
   - As `this`, `super` refers to the object on which a non-static method was called.
   - Differently from `this`, `super` refers to such object as an instance of the superclass. This is why we can use `super` to access attributes and methods of the superclass.
   - In general, it is not needed (since the subclass inherits all members of the superclass). It <u>must be used if</u> the method you want to access has been overridden or if the field has been hidden.

EXAMPLES – super



```
Dog
- owner: Person
public void bark() {
      print(“woof!”);
}
      :
```

extends

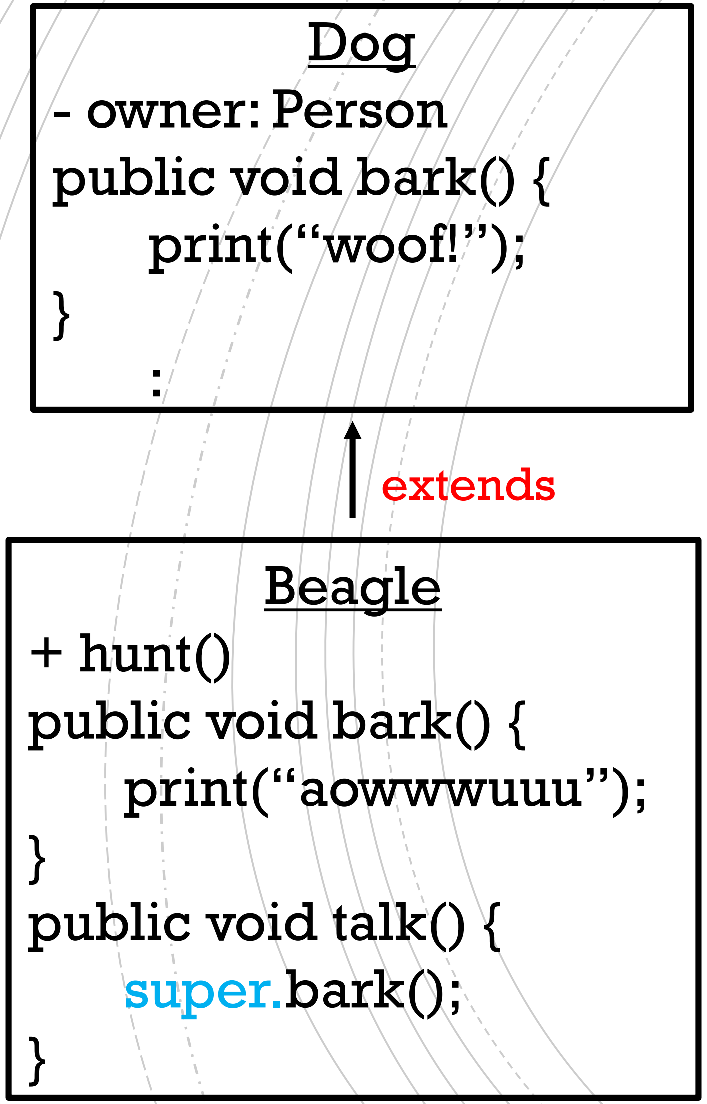
```
Beagle
+ hunt()
public void bark() {
      print(“aowwwuuu”);
}
public void talk() {
      bark();
}
```

```
public class Test {
   public static void main(String[] args) {
      Beagle snoopy = new Beagle();
      snoopy.talk();
   }
}
```

What prints?

- ➢ aowwwuuu

EXAMPLES – super



Dog

```
- owner: Person
public void bark() {
      print(“woof!”);
}
      :
```

extends

Beagle

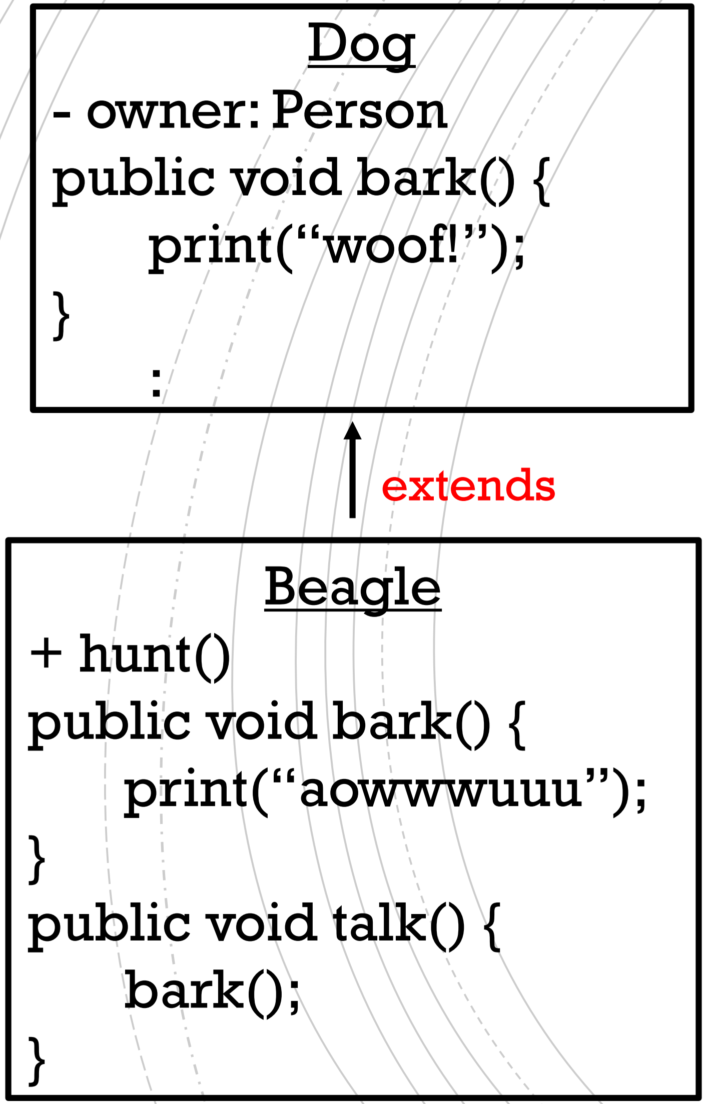
```
+ hunt()
public void bark() {
      print(“aowwwuuu”);
}
public void talk() {
      super.bark();
}
```

```
public class Test {
   public static void main(String[] args) {
      Beagle snoopy = new Beagle();
      snoopy.talk();
   }
}
```

What prints?

➢ `woof!`

EXAMPLES – super



```
Dog
- owner: Person
public void bark() {
      print(“woof!”);
}
      :
```

extends

```
Beagle
+ hunt()
public void bark() {
      print(“aowwwuuu”);
}
public void talk() {
      bark();
}
```

```
public class Test {
   public static void main(String[] args) {
      Dog snoopy = new Dog();
      snoopy.talk();
   }
}
```

What prints?

- ➢ **compile-time error!**

  There’s no method called `talk` inside the `Dog` class.

KEYWORD `super`

2. **Inside the subclass constructors to invoke a constructor from the superclass.**
   - **Syntax:**

```
super();
```

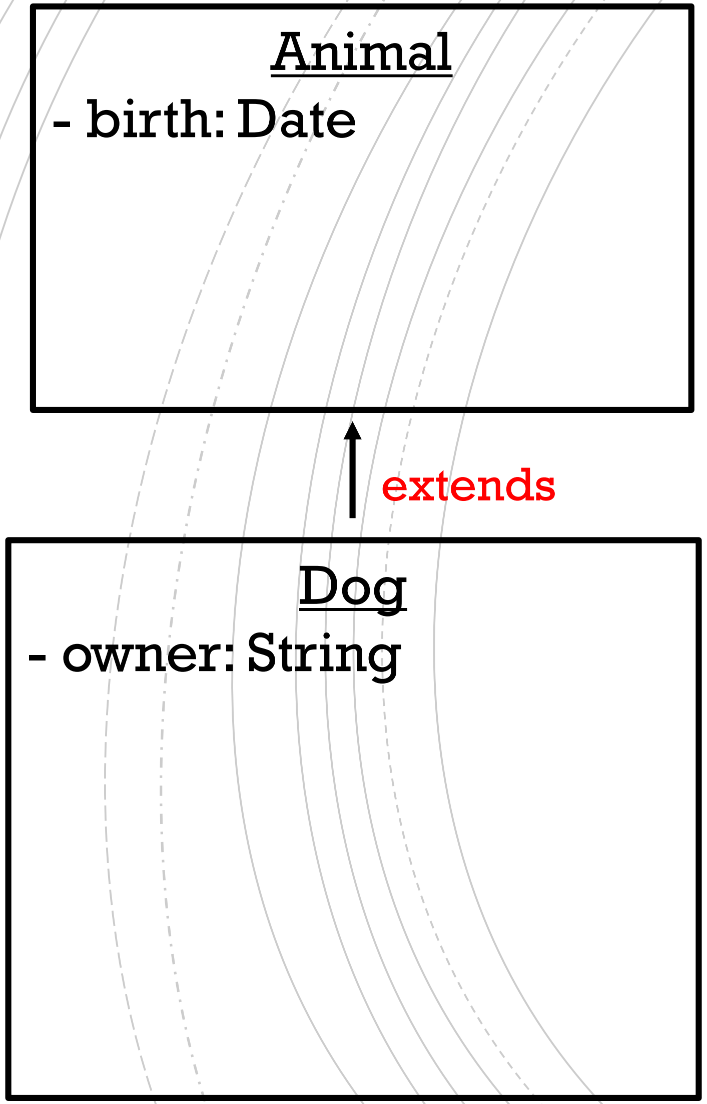
OR

```
super( parameter list);
```

   - **Example:**

```
public Dog(Person owner) {
   super();
   this.owner = owner;
}
```

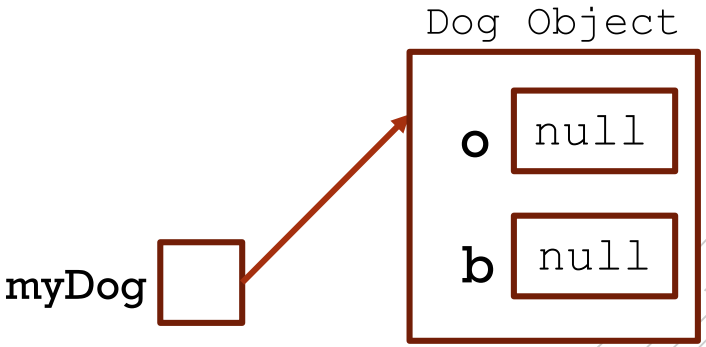
EXAMPLES – super



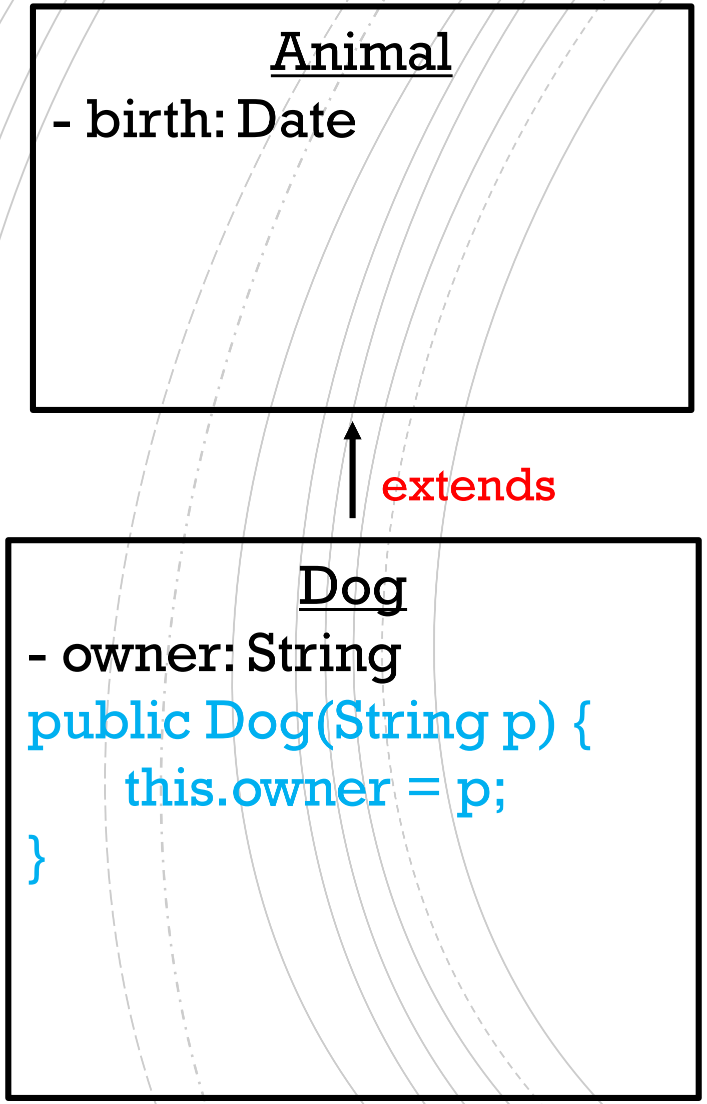
```
public class Test {
   public static void main(String[] args) {
      Dog myDog = new Dog();
   }
}
```

Is this allowed? If so, what is created?

- ➢ *Yes, the default constructor of Dog is used which implicitly calls on the default constructor from Animal.*



EXAMPLES – super



Animal

- birth: Date

extends

Dog

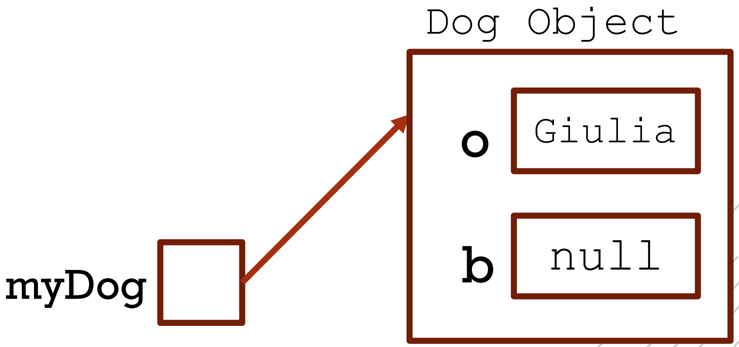
- owner: String

```
public Dog(String p) {
      this.owner = p;
}
```

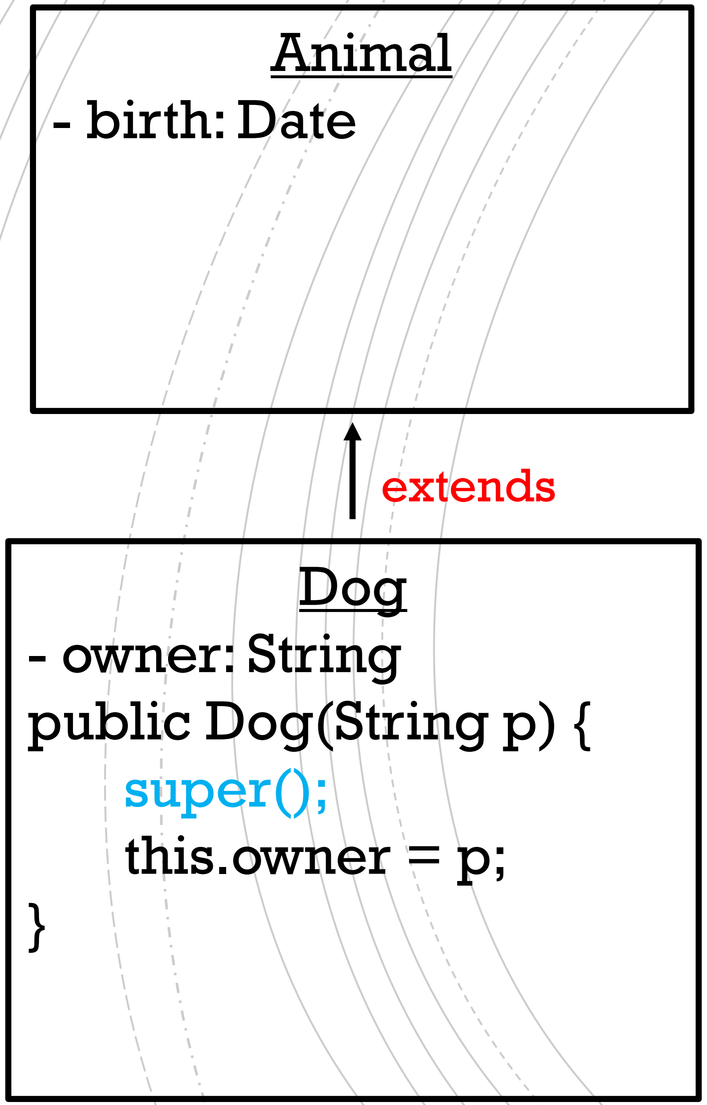
```
public class Test {
   public static void main(String[] args) {
      Dog myDog = new Dog(“Giulia”);
   }
}
```

Is this allowed? If so, what is created?

➢ *Yes, the constructor of Dog implicitly calls on the default constructor from Animal.*



EXAMPLES – super



Animal

- birth: Date

extends

Dog

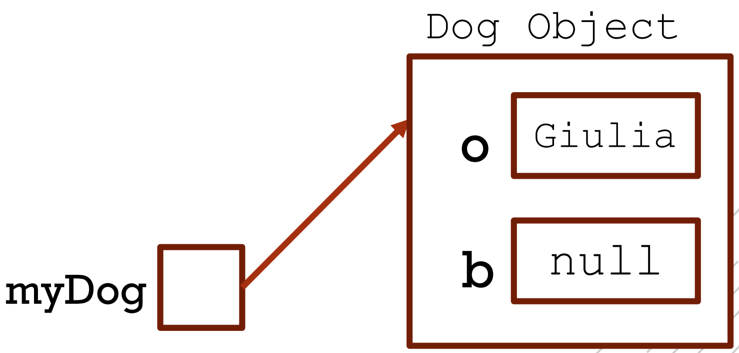
- owner: String

```
public Dog(String p) {
    super();
    this.owner = p;
}
```

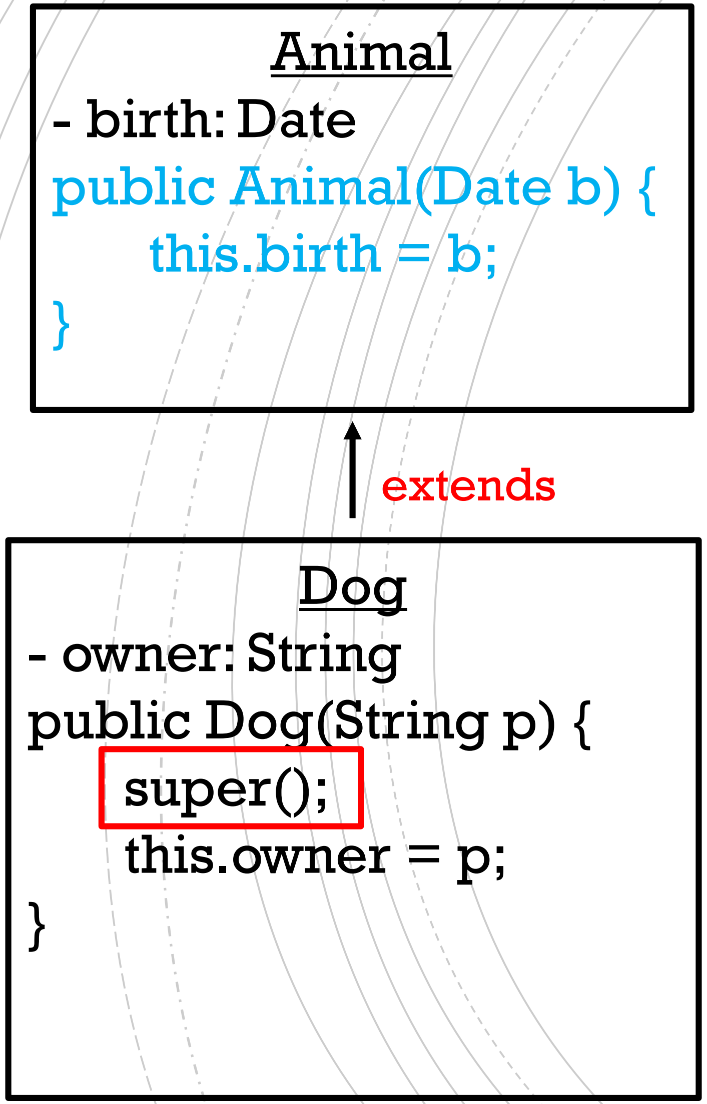
```
public class Test {
   public static void main(String[] args) {
      Dog myDog = new Dog("Giulia");
   }
}
```

Is this allowed? If so, what is created?

- ➢ *Yes, the constructor of Dog explicitly calls on the default constructor from Animal.*



EXAMPLES – super



Animal

- birth: Date

```
public Animal(Date b) {
    this.birth = b;
}
```

extends

Dog

- owner: String

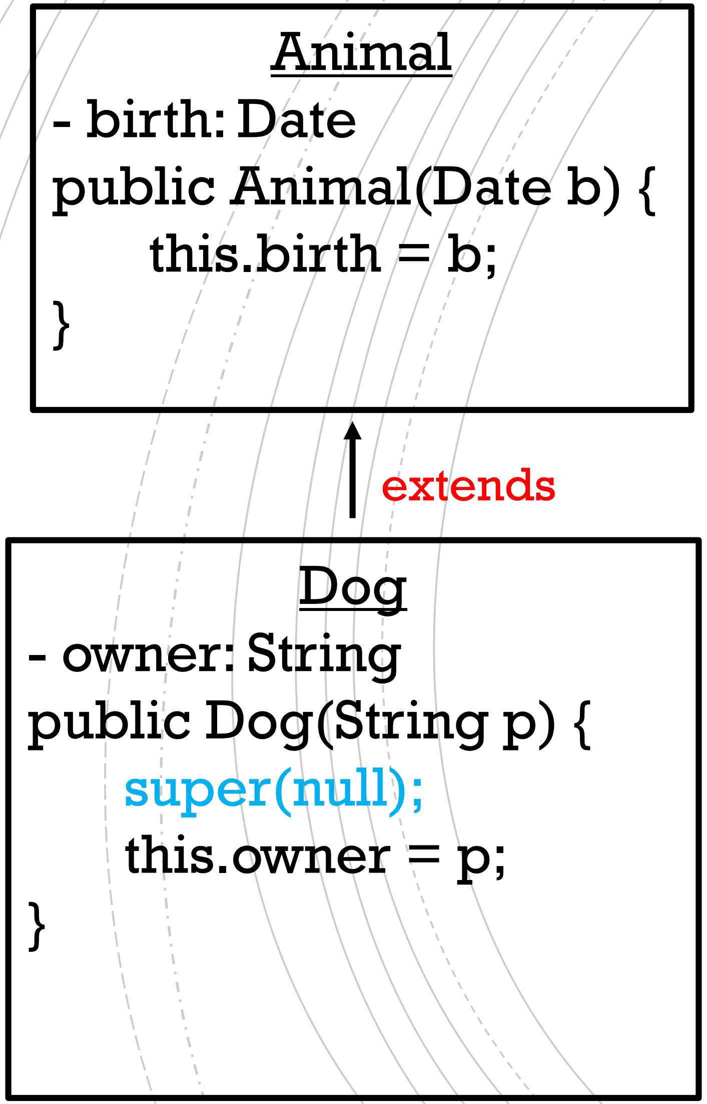
```
public Dog(String p) {
    super();
    this.owner = p;
}
```

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

Is this allowed? If so, what is created?

- *Compile-time error*.
  There's no constructor with no arguments in the Animal class!

EXAMPLES – super



Animal

```
- birth: Date
public Animal(Date b) {
      this.birth = b;
}
```

extends

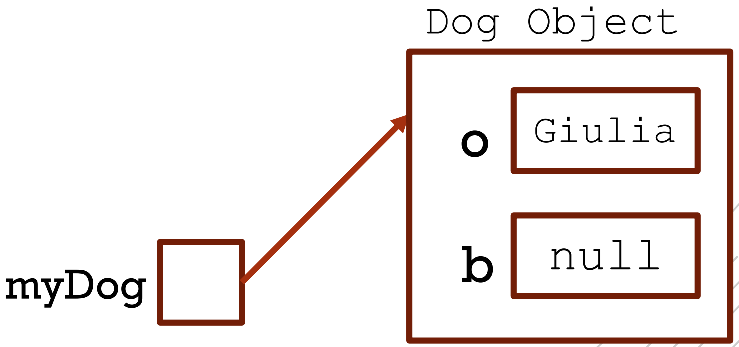
Dog

```
- owner: String
public Dog(String p) {
      super(null);
      this.owner = p;
}
```

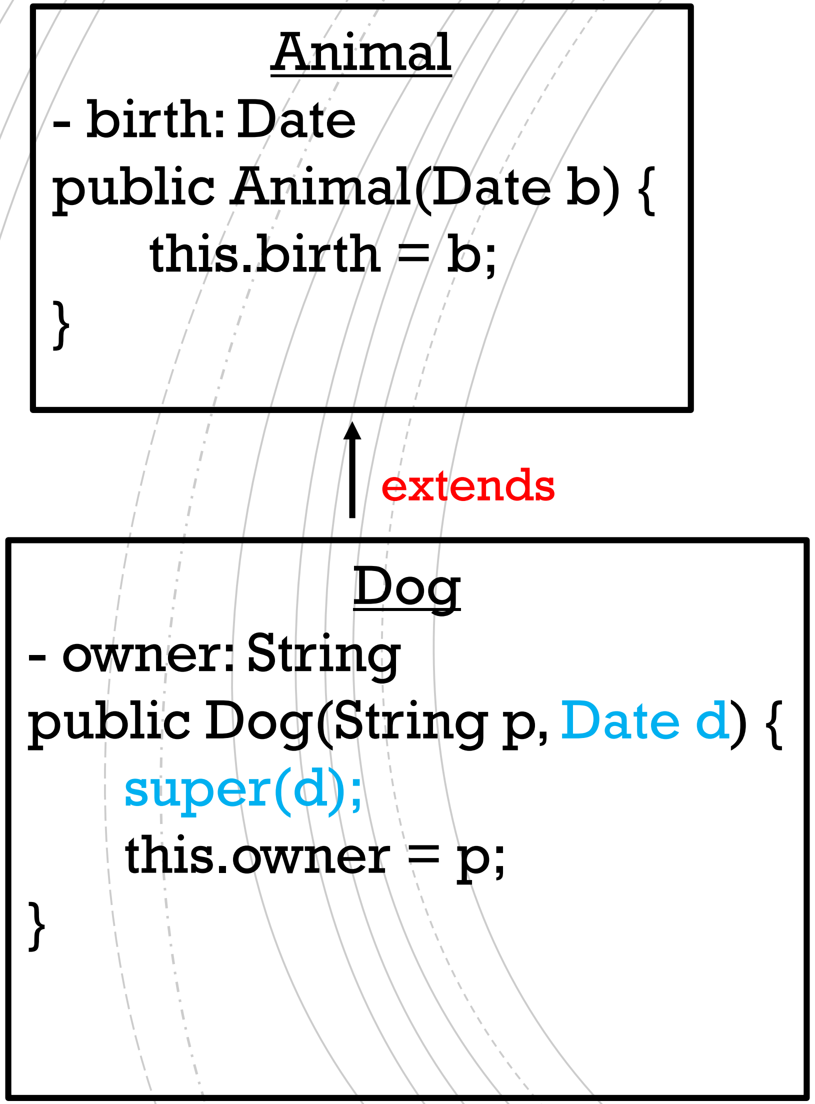
```
public class Test {
   public static void main(String[] args) {
      Dog myDog = new Dog("Giulia");
   }
}
```

Is this allowed? If so, what is created?

- *Yes*



EXAMPLES – super



```
Animal
- birth: Date
public Animal(Date b) {
      this.birth = b;
}
```

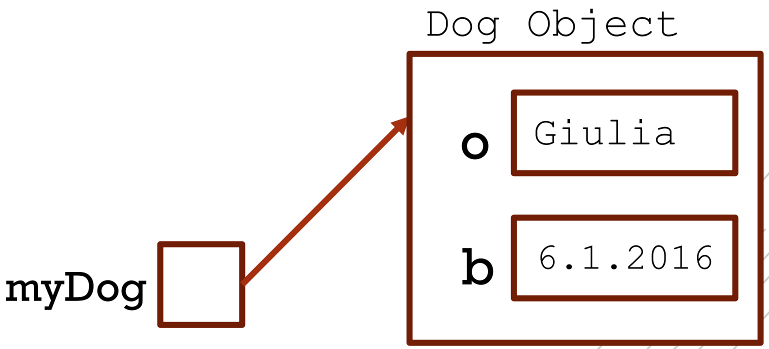
extends

```
Dog
- owner: String
public Dog(String p, Date d) {
      super(d);
      this.owner = p;
}
```

```
public class Test {
    public static void main(String[] args) {
        Dog myDog = new Dog("Giulia", 6.1.2016);
    }
}
```

Is this allowed? If so, what is created?

➢ *Yes*



TRY IT!

Go back to the three classes we have created and add appropriate constructors.

MODIFIERS and INHERITANCE

ACCESS CONTROL MODIFIERS

- Recall that a class can be declared to be either `public` or package-private (no keyword).
- A class can *extend* another class *if and only if* the latter is visible from where the former is located.

ACCESS CONTROL MODIFIERS

- Recall that a class can be declared to be either `public` or package-private (no keyword).
- A class can *extend* another class *if and only if* the latter is visible from where the former is located.

```
package assignments.a1;

public class A {
   :
}
```

```
package lectures;

public class B extends A{
   :
}
```

✓

All public classes can be extended (even across packages)

ACCESS CONTROL MODIFIERS

- Recall that a class can be declared to be either `public` or package-private (no keyword).
- A class can *extend* another class *if and only if* the latter is visible from where the former is located.

```
package assignments.a1;

class A {
   :
}
```

```
package lectures;

public class B extends A{
   :
}
```

✗

Not allowed,
since A is not
visible from B.

WHICH MEMBERS ARE INHERITED?

- Every superclass' member visible from where the subclass is located is inherited by the subclass. (with the exception of constructors)

- Members include: fields, methods, inner/ static nested classes.

- Note that *a subclass cannot reduce the visibility of an inherited method*. The visibility can only be increased. (we'll understand better why in the next few classes)

ASIDE: NESTED CLASSES

- Note that a nested class is not a subclass.

- Outer and inner classes have access to all fields and methods of each other. Details are out of the scope of this course.

final KEYWORD

- **A class that has been declared** `final` **cannot be *extended*.**

```
public final class Dog {
    :
}
```

```
public class Beagle extends Dog {
    :
}
```

✗ compile-time error!

`final` KEYWORD

- A method that has been declared `final` cannot be *overridden*.

```
public class Dog {
   public final void bark() {
      :
   }
}
```

```
public class Beagle extends Dog {
   public void bark() {
      :
   }
}
```

✗ compile-time error!

TO LOOK FORWARD TO

- Next class:
  - Let’s talk more about `Object`
  - Type conversion