Overloading, Overriding, This and Super

Nancy Hitschfeld
Matías Toro

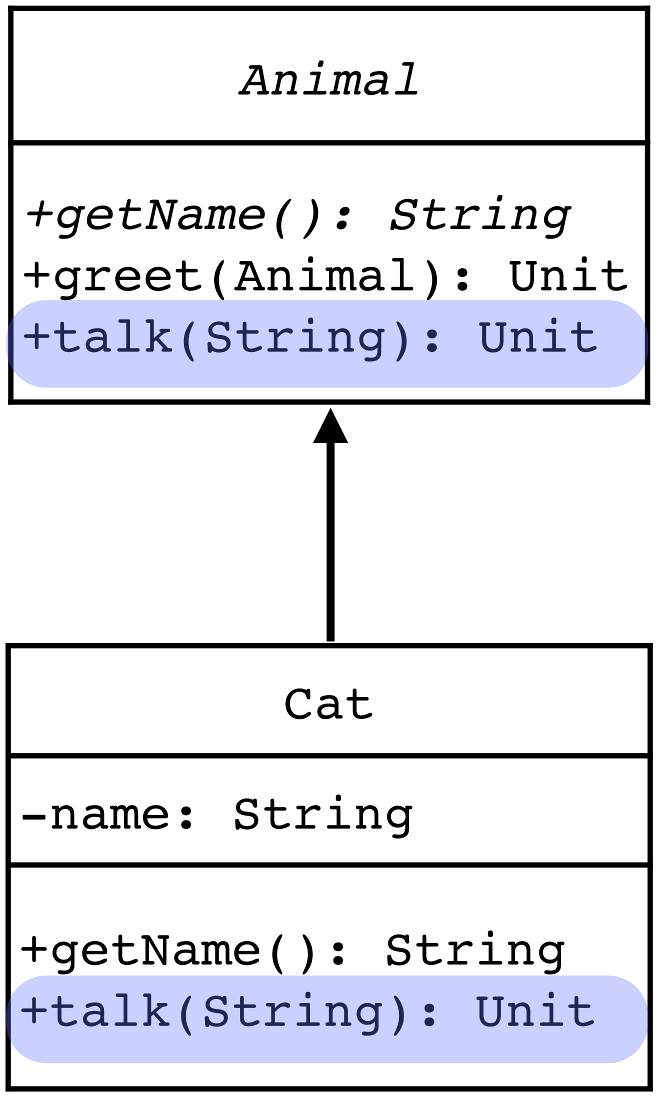
Outline

1. Overriding
2. Overloading
3. This and super pseudo variables

Outline

1. **Overriding**
2. Overloading
3. This and super pseudo variables

Method overriding



Method signature =
name + number of parameters +
type of its parameters

method overriding =
An instance method in a subclass with the *same signature* as an instance method in the superclass *overrides* the superclass's method.

A *message send* is always associated to a method signature. The signature is used to find the method to execute.

Method overriding

```
abstract class Animal{
  def getName(): String
  def greet(a: Animal): Unit = {
    talk("Good morning “+a.getName())
  }
  def talk(msg: String): Unit = println(msg)
}
```

```
class Cat(name: String) extends Animal{
  def getName(): String = name
  def talk(msg: String) = {
    println(msg.replaceAll("[a-z]+", "meow"))
  }
}
```

In some languages this is ok… but it is a bad practice.
Other languages force you to **explicitly mark** the operation as an **override**

Method overriding

```
abstract class Animal{
  def getName(): String
  def greet(a: Animal): Unit = {
    talk("Good morning "+a.getName())
  }
  def talk(msg: String): Unit = println(msg)
}
```

```
class Cat(name: String) extends Animal{
  def getName(): String = name
  override def talk(msg: String) = {
    println(msg.replaceAll("[a-z]+", "meow"))
  }
}
```

Method overriding

```
abstract class Animal{
  def getName(): String
  def greet(a: Animal): Unit = {
    talk("Good morning "+a.getName())
  }
  def talk(msg: String): Unit = println(msg)
}
```

```
class Cat(name: String) extends Animal{
  def getName(): String = name
  override def talk(msg: String) = {
    println(msg.replaceAll("[a-z]+", "meow"))
  }
}
```

```
val a: Cat = new Cat("Blair")
a.talk("give me food")
```

meow meow meow

```
val a: Animal = new Cat("Blair")
a.talk("give me food")
```

meow meow meow

We check at compile time if this makes sense… but at runtime we check which method to invoke

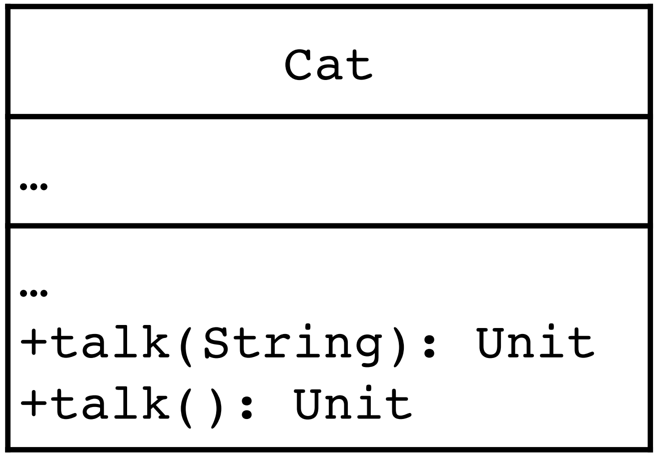
Outline

1. Overriding
2. **Overloading**
3. This and super pseudo variables

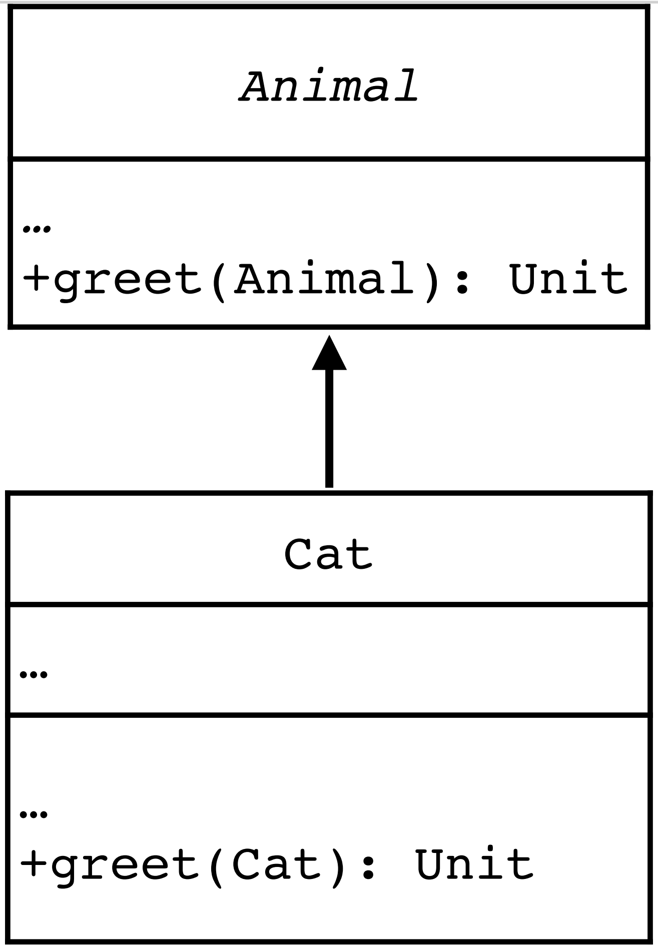
Method overloading

Two methods defined in a hierarchy

can have the same name but different signatures



Different **arity**



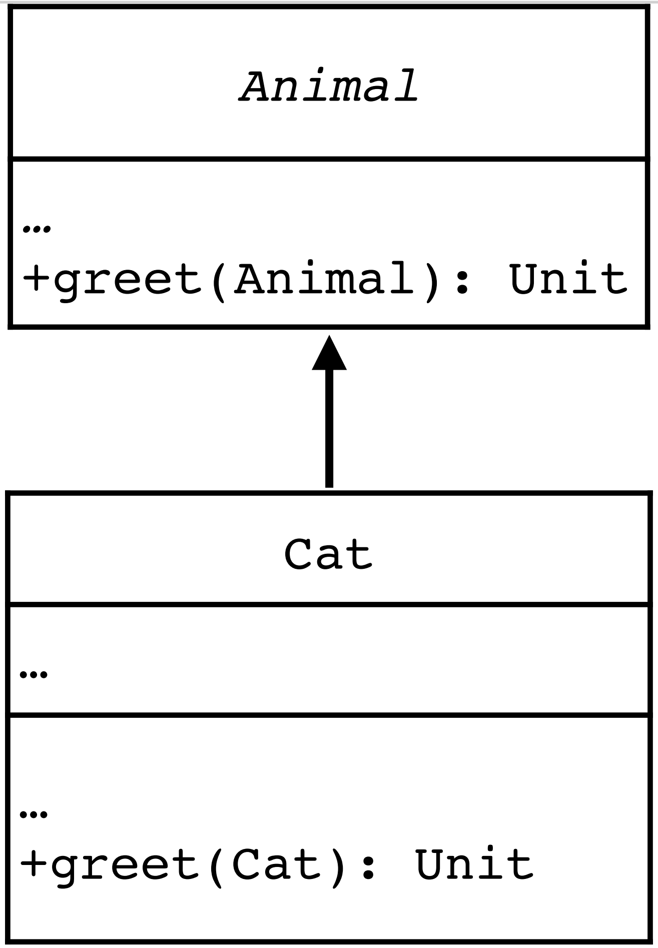
Different **types**

Method overloading

Two methods defined in a hierarchy

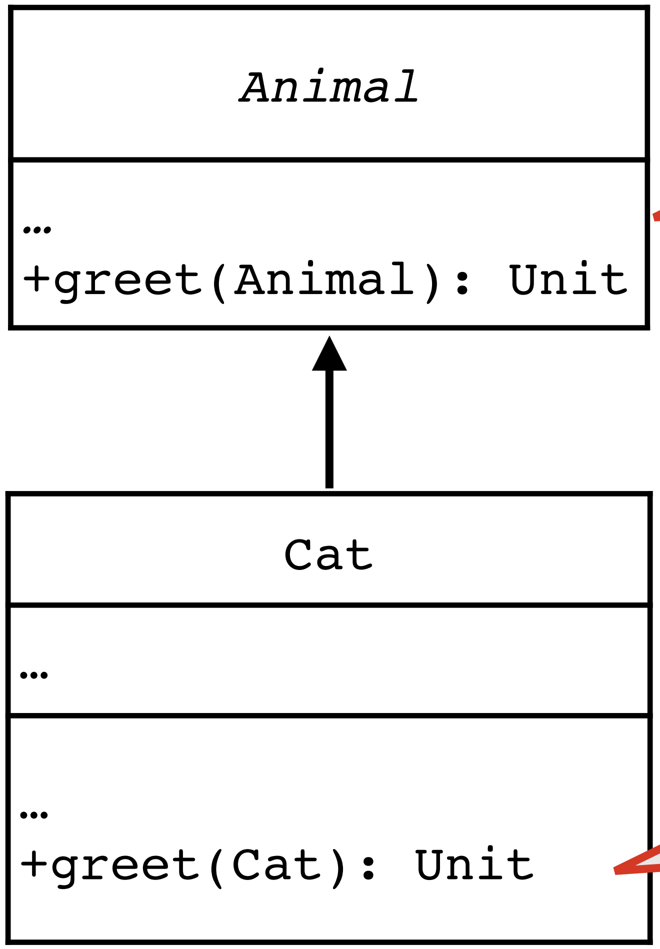
can have the same name but different signatures

Could generate complicated bugs



Different **types**

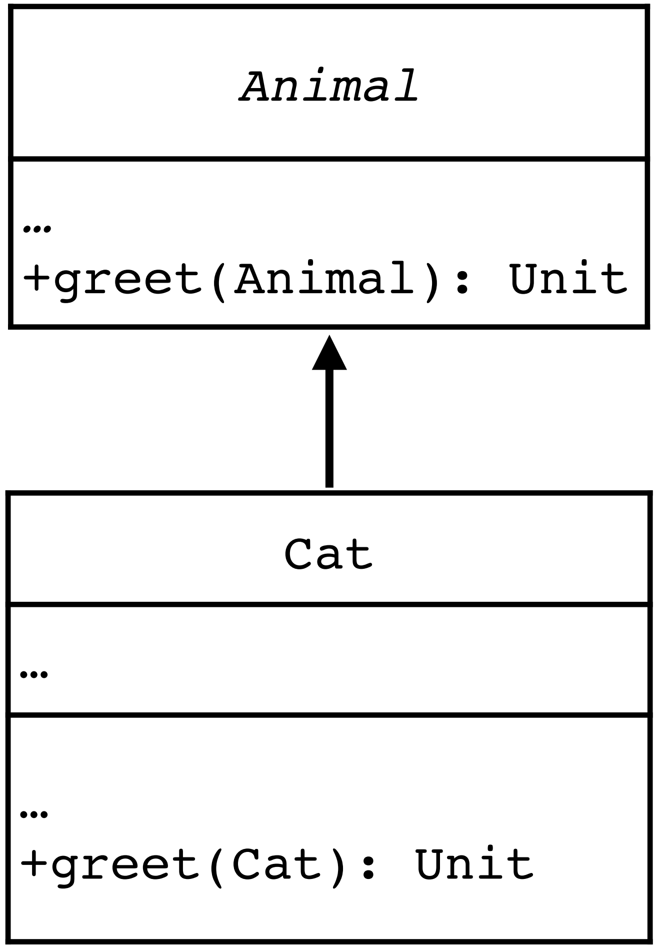
A typical situation



```
def greet(a: Animal): Unit = {
  talk("Good morning " + a.getName())
}
```

```
def greet(a: Cat): Unit = {
  println("Hello fellow cat")
}
```

A typical situation

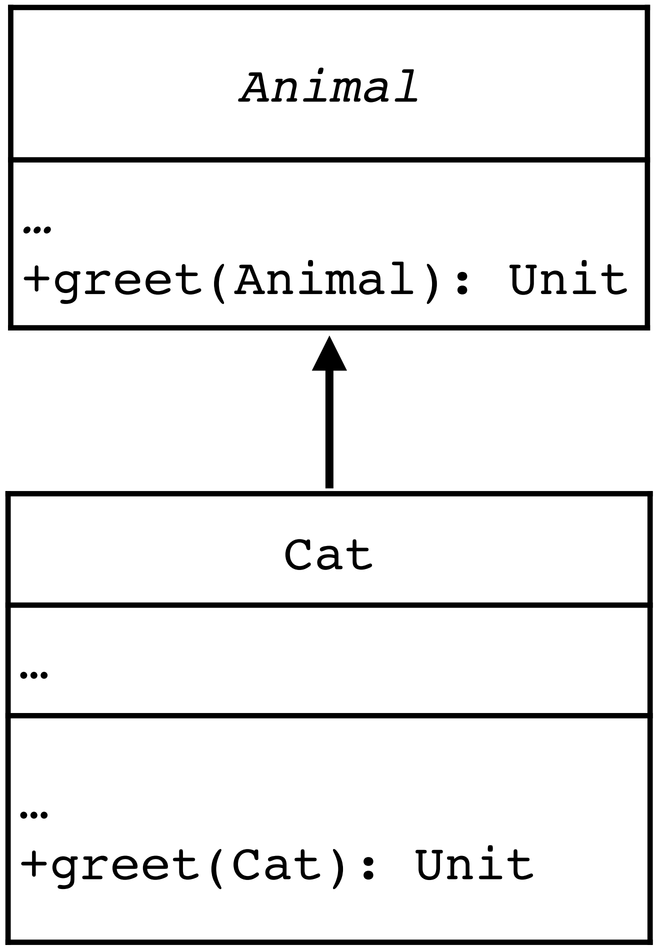


```
val c1: Cat = new Cat("Blair")
val c2: Cat = new Cat("Luna")

c1.greet(c2)
```

Hello fellow cat

A typical situation

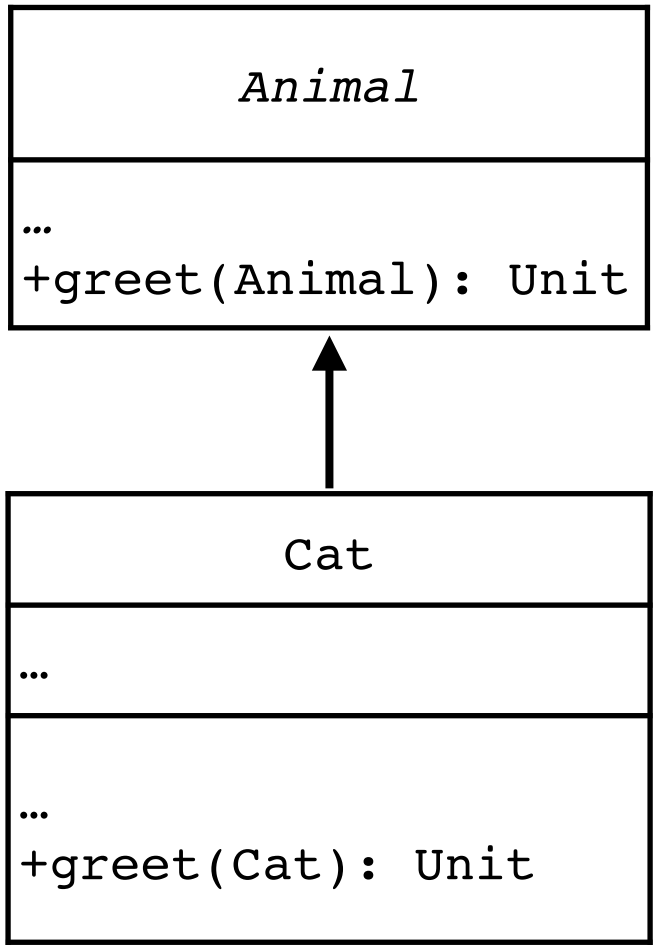


```
val c1: Animal = new Cat("Blair")
val c2: Cat = new Cat("Luna")

c1.greet(c2)
```

Good morning Luna

A typical situation



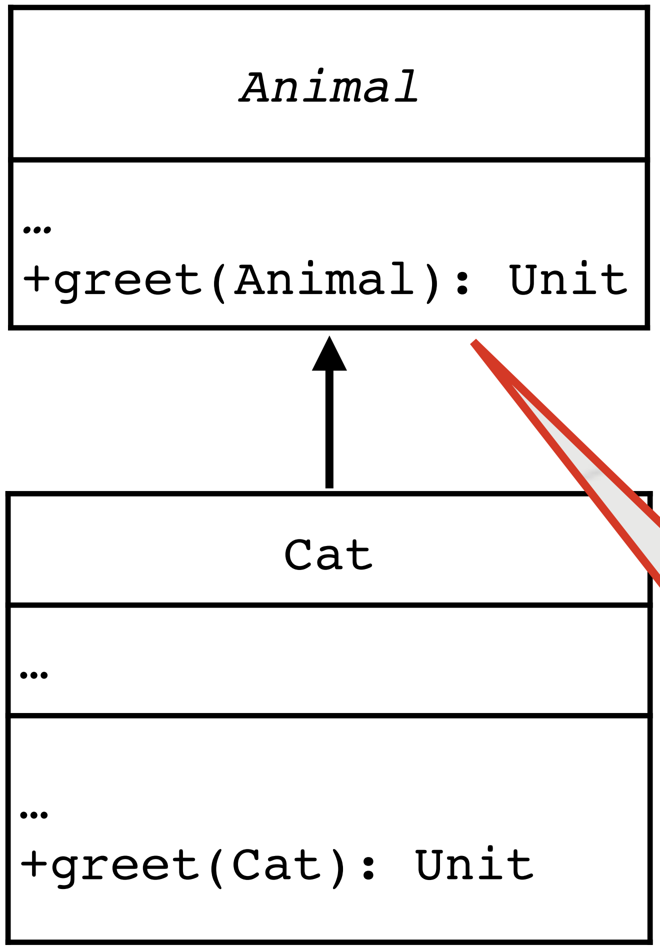
```
val c1: Cat = new Cat("Blair")
val c2: Animal = new Cat("Luna")

c1.greet(c2)
```

Good morning Luna

CIENCIAS DE LA COMPUTACIÓN
UNIVERSIDAD DE CHILE

Overloading is resolved STATICALLY



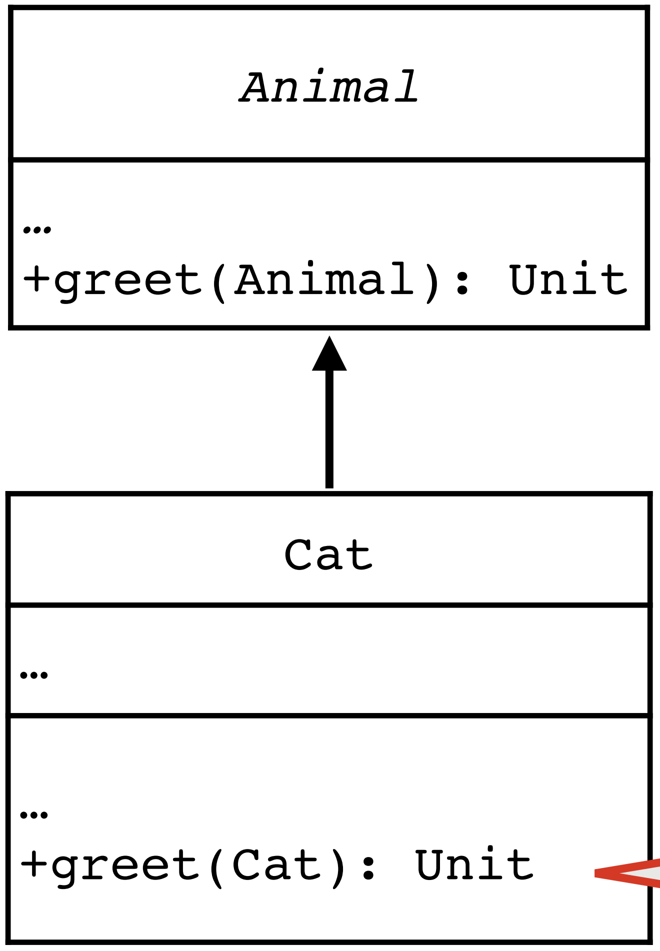
```
val c1: Cat = new Cat("Blair")
val c2: Animal = new Cat("Luna")

c1.greet(c2)
```

greet(Animal)

```
def greet(a: Animal): Unit = {
  talk("Good morning " + a.getName())
}
```

Overloading is resolved STATICALLY



```
val c1: Cat = new Cat("Blair")
val c2: Animal = new Cat("Luna")

c1.greet(c2)
```

greet(Animal)

Si la intención era hacer override…

```
override def greet(a: Cat): Unit =
```

… no hubiera compilado

Exercise #1

```
class AnimalClassifier {
  def classify(s: Dog): String = {
    "Dog"
  }
  def classify(l: Cat): String = {
    "Cat"
  }
  def classify(c: Animal): String = {
    "Unknown Animal"
  }
}
val cl = new AnimalClassifier();
val animals: List[Animal] = List(
  new Cat("C"),
  new Dog("D"),
  new Cat("F"));

for (a <- animals)
  println(cl.classify(a));
```

What does this program prints?

It prints:

```
Unknown Animal
Unknown Animal
Unknown Animal
```

a has declared type Animal

The actual (runtime) type of an objet is not used

Exercise #2: equals

```
class Cat(name: String) extends Animal{
  …
  override def equals(o: Animal): Boolean = {
    if(o.isInstanceOf[Cat]){
      val otherCat = o.asInstanceOf[Cat]
      this.name == otherCat.name
    } else false
  }
}
```

Is this ok?

No!, this overrides nothing

Exercise #2: equals

Is this ok?

```
class Cat(name: String) extends Animal{
  …
  override def equals(o: Any): Boolean = {
    if(o.isInstanceOf[Cat]){
      val otherCat = o.asInstanceOf[Cat]
      this.name == otherCat.name
    } else false
  }
}
```

Exercise #2: equals

```scala
class Cat(name: String) extends Animal{
  …
  override def equals(o: Any): Boolean = {
    if(o.isInstanceOf[Cat]){
      val otherCat = o.asInstanceOf[Cat]
      this.name == otherCat.name
    } else false
  }
}
```

```scala
class Persian(name: String) extends Cat(name)
val c = new Cat("Luna")
val p = new Persian("Luna")
println(p.equals(c))
```

true!!

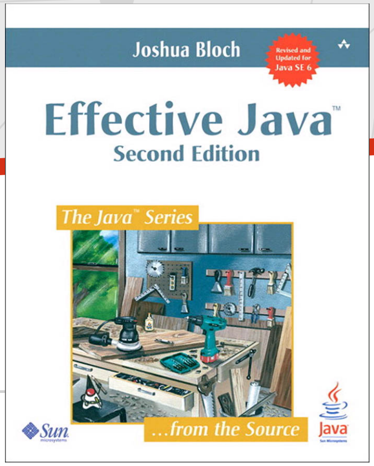
Exercise #2: equals

```
class Cat(name: String) extends Animal{
  …
  override def equals(o: Any): Boolean = {
    if(this.getClass().getName == o.getClass().getName){
      val otherCat = o.asInstanceOf[Cat]
      this.name == otherCat.name
    } else false
  }
}
```

```
class Persian(name: String) extends Cat(name)
val c = new Cat("Luna")
val p = new Persian("Luna")
println(p.equals(c))
```

false!!

#41 Use overloading Judiciously



Beware of overloading

> avoid “confusing” uses of overloading

> not confusing:

different arity

types are “unrelated” (none can be seen as a subtype of the other)

Outline

1. Overriding
2. Overloading
3. **This and super pseudo variables**

This and Super

the *this* pseudo-variable always refers to the object receiver

the *super* pseudo-variable always refers to the object receiver

a message sent to *super* makes the lookup begins in the superclass of the class in which the call is written

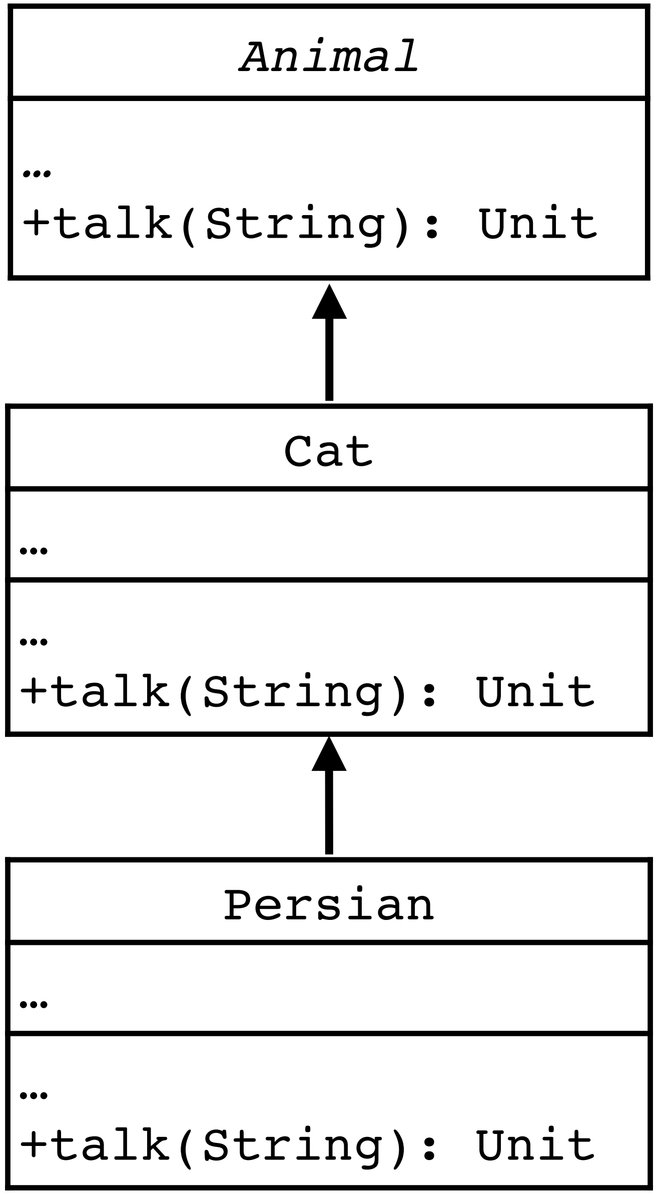
The Scala syntax prevents one from using super without being followed by ".identifier"

Class inheritance principle

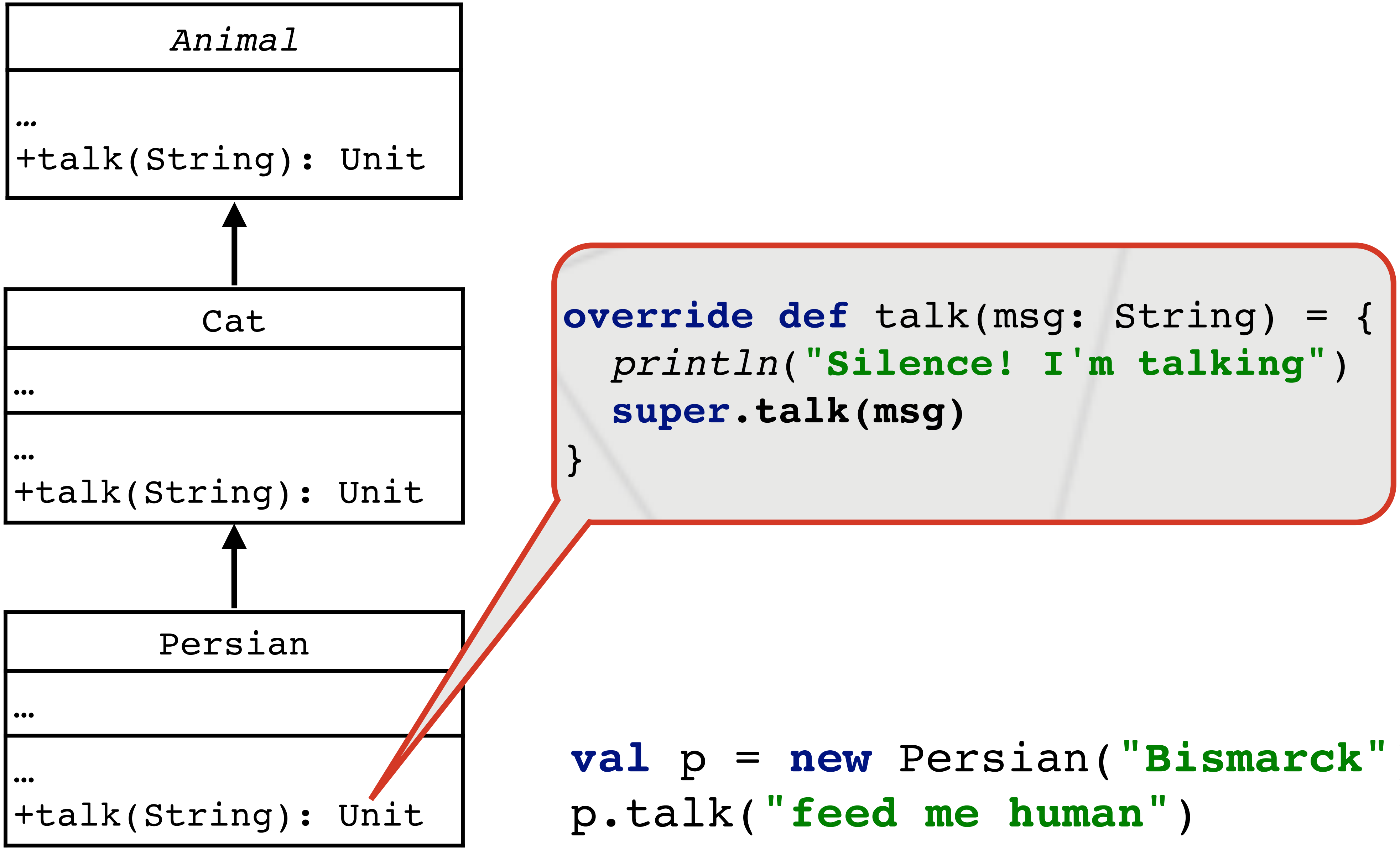
Sending a message to an object triggers a *lookup* along the *class hierarchy* of the class of the object

In a statically typed languages (e.g., Scala, Java, C#, C++), the lookup *always* find an appropriate method

This may not be the case in a dynamically typed language (e.g., Python, Ruby, Pharo, JavaScript)

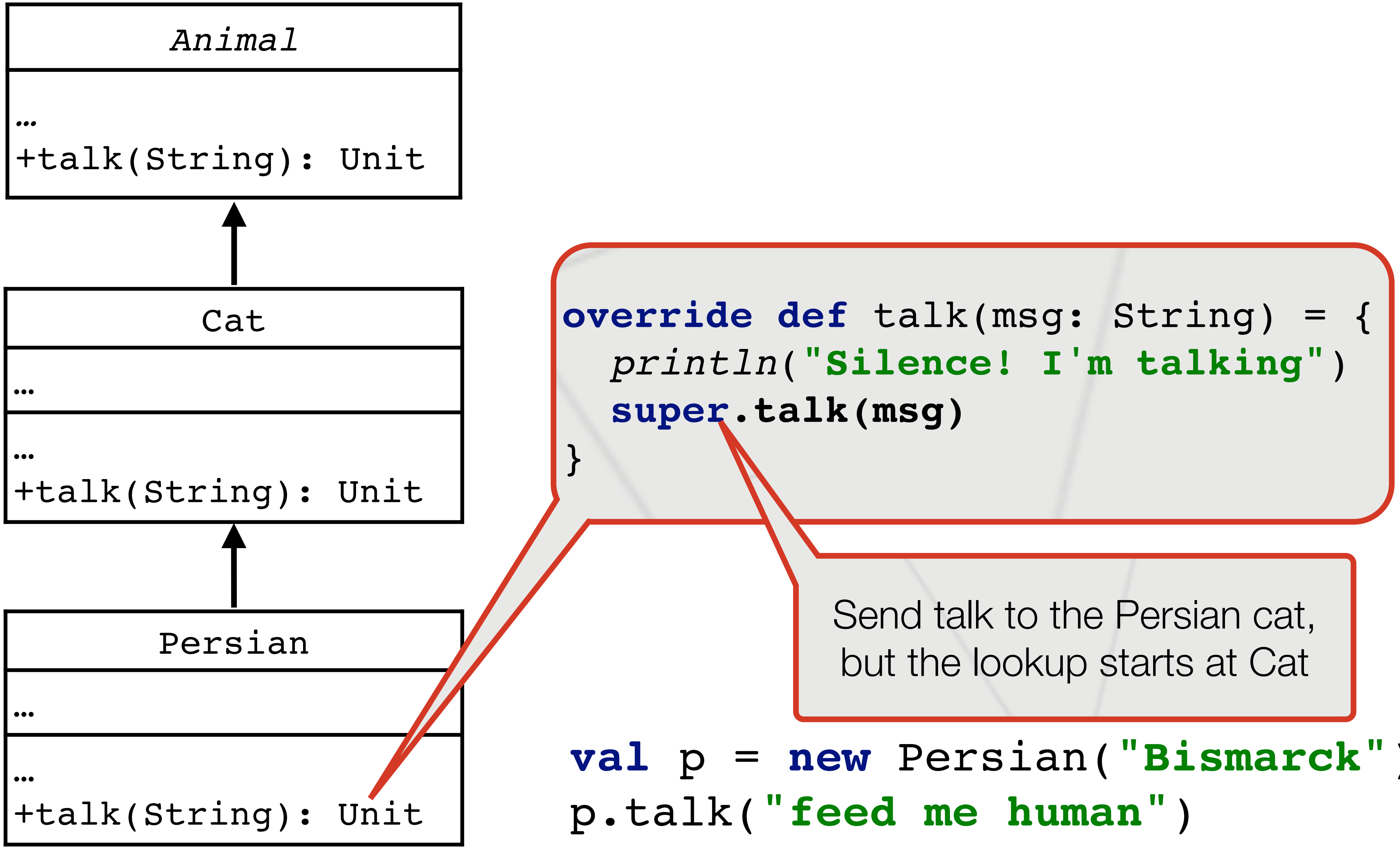


```
val p = new Persian("Bismarck")
p.talk("feed me human")
```



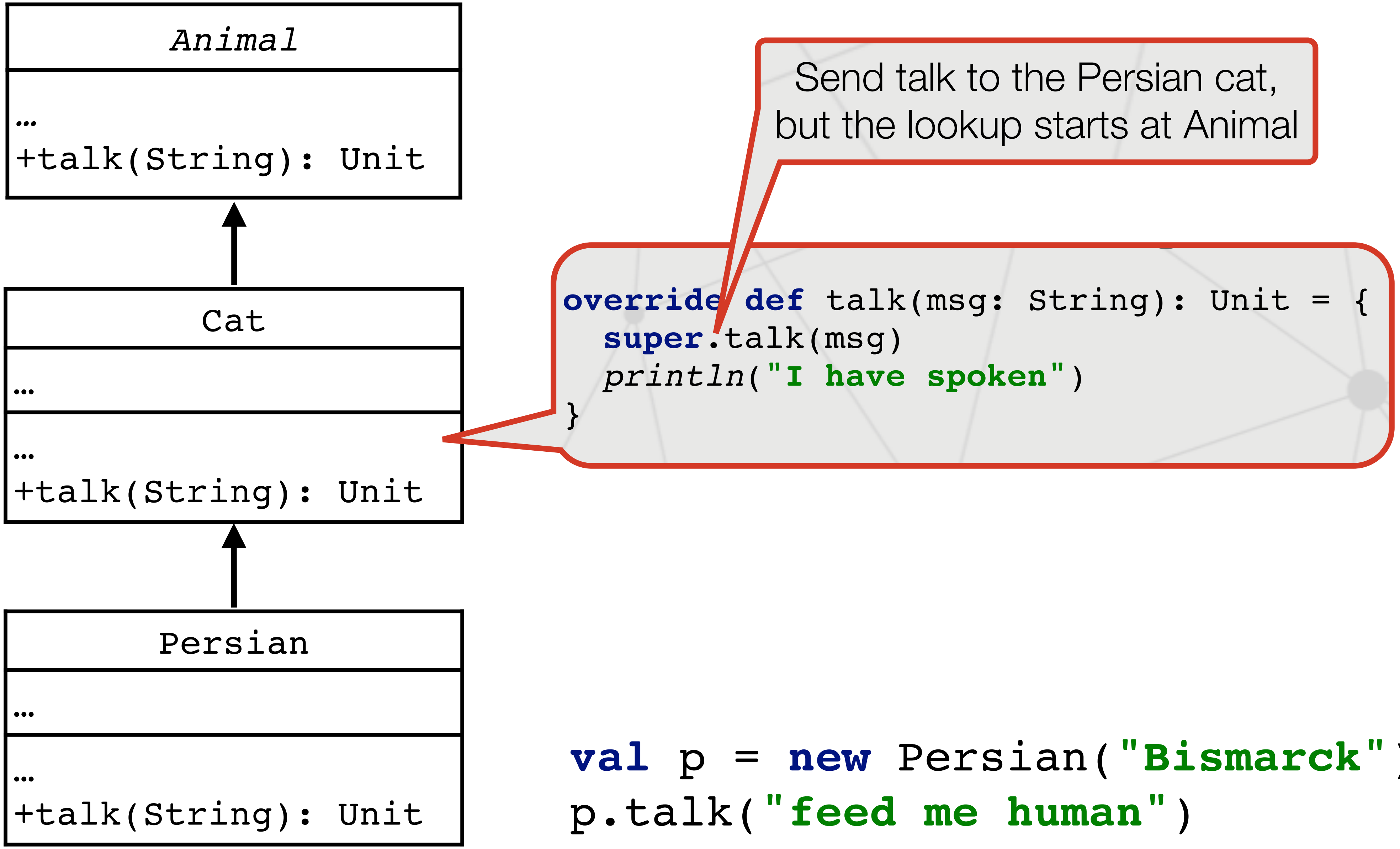
```
override def talk(msg: String) = {
  println("Silence! I'm talking")
  super.talk(msg)
}
```

```
val p = new Persian("Bismarck")
p.talk("feed me human")
```



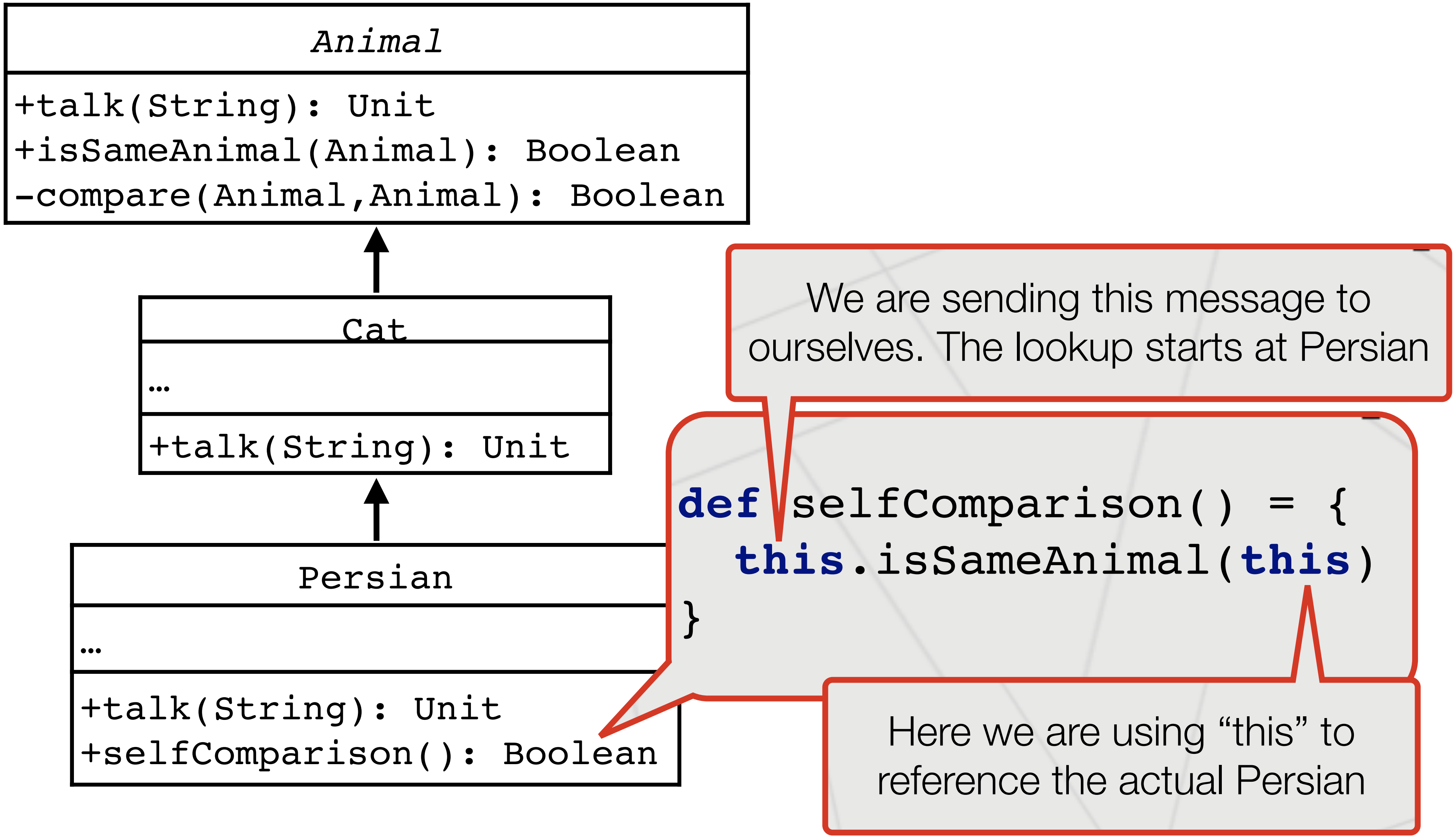
```
override def talk(msg: String) = {
  println("Silence! I'm talking")
  super.talk(msg)
}
```

```
val p = new Persian("Bismarck")
p.talk("feed me human")
```



```
override def talk(msg: String): Unit = {
  super.talk(msg)
  println("I have spoken")
}
```

```
val p = new Persian("Bismarck")
p.talk("feed me human")
```

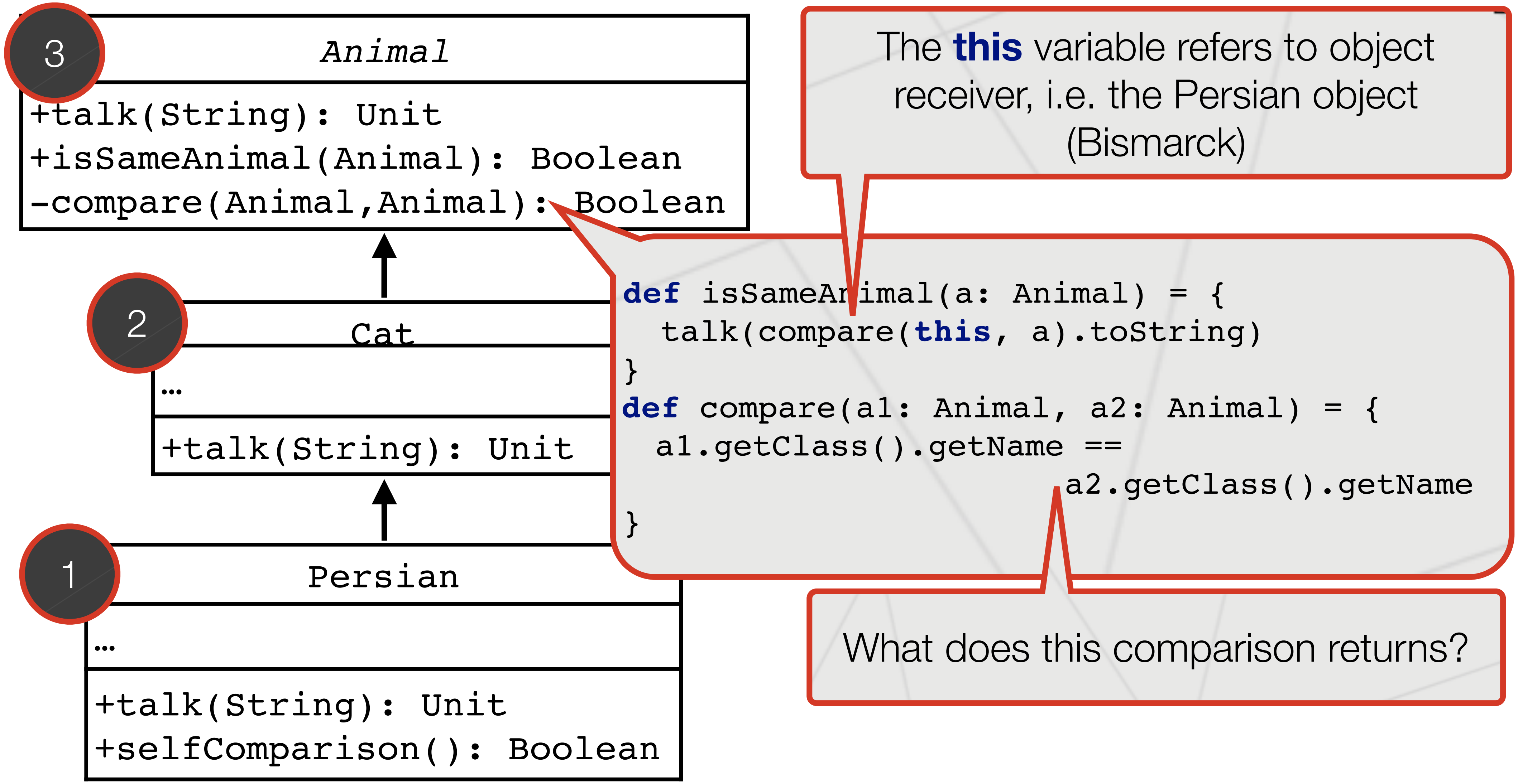

Animal
+talk(String): Unit
+isSameAnimal(Animal): Boolean
-compare(Animal,Animal): Boolean
Cat
…
+talk(String): Unit
Persian
…
+talk(String): Unit
+selfComparison(): Boolean

We are sending this message to ourselves. The lookup starts at Persian

```
def selfComparison() = {
  this.isSameAnimal(this)
}
```

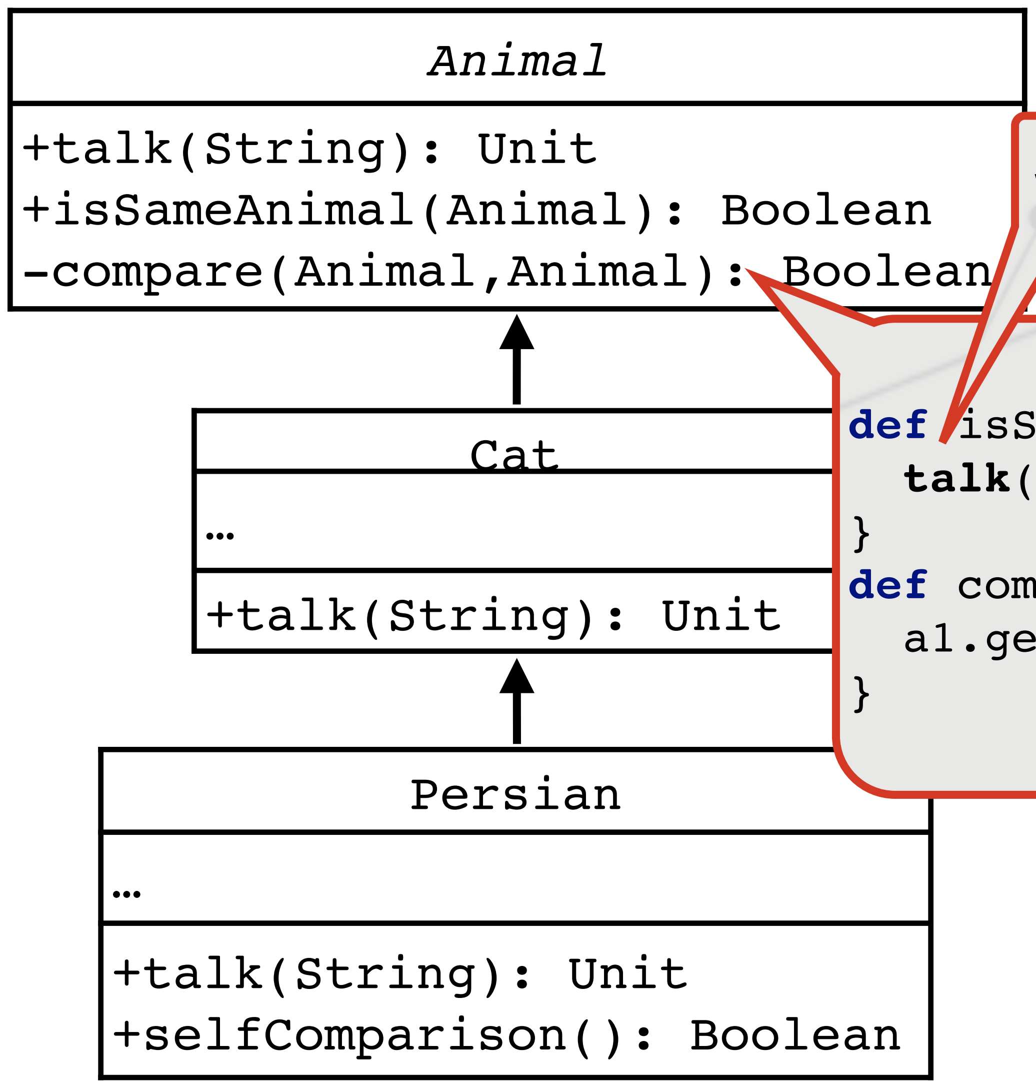
Here we are using “this” to reference the actual Persian

```
val p = new Persian("Bismarck")
p.selfComparison()
```


3
Animal
+talk(String): Unit
+isSameAnimal(Animal): Boolean
-compare(Animal,Animal): Boolean
2
Cat
…
+talk(String): Unit
1
Persian
…
+talk(String): Unit
+selfComparison(): Boolean
The this variable refers to object receiver, i.e. the Persian object (Bismarck)
What does this comparison returns?

```
def isSameAnimal(a: Animal) = {
  talk(compare(this, a).toString)
}
def compare(a1: Animal, a2: Animal) = {
  a1.getClass().getName ==
                               a2.getClass().getName
}
```

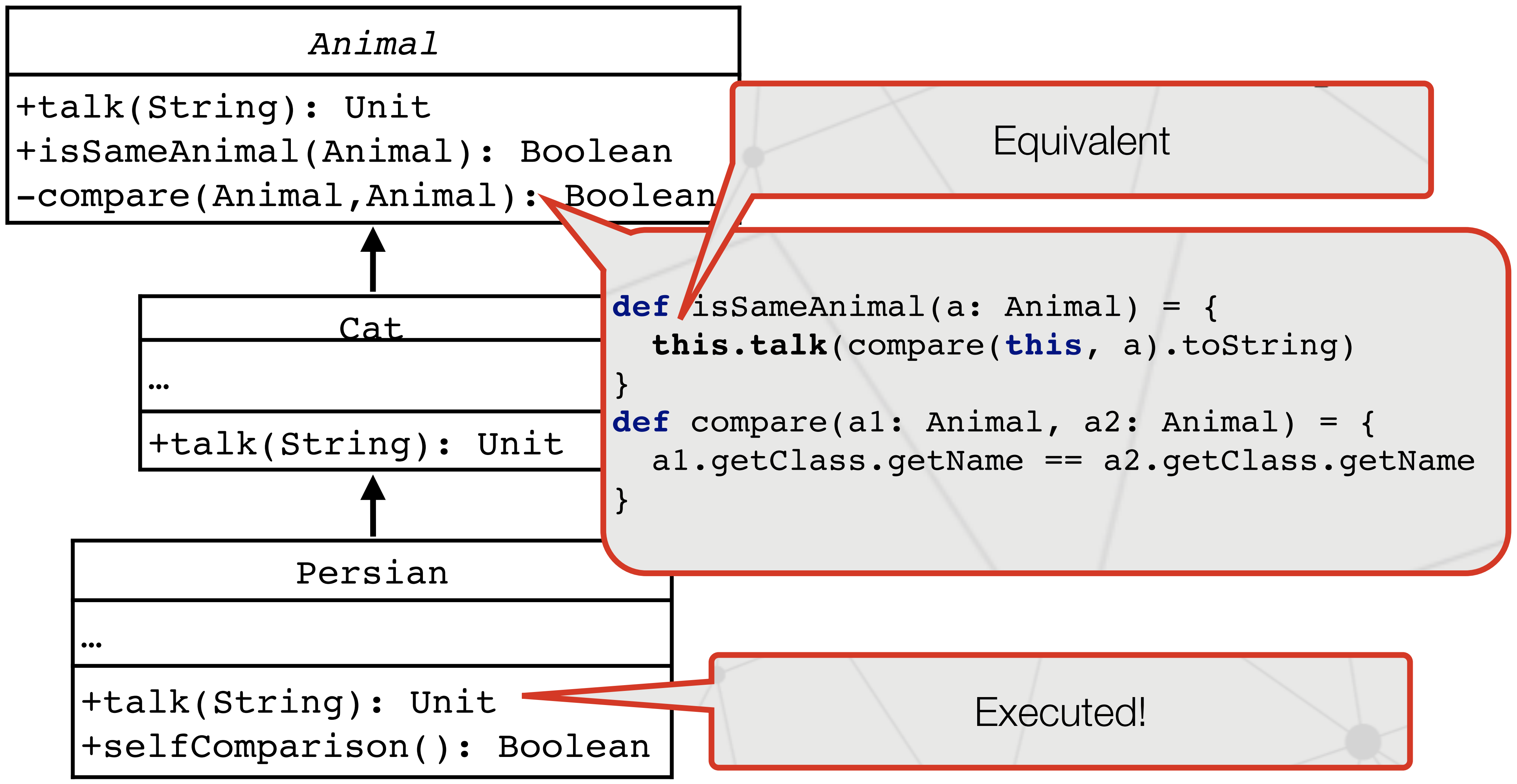
```
val p = new Persian("Bismarck")
p.selfComparison()
```


Animal
+talk(String): Unit
+isSameAnimal(Animal): Boolean
-compare(Animal,Animal): Boolean
Cat
…
+talk(String): Unit
Persian
…
+talk(String): Unit
+selfComparison(): Boolean

What version of talk are we executing?

```
def isSameAnimal(a: Animal) = {
  talk(compare(this, a).toString)
}
def compare(a1: Animal, a2: Animal) = {
  a1.getClass.getName == a2.getClass.getName
}
```

```
val p = new Persian("Bismarck")
p.selfComparison()
```



```scala
def isSameAnimal(a: Animal) = {
  this.talk(compare(this, a).toString)
}
def compare(a1: Animal, a2: Animal) = {
  a1.getClass.getName == a2.getClass.getName
}
```

```scala
val p = new Persian("Bismarck")
p.selfComparison()
```

```
class A {
  def foo(): Unit = {
    println("A.foo()")
    this.bar()
  }

  def bar(): Unit = {
    println("A.bar()")
  }
}
```

```
class B extends A {
  override def foo(): Unit = {
    super.foo()
  }

  override def bar(): Unit = {
    println("B.bar()")
  }
}
```

what `new B().foo()` prints?

```
class A {
  def test1(): Boolean = {
    super.equals(this)
  }

  def yourself(): A = {
    this
  }
}
```

```
class B extends A {
  def test2(): Boolean = {
    super.yourself() == this
  }

  def test3(): Boolean = {
    super.equals(super.yourself())
  }
}
```

```
new B().test1(), new B().test2(), new B().test3()  ??
```

```scala
class A {
  def test(): Boolean = {
    super.getClass() == this.getClass()
  }
}

class B extends A {}

object B {
  def main(arg: Array[String]): Unit = {
    println(new B().test())
  }
}
```

License

Attribution-ShareAlike 4.0 International (CC BY-SA 4.0)
You are free to:
- Share: copy and redistribute the material in any medium or format
- Adapt: remix, transform, and build upon the material for any purpose, even commercially

The licensor cannot revoke these freedoms as long as you follow the license terms

Attribution: you must give appropriate credit

ShareAlike: if you remix, transform, or build upon the material, you must distribute your contributions under the same license as the original

Complete license: https://creativecommons.org/licenses/by-sa/4.0/