CS/ENGRD 2110 FALL 2014

Lecture 6: Consequence of type, casting; function equals

http://courses.cs.cornell.edu/cs2110

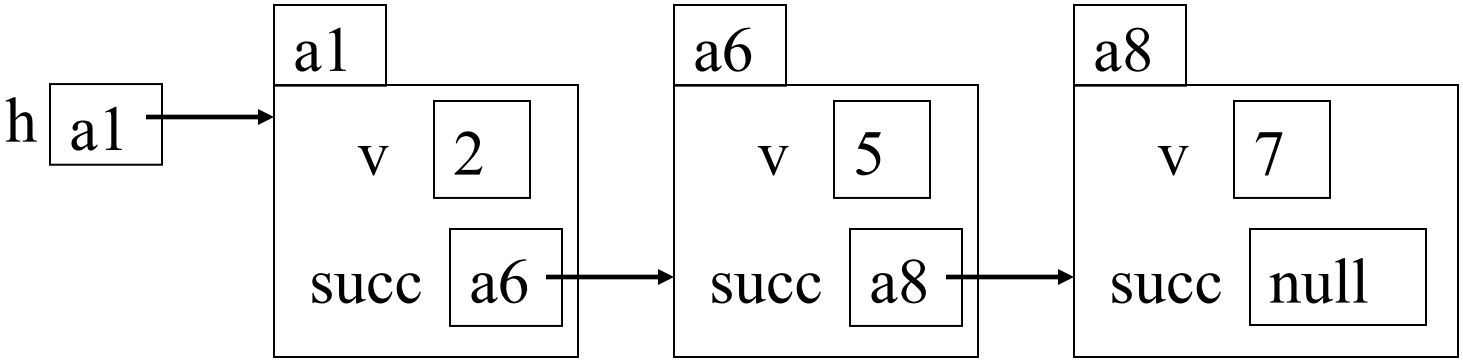
About prelim 1

- October 2: 5:30PM and at 7:30PM. We will tell you which to attend.
- Problems with that one? You go to the other one.
- Students with conflicts --having two prelims at 7:30 at that evening
  Take ours at 5:30 OR (take ours at 7:30 AND take makeup that the other class)

  New to Cornell? It is standard practice to take 2 prelims one evening
- People who HAVE to be out of town should email us the particulars --later, not now.
- Anyone who misses the prelim will have their grade based on prelim 2 and the final. They will HAVE to take the final (may be optional, in a way to be explained in November).
- Please don't email us about prelim 1 now. Too early. Too much going on now for us to handle it. We'll let you know when.

Assignment A3: Doubly linked Lists

Idea: maintain a list (2, 5, 7) like this:



Easy to insert a node in the beginning!

n a6

Also, if we have a variable that contains a pointer to a node, it's easy to remove that node or insert another value before or after that node.

Overview ref in text and JavaSummary.pptx

- Quick look at arrays slide 50-55
- Casting among classes C.33-C.36 (not good) slide 34-41
- Consequences of the class type slide 34-41
- Operator instanceof slide 40
- Function equals slide 37-41

Homework. Learn about while/ for loops in Java. Look in text.

while (<bool expr>) { … } // syntax

for (**int** k= 0; k < 200; k= k+1) { … } // example

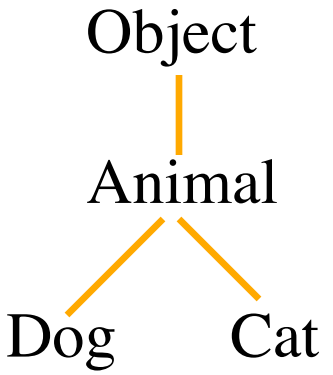
Classes we work with today

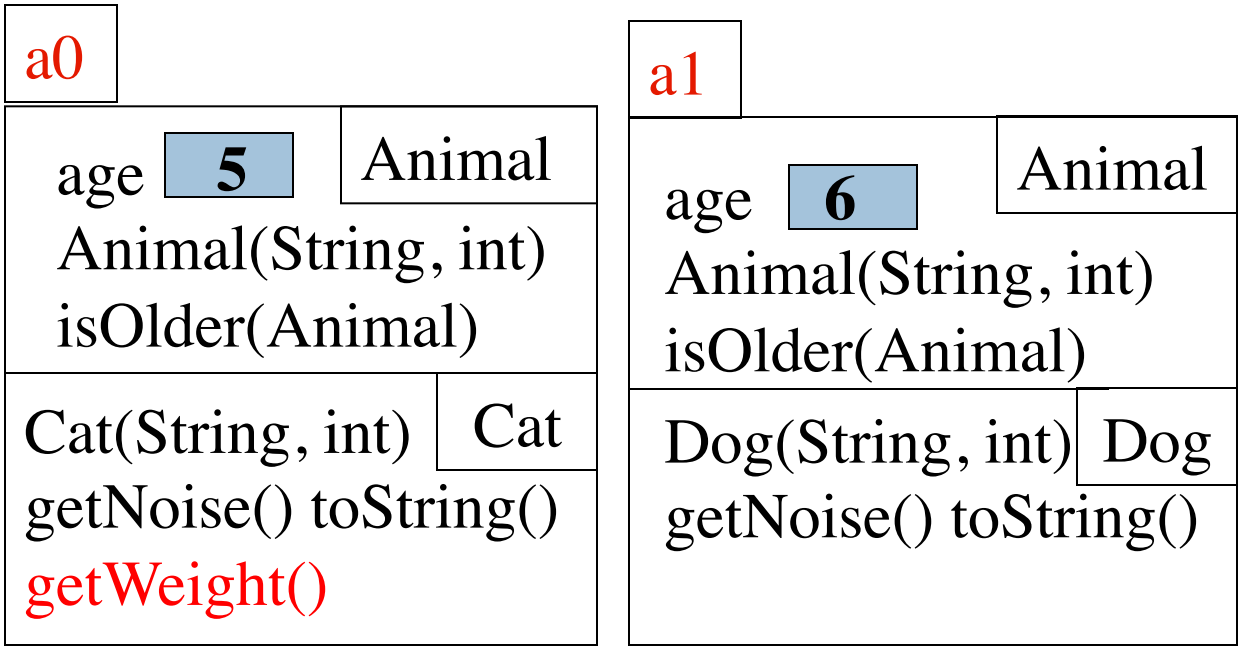
class hierarchy:

Work with a class Animal and subclasses like Cat and Dog

Put components common to animals in Animal

Object partition is there but not shown



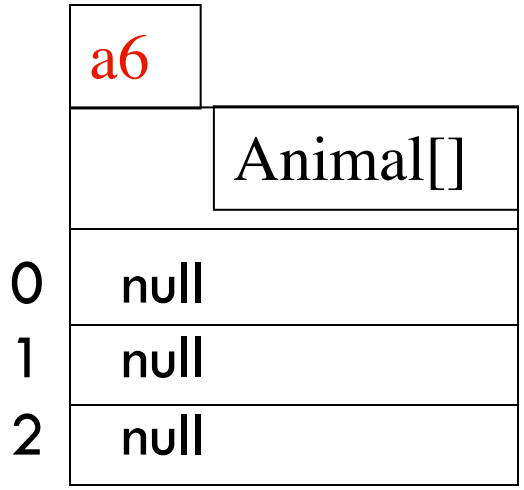


Animal[] v= **new** Animal[3];

declaration of array v

Create array of 3 elements

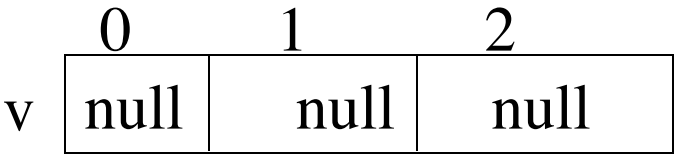
Assign value of new-exp to v



Assign and refer to elements as usual:

```
v[0]= new Animal(…);
…
a= v[0].getAge();
```

Sometimes use horizontal picture of an array:

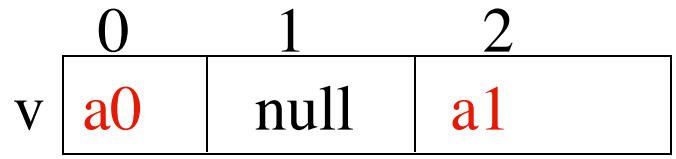


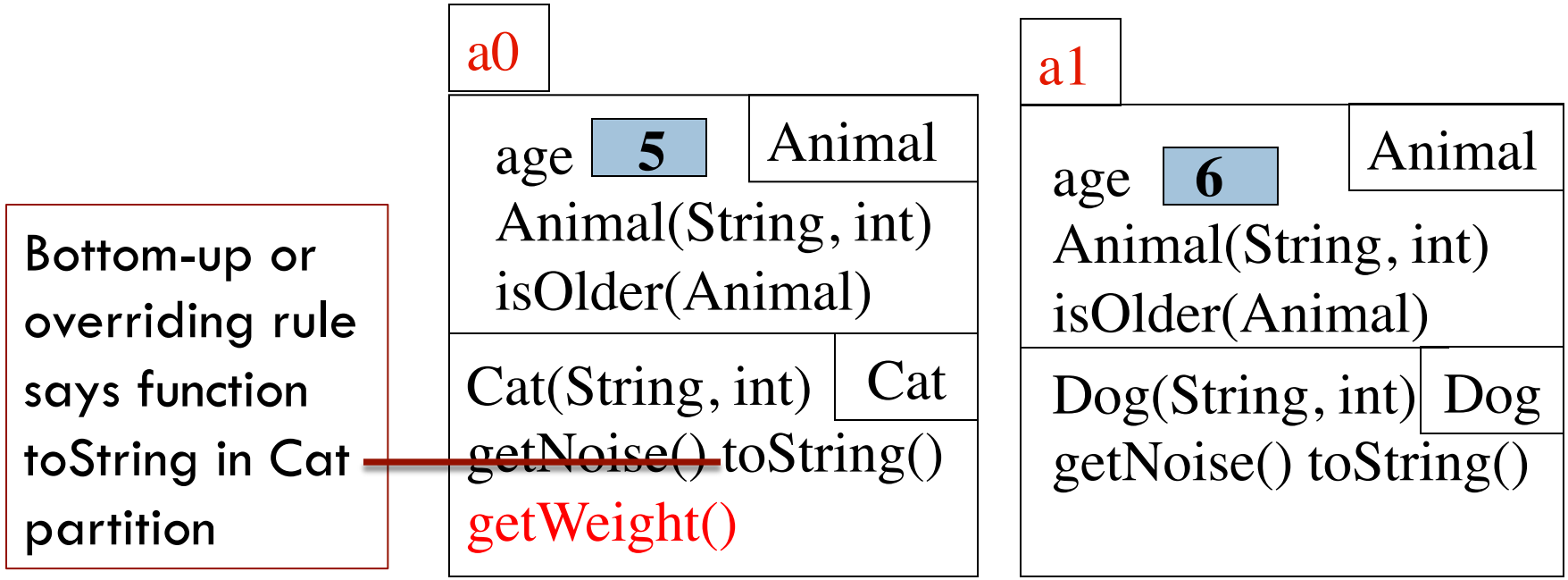
Which function is called?

Which function is called by

v[0].toString() ?

Remember, partition Object contains toString()

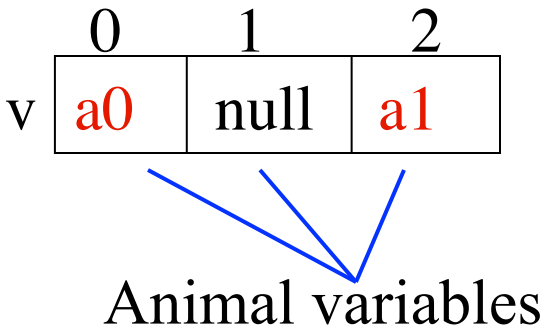




Consequences of a class type

Animal[] v; declaration of v. Also means that each variable v[k] is of type Animal

The type of v is Animal[]
The type of each v[k] is Animal
The type is part of the syntax/grammar of the language. Known at compile time.



As we see on next slide, the type of a class variable like v[k] determines what methods can be called

From an Animal variable, can use only methods available in class Animal

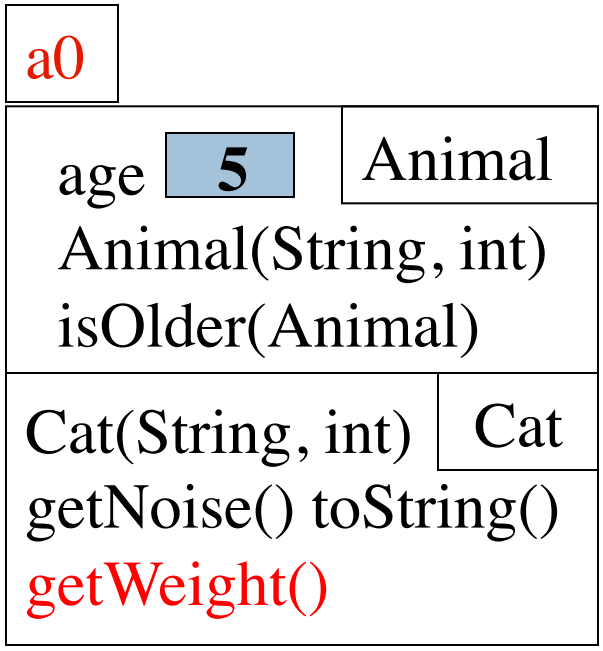
The same object a0, from the viewpoint of a Cat variable and an Animal variable

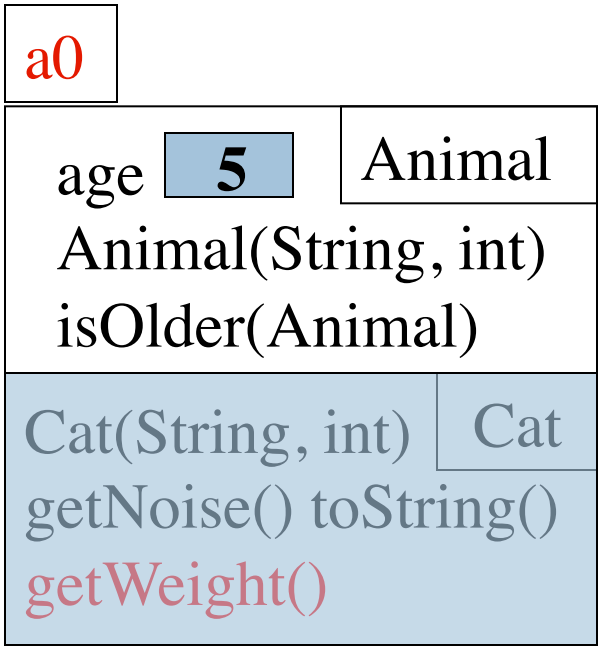
c a0 Cat

a a0 Animal

c.getWeight() is legal

a.getWeight() is illegal because getWeight Is not available in class Animal



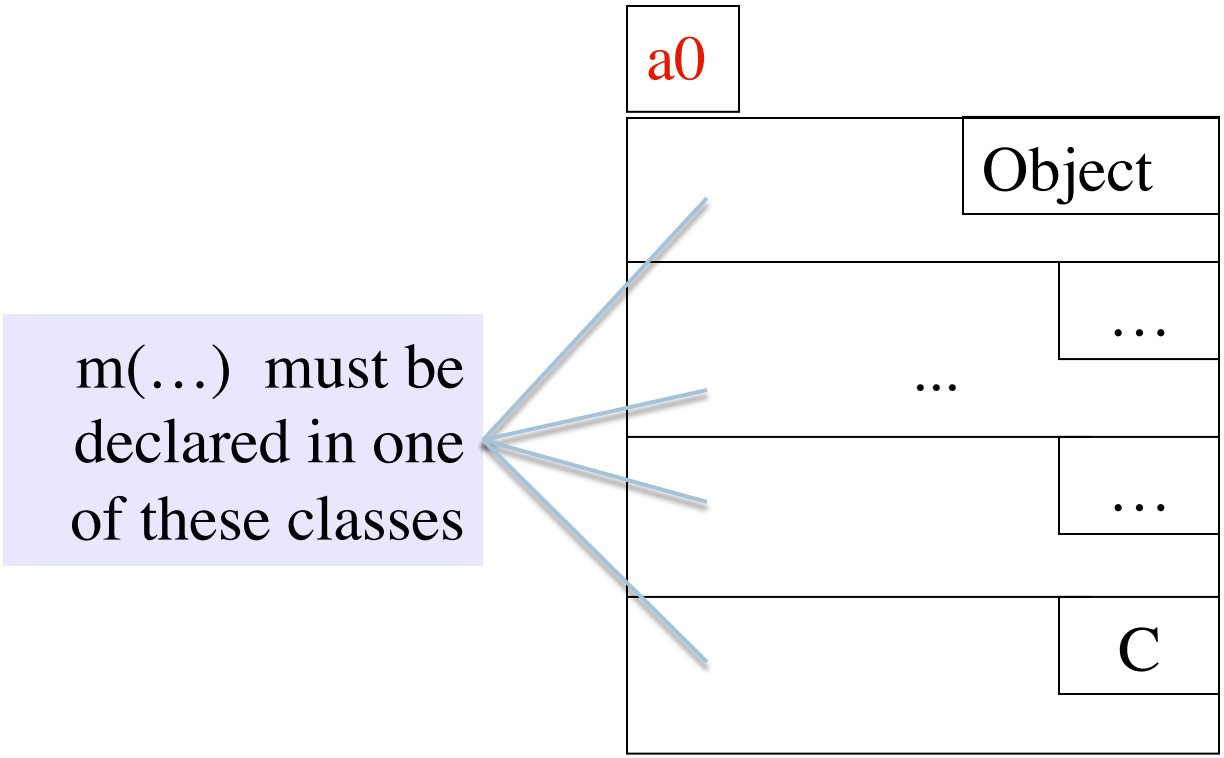


Rule for determining legality of method call

c a0

C

Rule: c.m(…) is legal and the program will compile ONLY if method m is declared in C or one of its superclasses



Another example

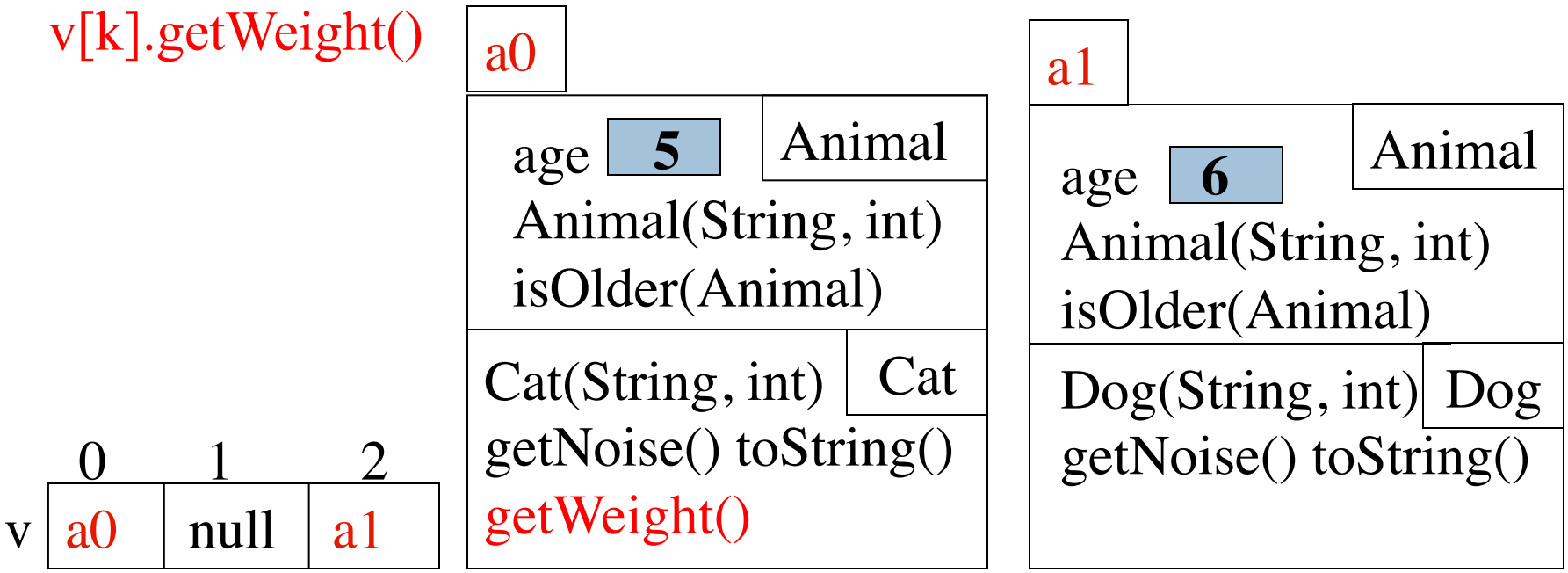
Type of v[0]: Animal

Should this call be allowed?
Should program compile?

v[k].getWeight()

Should this call be allowed?
Should program compile?

v[0].getWeight()



View of object based on the type

Each element v[k] is of type Animal.

From v[k], see only what is in partition Animal and partitions above it.

getWeight() not in class Animal or Object. Calls are illegal, program does not compile:

v[0].getWeight() v[k].getWeight()

Components are in lower partitions, but can't see them

	0	1	2
v	a0	null	a1

Animal

a0

age 5 | Animal

Animal(String, int)

isOlder(Animal)

Cat(String, int) | Cat

getNoise() toString()

getWeight()

a1

age 6 | Animal

Animal(String, int)

isOlder(Animal)

Dog(String, int) | Dog

getNoise() toString()

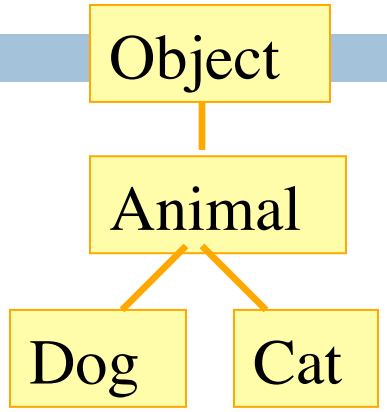
Casting up class hierarchy

You know about casts like

(int) (5.0 / 7.5)

(double) 6

double d= 5; // automatic cast

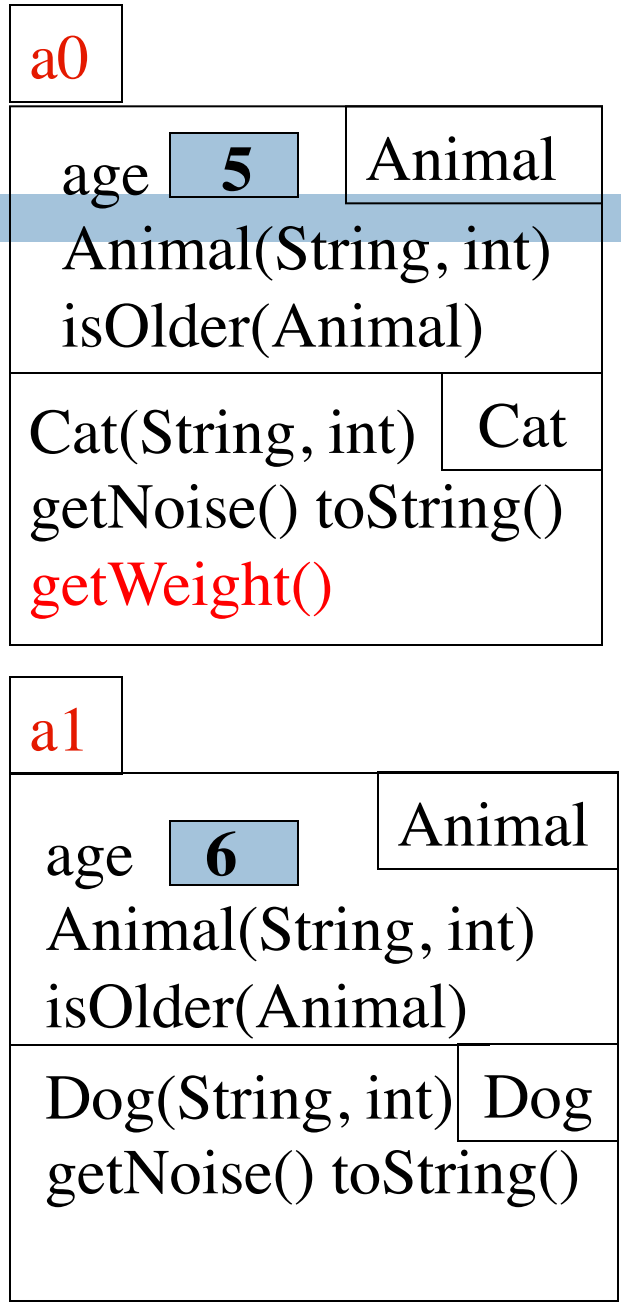


Discuss casts up/down class hierarchy.

Animal h= **new** Cat("N", 5);

Cat c= (Cat) h;

A class cast doesn't change the object. It just changes the perpective –how it is viewed!



Explicit casts: unary prefix operators

Rule: an object can be cast to the name of any partition that occurs within it — and to nothing else.

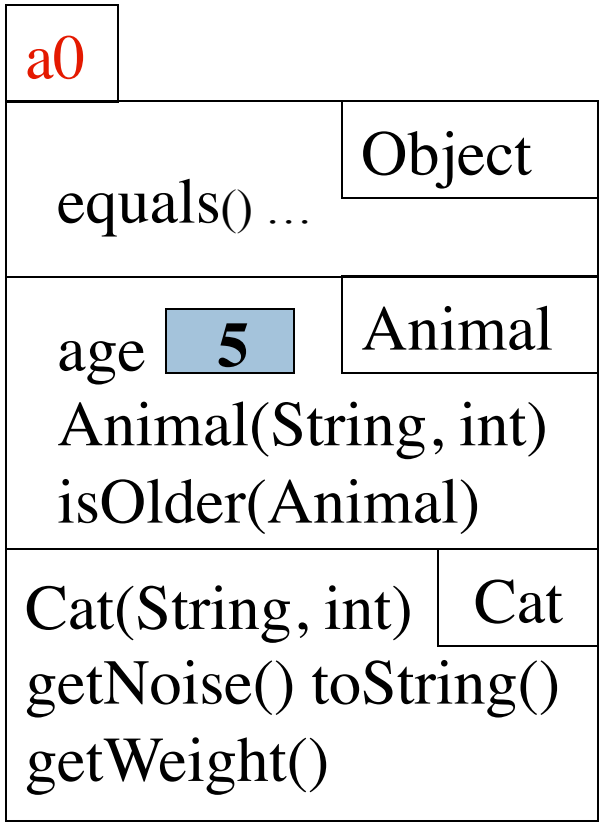
a0 maybe cast to Object, Animal, Cat.

An attempt to cast it to anything else causes an exception

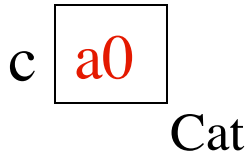
(Cat) c

(Object) c

(Animal) (Animal) (Cat) (Object) c



These casts don't take any time. The object does not change. It's a change of perception



Implicit upward cast

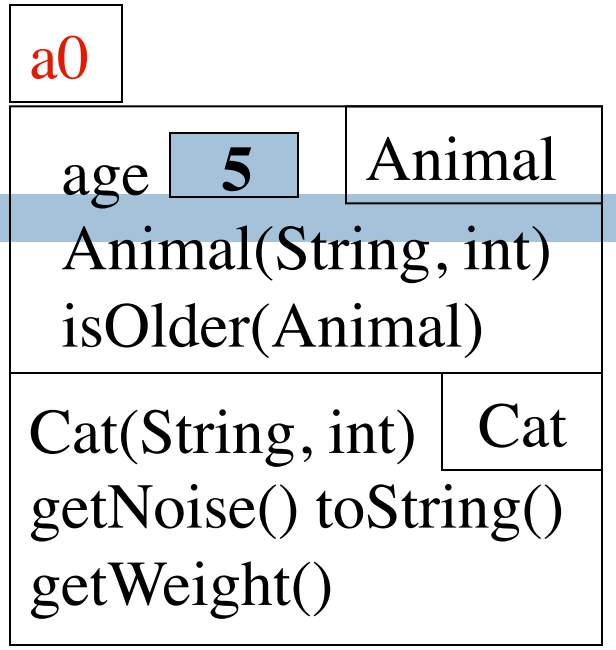

```
public class Animal {
    /** = "this Animal is older than h" */
    public boolean isOlder(Animal h) {
        return age > h.age;
    }
```

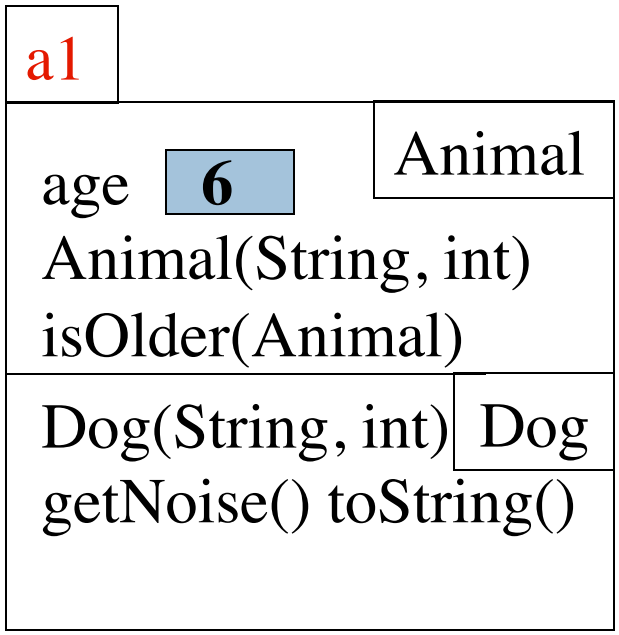
Call c.isOlder(d)

h is created. a1 is cast up to class Animal and stored in h

Upward casts done automatically when needed

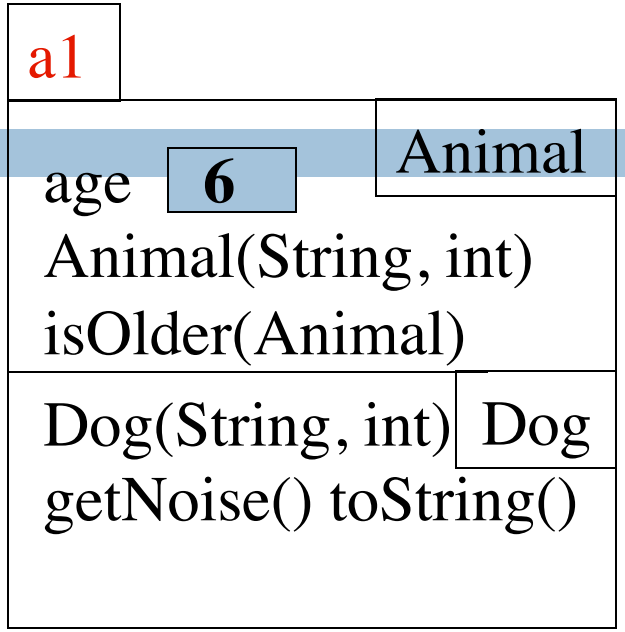
h a1 Animal c a0 Cat d a1 Dog





Example


```
public class Animal {
    /** = "this is older than h" */
    public boolean isOlder(Animal h) {
        return age > h.age;
    }
```



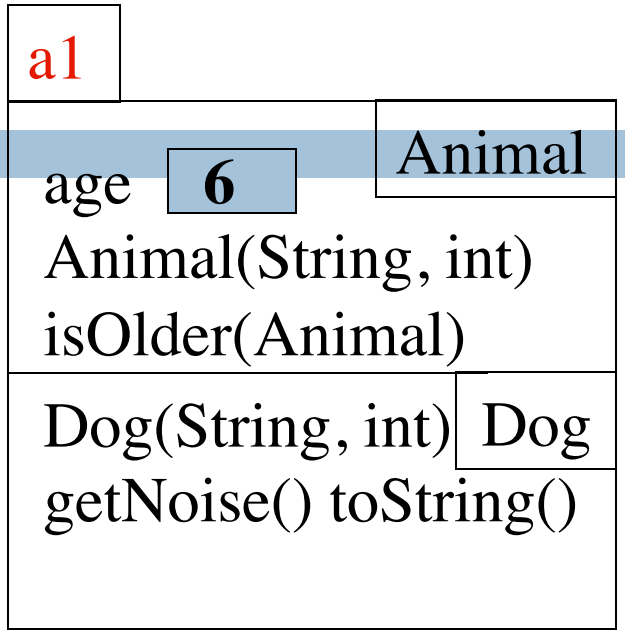
Type of h is Animal. Syntactic property.

Determines at compile-time what components can be used: those available in Animal

If a method call is legal, the overriding rule determines which method is called

Components used from h


```
public class Animal {
    /** = "this is older than h" */
    public boolean isOlder(Animal h) {
        return age > h.age;
    }
```



h.toString() OK —it's in class Object partition

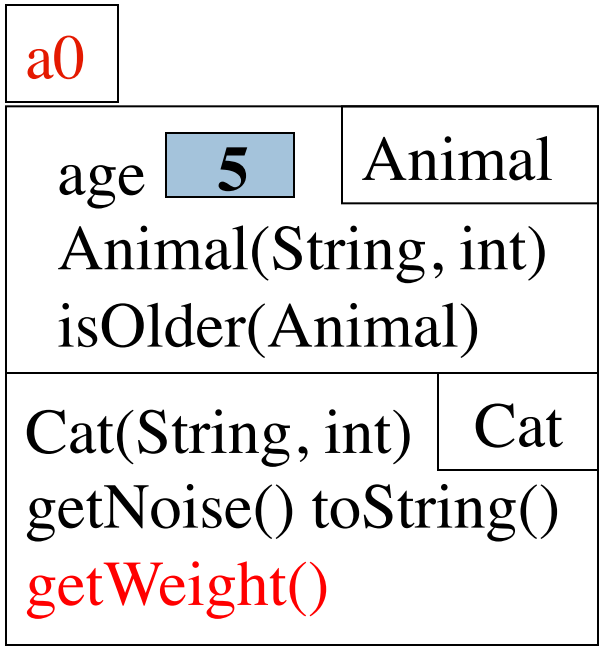
h.isOlder(…) OK —it's in Animal partition

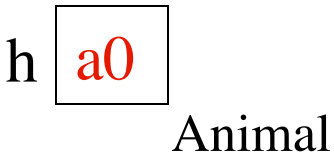
h.getWeight() ILLEGAL —not in Animal partition or Object partition

By overriding rule, calls toString() in Cat partition

Explicit downward cast


```
public class Animal {
   // If Animal is a Cat, return its weight;
       otherwise, return 0.
   public int checkWeight(Animal h) {
     if ( !                                )
          return 0;
     // { h is a Cat }
     Cat c= (Cat) h ;  // downward cast
     return c.getWeight();
}
```



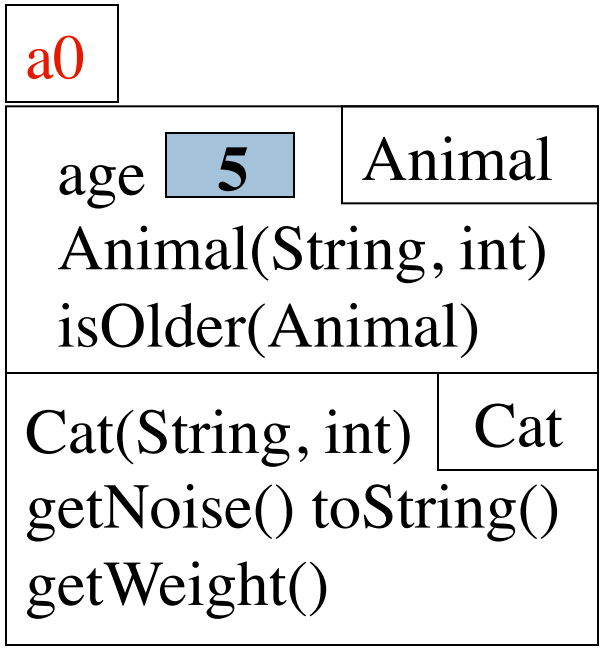


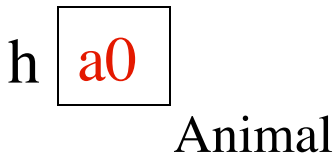
(Dog) h leads to runtime error.

Don't try to cast an object to something that it is not!

Operator instanceof, explicit downward cast


```
public class Animal {
   // If Animal is a cat, return its weight;
       otherwise, return 0.
   public int checkWeight(Animal h) {
     if ( ! (h instanceof Cat) )
          return 0;
     // { h is a Cat }
     Cat c= (Cat) h ;  // downward cast
     return c.getWeight();
}
```



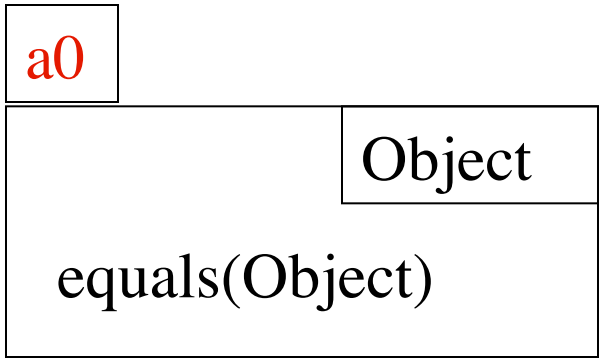


<object> **instanceof** <class>

true iff object is an instance of the class —if object has a partition for class

Function equals

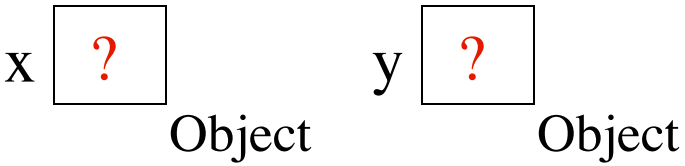

```
public class Object {
    /** Return true iff this object is
      * the same as ob */
    public boolean equals(Object b) {
        return this == b;
    }
}
```



x.equals(y) is same as
x == y
except when x is null!

This gives a null-pointer exception:

null.equals(y)



Overriding function equals

Override function equals in a class to give meaning to:

"these two (possibly different) objects of the class have the same values in some of their fields"

For those who are mathematically inclined, like any equality function, equals should be reflexive, symmetric, and transitive.

Reflexive: b.equals(b)

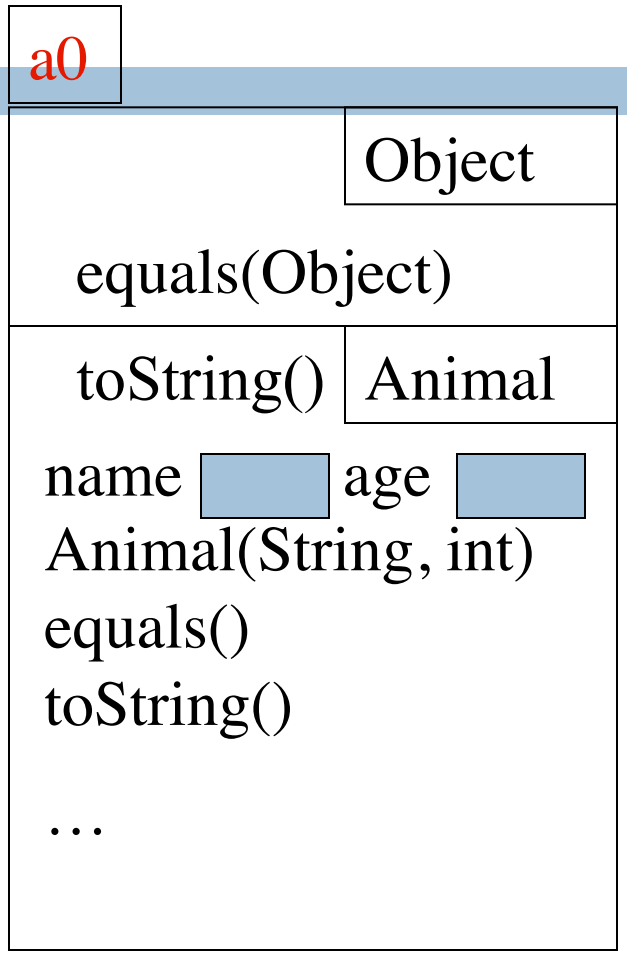
Symmetric: b.equals(c) = c.equals(b)

Transitive: if b.equals(c) and c.equals(d), then b.equals(d)

Function equals in class Animal


```
public class Animal {
  /** = "h is an Animal with the same
       values in its fields as this Animal" */
  public boolean equals (Object h) {

      if (!(h instanceof Animal))
          return false;
      Animal ob= (Animal) h;
      return name.equals(ob.name)  &&
                 age == ob.age;
}
```

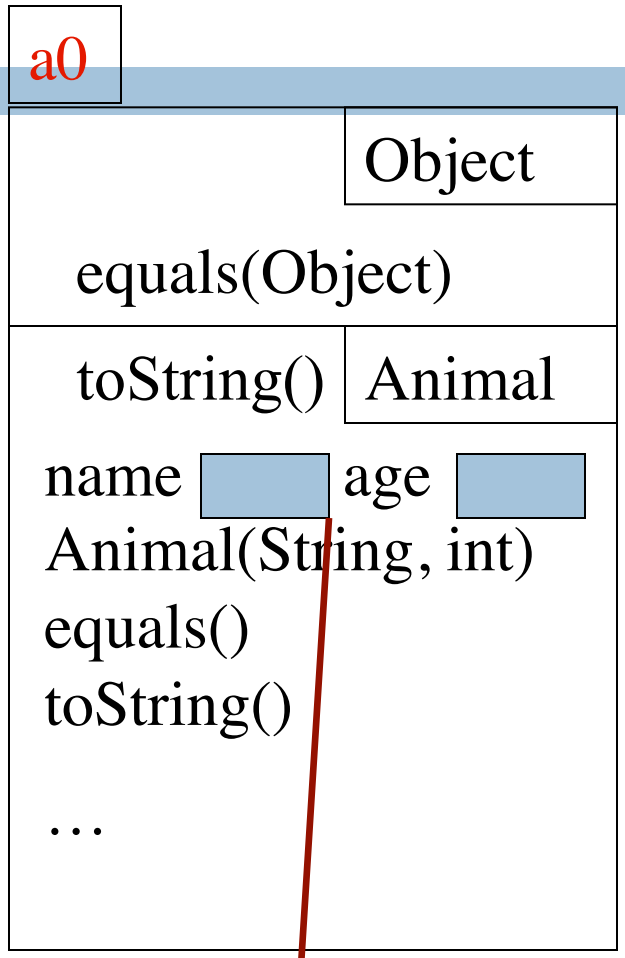


1. Because of h is an Animal in spec,
   need the test h **instanceof** Animal

Function equals in class Animal


```
public class Animal {
  /** = "h is an Animal with the same
       values in its fields as this Animal" */
  public boolean equals (Object h) {

      if (!(h instanceof Animal))
          return false;
      Animal ob= (Animal) h;
      return name.equals(ob.name)  &&
                  age == ob.age;
}
```

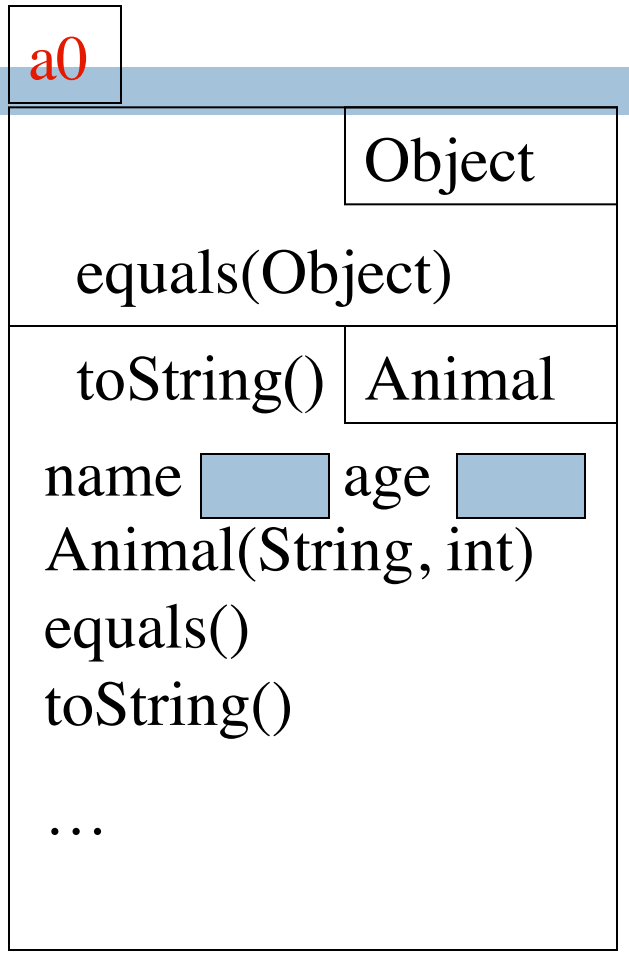


2. In order to be able to reference fields in partition Animal, need to cast h to Animal

Function equals in class Animal


```
public class Animal {
  /** = "h is an Animal with the same
       values in its fields as this Animal" */
  public boolean equals (Object h) {

      if (!(h instanceof Animal))
          return false;
      Animal ob= (Animal) h;
      return name.equals(ob.name)  &&
                 age == ob.age;
}
```

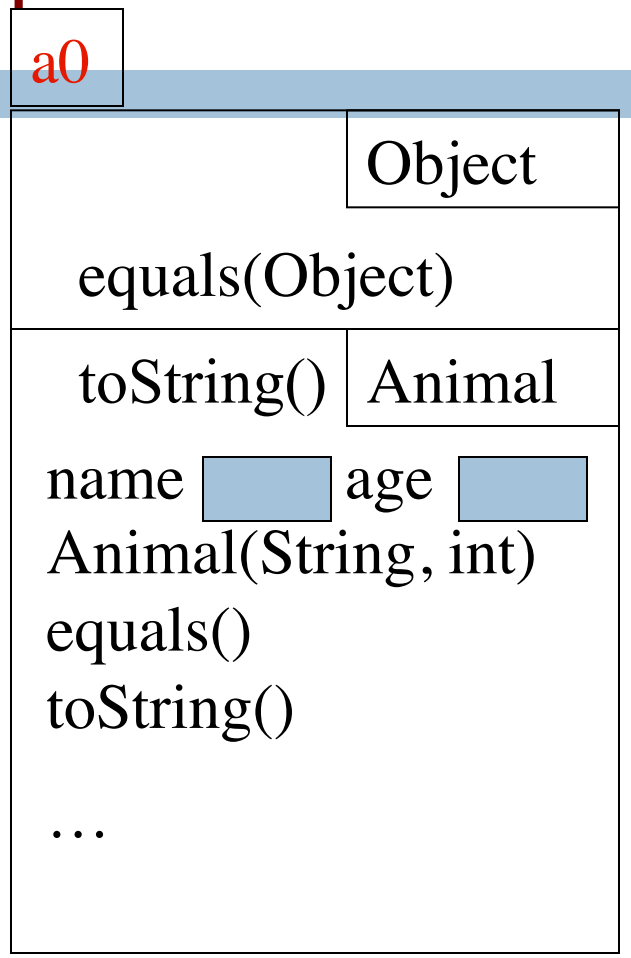


3. Use String equals function to check for equality of String values. Use == for primitive types

Why can't the parameter type be Animal?


```
public class Animal {
  /** = "h is an Animal with the same
       values in its fields as this Animal" */
  public boolean equals (Animal h) {

       if (!(h  instanceof  Animal))
           return false;
       Animal ob= (Animal) h;
       return name.equals(ob.name)  &&
                  age == ob.age;
}
```



What is wrong with this?