The fattest knight at King Arthur's round table was Sir Cumference. He acquired his size from too much pi.

CS/ENGRD 2110 SPRING 2018

Lecture 6: Consequence of type, casting; function equals

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

Overview references in JavaHyperText

- Quick look at arrays array
- Casting among classes cast, object-casting rule
- Operator instanceof
- Function getClass
- Function equals
- compile-time reference rule

Homework. JavaHyperText while-loop for-loop

```
while ( <bool expr> ) { … }                    // syntax

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

A2 is due Thursday

Everyone should get 100/100 since we gave you all the test cases you need.

Please look at the pinned Piazza note “Assignment A2” for information that is not in the handout and answers to questions.

Before Next Lecture…

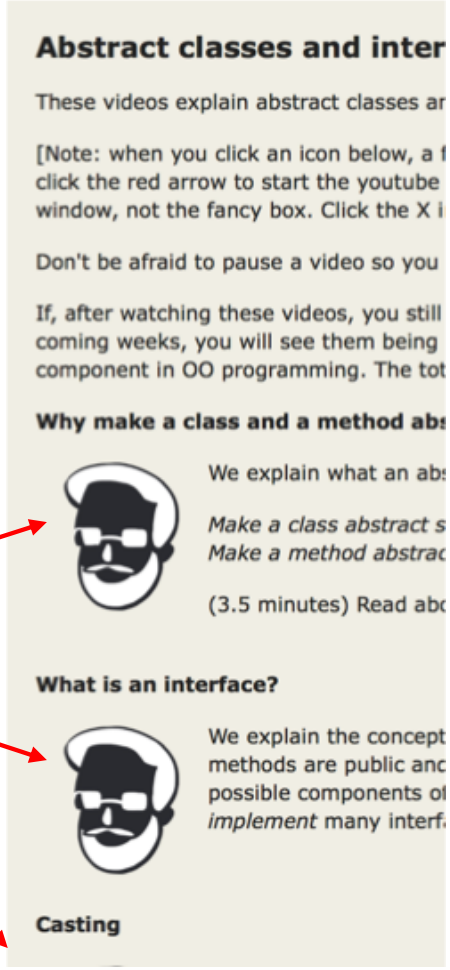
Follow the tutorial on **abstract classes and interfaces**, and watch less than 13 minutes of videos.

Visit JavaHyperText and click on

Abstract classes and interfaces

This will make Thursday's lecture far more understandable.

Click these

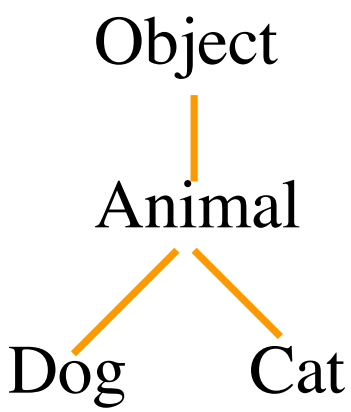


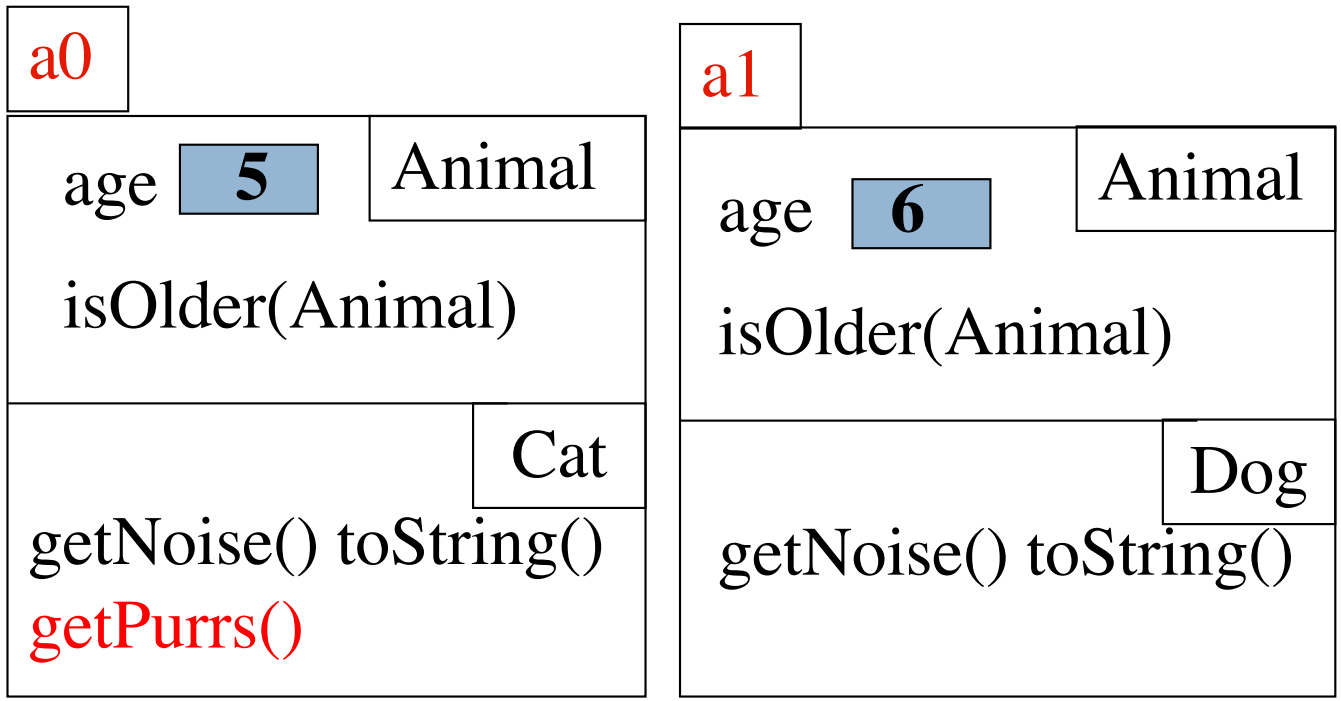
Classes we work with today

Work with a class Animal and subclasses like Cat and Dog

Put components common to animals in Animal

class hierarchy:





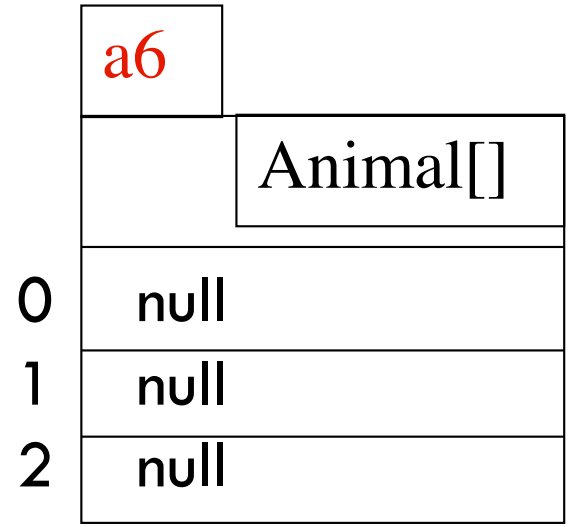
Object partition is there but not shown

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

declaration of array v

Create array of 3 elements

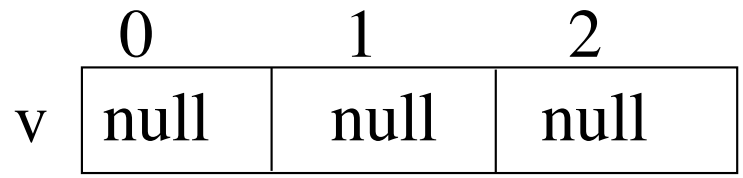
Assign value of new-exp to v



Assign and refer to elements as usual:

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

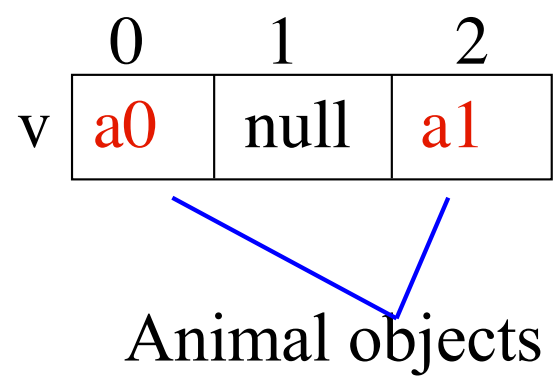
Sometimes use horizontal picture of an array:



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.



A variable's type:

- *Restricts* what values it can contain.
- Determines which methods are legal to call on it.

Dog and Cat objects stored in Animal variable

Which function is called by

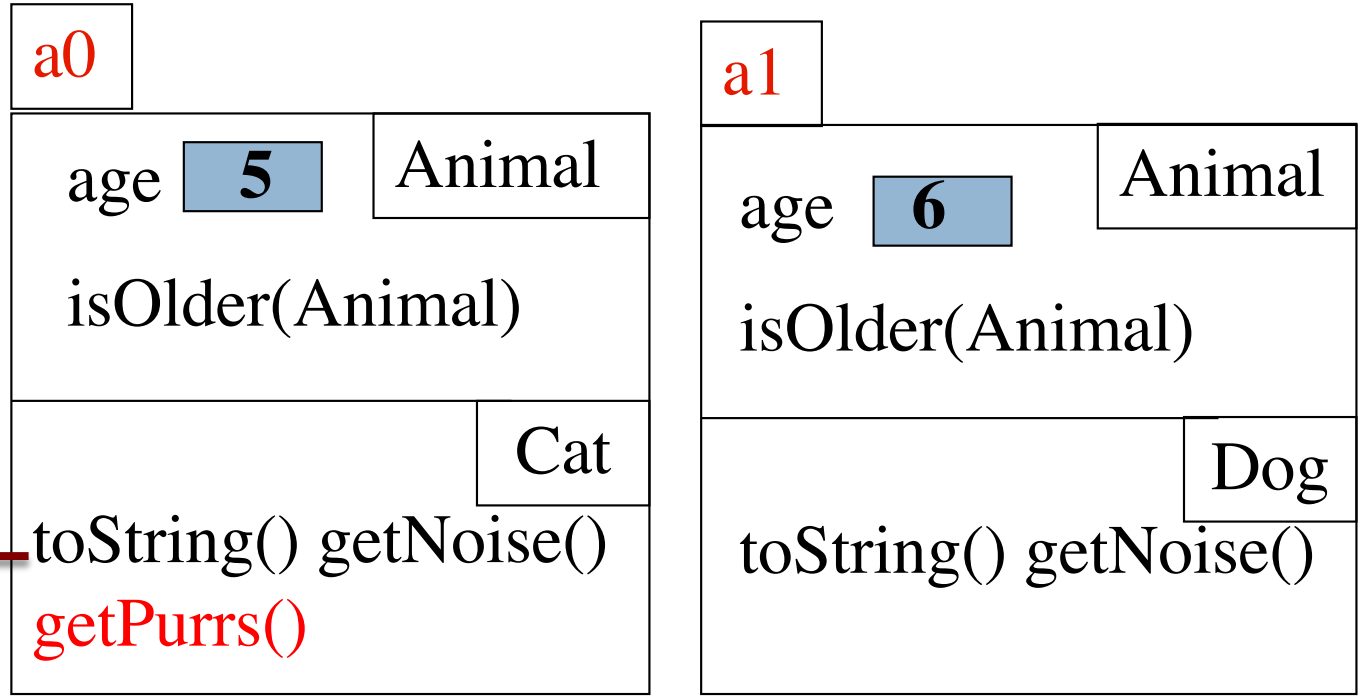
v[0].toString() ?

(Remember, the hidden Object partition contains toString().)

	0	1	2
v	a0	null	a1

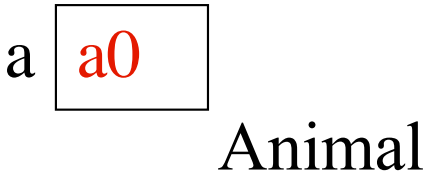
Can store (pointers to) subclass objects in superclass variable

Bottom-up or overriding rule says function toString in Cat partition

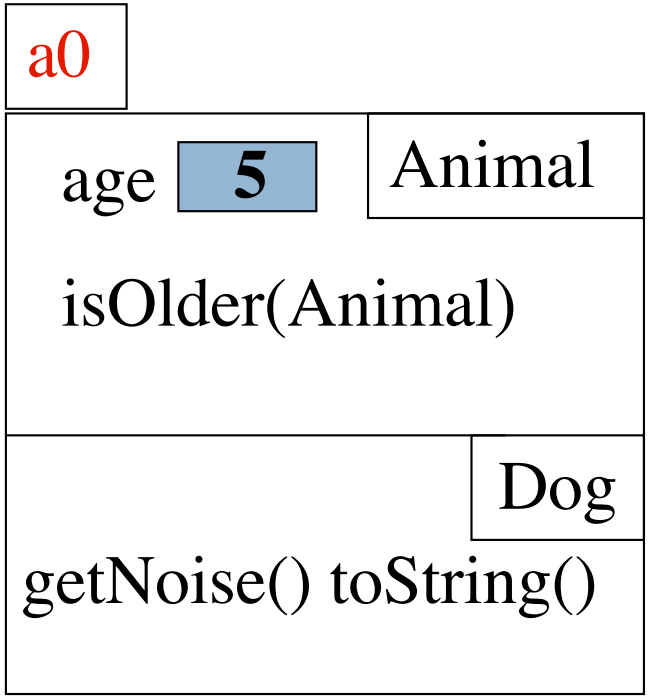


Compile-time reference rule: From a variable of type C, can reference only methods/fields that are available in class C.

a.getPurrs() is obviously illegal.
The compiler will give you an error.



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

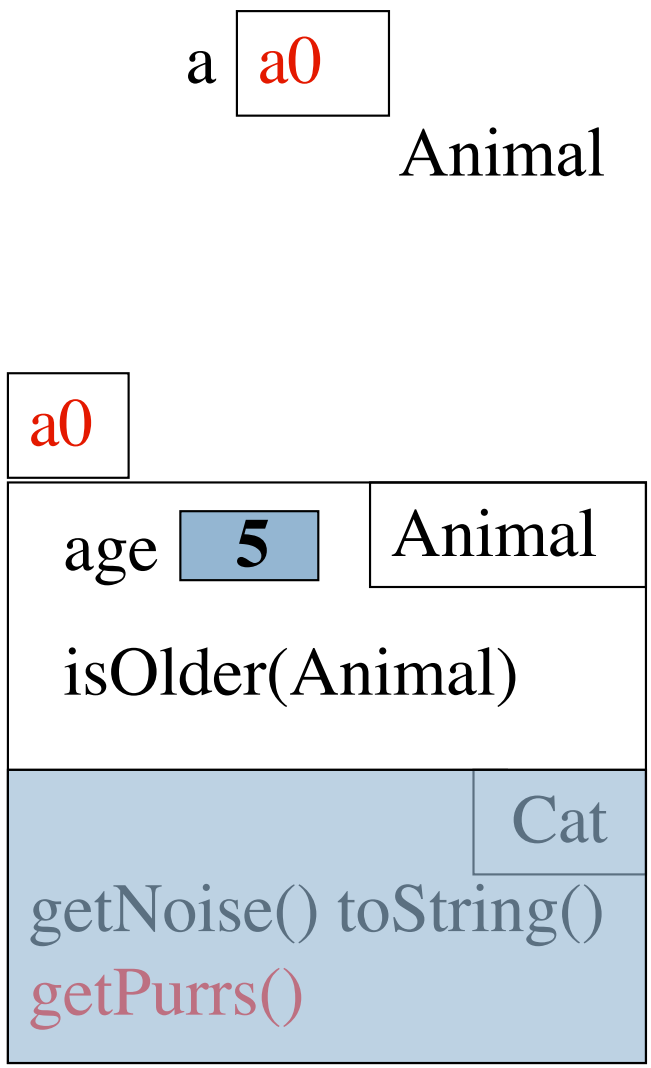


When checking legality of a call like
a.getPurrs(…)
since the type of a is Animal, method getPurrs must be declared in Animal or one of its superclasses.

see JavaHyperText: compile-time reference rule

Compile-time reference rule: From a variable of type C, can reference only methods/fields that are available in class C.

Suppose a0 contains an object of a subclass Cat of Animal. By the copmle-time reference rule below, a.getPurrs(…) is still illegal. Remember, the test for legality is done at compile time, not while the program is running.



When checking legality of a call like

a.getPurrs(…)

since the type of a is Animal, method getPurrs must be declared in Animal or one of its superclasses.

see JavaHyperText: compile-time reference rule

Compile-time reference rule: From a variable of type C, can reference only methods/fields that are available in class C.

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

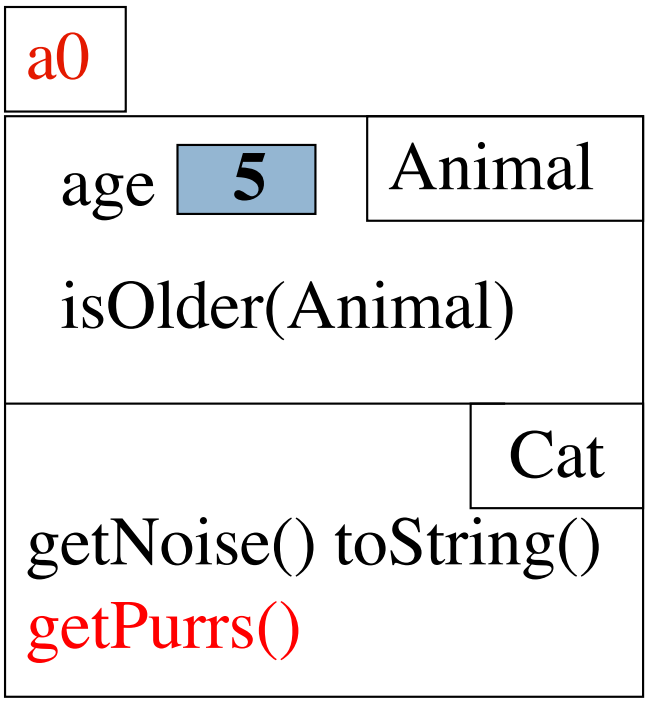
c a0 Cat

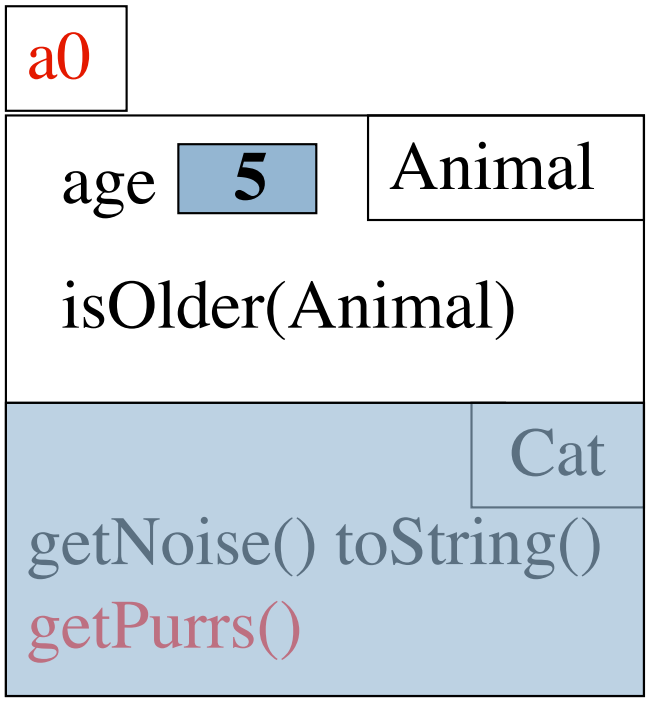
a a0 Animal

c.getPurrs() is legal

a.getPurrs() is illegal

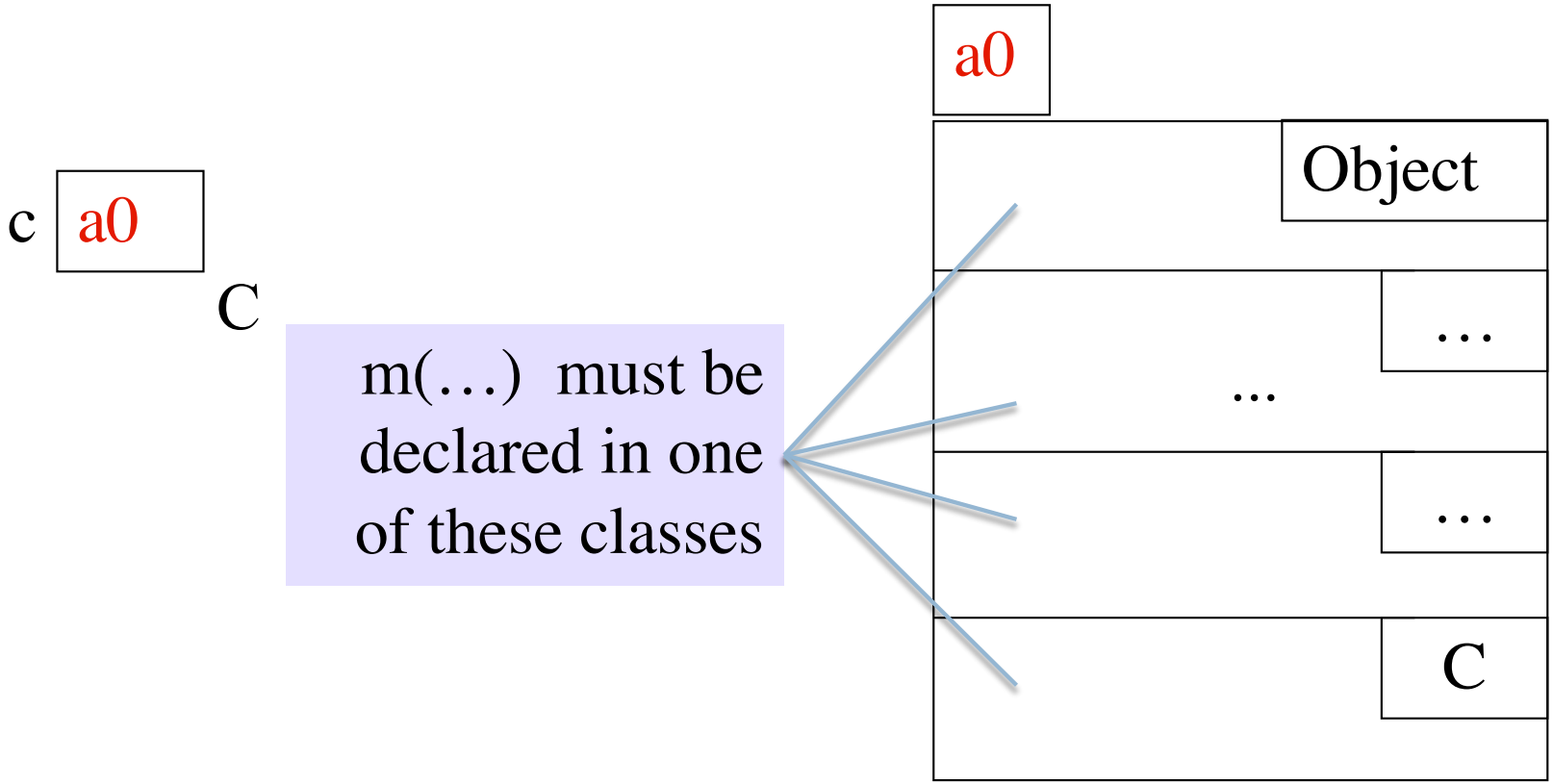
because getPurrs is not available in class Animal





Compile-time reference rule: From a variable of type C, can reference only methods/fields that are available in class C.

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



Another example

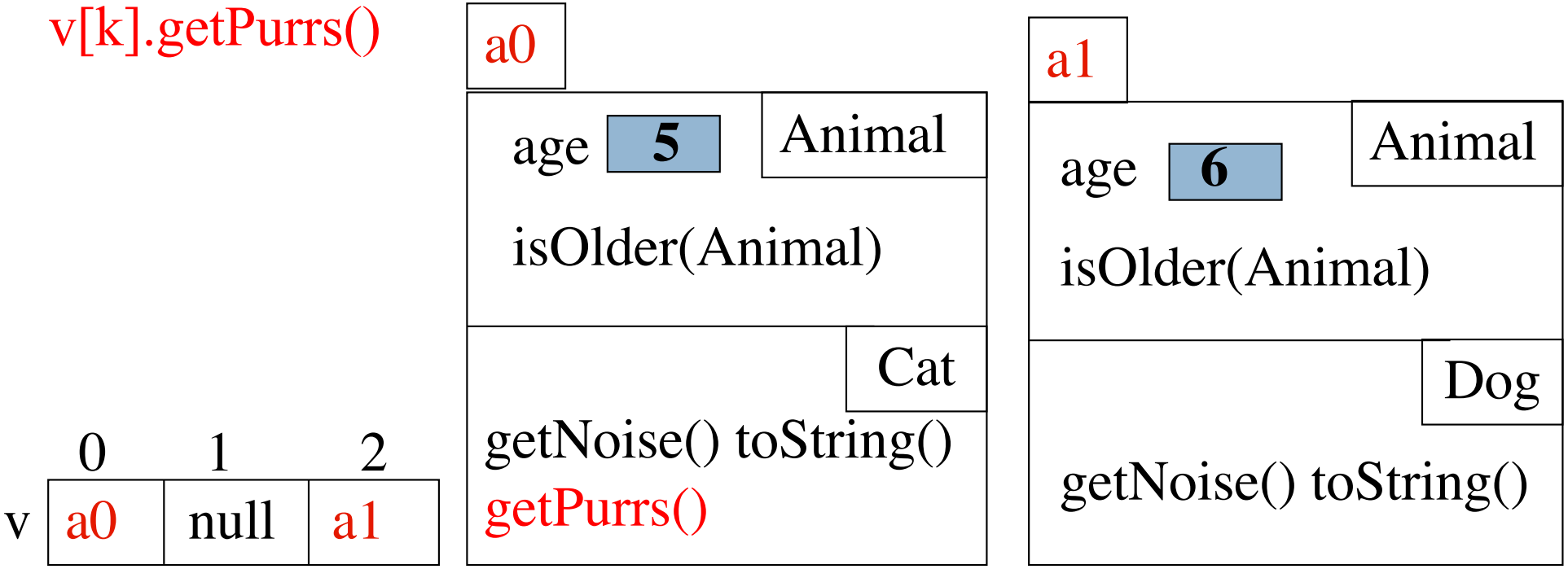
Type of v[0]: Animal

Should this call be allowed?
Should program compile?

v[0].getPurrs()

Should this call be allowed?
Should program compile?

v[k].getPurrs()



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.

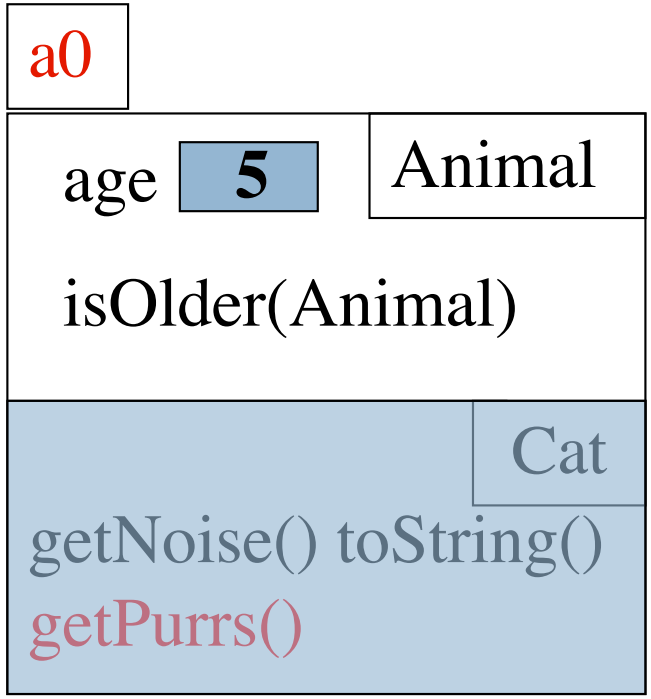
getPurrs() not in class Animal or Object. Calls are illegal, program does not compile:

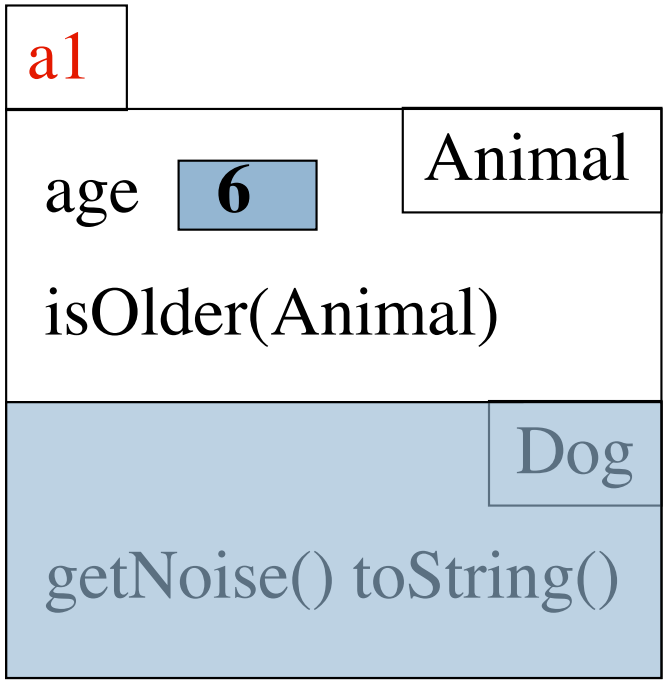
v[0].getPurrs() v[k].getPurrs()

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

	0	1	2
v	a0	null	a1

Animal

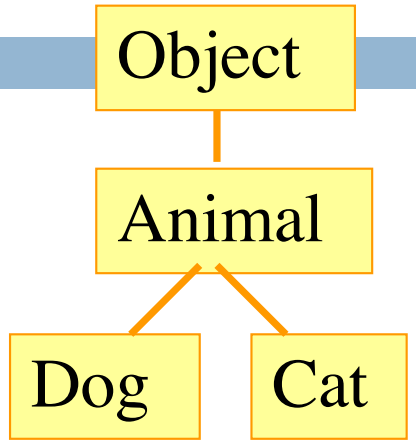


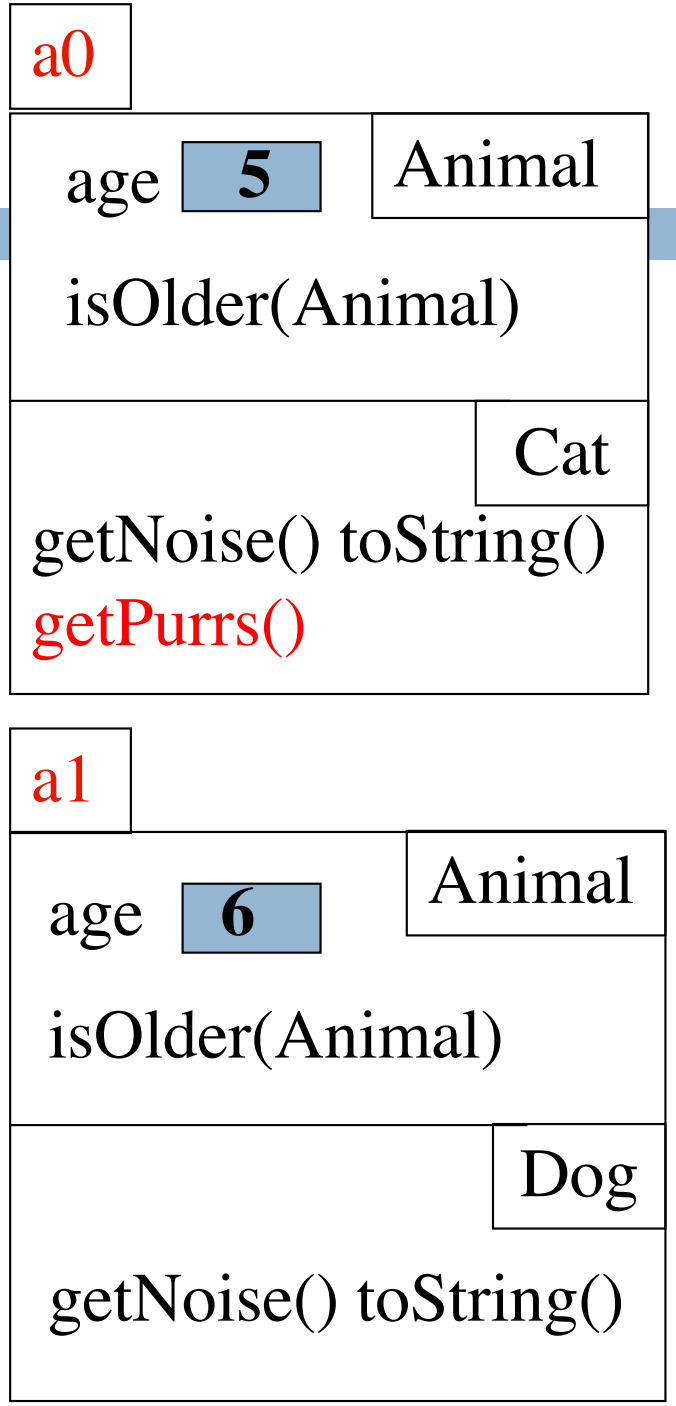


Casting objects

You know about casts like:

```
(int) (5.0 / 7.5)
(double) 6
double d= 5;     // automatic cast
```





You can also use casts with class types:

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

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

Explicit casts: unary prefix operators

Object-casting rule: At runtime, an object can be cast to the name of any partition that occurs within it —and to nothing else.

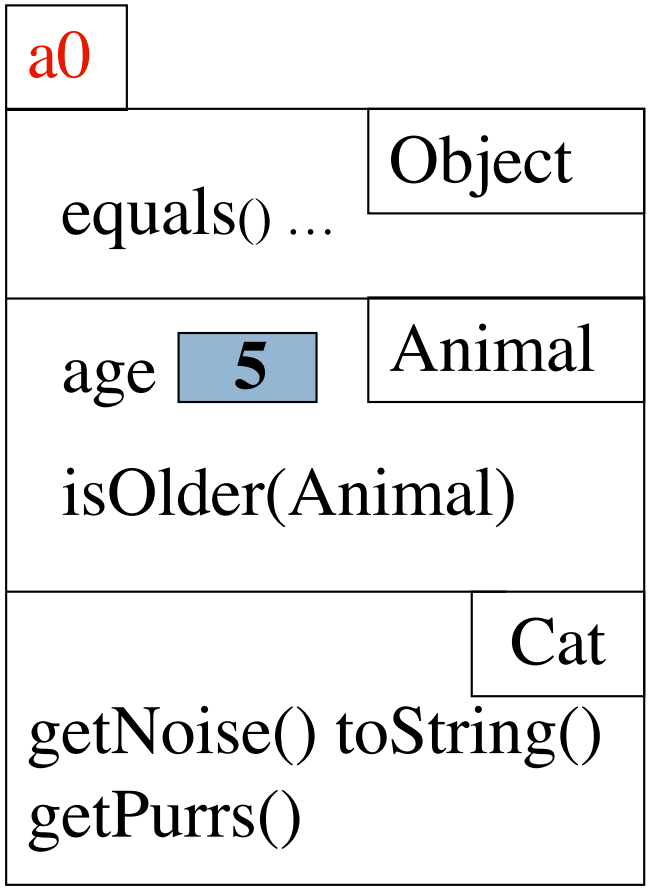
a0 can be cast to Object, Animal, Cat.

An attempt to cast it to anything else causes an exception

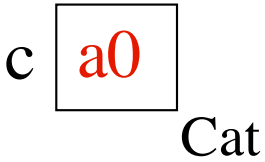
(Cat) c

(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

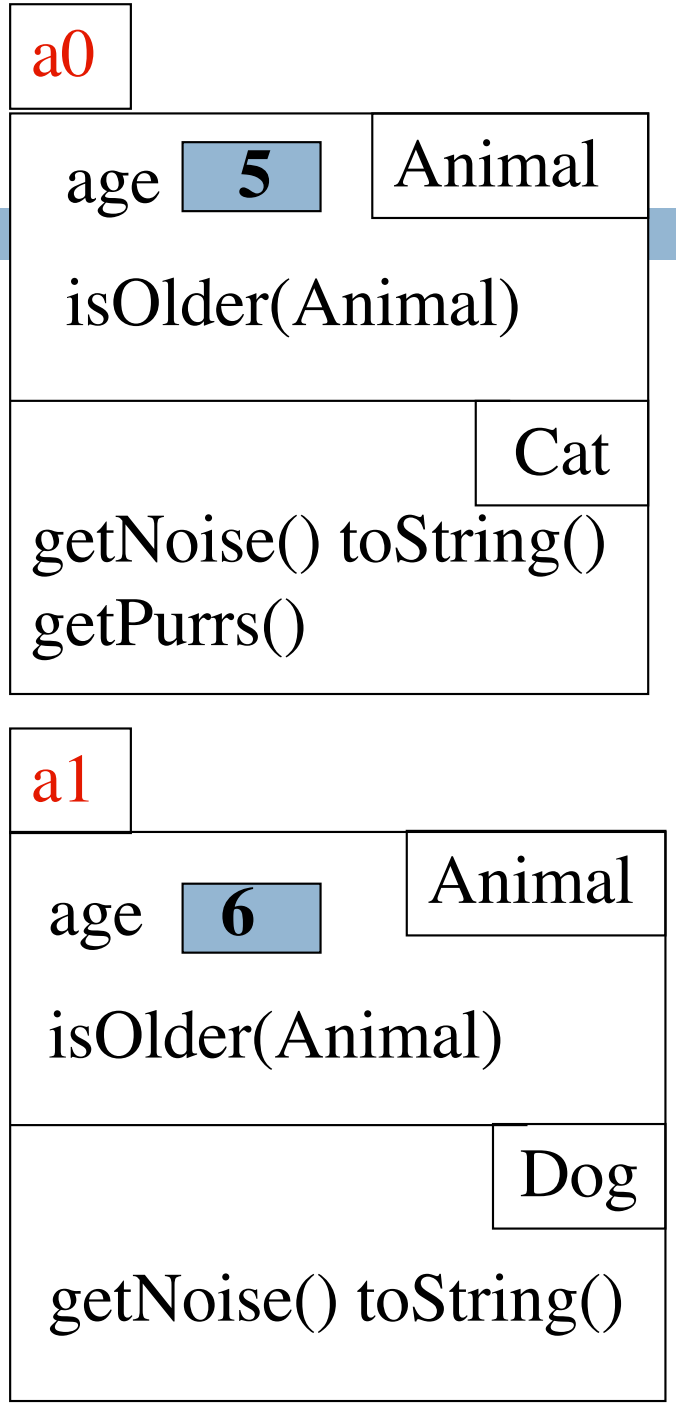

```
public class Animal {
    /** = "this Animal is older than h" */
    public boolean isOlder(Animal h) {
        return age > h.age;
    }
```

Call c.isOlder(d)

Variable 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



Function h.equals(ob)

Function h.equals(ob) returns true if objects h and ob are equal, where equality depends on the class. Here, we mean all corresponding fields are equal.

h a0

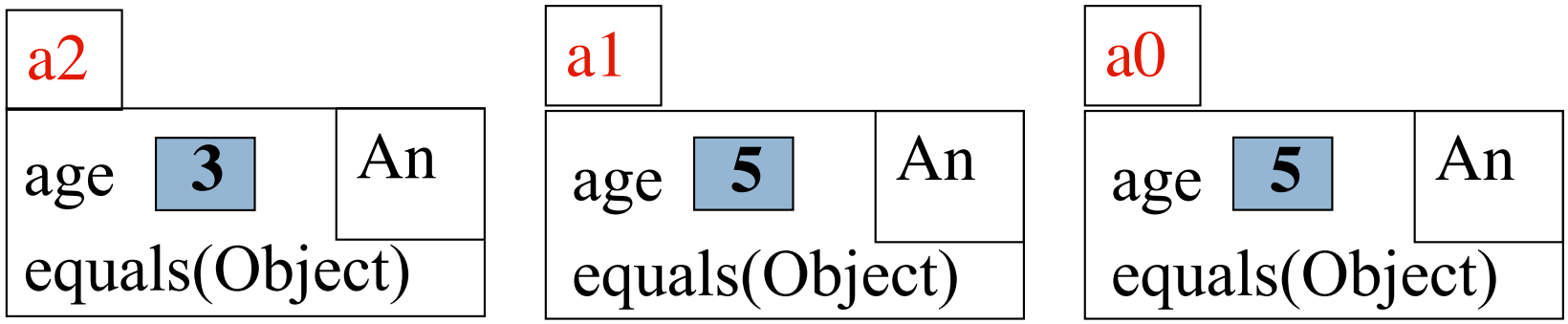
k a1

j a2

h.equals(h):	true
h.equals(k):	true
h.equals(j):	false

a0.equals(a0):	true
a0.equals(a1):	true
a0.equals(a2)	false

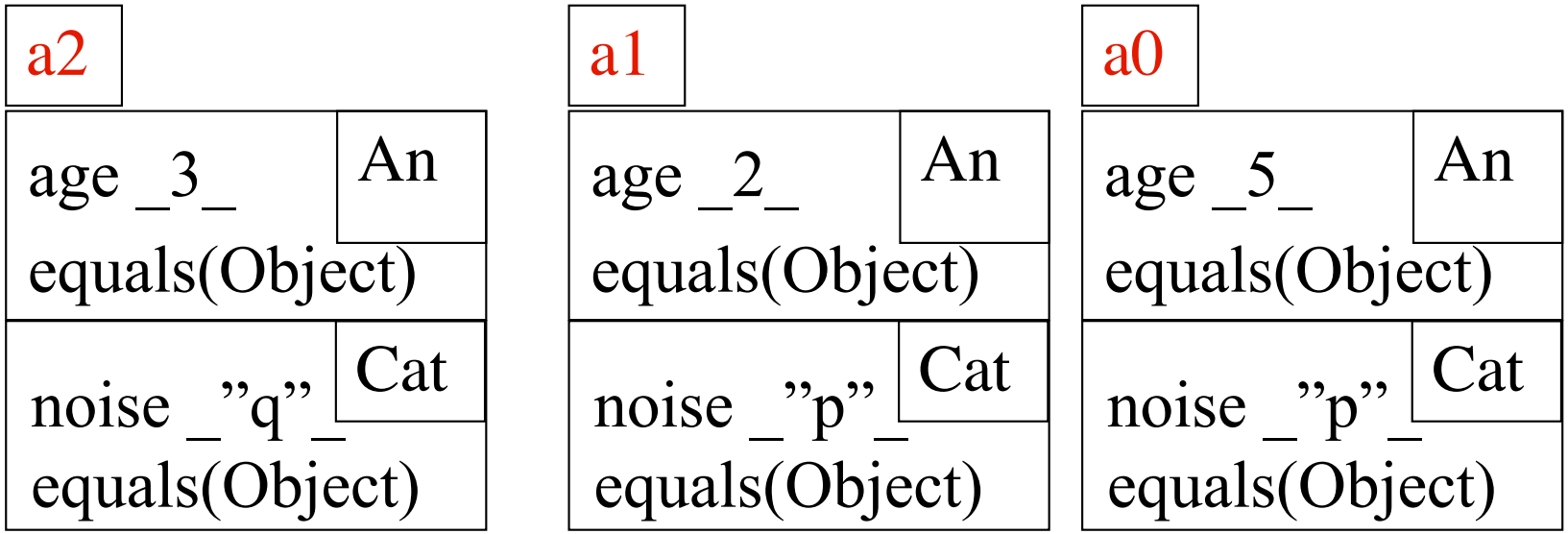
Not Java



Function h.equals(ob)

Function h.equals(ob) returns true if objects h and ob are equal, where equality depends on the class. Here, we mean all corresponding fields are equal.

a0.equals(a0): true
a0.equals(a1): false
a0.equals(a2): false

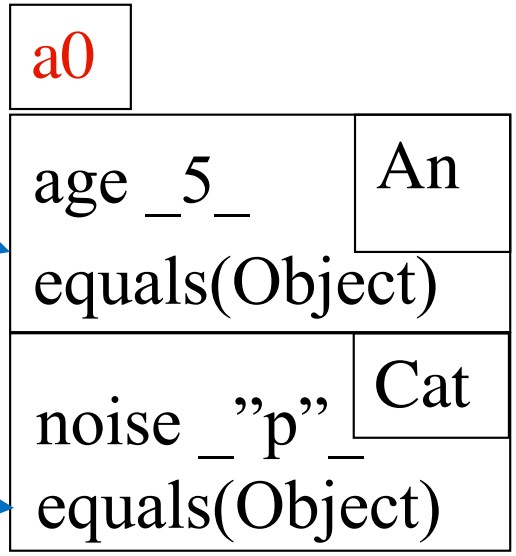


Function h.equals(ob)

Function h.equals(ob) returns true if objects h and ob are equal, where equality depends on the class. Here, we mean all corresponding fields are equal.

This function checks equality of age

This function
(1) Calls superclass equality
(2) checks equality of noise



Function h.equals(ob)

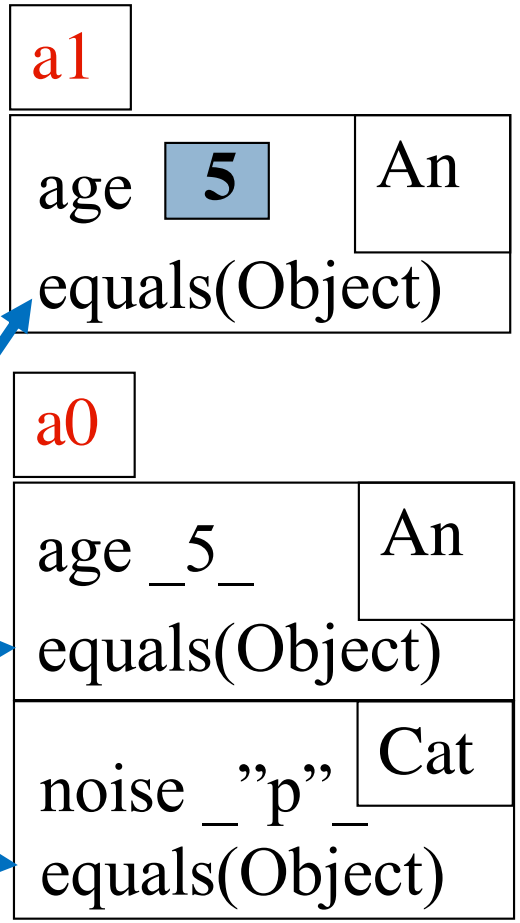
Function h.equals(ob) returns true if objects h and ob are equal, where equality depends on the class. Here, we mean all corresponding fields are equal.

What is value of a1.equals(a0)?
a0.equals(a1)?

Obviously, h.equals(0b) has to check that the classes of h and ob are the same

This function checks equality of age

This function
(1) Calls super-class equality
(2) checks equality of noise

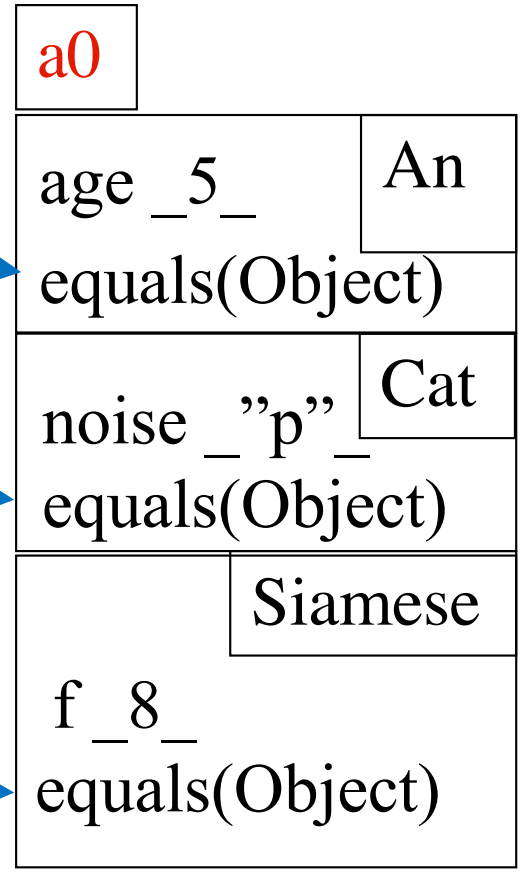


Function h.equals(ob)

(1) Check classes of this and parameter
(2) Check age of this and parameter

(1) Call super-class equality
(3) Check equality of noise

(1) Call super-class equality
(3) Check equality of noise



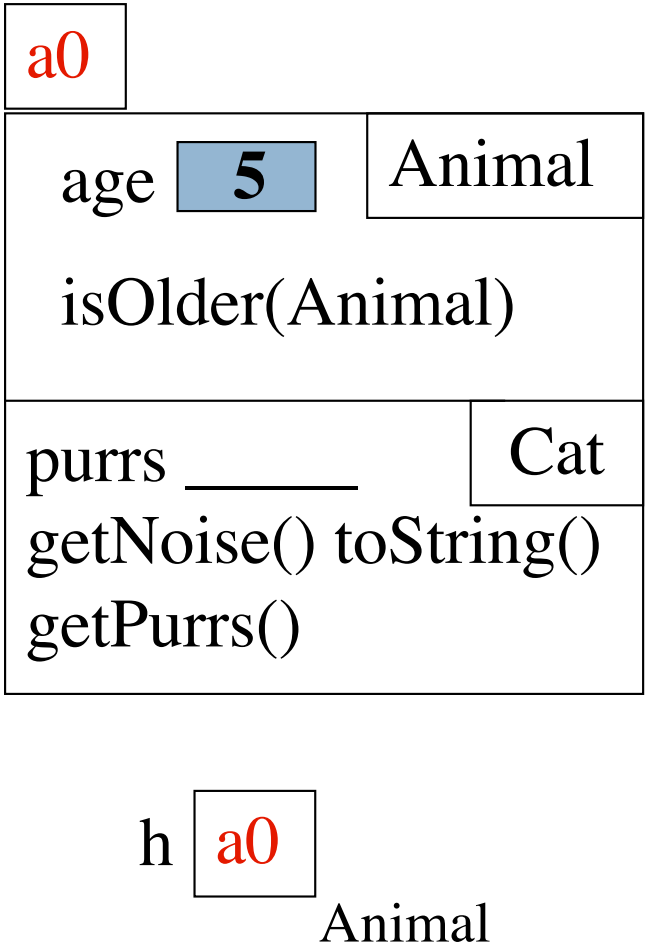
Use function getClass

h.getClass()

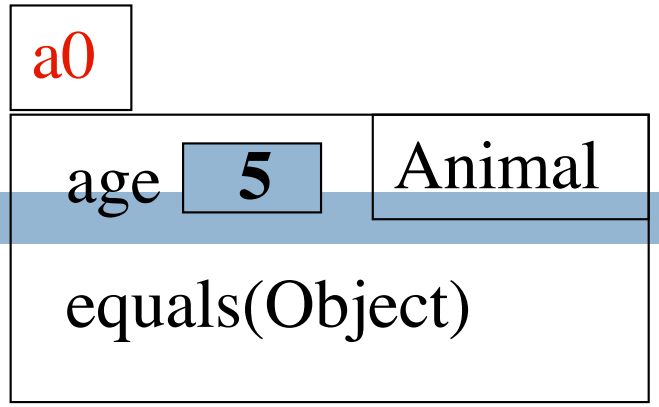
Let Cat be the lowest partition of object h

Then h.getClass == Cat.class

h.getClass != Animal.class



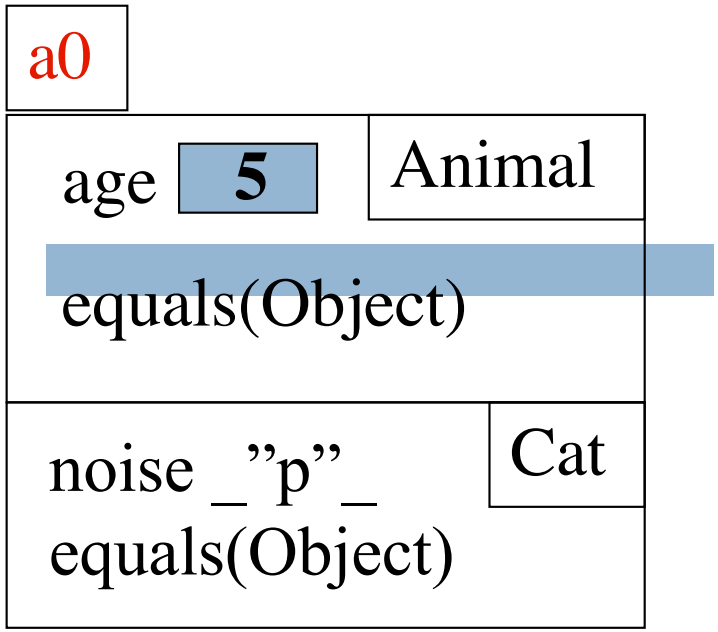
Equals in Animal




```
public class Animal {
  private int age;
  /** return true iff this and ob are of the same class
    * and their age fields have same values */
  public boolean equals(Object ob) {

      if (ob == null || getClass() != ob.getClass()) return false;

      Animal an= (Animal) ob;  // cast ob to Animal!!!!

      return age == an.age;  // downcast was needed to reference age
}
```

Equals in Cat

```
public class Animal {
   private int age;
   /** return true iff this and ob are of same class
      * and their age fields have same values */
   public boolean equals(Object ob) {}
```



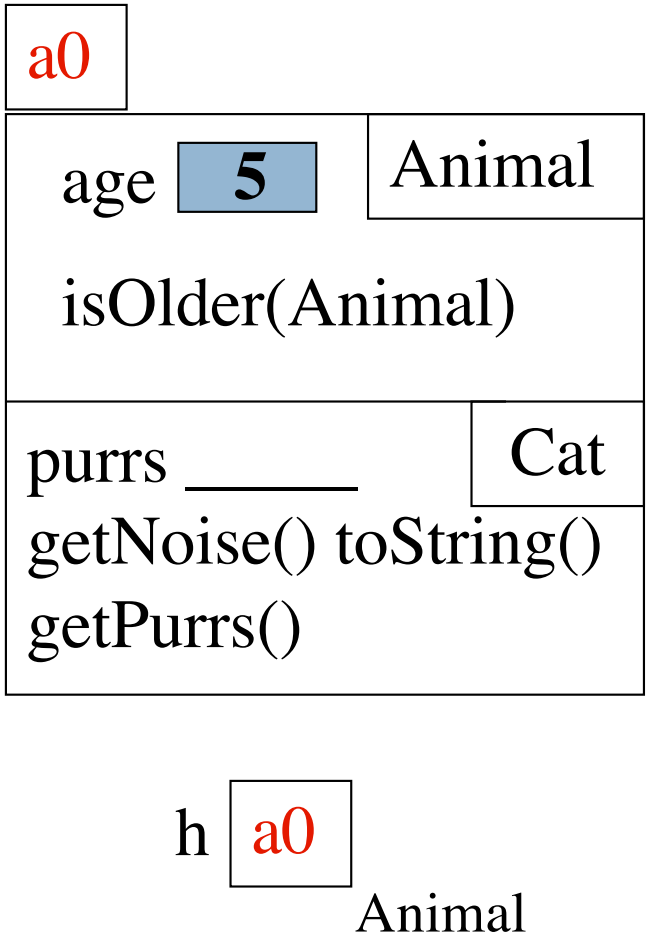

```
public class Cat extends Animal {
   private int age;
   /** return true iff this and ob are of same class
      * and their age and noise fields have same values */
   public boolean equals(Object ob) {}
         if (!super.equals(ob) return false;
         Cat ca= (Cat) ob;   // downcast is necessary!
         return noise == ca.noise;  // needed to reference noise
}
```

Use operator instanceof

ob **instanceof** C

true iff ob has a partition named C

h **instanceof** Object	true
h **instanceof** Animal	true
h **instanceof** Cat	true
h **instanceof** JFrame	false



Opinions about casting

Use of instanceof and downcasts can indicate bad design

DON'T:

```
if (x instanceof C1)
    do thing with (C1) x
else if (x instanceof C2)
    do thing with (C2) x
else if (x instanceof C3)
    do thing with (C3) x
```

DO:

```
x.do()
```

… where do is overridden in the classes C1, C2, C3

But how do I implement equals() ?

That **requires** casting!