CS/ENGRD 2110 FALL 2013

Lecture 6: Casting; function equals

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

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

- Static/Dynamic types (apparent/real types)   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

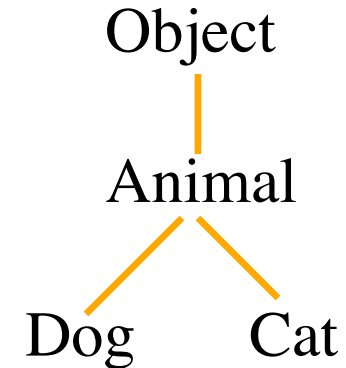**for** (**int** k= 0; k < 200; k= k+1) { … } // example

Classes we work with today

class hierarchy:
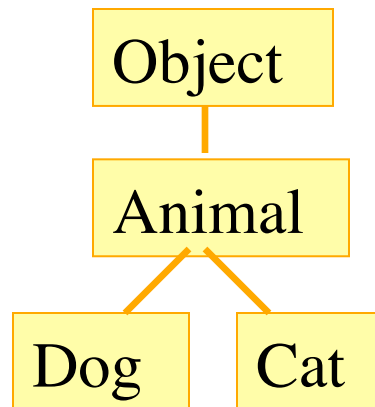
Work with a class Animal and subclasses like Cat and Dog

Put components common to animals

Object, partition is there but not shown

Object
|
Animal
/   \
Dog   Cat

class hierarchy:

Object
|
Animal
/   \
Dog   Cat

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()

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

declaration of array v

Create array of 3 elements

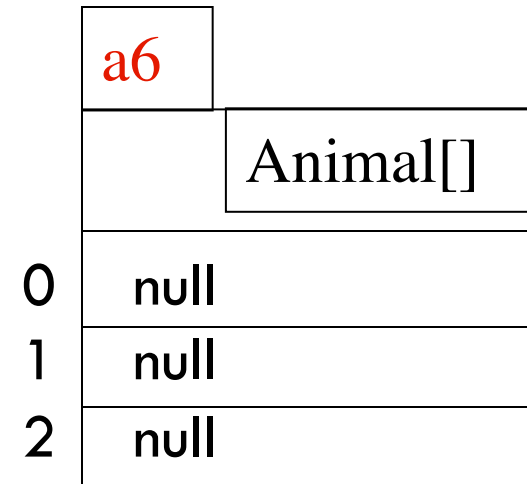| Assign value of new-exp to v |
|---|

v | ~~null~~   a6 |

a6

Animal[]

0 | null
1 | null
2 | null

Assign and refer to elements as usual:

v[0]= **new** Animal(…);

…

a= v[0].getAge();

Sometimes use horizontal picture of an array:

	0	1	2
v	null	null	null
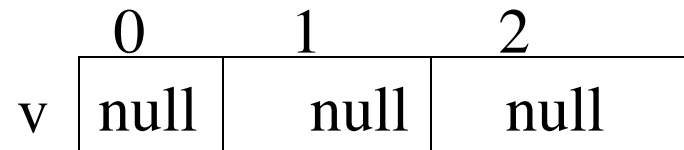
Which function is called?

Which function is called by

   v[0].toString() ?

Remember,
partition Object
contains

toString()

	0	1	2
v	a0	null	a1

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()

Static/apparent type
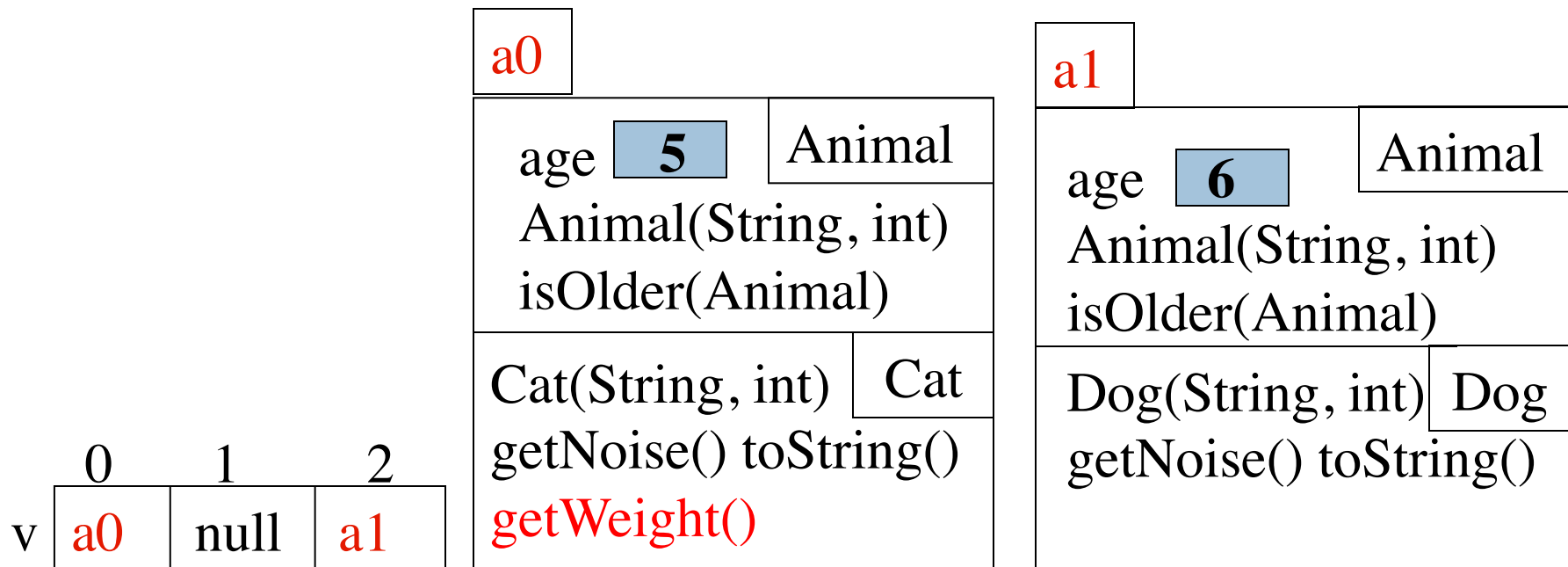
Each element v[k] is of type Animal.
Its declared type:

static type — known at
compile-time

apparent type

Should this call be allowed?
Should program compile?

v[0].getWeight()

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()

|	0	1	2
v	a0	null	a1

View of object from static type

Each element v[k] is of (static) 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:
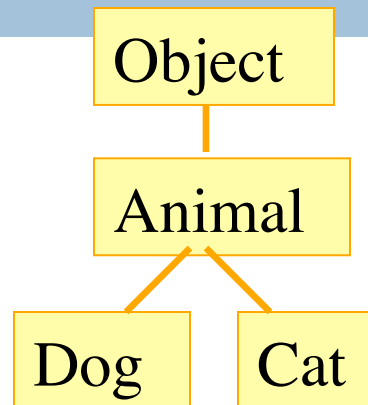
v[0].getWeight() v[k].getWeight()

Components still 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()
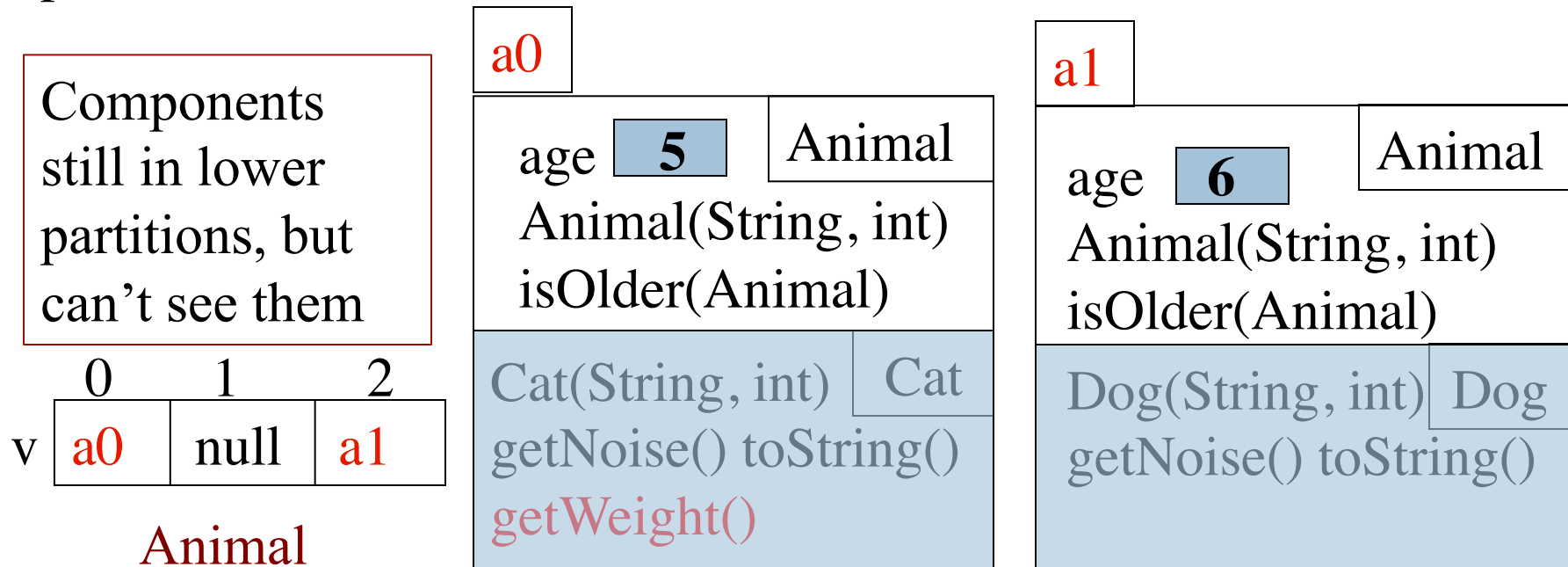
Casting up class hierarchy

You know about casts like

(**int**) (5.0 / 7.5)

(**double**) 6

double d= 5;   // automatic cast

We now discuss casts up and down the class hierarchy.

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

Cat c= (Cat) h;

Object
Animal
Dog   Cat

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
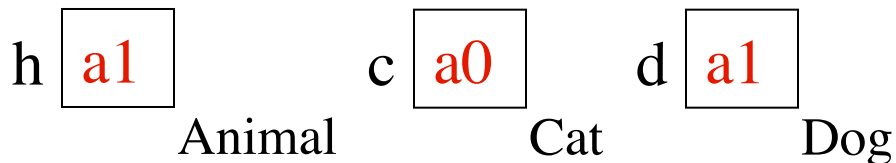getNoise() toString()

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]   c [a0]   d [a1]
  Animal   Cat   Dog

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()

Explicit casts: unary prefix operators

You may cast an object 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

a0

	Object
equals() ...	

	Animal
age **5**	
Animal(String, int)	
isOlder(Animal)	

Cat(String, int)	Cat
getNoise() toString() getWeight()	

c | a0

Cat

Static/dynamic types

a1

Animal

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

age   **6**

Animal(String, int)
isOlder(Animal)

Dog(String, int)   Dog
getNoise() toString()
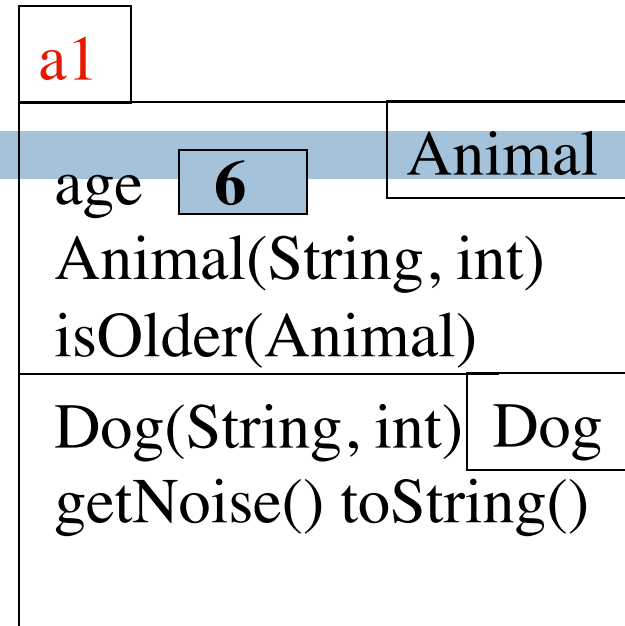
Static or apparent type of h is Animal. Syntactic property

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

h   a1

Animal

Dynamic or real type of h is Dog. Semantic/runtime property

If a method call is legal, dynamic type determines which one is called (overriding one)
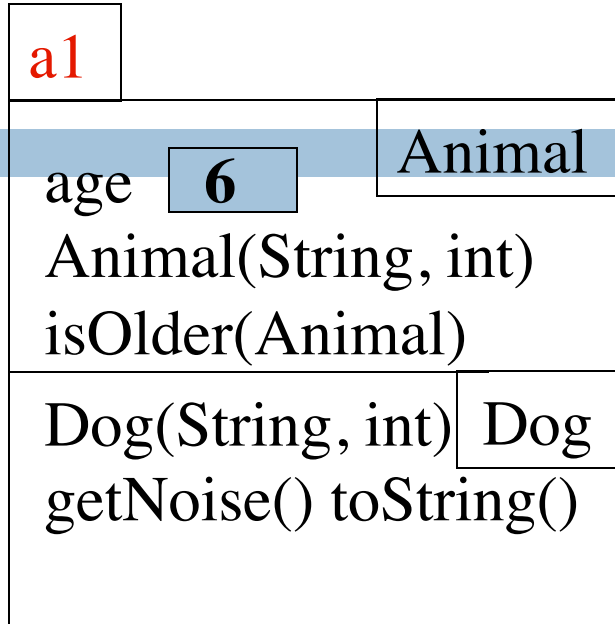
Components used from h

a1

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

Animal

age **6**

Animal(String, int)
isOlder(Animal)

Dog(String, int) Dog
getNoise() toString()

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

h a1
Animal

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();
}
```

a0

age	**5**	Animal

Animal(String, int)
isOlder(Animal)

Cat(String, int)	Cat

getNoise() toString()
getWeight()

(Dog) h leads to runtime error.

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

h a0

Animal

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();
}
```

a0

age	**5**	Animal
	Animal(String, int)	
	isOlder(Animal)	

Cat(String, int)	Cat
getNoise() toString()	
getWeight()	

h | a0
Animal

<object> **instanceof** <class>

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

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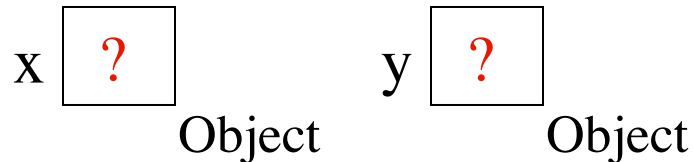
```
public class Object {
    /** Return true iff this object is the same as ob */
    public boolean equals(Object b) {
        return this == b;
    }
}
```

This gives a null-pointer exception:

$$null.equals(y)$$

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

x [?]   y [?]
    Object   Object

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

a0

Object

equals(Object)

toString() Animal

name ▭ age ▭
Animal(String, int)
equals()
toString()
…

```
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;

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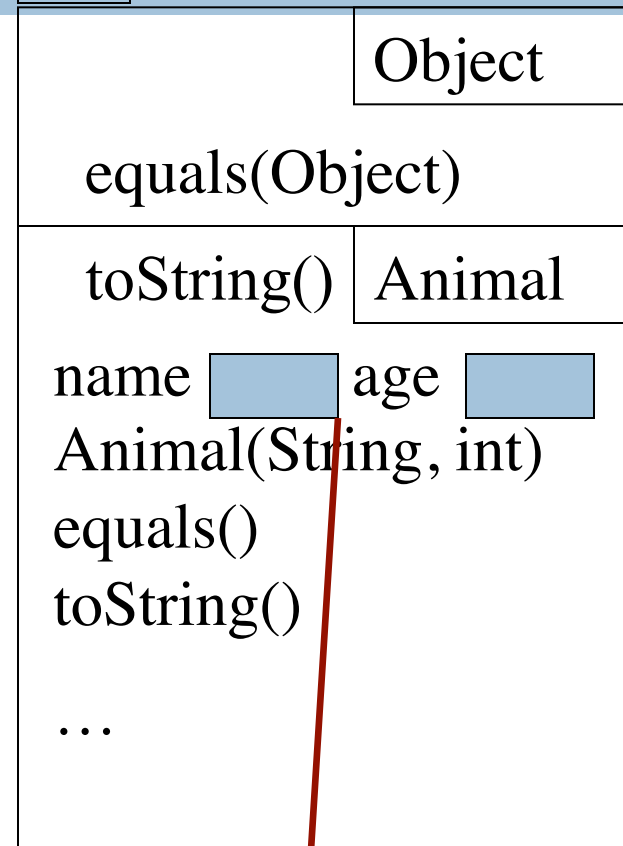
1. Because of h is an Animal in spec,
   need the test h instanceof Animal

Function equals in class Animal

a0

```
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;

}
```

Object

equals(Object)

toString() | Animal

name [ ] age [ ]
Animal(String, int)
equals()
toString()
…

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;

}
```
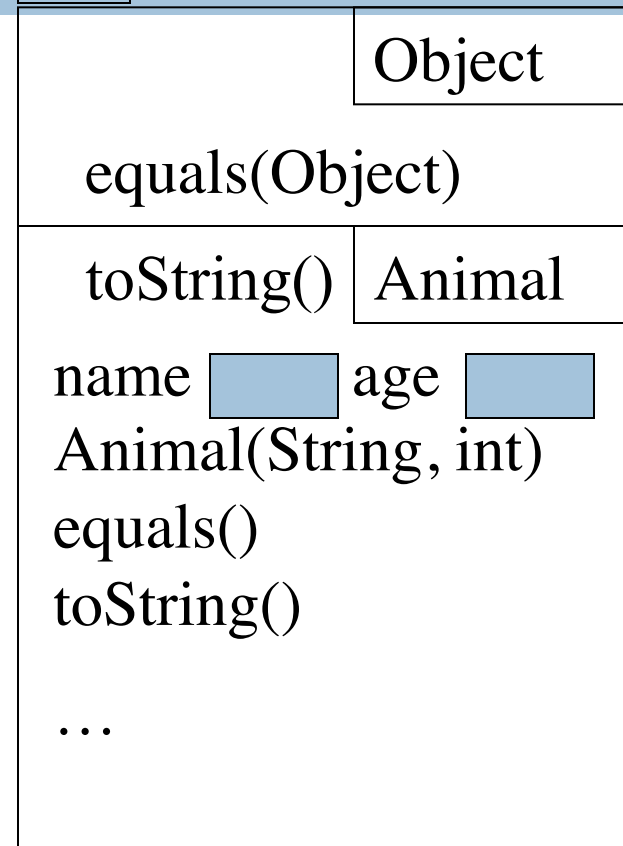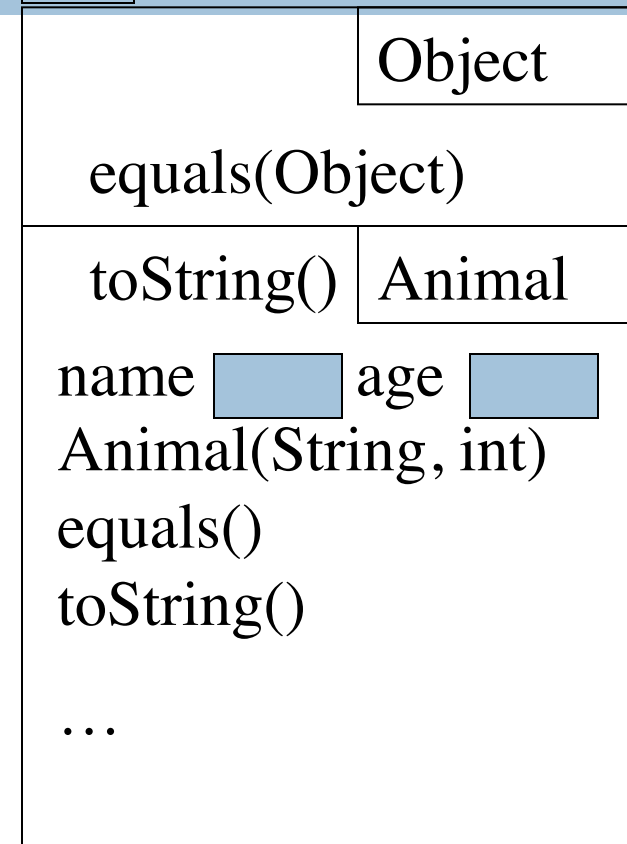
a0

Object

equals(Object)

toString() Animal

name ____ age ____
Animal(String, int)
equals()
toString()
…

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;

}
```

a0

Object

equals(Object)

toString()   Animal

name [   ] age [   ]
Animal(String, int)
equals()
toString()
…

What is wrong with this?

Recitation this week: VERY important

Recitation this week is about

      abstract classes

      interfaces

Don't miss recitation

Learn:

☐ Why we may want to make a class abstract

☐ Why we may want to make a method abstract

☐ An interface is like a very restricted abstract class, with different syntax for using it.