CS/ENGRD 2110 SPRING 2019

Lecture 6: Consequence of type, casting; function equals
http://courses.cs.cornell.edu/cs2110

Reminder: A1 due tonight

Today's topics

- Casting, object-casting rule
- Compile-time reference rule
- Quick look at arrays
- Implementing equals, method getClass

JavaHyperText

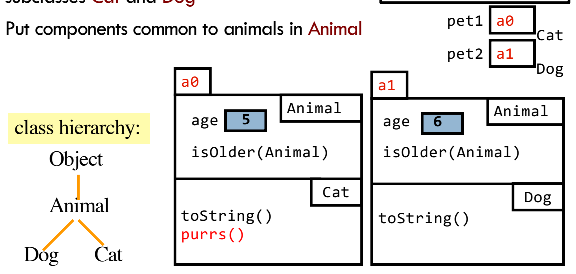
- Review on your own if you need to: while and for loop

Classes we work with today

class Animal
subclasses Cat and Dog

Put components common to animals in Animal

```
Cat pet1= new Cat(5);
Dog pet2= new Dog(6);
```



(Object partition is there but not shown)

DEMO

Casting

Casting objects

You know about casts like:

(int) (5.0 / 7.5)

(double) 6

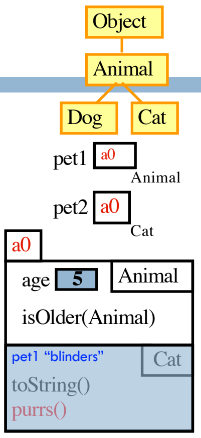
double d= 5; // cast implicit

You can also use casts with class types:

Animal pet1= **new** Cat(5); // cast implicit

Cat pet2= (Cat) pet1;

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



Explicit casts: unary prefix operators

7

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.

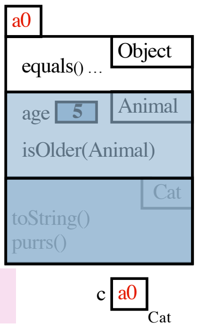
An attempt to cast it to anything else causes a ClassCastException.

(Cat) c

(Object) c

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

The **object** does not change.
The **perception** of it changes.



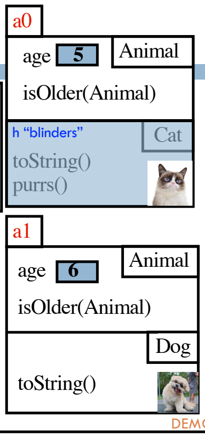
Implicit upward cast

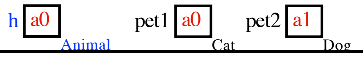
8

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

Cat pet1= new Cat(5);
Dog pet2= new Dog(6);
if (pet2.isOlder(pet1)) {…}

// pet1 is cast **up** to class
// Animal and stored in h





DEMO

9 Compile-time reference rule

Compile-time reference rule (v1)

see *JavaHyperText*

10

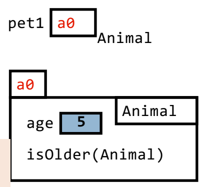
From a variable of type C, can reference only methods/fields that are available in class C.

```
Animal pet1= new Animal(5);
int m = pet1.purrs();
```

illegal

The compiler will give you an error.

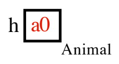
Checking the legality of pet1.purrs(…):
Since **pet1** is an Animal, purrs is legal only if it is declared in Animal or one of its superclasses.

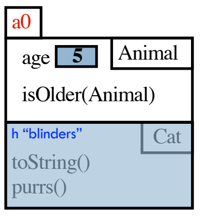


From an Animal variable, can use only methods available in class Animal

Quiz: Which references are legal?

11





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

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

C. `h.purrs()`
ILLEGAL —not in Animal partition or Object partition

12 Arrays

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

13

declaration of array v

Create array of 3 elements

Assign value of new-exp to v

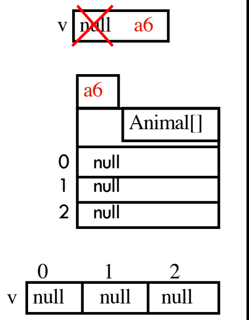
Assign and refer to elements as usual:

v[0]= **new** Animal(…);

…

a= v[0].getAge();

Sometimes use horizontal picture of an array:

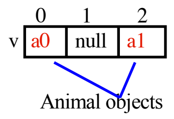


Array elements may be subclass objects

14

```
Animal[] v;          // declaration of v
v= new Animal[3];    // initialization of v
v[0]= new Cat(5);    // initialization of 1st elem
v[2]= new Dog(6);    // initialization of 2nd elem
```

The type of v is Animal[]
The type of each v[k] is Animal



Compile-time reference rule (CTRR), applied

15

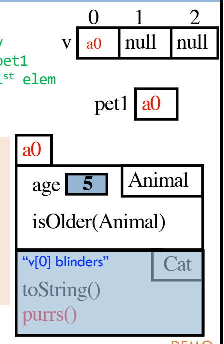
```
Animal[] v;           // declaration of v
v= new Animal[3];     // initialization of v
Cat pet1= new Cat(5); // initialization of pet1
v[0]= pet1;           // initialization of 1st elem
int m= v[0].purrs();  // is this allowed?
```

Not allowed!

Type of `v[0]` is Animal.

CTRR: May reference only methods available in Animal.

purrs is not declared in Animal or one of its superclasses.



DEMO

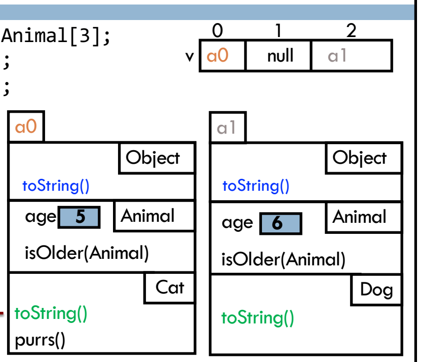
Contrast: Bottom-up rule, applied

16

```
Animal[] v= new Animal[3];
v[0]= new Cat(5);
v[2]= new Dog(6);
v[0].toString();
```

Which **toString()** gets called?

Bottom-up / Overriding rule says function toString in Cat partition



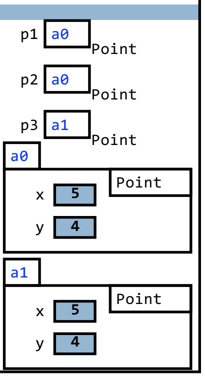
17 Equals

How **Object** defines **equals(o)**

18

```
public boolean equals(Object o) {
    return this == o;
}
```

```
Point p1= new Point(5,4);
Point p2= p1;

if (p1 == p2) {...}          // true?
if (p1.equals(p2)) {...}     // true?

Point p3= new Point(5,4);

if (p1 == p3) {...}          // true?
if (p1.equals(p3)) {...}     // true?
```



Defining equality for your own class

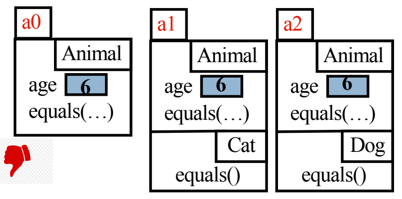
- **Specification:** Object.equals has a specification you must obey: reflexive, symmetric, transistive
  https://docs.oracle.com/javase/8/docs/api/java/lang/Object.html#equals-java.lang.Object-
- Reflexive x.equals(x)
- Symmetric x.equals(y) iff y.equals(x)
- Transitive if x.equals(y) and y.equals(z) then x.equals(z)

(Provided x and y are not null)

equals should say that x and y are equal iff they are indistinguishable

Are any of these equal?

Assume that Cat and Dog have no fields.



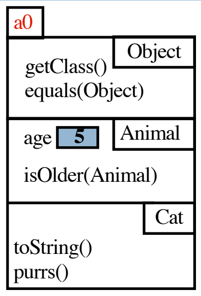
Can objects a1 and a2 be considered equal?

Can objects a0 and a1 be considered equal?

If the two objects are not of the same class (e.g. Cat, or Animal) they shouldn't be considered equal

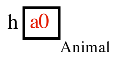
Function getClass and static field class

Instance method getClass() returns the class of the lowest partition in the object



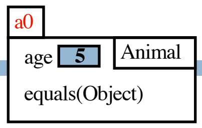
h.getClass() == Cat.class

h.getClass() != Animal.class



h.getClass() != Object.class

Equals in Animal

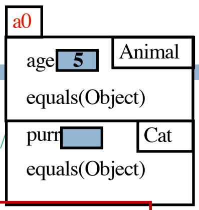


```
public class Animal {
  private int age;
  /** return true iff this and obj are of the same class
   * and their age fields have same values */
  public boolean equals(Object obj) {
      if (obj == null || getClass() != obj.getClass())  return false;
      Animal an= (Animal) obj;
      return age == an.age;
}
```

Almost every method equals that you write will have these three pieces

DEMO

Equals in Animal

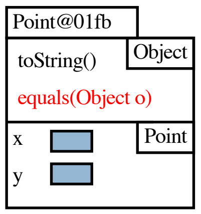


```
public class Animal {
  /** return true iff this and obj are of the
   * same class, age fields have same values */
  public boolean equals(Object obj) { … }

public class Cat extends Animal {
  /** return true iff this and obj are of the
   * same class and  age and purr fields have same values */
  public boolean equals(Object obj) {
      if (!super.equals(obj)) return false;
      Cat cob= (Cat) obj;
      return purr.equals(cob.purr);
}
```

DEMO

Object.equals

```
public class Point {
  public int x;
  public int y;

  public Point(int x, int y) {
    this.x= x;
    this.y= y;
  }
}
```



Equality for Points

25

```
public class Point {
   /** return "this and obj are of the same
       class, and this and obj have the same
       x and y fields"  */
   @Override
   public boolean equals(Object obj) {

     How can we tell whether this and obj are of the same class?
   }
}
```

Equality for Points

26

```
/** return "this and obj are of the same class and
     this and obj have the same x and y fields" */
@Override
 public boolean equals(Object obj) {
   if (obj == null || getClass() != obj.getClass())
      return false;
   Point p= (Point)obj;  // downcast to reference Point fields

   return  x == p.x && y == p.y;

}
```

Casting advice

27

function equals() requires casting

But, use of explicit down-casts can indicate bad design

DON'T:	DO:

if (…)
  do something with (C1) x
else if (…)
  do something with (C2) x
else if (…)
  do something with (C3) x
	x.do()
	
	… where do() is overridden in classes C1, C2, C3

Operator instanceof

28

obj instanceof C Is true if object obj has a partition named C.

```
if (s[k]  instanceof  Circle) {
   Circle cir=  Circle(s[k];

}
```