Subclasses: Private is Private!

```
public class Animal {
   private String name;
   private int age;

   public Animal(String n, int a) {
      name = n;
      age = a;
   } … }
```

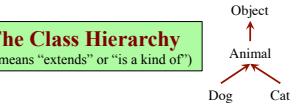
- Private = only in class
  - Excludes subclasses too!
- How access fields?
  - getters and setters
  - Use super() to initialize

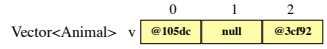
```
public class Cat extends Animal {
   public Cat(String n, int a) {
      name = n;
      age = a;
   } … }
```

```
public class Cat extends Animal {
   public Cat(String n, int a) {
      super(n,a);
   } …
}
```

Mixing Subclasses in Vector

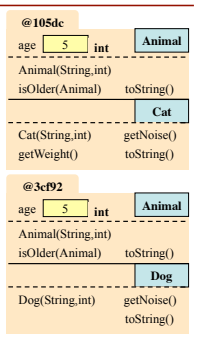
The Class Hierarchy
(→ means "extends" or "is a kind of")



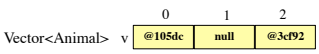


QUESTION:
Which method is called by v.get(0).toString()?

A: One in (hidden) Object part of **@105dc**
B: One in Animal part of **@105dc**
C: One in Cat part of **@105dc**
D: One in Dog part of **@3cf92**
E: None of these



Mixing Subclasses in Vector

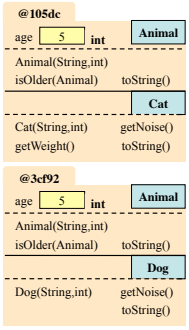


QUESTION:
Should the call

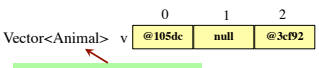
v.get(k).getWeight()

be allowed (**should the program compile**)?

A: Yes; v[0] has that method.
B: No; v[2] does not have that method.
C: No; it is not available in Animal.
D: None of these.



Apparent Type of an Expression



Apparently, v[k] is an Animal!

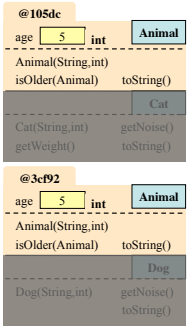
The call

v.get(k).getWeight()

is illegal (will not compile).

The apparent type of v[k] is Animal

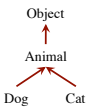
- Does not declare getWeight()
- Does not inherit getWeight()

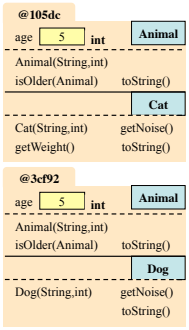


Casting Up and Down the Class Hierarchy

- Review of casting
  - (int) (5.0 / 7.5)
  - (double) 6
  - double d= 5; // automatic cast
- Can also cast class types:
  - Animal h = new Cat("N", 5);
  - Cat c = (Cat) h;

The Class Hierarchy
(→ means "extends" or "is a kind of")



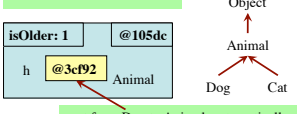


Implicit Casting in the Class Hierarchy

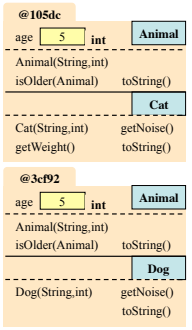
```
public class Animal {
   /** = "this is older than h" */
   public boolean isOlder(Animal h)
     { return this.age > h.age; }
}
```

Cat c = **new** Cat("C", 5);
Dog d = **new** Dog("D", 6);
c.isOlder(d) ?????

Casts **up** the hierarchy are automatic



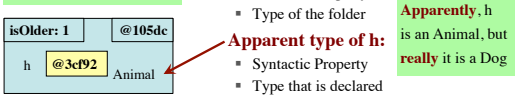
cast from Dog to Animal, automatically



Real vs. Apparent Type

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

```
Cat c = new Cat("C", 5);
Dog d = new Dog("D", 6);
c.isOlder(d)    ?????
```



Real type of h:
- Semantic Property
- Type of the folder

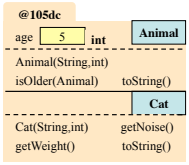
Apparent type of h:
- Syntactic Property
- Type that is declared

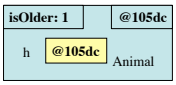
Apparently, h is an Animal, but **really** it is a Dog

What Can Variable h reference?

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

```
Cat c = new Cat("C", 5);
Dog d = new Dog("D", 6);
d.isOlder(c)    ?????
```



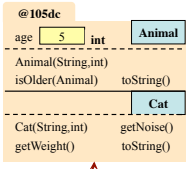


- **Apparent type** determines what methods calls are legal
- Cannot call h.getWeight();
  - This gives a syntax error
  - Even though real type is Cat

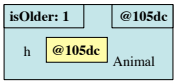
How Do We Resolve h.toString()?

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

```
Cat c = new Cat("C", 5);
Dog d = new Dog("D", 6);
d.isOlder(c)    ?????
```

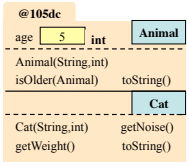


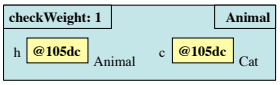
Determined by the real type of h



Casting Down the Class Hierarchy

```
public class Animal {
 /** If Animal is a cat, return weight; else return 0 */
 public static double checkWeight(Animal h) {
   if (!(h instanceof Cat)) {
     return 0;
   }
   // h is a Cat
   Cat c = (Cat)h;    // Downward cast
   return c.getWeight();
}}
```



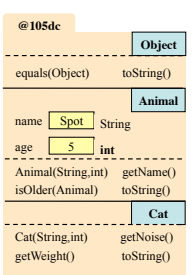


(Dog) h would lead to a **runtime error.**

You can't cast an object to something that it is not!

How to Override equals(Object)

```
public class Animal {
  /** Yields: "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;
  }
}
```



May want to define equals() in Cat and Dog.

A cat is not equal to a dog, even if they have the same name and age!

Overriding Versus Overloading

```
public class Animal { …
  public boolean equals(Object h) {
    if (!(h instanceof Animal)) {
      return false; }
    Animal ob= (Animal)h;
    return name.equals(ob.name) &&
           age == ob.age;
  }}
```

```
public class Cat extends Animal { …
  public boolean equals(Cat h) {
    return getName().equals(h.getName())
           && getAge() == h.getAge();
           && weight == h.weight;
}}
```

```
Cat c  = new Cat("C", 5);
Dog d = new Dog("C", 5);
c.equals(d)    ?????
```

- Method calls match on
  - Name of the method
  - Types of the parameters
- If no match:
  - Upcasts the arguments
  - Searches again for match