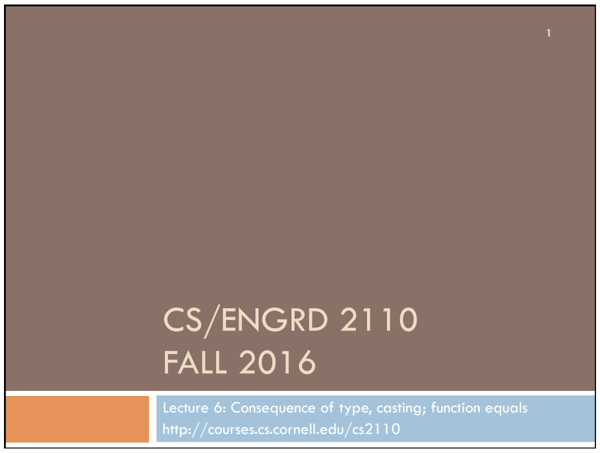
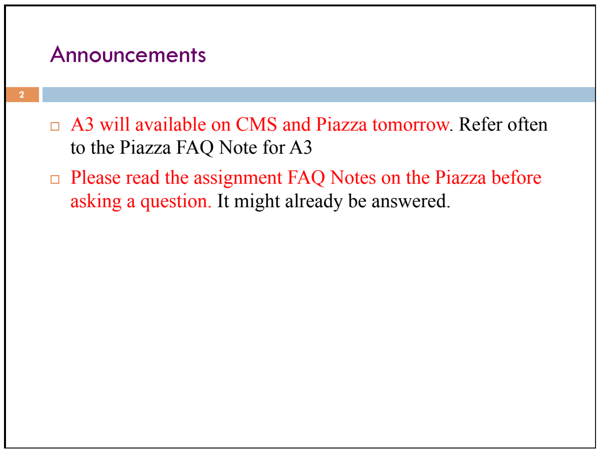


1

CS/ENGRD 2110
FALL 2016

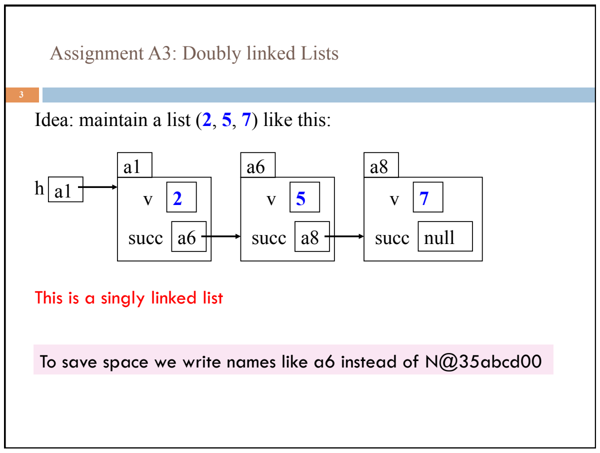
Lecture 6: Consequence of type, casting; function equals
http://courses.cs.cornell.edu/cs2110



## Announcements

2

- A3 will available on CMS and Piazza tomorrow. Refer often to the Piazza FAQ Note for A3
- Please read the assignment FAQ Notes on the Piazza before asking a question. It might already be answered.



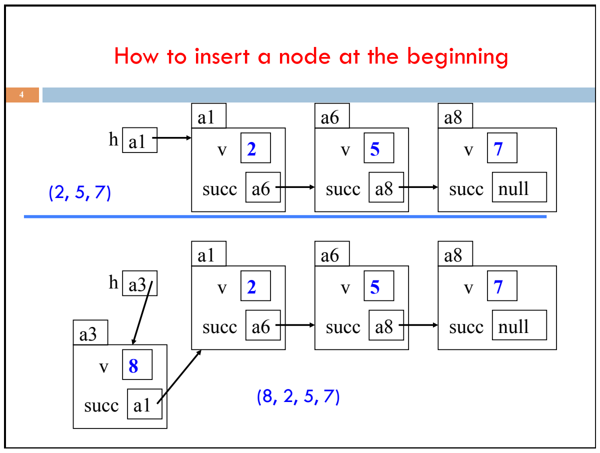
## Assignment A3: Doubly linked Lists

3

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

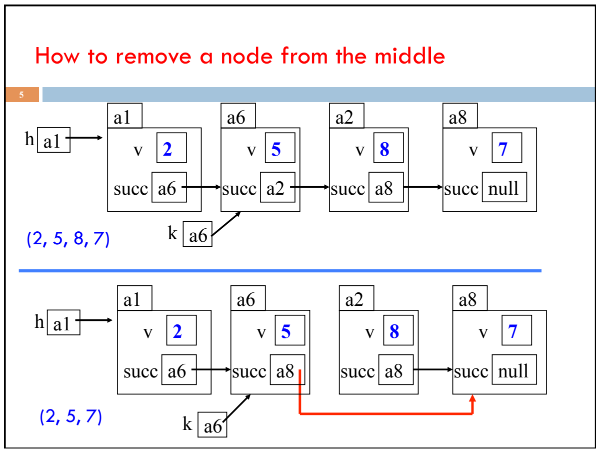
This is a singly linked list

To save space we write names like a6 instead of N@35abcd00



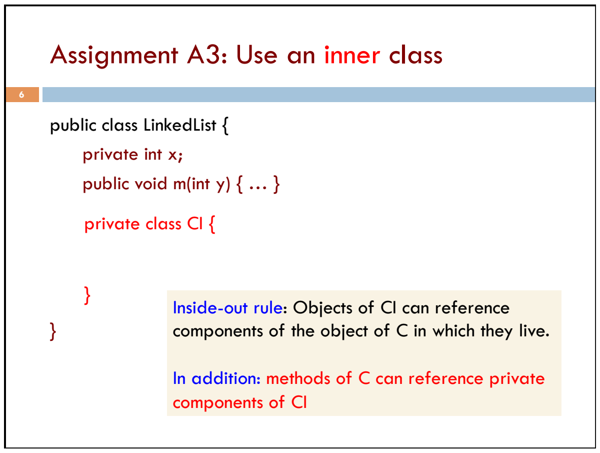
## How to insert a node at the beginning

4



## How to remove a node from the middle

5



## Assignment A3: Use an inner class

6

```
public class LinkedList {
    private int x;
    public void m(int y) { … }

    private class CI {


    }
}
```

Inside-out rule: Objects of CI can reference components of the object of C in which they live.

In addition: methods of C can reference private components of CI

## Assignment A3: Generics

7

```
public class LinkedList {
    void add(Object elem) {…}
    Object get(int index) {…}
}
```

Values of linked list are probably of class Object

```
public class LinkedList<E> {
    void add(E elem) {…}
    E get(int index) {…}
}
```

You can specify what type of values

```
ns = new LinkedList<Integer>();
ns.add("Hello"); // error
ns.add(5);
String s = ns.get(0); // error
int n = ns.get(0);
```

```
ss = new LinkedList<String>();
ss.add("Hello");
ss.add(5); // error
String s = ss.get(0);
int n = ss.get(0); // error
```

## Overview ref in text and JavaSummary.pptx

8

- 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

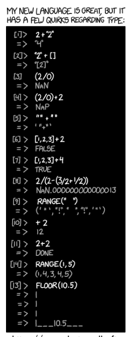
Homework. Learn about while/ for loops in Java. Look in text.

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

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

## Big Picture: Type Systems

9



MY NEW LANGUAGE IS GREAT, BUT IT HAS A FEW QUIRKS REGARDING TYPE:

```
[1]>  2+"2"
 =>   "4"
[2]>  "2"+[]
 =>   "[2]"
[3]   (2/0)
 =>   NaN
[4]>  (2/0)+2
 =>   NaP
[5]>  ""+""
 =>   '"+"'
[6]>  [1,2,3]+2
 =>   FALSE
[7]>  [1,2,3]+4
 =>   TRUE
[8]>  2/(2-(3/2+1/2))
 =>   NaN.0000000000000013
[9]>  RANGE(" ")
 =>   ('"', "!", " ", "!", '"')
[10]> +2
 =>   12
[11]> 2+2
 =>   DONE
[14]> RANGE(1,5)
 =>   (1,4,3,4,5)
[13]> FLOOR(10.5)
 =>   |
 =>   |
 =>   |
 =>   |___10.5___
```

https://www.destroyallsoftware.com/talks/wat

### Object types in Java

- Arrays
- Subtypes
- Method resolution
- Casts
- Binary methods

### Cornell Research

- Polyglot Compiler
- Object initialization
- Information-flow
- Pattern matching
- Decidability

## Classes we work with today

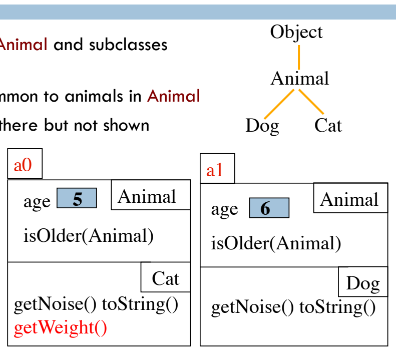
class hierarchy:

10

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];

11

declaration of array v

Create array of 3 elements

Assign value of new-exp to v

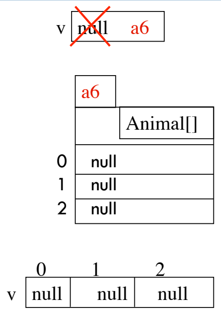
Assign and refer to elements as usual:

v[0]= **new** Animal(…);

…

a= v[0].getAge();

Sometimes use horizontal picture of an array:



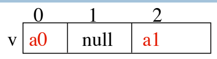
## Which function is called?

12

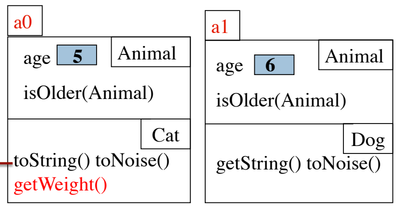
**Which function is called by**

**v[0].toString()  ?**

Remember, partition Object contains toString()



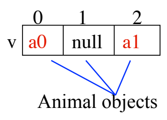
Bottom-up or overriding rule says function toString in Cat partition



## 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.

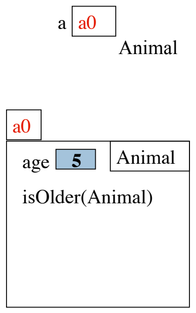


As we see on next slide, the type of a class variable like v[k] determines what methods can be called

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

a.getWeight() is obviously illegal. The class won't compile.

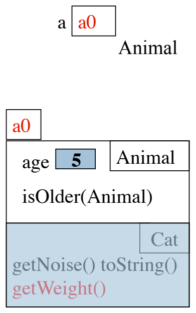
When checking legality of a call like
a.getWeight(…)
since the type of a is Animal, function getWeight must be declared in Animal or one of its superclasses.



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

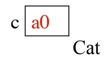
Suppose a0 contains an object of a subclass Cat of Animal. By the rule below, a.getWeight(…) is still illegal. Remember, the test for legality is done at compile time, not while the program is running. …

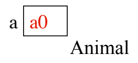
When checking legality of a call like
a.getWeight(…)
since the type of a is Animal, function getWeight must be declared in Animal or one of its superclasses.



## 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

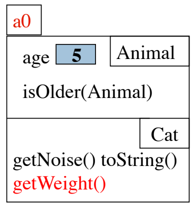


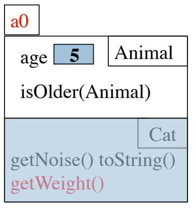


c.getWeight() is legal

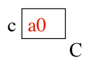
a.getWeight() is illegal

because getWeight is not available in class Animal

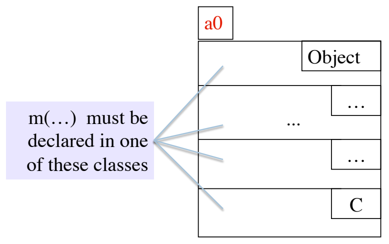




## Rule for determining legality of method call



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

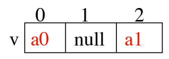
Type of v[0]: Animal

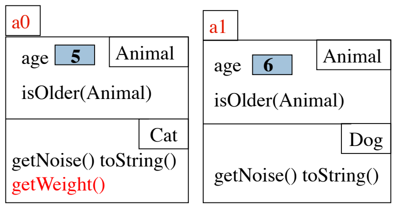
Should this call be allowed?
Should program compile?

v[k].getWeight()

Should this call be allowed?
Should program compile?

v[0].getWeight()





## View of object based on the type

19

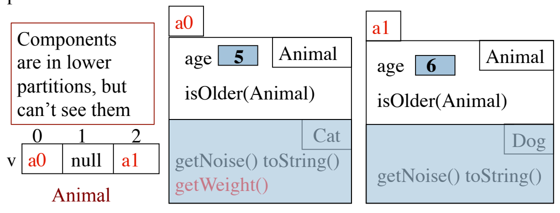
Each element v[k] is of type Animal.

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



## Casting objects

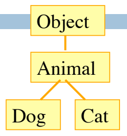
20

You know about casts like

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

**(double)** 6

**double** d= 5; // automatic cast

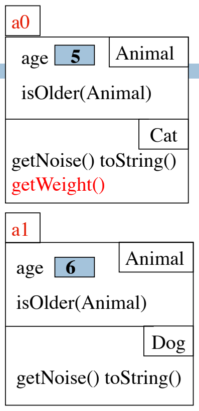


Discuss casts up/down class hierarchy.

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

21

**Rule**: 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.

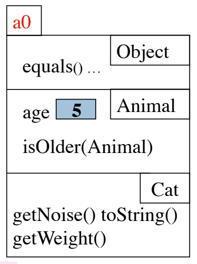
An attempt to cast it to anything else causes an exception

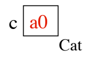
(Cat) c

(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

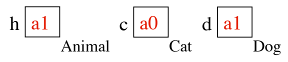
22

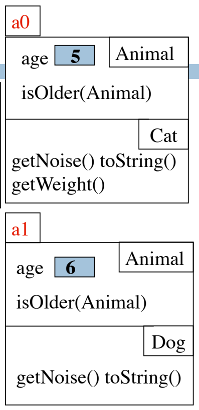
```
public class Animal {
   /** = "this Animal is older than h" */
   public boolean isOlder(Animal h) {
     return age > h.age;
   }
```

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





## Example

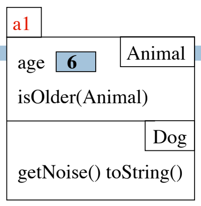
23

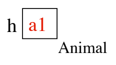
```
public class Animal {
   /** = "this is older than h" */
   public boolean isOlder(Animal h) {
     return age > h.age;
   }
```

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 implementation is called





## Components used from h

24

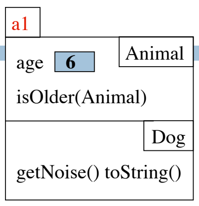
```
public class Animal {
   /** = "this is older than h" */
   public boolean isOlder(Animal h) {
     return age > h.age;
   }
```

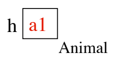
h.toString() OK —it's in class Object partition

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

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

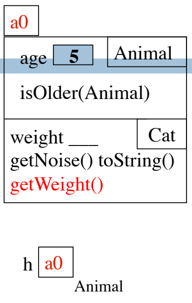
By overriding rule, calls toString() in Dog partition





## Explicit downward cast

```
public class Cat extends Animal {
  private float weight;
  /** return true if o is a Cat and its
    * fields have same values as this */
  public boolean equals(Object o) {
             ?
    // { h is a Cat }
    if ( ! super.equals(o) ) return false;
    Cat c= (Cat) o ;  // downward cast
    return weight == c.getWeight();
}
```

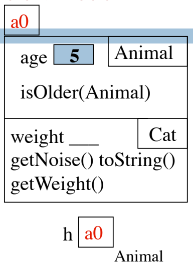


**(Dog) o** leads to runtime error.

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

## Operator instanceof, explicit down cast

```
public class Cat extends Animal {
  private float weight;
  /** return true if o is a Cat and its
    * fields have same values as this */
  public boolean equals(Object o) {
    if ( ! (o instanceof Cat) ) return false;
    // { h is a Cat }
    if ( ! super.equals(o) ) return false;
    Cat c= (Cat) o ;  // downward cast
    return weight == c.getWeight();
}
```



<object> **instanceof** <class>

true iff object has a partition for class

## Opinions about casting

- Using instanceof and downcasts often indicates bad design

| DON'T: | DO: |
|---|---|
| if (x instanceof C1)<br>  // do thing with (C1) x<br>else if (x instanceof C2)<br>  // do thing with (C2) x<br>else if (x instanceof C3)<br>  // do thing with (C3) x<br>… | x.doThing()<br><br>(doThing overridden in C1,C2,C3) |

- But how do I implement equals() ?
  - Object.equals has issues (but we're stuck with it)
    - Try guaranteeing that o1.equals(o2) implies o2.equals(o1)