

1

## CS/ENGRD 2110 SPRING 2015

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

### Announcements

- **A1 feedback available.** 30 still to be graded. Hopefully today. If yours is not graded yet, we appreciate your patience.  
Average 93.1 Median 95
- You can ask for a regrade of A1 on the CMS. State what you think we graded unfairly or incorrectly.
- **Your file A2.java must be in the default package.** Do NOT put a package statement at the top of it.
- **A3 now available on CMS and Piazza.** 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.
- **Read Note @282 on study habits!** Take its message to heart.

### Assignment A3: Doubly linked Lists

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

### Easy to insert a node in the beginning!

(2, 5, 7)

(8, 2, 5, 7)

### Easy to remove a node if you have its predecessor!

(2, 5, 8, 7)

(2, 5, 7)

### Assignment A3: Use an inner class

```
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

```

public class LinkedList {
}

public class LinkedList<E> {
}

new LinkedList<Integer>(...)
new LinkedList<String>(...)
new LinkedList<JFrame>(...)
    
```

Values of linked list are probably of class Object

You can specify what type of values

### 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
- 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
    
```

### Classes we work with today

class hierarchy:

```

graph TD
    Object --> Animal
    Animal --> Dog
    Animal --> Cat
    
```

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

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

Assign and refer to elements as usual:

```

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

Sometimes use horizontal picture of an array:

v	0	1	2
	null	null	null

a6	Animal[]
0	null
1	null
2	null

### Which function is called?

Which function is called by **v[0].toString()** ?

Remember, partition Object contains **toString()**

v	0	1	2
	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()	

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.

v	0	1	2
	a0	null	a1

Animal variables

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

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

**c** a0 **Cat**      **a** a0 **Animal**

**c.getWeight()** is legal      **a.getWeight()** is illegal because getWeight is not available in class Animal

age	5	Animal
Animal(String, int) isOlder(Animal)		
Cat(String, int)	Cat	
getNoise() toString() getWeight()		

age	5	Animal
Animal(String, int) isOlder(Animal)		
Cat(String, int)	Cat	
getNoise() toString() getWeight()		

### 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

**m(...)** must be declared in one of these classes

### Another example

Type of **v[0]**: Animal      Should this call be allowed? Should program compile? **v[0].getWeight()**

Should this call be allowed? Should program compile? **v[k].getWeight()**

age	5	Animal
Animal(String, int) isOlder(Animal)		
Cat(String, int)	Cat	
getNoise() toString() getWeight()		

age	6	Animal
Animal(String, int) isOlder(Animal)		
Dog(String, int)	Dog	
getNoise() toString()		

0 1 2  
v a0 null a1

### 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.

**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

age	5	Animal
Animal(String, int) isOlder(Animal)		
Cat(String, int)	Cat	
getNoise() toString() getWeight()		

age	6	Animal
Animal(String, int) isOlder(Animal)		
Dog(String, int)	Dog	
getNoise() toString()		

0 1 2  
v a0 null a1

Animal

### Casting up class hierarchy

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 perspective — how it is viewed!

age	5	Animal
Animal(String, int) isOlder(Animal)		
Cat(String, int)	Cat	
getNoise() toString() getWeight()		

age	6	Animal
Animal(String, int) isOlder(Animal)		
Dog(String, int)	Dog	
getNoise() toString()		

### Explicit casts: unary prefix operators

Rule: an object can be cast 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

equals() ...	Object	
age	5	Animal
Animal(String, int) isOlder(Animal)		
Cat(String, int)	Cat	
getNoise() toString() getWeight()		

**c** a0 **Cat**

### Implicit upward cast

```

19 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

age	5	Animal
Animal(String, int)	isOlder(Animal)	
Cat(String, int)	Cat	
getNoise() toString()	getWeight()	

age	6	Animal
Animal(String, int)	isOlder(Animal)	
Dog(String, int)	Dog	
getNoise() toString()		

h `a1` Animal    c `a0` Cat    d `a1` Dog

### Example

```

20 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 method is called

age	6	Animal
Animal(String, int)	isOlder(Animal)	
Dog(String, int)	Dog	
getNoise() toString()		

h `a1` Animal

### Components used from h

```

21 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 `Cat` partition

age	6	Animal
Animal(String, int)	isOlder(Animal)	
Dog(String, int)	Dog	
getNoise() toString()		

h `a1` Animal

### Explicit downward cast

```

22 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();
    }
}
    
```

(Dog) `h` leads to runtime error.  
 Don't try to cast an object to something that it is not!

age	5	Animal
Animal(String, int)	isOlder(Animal)	
Cat(String, int)	Cat	
getNoise() toString()	getWeight()	

h `a0` Animal

### Operator instanceof, explicit downward cast

```

23 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();
    }
}
    
```

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

age	5	Animal
Animal(String, int)	isOlder(Animal)	
Cat(String, int)	Cat	
getNoise() toString()	getWeight()	

h `a0` Animal

### Function equals

```

24 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 `?` Object    y `?` Object

### Overriding function equals

25

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

26

```

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()
...

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

### Function equals in class Animal

27

```

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

28

```

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()
...

3. Use `String equals` function to check for equality of `String` values. Use `==` for primitive types

### Why can't the parameter type be Animal?

29

```

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;
    }
    
```

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

What is wrong with this?