

1

CS/ENGRD 2110
SPRING 2017

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

Announcements

- A3 will available on Piazza tomorrow. Refer often to the Piazza FAQ Note for A3
- Please read the assignment A1 FAQ Notes on the Piazza before asking a question. It might already be answered.

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

How to insert a node at the beginning

How to remove successor of a node in the middle

Assignment A3: Use an inner class

```

public class C {
    private int x;
    public void m(int y) { ... ob= new Cin(...); ob.b= 5; }

    private class Cin {
        private int b;
        public void mm() { ... x= b; ... }
    }
}
    
```

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

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

Assignment A3: Generics

```

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

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

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

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
	isOlder(Animal)	
		Cat
	getNoise() toString() getWeight()	

a1	age 6	Animal
	isOlder(Animal)	
		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

Animal[]

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
	isOlder(Animal)	
		Cat
	toString() toNoise() getWeight()	

a1	age 6	Animal
	isOlder(Animal)	
		Dog
	getString() toNoise()	

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 objects

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

13

`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

14

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

15

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

16

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

17

Type of `v[0]`: `Animal`

Should this call be allowed? Should program compile?

`v[0].getWeight()`

`v[k].getWeight()`

View of object based on the type

18

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

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!

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 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

```
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

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

Example

```
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

```
h [a1] Animal
```

Components used from h

```
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

```
h [a1] Animal
```

Explicit downward cast

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

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

