Assignment A3: Doubly linked Lists

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

```
      a1
       v
     succ a6
      v
     succ a8
      v
     succ null
```

Easy to insert a node in the beginning!

Also, if we have a variable that contains a pointer to a node, it's easy to remove that node or insert another value before or after that node.

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

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

```
  a1
  Animal
  getWeight()
  getNoise()
  toString()
  age

  v
  Animal
  getWeight()
  getNoise()
  toString()
  age

  succ
  a6

  v
  Animal
  getWeight()
  getNoise()
  toString()
  age

  succ
  a8

  v
  Animal
  getWeight()
  getNoise()
  toString()
  age

  succ null
```

Sometimes use horizontal picture of an array:

```
[null null null]
```

About prelim 1

- October 2: 5:30PM and at 7:30PM. We will tell you which to attend.
- Problems with that one? You go to the other one.
- Students with conflicts -- having two prelims at 7:30 at that evening
  Take ours at 5:30 OR (take ours at 7:30 AND take makeup that the
  other class)
- New to Cornell? It is standard practice to take 2 prelims one evening
- People who HAVE to be out of town should email us the particulars --later,
  not now.
- Anyone who misses the prelim will have their grade based on prelim 2 and
  the final. They will HAVE to take the final (may be optional, in a way to be
  explained in November).
- Please don't email us about prelim 1 now. Too early. Too much going on
  now for us to handle it. We'll let you know when.
Which function is called by v[0].toString()?

Remember, partition Object contains toString.

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

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

From an Animal variable, can use only methods 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.

Animal[] v;

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.

Consequences of a class type

The type of v[k] is of type 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.

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.

Components are in lower partitions, but can’t see them.

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

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

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()

getWeight() not in class Animal or Object. Calls are illegal, program does not compile:

v[0].getWeight() v[k].getWeight()
Casting up class hierarchy

You know about casts like
(int) (5.0 / 7.5)
(double) 6
double de=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 maybe cast to Object, Animal, Cat.
An attempt to cast it to anything else causes an exception
(Cat) c
(Animal) c
(Dog) h

Explicit downward cast

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
Animal h

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

h a1 Animal

%0
age 5 Animal
Object
Animal
Cat
Dog

%1
age 6 Animal
Animal(String, int)
isOlder(Animal)
getNoise() toString()
getWeight()

%2
age 1 Animal
Animal(String, int)
isOlder(Animal)
getNoise() toString()
getWeight()

%3
age 6 Animal
Animal(String, int)
isOlder(Animal)
getNoise() toString()
getWeight()

%4
age 6 Animal
Animal(String, int)
isOlder(Animal)
getNoise() toString()
getWeight()
Operator instanceof, explicit downward cast

```java
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();
    }
}
```

Function equals

```java
public class Object {
    /** Return true iff this object is the same as ob */
    public boolean equals(Object b) {
        return this== b;
    }
}
```

Overriding function equals

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

```java
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;
    }
}
```

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

What is wrong with this?