Announcements

- A3 will be 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

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

```
   h a1 v 2 a6 v 5 a8 v 7
   succ a6 succ a8 succ null
```

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

```
   h a3 v 8 a2 v 2 a6 v 5 a8 v 7
   succ a6 succ a8 succ null
```

(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 |

How to remove a node from the middle

```
   h a1 v 2 a6 v 5 a2 v 8 a8 v 7
   succ a6 succ a2 succ a8 succ null
```

(2, 5, 8, 7)

```
   h a6 v 2 a2 v 5 a8 v 7
   succ a2 succ a8 succ null
```

(2, 5, 7)
Assignment A3: Generics

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

Values of linked list are probably of class Object

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

You can specify what type of values

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

```java
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

- 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.

```java
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

```
Object

Animal

Dog

Cat
```

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

```
Which function is called?
```

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

Remember, partition Object contains toString()
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

a0.getWeight() is obviously illegal.
The class won’t compile.

When checking legality of a call like
a0.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, a0.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
a0.getWeight(…) since the type of a is Animal, function getWeight must be declared in Animal
or one of its superclasses.

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[0].getWeight()

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

- $v[0]$: getWeight() not in class Animal or Object. Calls are illegal, program does not compile: $v[0].getWeight()$ $v[k].getWeight()$

View of object based on the type

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. $a_0$ 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.

Explicit casts: unary prefix operators

Casting objects

You know about casts like:

- (int) (5.0 / 7.5)
- (double) 6

A class cast doesn’t change the object. It just changes the perspective —how it is viewed!

Implicit upward cast

- (int) $a_0$
- (double) $a_1$

By overriding rule, calls toString() in Dog partition.

Components used from $h$

- (int) $a_0$
- (double) $a_1$

If a method call is legal, the overriding rule determines which implementation is called.

Type of $h$ is Animal. Syntactic property. Determines at compile-time what components can be used: those available in Animal.

Call isOlder($d$)

Variable $h$ is created. $a_1$ is cast up to class Animal and stored in $h$

Upward casts done automatically when needed:

- $h$
- $c$
- $a_0$
- $d$
- $a_1$

Call isOlder($h$)
Explicit downward cast

```java
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

```java
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:**
  ```
  if (x instanceof C1) { // do thing with (C1) x
  } else if (x instanceof C2) { // do thing with (C2) x
  } else if (x instanceof C3) { // do thing with (C3) x
  }
  ```

  **DO:**
  ```
  x.doThing();
  ```

- 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)