Announcements

- 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.

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 N035abcd00

Easy to insert a node at the beginning!

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 {
    Values of linked list are probably of class Object
}

public class LinkedList<E> {
    You can specify what type of values
}

new LinkedList<Integer>(…)
new LinkedList<String>(…)
new LinkedList<JFrame>(…)

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

Big Picture: Type Systems

Object types in Java
- Arrays
- Subtypes
- Method resolution
- Costs
- Binary methods

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

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

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

Which function is called?

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

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()
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.

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>a0</td>
<td>null</td>
<td>a1</td>
</tr>
<tr>
<td>Animal</td>
<td>Animal(String, int)</td>
<td>Animal(String, int)</td>
</tr>
<tr>
<td>getWeight()</td>
<td>isOlder(Animal)</td>
<td>isOlder(Animal)</td>
</tr>
</tbody>
</table>

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

v[0].getWeight() v[k].getWeight()

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.

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>a0</td>
<td>a1</td>
</tr>
<tr>
<td>Object</td>
<td>Animal</td>
</tr>
<tr>
<td>equals()</td>
<td>getWeight()</td>
</tr>
<tr>
<td>age</td>
<td>age</td>
</tr>
<tr>
<td>Animal</td>
<td>Animal</td>
</tr>
<tr>
<td>Animal(String, int)</td>
<td>Animal(String, int)</td>
</tr>
<tr>
<td>isOlder(Animal)</td>
<td>isOlder(Animal)</td>
</tr>
<tr>
<td>Cat(String, int)</td>
<td>Cat(String, int)</td>
</tr>
<tr>
<td>getNoise() toString() getWeight()</td>
<td>getNoise() toString()</td>
</tr>
</tbody>
</table>

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)

h is created. a1 is cast up to class Animal and stored in h

Upward casts done automatically when needed

h a1 c a0 d a1

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>a0</td>
<td>a1</td>
<td>a1</td>
</tr>
<tr>
<td>Animal</td>
<td>Cat</td>
<td>Dog</td>
</tr>
</tbody>
</table>

Casting up class hierarchy

You know about casts like

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

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!

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>a0</td>
<td>a0</td>
<td>a1</td>
</tr>
<tr>
<td>Animal</td>
<td>Animal</td>
<td>Cat</td>
</tr>
<tr>
<td>Animal(String, int)</td>
<td>Animal(String, int)</td>
<td>Animal(String, int)</td>
</tr>
<tr>
<td>isOlder(Animal)</td>
<td>isOlder(Animal)</td>
<td>isOlder(Animal)</td>
</tr>
<tr>
<td>Cat(String, int)</td>
<td>Cat(String, int)</td>
<td>Cat(String, int)</td>
</tr>
<tr>
<td>getNoise() toString() getWeight()</td>
<td>getNoise() toString() getWeight()</td>
<td></td>
</tr>
</tbody>
</table>

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 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

Type of h is Animal. Syntactic property.

Determines at compile-time what components can be used; those available in Animal

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1</td>
<td>a1</td>
</tr>
<tr>
<td>Animal</td>
<td>Animal</td>
</tr>
</tbody>
</table>

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

By overriding rule, calls toString() in Dog partition

[Animal]
### 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;
        } else {
            Cat c = (Cat) h; // downward cast
            return c.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;
        } else {
            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!*