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:

```
  h   a1
      ↓ 2
      ↓   6
      ↓   a6
      ↓ next a8
      ↓ v   next null
  a1
      v  5
      ↓ next   a2
      ↓ v  8
      ↓ next   a8
      ↓ v   next null
  a6
      v  null
```

This is a singly linked list

To save space we write names like a6 instead of N@35abco0

Easy to insert a node in the beginning!

```
  h   a1
      ↓ 2
      ↓   6
      ↓   a6
      ↓ next a8
      ↓ v   next null
  a1
      v  5
      ↓ next   a2
      ↓ v  8
      ↓ next   a8
      ↓ v   next null
  a6
      v  null
```

(2, 5, 7)

```
  h   a2
      ↓ 8
      ↓   6
      ↓   a6
      ↓ next a8
      ↓ v   next null
  a1
      v  5
      ↓ next   a2
      ↓ v  8
      ↓ next   a8
      ↓ v   next null
  a6
      v  null
```

(8, 2, 5, 7)

Easy to remove a node if you have its predecessor!

```
  h   a1
      ↓ 2
      ↓   6
      ↓   a6
      ↓ next a8
      ↓ v   next null
  a1
      v  5
      ↓ next   a2
      ↓ v  8
      ↓ next   a8
      ↓ v   next null
  a6
      v  null
```

(2, 5, 8, 7)

```
  h   a1
      ↓ 2
      ↓   6
      ↓   a6
      ↓ next a8
      ↓ v   next null
  a1
      v  5
      ↓ next   a2
      ↓ v  8
      ↓ next   a8
      ↓ v   next null
  a6
      v  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.
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

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

class hierarchy:

v = new Animal[3];

Which function is called?

Which function is called by v[0].toString()?
Remember, partition Object contains 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.

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

<table>
<thead>
<tr>
<th>v[k]</th>
<th>a[k]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Animal(String, int) isOlder(Animal)</td>
<td>Animal(String, int) isOlder(Animal)</td>
</tr>
<tr>
<td>Cat(String, int)</td>
<td>Cat(String, int)</td>
</tr>
<tr>
<td>getWeight()</td>
<td>getWeight()</td>
</tr>
</tbody>
</table>

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

- a0.getWeight() is legal because getWeight is not available in class Animal
- a0.getWeight() is illegal because getWeight is not available in class Animal

Should program compile?
Yes

Should this call be allowed?
Yes

Type of v[0]: Animal

Discuss casts up/down class hierarchy.

- Animal he = 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
(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

Another example

Type of v[0]: Animal

Should this call be allowed?
Should program compile?
v[0].getWeight()

Once a partition occurs above it, we can’t see them. If we have a Cat, we can’t look at the Cat’s methods.

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.

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

<table>
<thead>
<tr>
<th>a0</th>
<th>c</th>
<th>v[k]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Object</td>
<td>Animal</td>
<td>null</td>
</tr>
<tr>
<td>Animal</td>
<td>Animal</td>
<td>null</td>
</tr>
<tr>
<td>Animal(String, int) isOlder(Animal)</td>
<td>Animal(String, int) isOlder(Animal)</td>
<td>null</td>
</tr>
<tr>
<td>Cat(String, int)</td>
<td>Cat(String, int)</td>
<td>null</td>
</tr>
<tr>
<td>getWeight()</td>
<td>getWeight()</td>
<td>null</td>
</tr>
</tbody>
</table>

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

Casting up class hierarchy

You know about casts like

(int) 5.0 / 7.5
(double) 6

Discuss casts up/down class hierarchy.

Animal he = 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!
**Implicit upward cast**

```java
public class Animal {
    /** = "this Animal is older than h" */
    public boolean isOlder(Animal h) {
        return age > h.age;
    }
}
```

**Example**

```java
public class Animal {
    /** = "this is older than h" */
    public boolean isOlder(Animal h) {
        return age > h.age;
    }
}
```

**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();
    }
}
```

**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;
    }
}
```

**Components used from h**

```java
public class Animal {
    /** = "this is older than h" */
    public boolean isOlder(Animal h) {
        return age > h.age;
    }
}
```

**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;
    }
}
```

**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;
    }
}
```

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

2. In order to be able to reference fields in partition Animal, need to cast h to Animal

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

Why can’t the parameter type be Animal?

```java
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?