CS1110 15 Oct 2009 **Another classy lecture**

Casting about (secs 4.2, 4.3)

1. the class hierarchy
2. apparent and real classes
3. casting between classes
4. operator **instanceof**
5. function equals

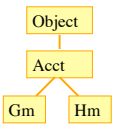
Reading for next time: Sec. 2.3.8 and chapter 7 on loops.

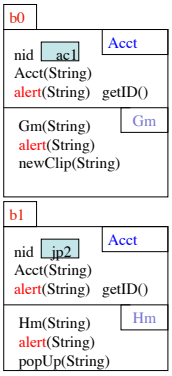
A4 due Friday

Time management tip #42: schedule deadlines on your calendar; *also schedule the time it will take to do the work*.

Setting: maintaining Cornell email accounts (Acct).
Two kinds so far: GMail (Gm) and Hotmail (Hm).

the class hierarchy:





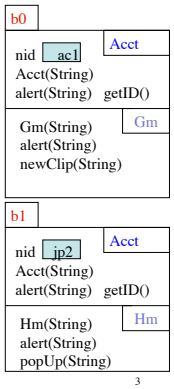
Both Gm and Hm override method alert(String), presumably in an application-specific way.
(Gmail might show a new "*Web clip*", whereas Hotmail might create a *popup*).

g b0 Gm h b1 Hm

a b0 Acct

The **apparent** (declared) type of **a** is Acct, and *will always be* Acct.
This is a syntactic property having to do with compiling.

The **real** type of **a**, the real class of the object whose name is *currently* in **a**, is Gm, but *could change* via assignment: **a**= h;
This is a semantic property having to do with the current value of **a**.

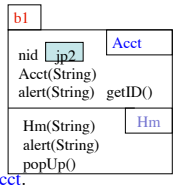


Implicit casting up the class hierarchy (good news)

h b1 Hm

Vector<Acct> v

0	1	2 …
b0	b14	…



h has apparent type Hm,
but our list v has an apparent type based on Acct.

Does this mean we *must* do an explicit cast to add h to v?

v.add((Acct) h);

Nope; luckily, casts **up** the hierarchy are automatic, allowing this:

v.add(h);

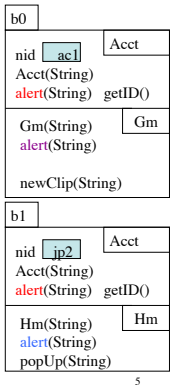
More good news:
Overriding (still) has the correct behavior

Vector<Acct> v

0	1	2
b0	null	b1

v.get(0).alert() will call the over-riding, Gmail-specific alert() method.

v.get(2).alert() will call the over-riding, Hotmail-specific alert() method.



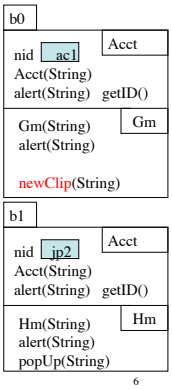
A sensible policy with an embedded "gotcha": The apparent type can rule out some available methods.

Vector<Acct> v

0	1	2
b0	null	b1

The *apparent* type of v, based on Acct, does *not* have a newClip method.

Therefore, the compiler rules the call v.get(0).newClip("FLOOD") **illegal**, even though in practice, the real type of v.get(0) might mean that newClip(…) would be available.



Workaround: check the real type.

a b0 Acct

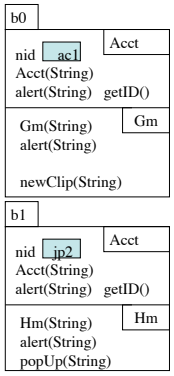
If we insist on calling newClip at all costs, then we need fresh variables of the right apparent type (Gm, not Acct).

To assign correctly to these fresh variables, we need to check the real type:

```
if ( a instanceof Gm) {
      Gm newG= (Gm) a;
      …
}
```

need this cast

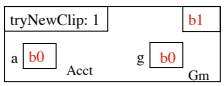
(can't just wedge "big" class into small)



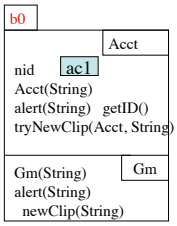
7

Example

```
public class Acct {
    // If Acct is a Gm, apply newClip,
       o.w, do nothing. (instance method just for lecture)
    public void tryNewClip(Acct a, String msg) {
      if ( ! (a instanceof Gm) )
          return;
      // a is a Gm
      Gm g= (Gm) a ;     // downward cast
      return g.newClip(msg);
}
```



Apparent type of a: Acct
Real type of a: Gm



Here, **(Hm) a** would lead to a runtime error.

Don't try to cast an object to something that it is not!

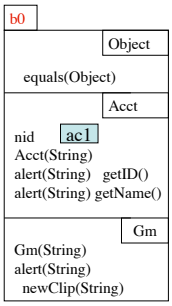
8

The correct way to write method equals

Note that method equals should take arbitrary Objects as arguments.

```
public class Acct {
    …
    /** = "h is an Acct with the same
           values in its fields as this Acct */
    public boolean equals (Object h) {

      if (!(h instanceof Acct)) return false;
      Acct a= (Acct) h;
      return  nid == a.nid;
}
```



9