CS110  Lec. 13  14 Oct 2010
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
6. abstract methods/classes (section 4.7, labs next week)

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

A4 due Saturday; make sure you're using Monday's a4.zip files (see assignments page on the course website for description of updates)

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

---

Implicit casting up the class hierarchy (good news)

<table>
<thead>
<tr>
<th></th>
<th>Cm</th>
<th></th>
<th>Um</th>
</tr>
</thead>
<tbody>
<tr>
<td>u</td>
<td>b1</td>
<td></td>
<td>b1</td>
</tr>
</tbody>
</table>

Vectors<Acct>

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>b0</td>
<td>0</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

[Not drawing Vectors as objects to save space]

u has apparent type Um, but our list v has an apparent type based on Acct.

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

v.add((Acct) u);

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

v.add(u);

---

Sources of apparent and real types

Apparent types come from declarations

real types come from assignment

---

More good news:

Overriding (still) has the correct behavior

First, the compiler checks that apparent type Acct has an alert method; if that succeeds, then the bottom-up rule is applied.

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

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

---

Setting: Cmail (Cm) and Umail (Um) accounts.

They have commonalities, like netIDs and an "alert" ability, so we make them subclasses of class Acct.

```
<table>
<thead>
<tr>
<th></th>
<th>Cm</th>
<th></th>
<th>Um</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cm(String)</td>
<td>Alert(String)</td>
<td>getID()</td>
<td></td>
</tr>
<tr>
<td>Um(String)</td>
<td>Alert(String)</td>
<td>newClip(String)</td>
<td></td>
</tr>
</tbody>
</table>
```

But, Cm and Um override Acct method alert(String), due to system differences.

Cmail shows a "Web clip"; Umail creates a popup.

Why do we keep drawing the over ridden alert?

[Note: Acct might best be made an abstract class; see last slide and next lab]
A sensible policy with an embedded “gotcha”: The apparent type can rule out some available methods.

Example

```
public class Acct {
    // If Acct is a Cm, apply newClip, o.w., do nothing.
    public static void tryClip(Acct a, String msg) {
        // a is a Cm
        if ( !a instanceof Cm ) { return; }
        Cm c = (Cm) a; // downward cast
        // why? bc objects don’t generally have nids
        return c.newClip(msg);
    }
}
```

Workaround: check the real type.

```
if ( a instanceof Cm) {
    Cm newG = (Cm) a;
    // downward cast
    Cm.c.newClip(msg);
```

Abstract methods and classes (see lab next week)

Make a (superclass) method abstract to force (non-abstract) subclasses to override it (and hence define it):

```
Example: In Acct (note stranded semi-colon!):

    public abstract void alert(String s);
```

means every sub-type of email account must have an alert method — different for different systems.

Make a (super)class abstract if there can only be subclass objects, but you still want default behaviors/info.

```
Example: Nothing is just a generic Animal (it’s a Pig, or a Butterfly, or a Person),and all live somewhere; but all Animals breathe oxygen. So, can’t create an "Animal":

    public abstract class Animal() {
        public abstract String habitat() ;
        public boolean breathesOxygen() (return true); }