Overriding (still) has the correct behavior.

v.get(0).alert().
v.get(2).alert().

Current value of this is a but object whose name is The compiling.

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

v.get(0).alert() will call the over-riding,
GMail (Gm) and Hotmail (Hm).

A sensible policy with an embedded “gotha”:
The apparent type can rule out some available methods.

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.

Implicit casting up the class hierarchy (good news)

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);

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

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).

Overriding (still) has the correct behavior.

The apparent (declared) type of a is Acct, and will always be Acct.
This is a semantic 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 = b;
This is a semantic property having to do with the current value of a.

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

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

```java
Acct a

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:
```java
if (a instanceof Gm) {
    Gm newG = (Gm) a;
    // need this cast (can't just wedge "big" class into small)
    }
```

Example

```java
public class Acct {
    // If Acct is a Gm, apply newClip, o.w., do nothing. (instanceof 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!

The correct way to write method equals

```java
public class Acct {
    /** = "h is an Acct with the same values in its fields as this Acct" */
    public boolean equals (Object h) {
        Acct a = (Acct) h;
        return nid == a.nid;
    }
}
```