Index of /~cs61b/lectures/11

Name	Last modified	Size	Description
Parent Directory	24-Sep-98 21:22	-	

```
                     CS61b: Lecture 11
                    Friday, September 18

ANNOUNCMENTS: HW5 will be online tonight; due 9/25 at 5pm.

INHERITANCE
-----------
So far, we've started from scratch in defining each class, its methods and data.

    class Worker {
      private int hunger;
      public Worker() { hunger = 0; }
      public void eat() {
        hunger -= 1;
        System.out.println("yum");
      }
      public void work() {
        hunger += 1;   // Or this.hunger
        System.out.println("whistle");
      }
      public void live() {
        this.work(); this.eat();   // OR just work(); eat();
      }
    }

In large programs, there are often many related classes.
We can identify at least two common relationships between classes

  Is-a: All objects in one class are also in another class
    E.g., a TA is a Worker, a Gardner is a Worker
  Has-a: All objects of one class contain and object of another
    E.g., an Album contains a List object, and a List contains a ListNode

We know how to implement a Has-a relationship by adding a data field
  of the given type to the class in question.
We could implement Is-a in the same way, but you are likely to
  add all of the data and methods from one class (TwoD) to the other (Square)

           Worker
         /      \             * a picture of an inheritance hierarchy
      TA      Contractor      * superclasses are above their subclasses
                /  \
      BrickLayer   Roofer

Java inheritance terminology:
  super class - a general class from which other classes inherit things
  extended class or subclass - a special class that inherits from a super class
```

EXTENDING CLASSES

A TA is a Worker with a class list used to verify that student are in their lab

```
    class TA extends Worker {           /* defines an edge in the hierarchy */
      public TA() { super(); }
      public void work() {
        super.work();      /* do whatever a Worker does to work */
        System.out.println("(let me help)");
      }
      public void add(String student) {
        classList =
          new NameList(student, classList);
      }
      private List classList = null;
    }
```

The advantage of inheritance: code re-use.
But care must be taken to avoid deep, complicated inheritance hierarchies.

PUBLIC, PRIVATE, and PROTECTED

In the above example, code in the TA class can access the public things
in Worker (as can any class), but not the hunger field, which is private.
If hunger were public in Worker, then any class (including TA) could access it.

If hunger is instead "protected", then TAs and subclasses of TA and other
subclasses of Worker can access it, but no other classes can.

```
      protected int hunger;
```

inside the add method on TA, we could then have:

```
      hunger += 1;    /* TAs get hungry at the thought of more students */
```

DYNAMIC VS STATIC TYPES

We can pass objects of a subtype to methods expecting the supertype.

```
    void foo (Worker w) { ... }    // it's OK to pass TA here too.
    void bar (TA t)     { ... }    // cannot pass Worker here.

  Worker aWorker = new Worker ();
  TA clark = new TA();
  Worker superman = clark; // an alias
  clark.live();            // Same as superman.live();
```

The add method is only defined on TA, not on Worker, so:

```
  clark.add("joe");        // OK
  aWorker.add("sue");      // ERROR
  superman.add("fred");    // ERROR
```

If we defined an add method on Worker, then both the ERRORs are OK

```
  aWorker.add("sue");      // Worker's add method
  superman.add("fred");    // TA's add method
```

OVERRIDING METHODS

A subclass may override a superclass method by defining one w/ same name/signature

```
  public void work() {           // inside the TA class
    hunger += 2;
```

```
    System.out.println("yawn");
  }
```

CLASS OBJECT

```
We have seen that Object can be used to declare variables       Object
  when we don't know what type the objects will be.          /    |     \
                                                       Integer  Worker  ...
This is because "Object" is automatically the top                 |  \
  of all inheritance hierarchies.                                 TA  ...
```

The primitive types in Java (int, boolean, etc.) are NOT extensions of Object.
The standard library implementations (Integer, Boolean, etc.) are.

What does the Object class really look like?

```
     public class Object {
        public String toString() { ... }   // returns a String representation
        public boolean equals(Object obj)  // test for equality
    }    ... lots of other stuff  omitted here
```