CS1110 Classes, stepwise refinement   23 Sep 2009

Miscellaneous points about classes.

More on stepwise refinement.

Prelim 7:30-9:00 Thursday, 7 October, Olin Hall 155 & 255
Review session: 1:00-3:00, Sunday, 3 Oct., Philips 101

Prelim conflict? Email Maria Witlox by Friday. Tell her what the conflict is (which course, work, reason for being out of town, etc.)
mwitlox@cs.cornell.edu

Next: wrapper classes. Section 5.1 of class text

A1: 130 done
82 to go

Let’s finish early this weekend.
You can still work on A3 as your A1 iterative process proceeds. But the A1 part must be completed before you submit A3.
Points are deducted on A3 if there are errors in A1.

Help: Get it now if you need it!!

• Call Cindy 255-8240 for an aptmnt with David Gries.
• Email Lillian Lee to make an aptmnt: lle@c.cornell.edu

• See a consultant in the ACCEL Lab:
  Sun, Mon, Tues, Wed, Thurs during office hours.

• See a TA.

• Peer tutoring (free). Ask in Olin 167 or visit http://www.engineering.cornell.edu, click on "student services". On the page that comes up, click on "Engineering Learning Initiatives (ELI.)" in the left column, upper part. Then, click on "peer tutoring" in the left column.

Content of this lecture

Go over miscellaneous points to round out your knowledge of classes and subclasses. There are a few more things to learn after this, but we will handle them much later.

• Inheriting fields and methods and overriding methods. Sec. 4.1 and 4.1.1; pp. 142-145
• Purpose of super and this. Sec. 4.1.1, pp. 144-145.
• More than one constructor in a class; another use of this. Sec. 3.1.3, pp. 110–112.
• Constructors in a subclass — calling a constructor of the super-class; another use of super. Sec. 4.1.3, pp. 147-148.

Then, we develop a nice function to anglicize integers, e.g. for 235, produce "two hundred thirty five".

Employee c = new Employee("Gries", 1969, 50000);
c.toString()

Which method toString() is called?

Overriding rule, or bottom-up rule:
To find out which is used, start at the bottom of the class and search upward until a matching one is found.

Terminology. Employee inherits methods and fields from Object. Employee overrides function toString.

Purpose of super and this
this refers to the name of the object in which it appears.
super is similar but refers only to components in the partitions above.

Employee:

/** = toString value from superclass */
public String toStringUp() {
  return super.toString();
}

Employee:

/** = String representation of this Employee */
public String toString() {
  return this.getName() + ", year " +
      getStart() + ", salary " + salary;
}

ok, but unnecessary

This class is on page 105 of the text.
A second constructor in Employee

public class Executive extends Employee {
    public Executive(String n, int d, double b) {
        bonus = b;
    }
}

/** Constructor: a person with name n, year hired d, salary s */
public Employee(String n, int d, double s) {
    name = n; start = d; salary = s;
}

/** Constructor: a person with name n, year hired d, salary 50,000 */
public Employee(String n, int d) {
    name = n; start = d; salary = 50000;
}

/** Constructor: a person with name n, year hired d, salary 50,000 */
public Employee(String n, int d) {
    this(n, d, 50000);
}

Here, this refers to the other constructor. You HAVE to do it this way

public class Executive extends Employee {
    private double bonus;

    /** Constructor: name n, year hired d, salary 50,000, bonus b */
    public Executive(String n, int d, double b) {
        super(n, d);
        bonus = b;
    }
}

The first (and only the first) statement in a constructor has to be a call on another constructor. If you don’t put one in, then this one is automatically used: super();

Principle: Fill in superclass fields first.

/** = the anglicization of n.
   Precondition: 0 < n < 1,000,000 */
public static String anglicize(int n) {
}

Principles and strategies

- Nicolaus Copernicus Principle
- Mathematica Principle
- Mañana Principle.

During programming, you may see the need for a new method. A good way to proceed in many cases is to:
1. Write the specification of the method.
2. Write just enough of the body so that the program can be compiled and so that the body method does something reasonable, but no the complete task. So you put off completing this method until another time — mañana (tomorrow) — but you have a good spec for it.
3. Return to what you were doing and continue developing at that place, presumably writing a call on the method that was just “stubbed in”, as we say.
<table>
<thead>
<tr>
<th>What numbers should we look at first?</th>
<th>What numbers should we look at first?</th>
</tr>
</thead>
<tbody>
<tr>
<td>A: Small numbers</td>
<td>A: 0..9</td>
</tr>
<tr>
<td>B: Numbers $\geq 100$</td>
<td>B: 1..9</td>
</tr>
<tr>
<td>C: Numbers $\geq 1000$</td>
<td>E: 1..10</td>
</tr>
<tr>
<td></td>
<td>C: 0..19</td>
</tr>
<tr>
<td></td>
<td>D: 1..19</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>How many test cases do we need to test ang19?</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A: 1</td>
<td></td>
</tr>
<tr>
<td>B: 2</td>
<td></td>
</tr>
<tr>
<td>E: 5</td>
<td></td>
</tr>
<tr>
<td>C: 10</td>
<td></td>
</tr>
<tr>
<td>D: 19</td>
<td></td>
</tr>
</tbody>
</table>