CS1110 22 Sep 2011

Miscellaneous points about classes.
More on stepwise refinement.

Next: wrapper classes.
Section 5.1 of class text

Need Help?

- Make apptmnt with Marschner or Gries.
- See consultant in ACCEL Lab
- See a TA.
- Peer tutoring (free). Olin 167

Prelim,
Thurs, 6 Oct, 7-9:30PM
Conflict?
Complete assignment P1Conflict on CMS

1

The new-expression

Some of you still are confused about how the new-expression is evaluated. If you do not understand how it is evaluated and cannot evaluate it yourself, you do not understand classes and objects. So much of object-oriented (OO) programming is embodied in evaluation of the new expression —that you *must* understand it.

Next slides: We again go through evaluation of a new-expression. As we do it, copy everything we do onto your own paper. Don't understand? Ask a question!!

After the lecture, memorize what we did!

2

t= **new** Book("Truth is all", 2345) ;

Above is an assignment statement. If we ask you to evaluate the expression in the assignment statement, you do that and nothing more. You don't mention variable t at all!

Execution of the assignment consists of 2 steps:

1. Evaluate the expression (here, **new** Book(…)) and
2. Store the value of the expression in the variable (here, t)

Why mention t when discussing evaluation of the expression?

We ask you to be more precise and careful in what you do than you have ever been —because programming requires it. Remember what the Director of Google Research said (see assignment A1 handout).

3

t

b7
constructor call

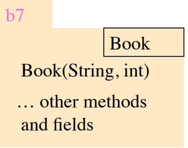
t= **new** Book("Truth is all", 2345) ;

In explaining how to evaluate the new-expression, you do NOT need to know what the definition of class Book is. You do NOT have to know what fields class Book has.

Step 1. Draw an object of class Book;

Step 2. Execute the constructor call; (you expect that it initializes the fields of b7)

Step 3. Use the name of the new object (b7) as the value of the new-expression.


b7

Book

Book(String, int)

… other methods and fields

If you want to complete the assignment, store the value in t. But it is NOT part of evaluating the new-expression.

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.

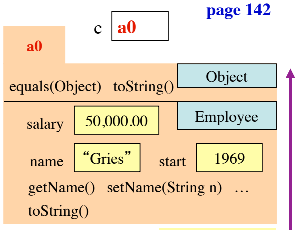
5

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

Sec. 4.1, page 142

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.



This class is on page 105 of the text.

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

6

Purpose of super and this

Sec. 4.1, pages 144-145

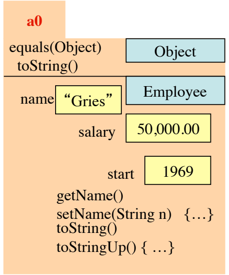
this refers to the name of the object in which it appears.
super is similar but refers only to components in the partitions above.

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

ok, but unnecessary

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

necessary



A second constructor in Employee
Provide flexibility, ease of use, to user

Sec. 3.1.3, page 110

```
/** 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;
}
```

First constructor

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

Second constructor; salary is always 50,000

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

Another version of second constructor; calls first constructor

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;
   }
}
```

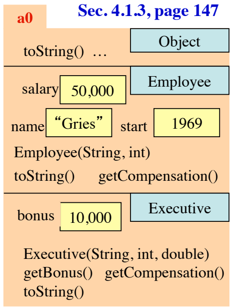
Calling a superclass constructor from the subclass constructor

Sec. 4.1.3, page 147

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.



Anglicizing an Integer

anglicize(“1”) is “one”
anglicize(“15”) is “fifteen”
anglicize(“123”) is “one hundred twenty three”
anglicize(“10570”) is “ten thousand five hundred seventy”

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

 }
```

Principles and strategies

Develop algorithm step by step, using principles and strategies embodied in “stepwise refinement” or “top-down programming. READ Sec. 2.5 and Plive p. 2-5.

- **Take small steps**. Do a little at a time
- **Refine**. Replace an English statement (what to do) by a sequence of statements to do it (how to do it).
- **Refine**. Introduce a local variable —but only with a reason
- **Compile often**
- **Intersperse programming and testing**
- **Write a method specifications** —before writing the bodies
- **Separate your concerns**: focus on one issue at a time

Principles and strategies

- 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 method body 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.