Slide 1

Congrats!! You now know the basics of OO (object-orientation).

Discussion of Methods: Executing method calls. If-statements. The return statement in a function. Local variables.

For this and next lecture: **Read chapter 2, but NOT 2.3.8!!!!**
Do the self-review exercises in 2.3.4

The last slide: local variables –variables declared within a method body. No time to discuss them. You are responsible for local variables. Read pp. 76-78 (sec. 2.3.7).

Sit next to someone. Today, we do some work in pairs. Get out a blank sheet of paper.

Get out Iclicker.

Conflict with Thursday Oct. 7th 7:30-9pm prelim? Email mwitlox@cs.cornell.edu with netid, other course name, other course instructor's email address BY NEXT THURSDAY.

1

Slide 2

CS1110   Thursday, 16 February 2010

A2 will take you 10-15 minutes.

Due on Thursday, in class.

Do it this weekend, not two minutes before class on Thursday

Sit next to someone. Today, we do some work in pairs. Get out a blank sheet of paper.

Get out Iclicker.
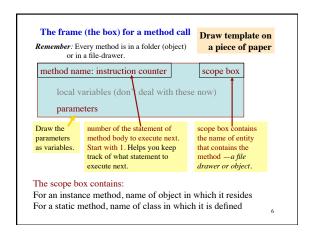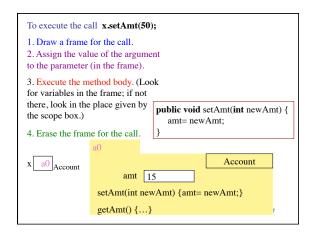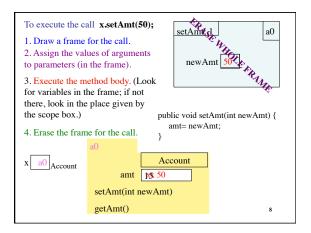
2

Slide 3

/** An instance keeps info about a book chapter */
public class Chapter {
    // class invariant: meanings of fields and constraints on them
    private int numb; // the chapter number, in range 0..100
    private String title; // chapter title
    private Chapter prev; // instance for the previous chapter
    …   // (null if no previous chapter)

c a0

a0
numb 2   Chapter
title "objects"
} prev a5   …

a5
numb 1   Chapter
title "intro"
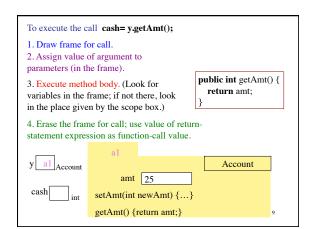prev **null**   …

Slide 4

/** An instance keeps info about a book chapter */
public class Chapter {
    // class invariant: meanings of fields and constraints on them
    private int numb; // the chapter number, in range 0..100
    private String title; // chapter title
    private Chapter prev; // instance for the previous chapter
    …   // (null if no previous chapter)

/** Change the previous chapter title to t.
    Precondition: previous chapter not null. */
public void changePrevTitle
                (String t) {
    prev.title= t;
} }

A. It will work
B. It won't compile
   --it has a syntax error
C. It will compile
   --but it won't run right

Slide 5
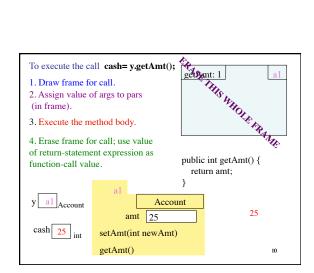
We write programs in order to do things.
Methods are the key "doers".

/** Constructor: a chapter with title t,
    number n, and previous chapter null.*/
public Chapter(String t, **int** n) {
    title= t;
    numb= n;
    previous= null;
}

Within the body (between { }) is the sequence of statements to execute when the method is called. ("Follow the recipe".)

parameters: t and n

declaration of parameter t

Memorize: a parameter is a variable that is declared within the parentheses of a method header.

But how is a method call executed?
How do parameters and arguments work?

5

Slide 6

The frame (the box) for a method call

Draw template on a piece of paper

Remember: Every method is in a folder (object) or in a file-drawer.

method name: instruction counter   scope box

local variables (don't deal with these now)

parameters

Draw the parameters as variables.

number of the statement of method body to execute next. Start with 1. Helps you keep track of what statement to execute next.

scope box contains the name of entity that contains the method —*a file drawer or object*.

The scope box contains:
For an instance method, name of object in which it resides
For a static method, name of class in which it is defined

6

1

Slide 7

To execute the call **x.setAmt(50);**

1. Draw a frame for the call.
2. Assign the value of the argument to the parameter (in the frame).

3. Execute the method body. (Look for variables in the frame; if not there, look in the place given by the scope box.)

```
public void setAmt(int newAmt) {
    amt= newAmt;
}
```

4. Erase the frame for the call.

a0

x | a0 | Account

Account

amt | 15

setAmt(int newAmt) {amt= newAmt;}

getAmt() {…}

7

Slide 8

To execute the call **x.setAmt(50);**

1. Draw a frame for the call.
2. Assign the values of arguments to parameters (in the frame).

3. Execute the method body. (Look for variables in the frame; if not there, look in the place given by the scope box.)

4. Erase the frame for the call.

setAmt: d ⟍ a0
newAmt | 50 ⟍ *ERASE WHOLE FRAME*

```
public void setAmt(int newAmt) {
    amt= newAmt;
}
```

a0

x | a0 | Account

Account

amt | 15 50

setAmt(int newAmt)

getAmt()

8

Slide 9

To execute the call **cash= y.getAmt();**

1. Draw frame for call.
2. Assign value of argument to parameters (in the frame).

3. Execute method body. (Look for variables in the frame; if not there, look in the place given by the scope box.)

```
public int getAmt() {
    return amt;
}
```

4. Erase frame for call; use value of return-statement expression as function-call value.

a1

y | a1 | Account

Account

amt | 25

cash | | int

setAmt(int newAmt) {…}

getAmt() {return amt;}

9

Slide 10

To execute the call **cash= y.getAmt();**

1. Draw frame for call.
2. Assign value of args to pars (in frame).

3. Execute the method body.

4. Erase frame for call; use value of return-statement expression as function-call value.

getAmt: 1 ⟍ a1
ERASE THIS WHOLE FRAME

```
public int getAmt() {
    return amt;
}
```

a1

y | a1 | Account

Account

amt | 25

cash | 25 | int

setAmt(int newAmt)

getAmt()

25

10

Slide 11

new Chapter("Intro", 1)

1. Draw a frame for the call.

2. Assign arg values to pars.

3. Execute the method body.

4. Erase the frame for the call.

```
Chapter(String t, int n) {
    String d;
    1: d= t;
    2: title= d;
    3: number= n;
    4: previous= null;
}
```

a8

Chapter

title | null

number | 0

previous | null

Chapter(String t, int n) { … }

Note *local* variable d declared within method body. It should be drawn in frame for call.

11

Slide 12

new Chapter("Intro", 1)

1. Draw a frame for the call.

2. Assign arg values to pars.

3. Execute the method body.

4. Erase the frame for the call.

Chapter: 1 2 ⟍ a8
t | "Intro"
n | 1   d | "Intro"
ERASE THIS WHOLE FRAME

```
Chapter(String t, int n) {
    1: String d= t;
    2: title= d;
    3: number= n;
    4: previous= null;
}
```

a8

Chapter

title | null "Intro"

number | 0 1

previous | null null

Chapter(…)

Note *local* variable d declared within method body. It should be drawn in frame for call.

12

Slide 13

```
/* swap x, y to put larger        /* Put smaller of x, y in  z */
   in y */                        if (x < y) {
if (x > y) {                          z= x;
   int t;                         }
   t= x;        if statement      else {        if-else statement
   x= y;                             z= y;
   y= t;                          }
}
```
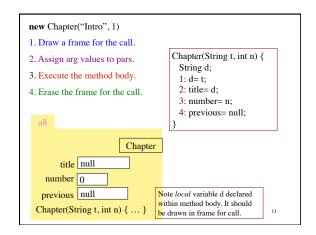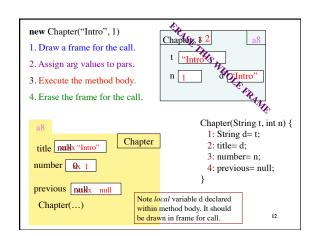
Syntax:
if (<boolean expression>)
  <statement>

Execution: if the <boolean expression> is true, then execute the <statement>

Syntax:
if (<boolean expression>)
  <statement1>
else <statement2>

Execution: if the boolean expression is true, then execute <statement1>; otherwise, execute <statement2>

13

Slide 14

Idiom: if statements and multiple return staements

```
/** = smallest of b, c, d */
public static int smallest(int b, int, c, int d) {
    if (b <= c  && b <= d) {
        return b;
    }
    // { The smallest is either c or d }
    if (c <= d) {
        return c;
    }
    // { the smallest is d }
    return d;
}
```

Execution of statement

    return <expr> ;

terminates execution of the procedure body and yields the value of <expr> as result of function call

Assertion

Execution of function body must end by executing a return statement. 14

Slide 15

Syntax of procedure/function/constructor and calls

public <result type> <name> (<parameter declarations>) { … } **function**

public void <name> (<parameter declarations>) { … } **procedure**

public <class-name> (<parameter declarations>) { … } **constructor**

Exec. of a function body *must* terminate by executing a statement "**return** <exp> ;", where the <exp> has the <result type>.

Exec. of a proc body *may* terminate by executing statement "**return** ;"

Exec. of a constructor body initializes fields of a new object in order to make the class invariant true.

<name> (<arguments>) **function call**

<name> (<arguments>) ; **procedure call**

new <class-name> (<arguments>) **constructor call**

<arguments>: <expression>, <expression>, …, <expression> 15

Slide 16

Scope of local variable is the places where it can be used. The scope is the sequence of statements following it within the containing "block".

```
/** = the max of x and y */
public static int max(int x, int y) {
    // Swap x and y to put the max in x
    if (x < y) {
        int temp;      scope of temp
        temp= x;
        x= y;
        y= temp;
    }                  You can't use temp down here

                       This is an error.

    return x;
}
```

16

3