CS1110 29 Jan. Customizing a class & testing

- Fields; getter & setter methods.
  Secs 1.4.1 (p. 45) & 3.1 (pp. 105–110 only)
- Constructors. Sec. 3.1.3 (p. 111–112)
- Testing methods. Appendix I.2.4 (p. 486)

Next time:

Testing using JUnit.

Object: the superest class of them all. pp 153–154.

Function toString.

Static components Sec. 1.5 (p. 47).

Quiz 2 on Tuesday (3 February):
Purpose of a constructor (slide 6)
Evaluating a new expression (slide 7)

Quote for the day:
There is no reason anyone would want a computer in their home.
--Ken Olson, founder of Digital Equipment Corp. (DEC), 1977.
The company was a huge player in computer hardware and software in CS academia in the 1970's. The PDP machines were well known. The VAX had unix on it, and C, and Lisp. It was the main computer in most CS departments of any stature. DEC was bought by COMPAQ in the late 1990's.

One-on-One Sessions

Next two weeks, hold a 1/2-hour one-on-one session on a computer with each student in CS1110.

Purpose: See how well you understand what we have done, let you ask questions, give you help. Graded 0-1: you get 1 if you took part in a session. Not counted in course grade. Purpose: to help you.

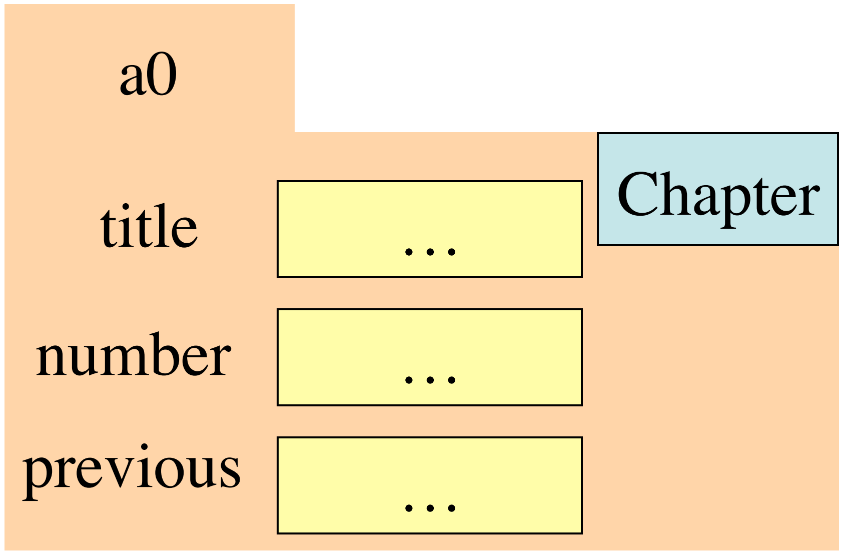
Instructors : Gries, Lee, TAs, consultants.

How to sign up: Visit the course CMS (from column of links on course home page). Click on assignment One-on-one. You will see a list of times and instructors. Choose one. First-come-first-served.

Not registered in the CMS? Email Maria Witlox immediately and ask her to register you: mwitlox@cs.cornell.edu

Field: a variable that is in each folder of a class.

We generally make fields **private** instead of **public**, so that they cannot be referenced from methods that are outside the class.



```
public class Chapter {

    private String title; // Title of the chapter

    private int number; // Number of the chapter

    private Chapter previous; // previous chapter (null if none)
}
```

Declarations of fields

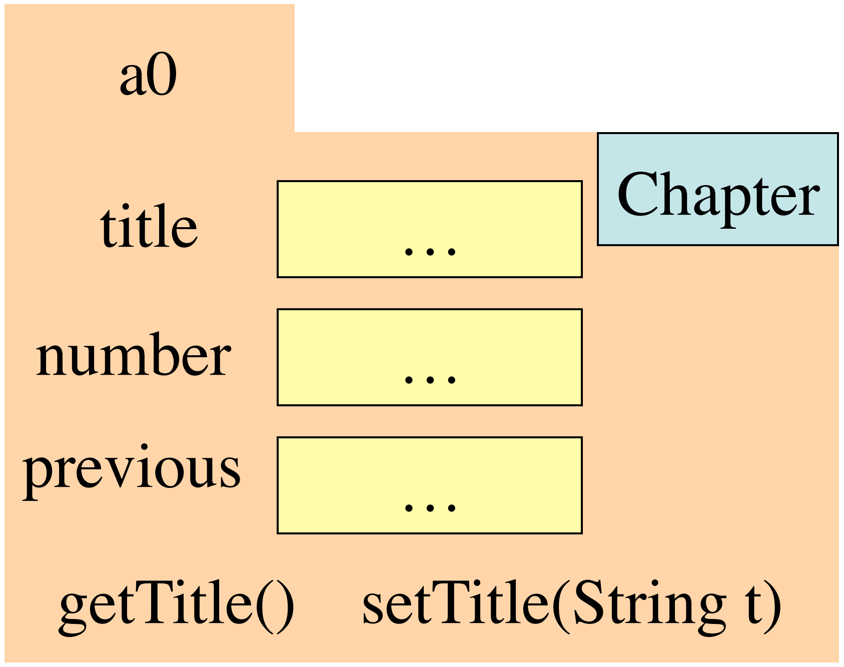
Getter and setter methods

```
/** An instance describes a chapter
    of a book */
public class Chapter {
  // Title of the chapter
  private String title;

  /** = title of the chapter */
  public String getTitle() {
    return title;
  }

  /** Set chapter title to t */
  public void setTitle(String t) {
    title= t;
  }


}
```



Getter methods (functions) **get** or retrieve values from a folder.

Setter methods (procedures) **set** or change fields of a folder

Initialize fields when a folder is first created

new Chapter()

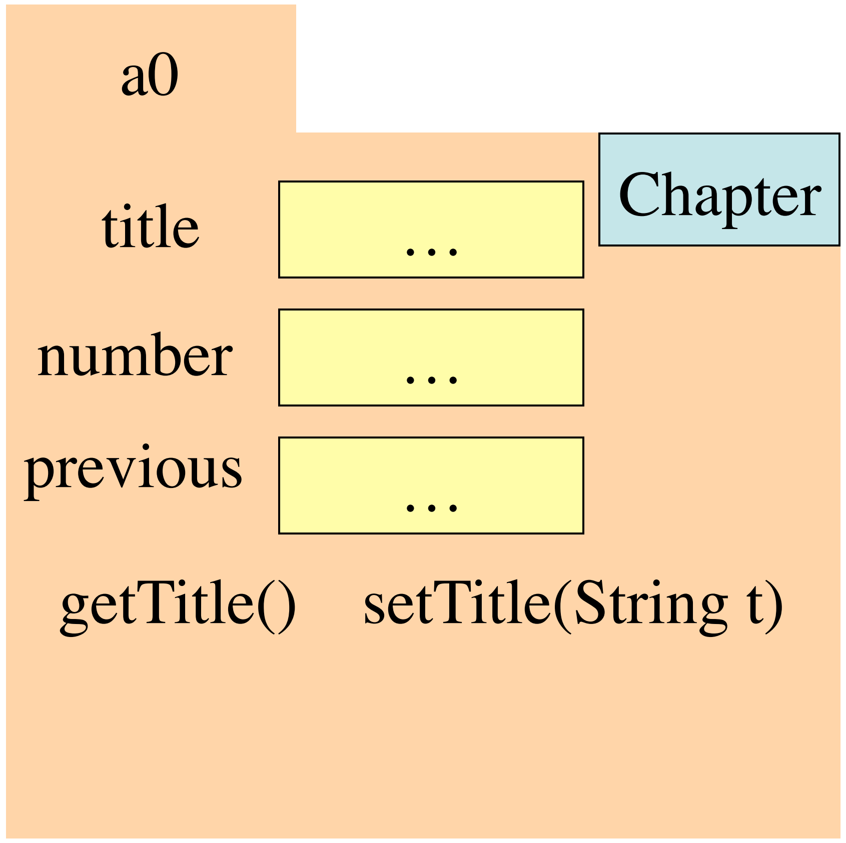
create an object but doesn't allow us to say what values should be in it.

We would like to be able to use:

new Chapter("I am born", 1, **null**)

to set the title to "I am born", the chapter number to 1, and the previous chapter to **null**.

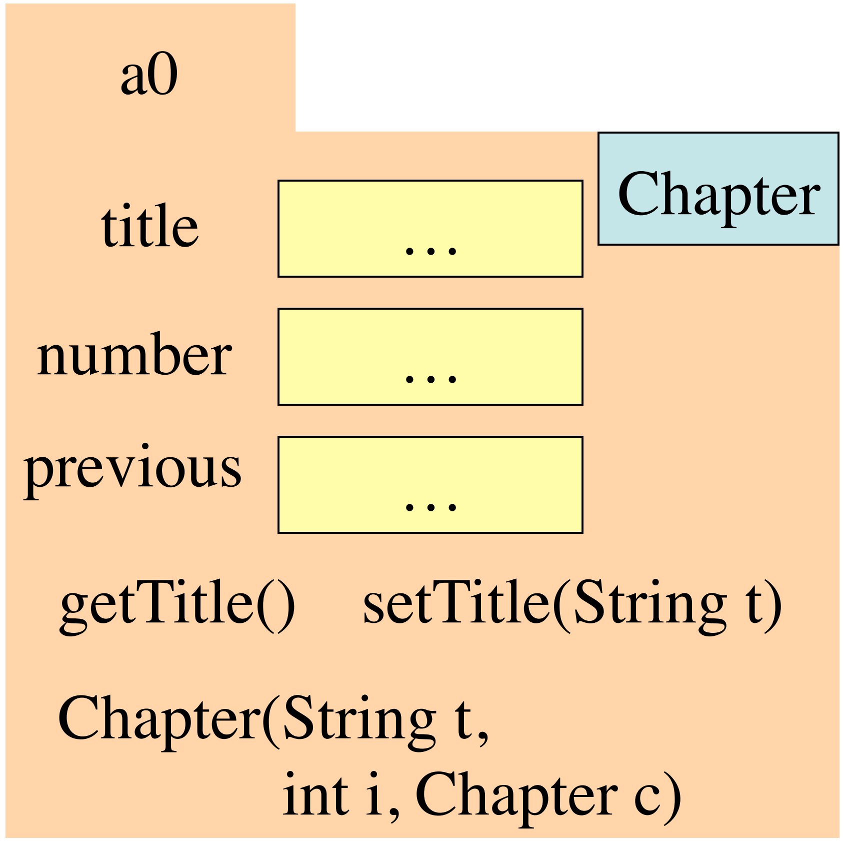
For this, we use a new kind of method, the **constructor**.



Purpose of a constructor:
To initialize (some) fields of a newly created object

```
/** An instance describes a chapter of a book */
public class Chapter {
    private String title; // Title of chapter
    private int number; // No. of chapter
    private Chapter previous; // previous
                          // chapter (null if none)

    /** Constructor: an instance with title
          chapter number i, and previous
          chapter p (null if none) */
    public   Chapter(String t, int i,
                             Chapter p) {
        title= t;
        number= i;
        previous= p;
    }
}
```



The name of a constructor: the name of the class.

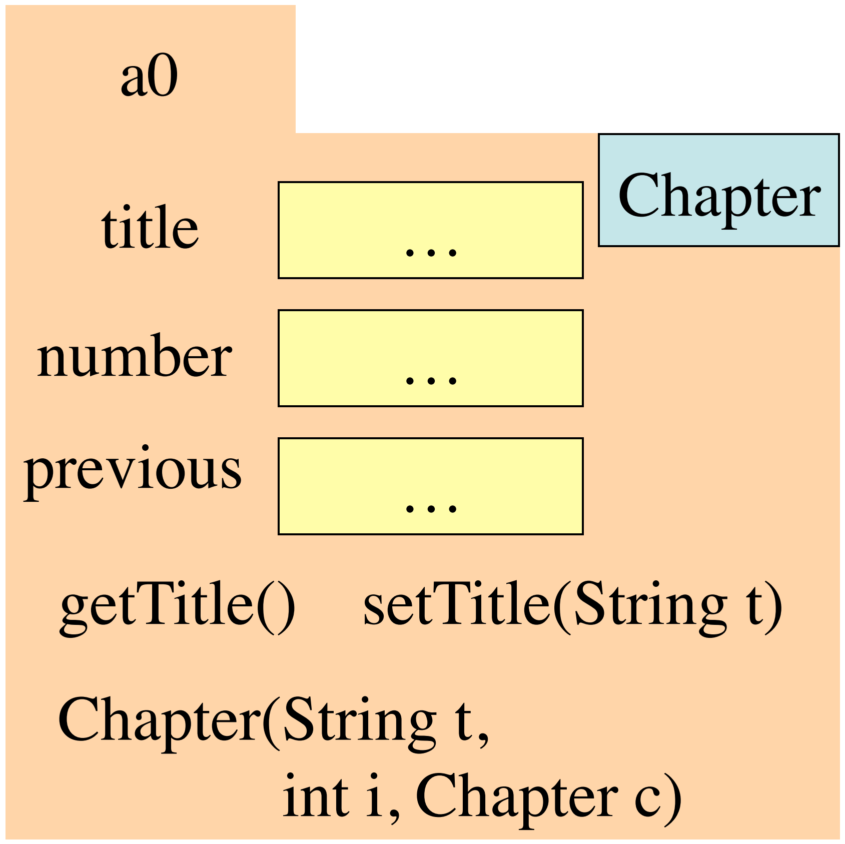
Do not put a type or **void** here

New description of evaluation of a new-expression

new Chapter("I am born", 1, **null**)

1. Create a new folder of class Chapter, with fields initialized to default values (e.g. 0 for **int**) –of course, put the folder in the file drawer.
2. Execute the constructor call

   Chapter("I am born", 1, null)
3. Use the name of the new object as the value of the new-expression.



Memorize this new definition! Today! Now!

Testing —using JUnit

Bug: Error in a program.

Testing: Process of analyzing, running program, looking for bugs.

Test case: A set of input values, together with the expected output.

Debugging: Process of finding a bug and removing it.

Get in the habit of writing test cases for a method from the specification of the method even before you write the method.

A feature called **Junit** in DrJava helps us develop test cases and use them. You *have* to use this feature in assignment A1.

1. c1= **new** Chapter(“one”, 1, **null**);
   Title should be: “one”; chap. no.: 1; previous: **null**.

Here are two test cases

2. c2= **new** Chapter(“two”, 2, c);
   Title should be: “two”; chap. no.: 2; previous: c1.

Need a way to run these test cases, to see whether the fields are set correctly. We could use the interactions pane, but then repeating the test is time-consuming.

To create a testing framework: select menu **File** item **new Junit test case…**. At prompt, put in class name **ChapterTester**. This creates a new class with that name. Save it in same directory as class Chapter.

The class imports **junit.framework.TestCase**, which provides some methods for testing.

```
/** A JUnit test case class.
 * Every method starting with "test" will be called when running
 * the test with JUnit. */
public class ChapterTester extends TestCase {

    /**  A test method.
     * (Replace "X" with a name describing the test.  Write as
     * many "testSomething" methods in this class as you wish,
     * and each one will be called when testing.) */
    public void testX() {
    }
}
```

One method you can use in testX is

assertEquals(x,y)

which tests whether expected value x equals y

A testMethod to test constructor and getter methods

```
/** Test first constructor and getter methods getTitle,
    getNumber, and getPrevious */
public void testConstructor() {
    Chapter c1= new Chapter("one", 1, null);
    assertEquals("one", c1.getTitle(), );
    assertEquals(1, c1.getNumber());
    assertEquals(null, c1.getPrevious());

    Chapter c2= new Chapter("two", 2, c1);
    assertEquals("two", c2.getTitle());
    assertEquals(2, c2.getNumber());
    assertEquals(c1, c2.getPrevious());
}
```

first test case

second test case

assertEquals(x,y):

test whether **x** equals **y** ; print an error message and stop the method if they are not equal.

x: expected value,
y: actual value.

A few other methods that can be used are listed on page 488.

Every time you click button Test in DrJava, this method (and all other testX methods) will be called.