CS100J 7 February 2006

In 1968, the Defense Department hired Bolt Beranek and Newman (BBN) of Boston to help develop the ARPANET, which later turned into the internet. In 1971, Ray Tomlinson of BBN was given the task of figuring out how to send files from one person to another. He created email with file attachments. He selected @ as the separator between an email name and location. Names for @ in other languages:

Italian: chiocciolina = little snail
French: petit escargot = little snail
German: klammeraffe = spider monkey
Dutch: api = short for apestaart (monkey's tail)
Finnish: miau = cat tail
Israeli: strudel = a pastry
Danish: snabel = an "A" with a trunk
Spanish: un arroba = a unit of about 25 pounds
Norwegian: kanel-bolle = spiral-shaped cinnamon cake

For more info: http://www.mailmsg.com/history.htm

TODAY:
- Note on method specs
- Static variables. Sec. 1.5 (p. 47)
- Testing and the use of Junit testing. Sec. 14.1.1 (p. 385-388)


```
/** Each instance describes a chapter in a book * */
public class Chapter {
  private String title; // The title of the chapter
  private int number; // The number of chapter
  private Chapter previous; // previous chapter (null if none)

  /** Constructor: an instance with title t, chap n, previous chap c */
  public Chapter(String t, int n, Chapter c)
    { title= t; number= n; previous= c; }

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

  /** = number of this chapter */
  public int getNumber() { return number; }

  /** = (name of) the previous chapter (null if none) */
  public Chapter getPrevious() { return previous; }
}
```

Download class from course web page.

Today, we will use a class **Chapter**: an instance of which describes a book. Here, we have a constructor and three getter methods

About method specifications

A **precondition** is a restriction that a call of a method must satisfy. Our convention is that the method need not check that the precondition holds.

```
/** = the chapter number of Chapter c.
      Precondition: c should not be null */
public boolean chapterNumber(Chapter c) {
        return c.number;
}
```

Up to caller to make sure c is not **null**; No need in this class to check in method body.

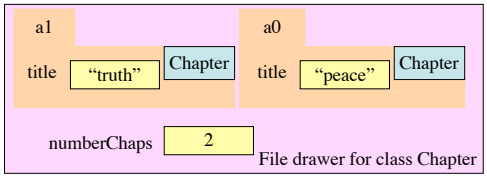
```
/** = "c is not null and has chapter number 0" */
public static boolean isZero(Chapter c) {
        return c != null  && c.number == 0;
}
```

The fact that c is not **null** is not given as a precondition but as something that the method body should check.

A static field does not appear in each folder.
It appears in the file drawer, by itself, on a piece of paper.
There is only ONE copy of it.

```
public class Chapter {
  private String title; // title of chapter
  private static int numberChaps= 0; // no. of folders created
}
```

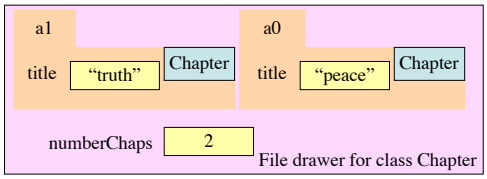
Reference the static variable using **Chapter.numberChaps**



A static field does not appear in each folder.
It appears in the file drawer, by itself, on a piece of paper.
There is only ONE copy of it.

```
public class Chapter {
  private int title; // title of chapter
  private static int numberChaps= 0; // no.of folders created
}
```

Use a static variable when you want to maintain information about all (or some) folders.



Make a method **static** when it does not refer to any of the fields or methods of the folder.

```
public class Chapter {
  private int number; // Number of chapter
  private static int numberOfChapters= 0;

   /** = "This chapter has a lower chapter number than Chapter c".
         Precondition: c is not null. */
   public boolean isLowerThan(Chapter c) {
       return number < c.number;
   }
   /** = "b's chapter number is lower than c's chapter number".
         Precondition: b and c are not null. */
   public static boolean isLower(Chapter b, Chapter c) {
       return b.number < c.number;
   }
}
```

CS100J 13 September 2005. Testing.

1. **Testing** --using Junit. Pages 385-388 (through Sec. 14.1.1).

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 A2.

1. c1= **new** Chapter("one", 1, **null**);
   Title should be: "one"; chap. no.: 1; previous: **null**.
2. c2= **new** Chapter("two", 2, c1);
   Title should be: "two"; chap. no.: 2; previous: c1.

Here are two test cases

/** = a String that consists of the first letter of each word in s.
E.g. for s = "Juris Hartmanis", the answer is "JH".
Precondition: s consists of a name in the form "first last" or "first middle last", with one or more blanks between each pair of names. There may be blanks at the beginning and end.
public String initialsOf(String s) {

}

1. c1= **new** Chapter("one", 1, **null**);
   Title should be: "one"; chap. no.: 1; previous: **null**.
2. c2= **new** Chapter("two", 2, c);
   Title should be: "two"; chap. no.: 2; previous: c1.

Here are two test cases

We 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 framework for testing in DrJava, select menu **File** item **new Junit test case…**. At the prompt, put in the class name **ChapterTester**. This creates a new class with that name. Immediately save it —in the 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 the word "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.  You may 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 first constructor

```
/** Test first constructor and getter methods getTitle,
    getNumber, and getPrevious */
public void testFirstConstructor() {
    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.

A testMethod to test setter methods

```
/** Test Setter methods setTitle, setNumber, and setPrevious */
public void testSetters() {
    Chapter c1= new Chapter("one", 1, null);
    c1.setTitle("new title");
    c1.setNumber(18);
    Chapter c2= new Chapter("two", 2, null);
    c1.setPrevious(c2);
    assertEquals("new title", c1.getTitle());
    assertEquals(18, c1.getNumber());
    assertEquals(c2, c1.getPrevious());
}
```

assertEquals(x,y):

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

x is the expected value, **y** the actual value.

For the method below, use THREE test cases: one when <, one when =, one when >

```
/** = the chapter no of this chapter is <= c's chapter number.
      Precondition: c is not null */
public boolean isAtMost(Chapter c) {…}
```