CS100J 6 February 2007

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

TODAY:
- Object: the superest class of them all. pp 153-154.
- Function toString.
- Static variables and methods. Sec. 1.5 (p. 47).
- Testing using JUnit.

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

1

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

2

Class Object: The superest class of them all

Every class that does not extend another one automatically extends class Object.

public class C { … }

is equivalent to

public class C **extends** Object { …}

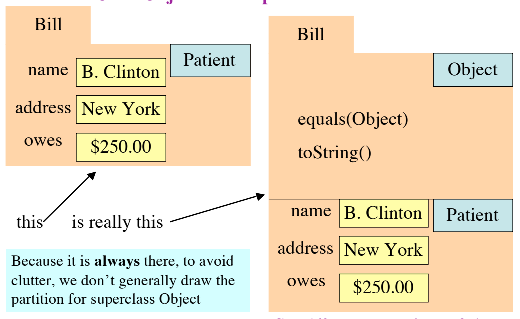
See 1/2-page section 4.3.1 on page 154.

The reason for this will become clear later.

You need this information to do assignment A2.

3

Class Object: The superest class of them all



Because it is **always** there, to avoid clutter, we don't generally draw the partition for superclass Object

See 1/2-page section 4.3.1 on page 154.

4

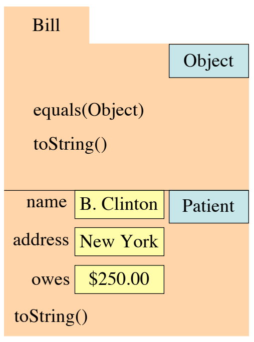
Method toString()

Convention: c.toString() returns a representation of folder c, giving information on the values in its fields.

Put following method in Patient.

```
public String toString() {
   return name + "  " + address +
         "  " + owes;
}
```

In appropriate places, the expression c automatically does c.toString()



5

Example of toString in another class

```
/** An instance represents a point (x, y) in the plane */
public class Point {
   private int x;  // the x-coordinate
   private int y; // the y-coordinate
   /** Constructor: An instance for point(xx, yy) */
   public Point(int xx, int yy) {
      x= xx;   y= yy;
   }
   /** = a representation of this point */
   public String toString() {
      return "(" + x + ", " + y + ")";
   }
}
```

Getter and setter methods are not given on this slide

Function toString should give the values in the fields in a format that makes sense for the class.

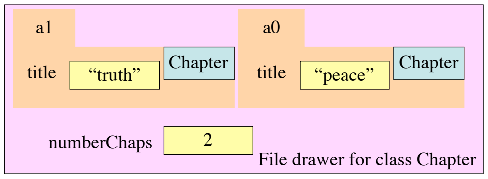
6

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 static variable using **Chapter.numberChaps**

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;
    }
}
```

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.

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.

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) {
   …
}
```

```
/** 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() {
    }
}
```

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.

Other methods listed on page 488.

testMethods to test getters and setters

```
/** 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());
}
```

one test case

```
/** 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());
}
```

Every time you click button Test in DrJava, all methods with a name testXXX will be called.