CS100J 05 February 2005

Today's topic: **Customizing a class (continued)**

Quiz 1 is today

Quiz 2 is next Tuesday

Quote for the day:
There is no reason anyone would want a computer in their home. --Ken Olson, president, chairman and founder of Digital Equipment Corp., 1977.
The company was a huge player in computer hardware and software in CS academia in the 1970's. The old 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. The company was bought by COMPAQ in the late 1990's.

CS100J, 03 February 2005

Reading for this lecture: Section 1.4, 1.5, and 1.7 (not 1.6).

Read all the "style notes", too.

Summary of lectures: On course home page, click on "Handouts" and then "Outline of lectures held so far".

Today: Class Object, method toString()
Fields (variables in a folder).
Constructors.
Static components.

CS100J, 03 February 2005

Reading for this lecture: Section 1.4, 1.5, and 1.7 (not 1.6).

Read all the "style notes", too.

Summary of lectures: On course home page, click on "Handouts" and then "Outline of lectures held so far".

Today: Fields (variables in a folder).
Constructors.
Static components.

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

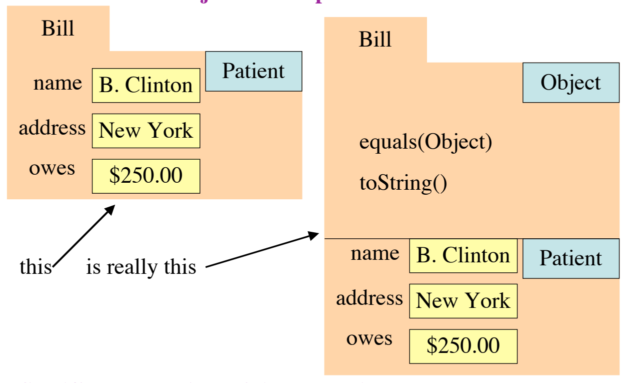
public class C **extends** Object { …}

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 A1.

Class Object: The superest class of them all



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

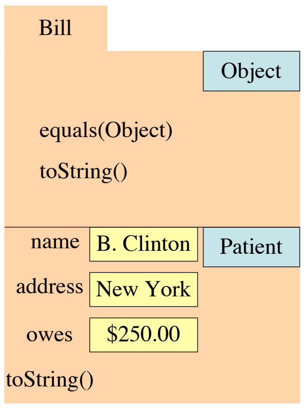
Method toString()

Convention: c.toString() prints a representation of folder c.

Put following method in Patient.

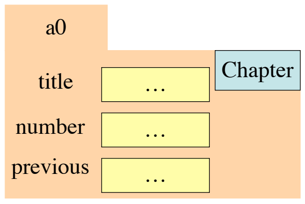
```
public String tostring() {
   return name + " " + address +
         " " + owes;
}
```

The expression c
automatically does c.toString()



Field: a variable that is in each folder

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)
}
```

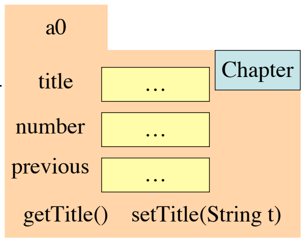
Getter and setter methods

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

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

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

}
```



Getter methods get or retrieve values from a folder.

Setter methods set or change fields of a folder

We need a way to initialize fields when a folder is first created

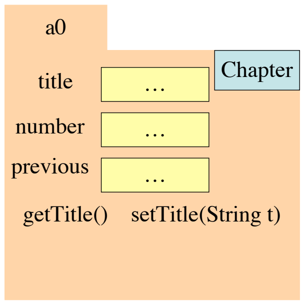
new Chapter()

creates a folder but doesn't allow us to say what values should be in it.

We would like to be able to say:

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**.

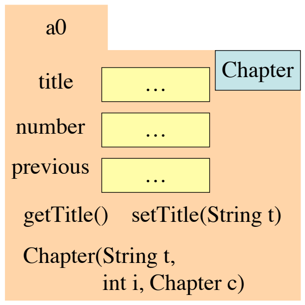


For this, we use a new kind of method, the **constructor**.

The purpose of a constructor is to initialize (some) fields of a newly created folder

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

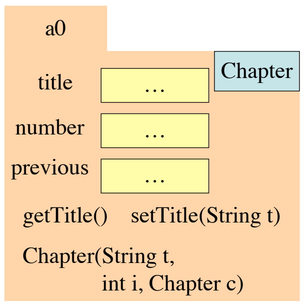


The name of a constructor is the name of the class.

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

New description of execution 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 (0 for **int**, for example).
2. Put the folder in file-drawer Chapter.
3. Execute the constructor call

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

4. Use the name of the new folder as the value of the new-expression.

Memorize this new definition! Today! Now!

You can have more than one constructor

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

/** Constructor: an instance with title t,
                  chapter number i, and previous chapter null */
public Chapter(String t, int i) {
      title= t;
      number= i;
      previous= null;
}
```

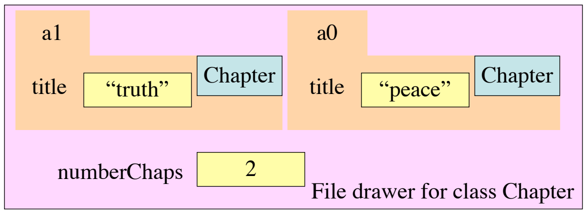
Makes it easier, more flexible, for the "user" who is using the class

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; // Number of chapter
   private static int numberOfChapters= 0;
}
```

Reference the static variable using

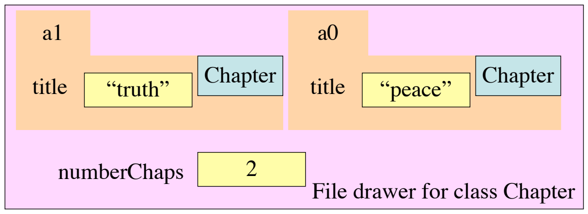
Chapter.numberChaps



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; // Number of chapter
   private static int numberOfChapters= 0;
}
```

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



a1
title
"truth"
Chapter
a0
title
"peace"
Chapter
numberChaps
2
File drawer for class Chapter