

CS 61B: Lecture 2
Wednesday, August 30, 2006

Today's reading: Sierra & Bates, Chapter 2; pp. 54-58, 154-160, 661, 669.

OBJECTS AND METHODS

```
=====
String s1;           // Step 1: declare a String variable.
s1 = new String();   // Step 2: assign it a value: a new empty string.
String s2 = new String(); // Steps 1 & 2 combined.
```

```

      ---      ---      ---
At this point, s1 and s2 are both  s1 |.+---->| "" |  s2 |.+---->| "" |
variables that reference empty strings.  ---      ---      ---
```

```
s1 = "Yow!";           // Construct a new String; make s1 a reference to it.
s2 = s1;               // Assign s2 the value of s1.
```

```

      ---      ---      ---
s1 |.+---->| Yow! |<----+.| s2
      ---      ---      ---
```

Now s1 and s2 reference the same object. What if we'd prefer to have a copy of the object?

```
s2 = new String(s1);   // Construct a copy of s1; make s2 a reference to it.
```

```

      ---      ---      ---
s1 |.+---->| Yow! |  s2 |.+---->| Yow! |
      ---      ---      ---
```

Now they refer to two different, but identical, objects.

Think about that. When Java executes that line, it does the following things, in the following order.

- Java looks inside the variable s1 to see where it's pointing.
- Java follows the pointer to the String object.
- Java reads at the characters stored in that String object.
- Java creates a new String object that stores a copy of those characters.
- Java makes s2 reference the new String object.

We've seen three String constructors:

- (1) new String() constructs an empty string--it's a string, but it contains no characters.
- (2) "cs 4" constructs a string containing the characters cs 4.
- (3) new String(s1) takes a parameter s1. Then it makes a copy of the object that s1 references.

Constructors always have the same name as their class, except the special constructor "stuffinquotes". That's the only exception.

Observe that "new String()" can take no parameters, or one parameter. These are two different constructors--one that is called by "new String()", and one that is called by "new String(s1)". (Actually, there are many more than two--check out the online Java API to see all the possibilities.)

Let's look at some methods that aren't constructors.

```
s2 = s1.toUpperCase();           // Create a string like s1, but in all upper case.
String s3 = s2.concat("!!!");    // Also written: s3 = s2 + "!!!";
String s4 = "".concat(s2).concat("***"); // Also written: s4 = "" + s1 + "***";
```

```

      ---      ---      ---      ---
s2 |.+---->| YOW! |  s3 |.+---->| YOW!!! |  s4 |.+---->| *YOW!* |
      ---      ---      ---      ---
```

Now, here's an important fact: when Java executed the line

```
s2 = s1.toUpperCase();
```

the String object "Yow!" did not change. Instead, s2 itself changed to reference a new object. Java wrote a new "pointer" into the variable s2, so now s2 points to a different object than it did before.

Unlike in C, in Java Strings are immutable--once they've been constructed, their contents never change. If you want to change a String object, you've got to create a brand new String object that reflects the changes you want. This is not true of all objects; most Java objects let you change their contents.

I/O Classes and Objects in Java

Here are some objects in the System class for interacting with a user:

```
System.out is a PrintStream object that outputs to the screen.
System.in is an InputStream object that reads from the keyboard.
[Reminder: this is shorthand for "System.in is a variable that references
an InputStream object."]
```

But System.in doesn't have methods to read a line directly. There is a method called readLine that does, but it is defined on BufferedReader objects.

- How do we construct a BufferedReader? One way is with an InputStreamReader.
- How do we construct an InputStreamReader? We need an InputStream.
- How do we construct an InputStream? System.in is one.

(You can figure all of this out by looking at the constructors in the online Java libraries API--specifically, in the java.io library.)

Why all this fuss?

InputStream objects (like System.in) read raw data from some source (like the keyboard), but don't format the data.

InputStreamReader objects compose the raw data into characters (which are typically two bytes long in Java).

BufferedReader objects compose the characters into entire lines of text.

Why are these tasks divided among three different classes? So that any one task can be reimplemented (say, for improved efficiency) without changing the other two.

Here's a complete Java program that reads a line from the keyboard and prints it on the screen.

```
import java.io.*;

class SimpleIO {
    public static void main(String[] arg) throws Exception {
        BufferedReader keybd =
            new BufferedReader(new InputStreamReader(System.in));
        System.out.println(keybd.readLine());
    }
}
```

Don't worry if you don't understand the first three lines; we'll learn the underlying ideas eventually. The first line is present because to use the Java libraries, other than java.lang, you need to "import" them. java.io includes the InputStreamReader and BufferedReader classes.

The third line is present because any Java program always begins execution at a method named "main", which is usually defined more or less as above. When you write a Java program, just copy the line of code, and plan to understand it a few weeks from now.

Classes for Web Access

Let's say we want to read a line of text from the White House Web page. (The line will be HTML, which looks ugly. You don't need to understand HTML.)

How to read a line of text? With readLine on BufferedReader.
How to create a BufferedReader? With an InputStreamReader.
How to create an InputStreamReader? With an InputStream.
How to create an InputStream? With a URL.

```
import java.net.*;
import java.io.*;

class WHWWW {
    public static void main(String[] arg) throws Exception {
        URL u = new URL("http://www.whitehouse.gov/");
        InputStream ins = u.openStream();
        InputStreamReader isr = new InputStreamReader(ins);
        BufferedReader whiteHouse = new BufferedReader(isr);
        System.out.println(whiteHouse.readLine());
    }
}
```