CS61B Summer 2006
Instructor: Erin Korber
Lecture 4, 29 June
Reading for tomorrow: Ch. 10, pp. 273-293, 307-309

1 I/O (based on notes from JRS)

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());
  }
}
```

1.1 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 a 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());
  }
}
```

2 Encapsulation

- Purpose: to allow the programmer to maintain control of the data.
- Until now, have been just using the dot operator to access instance variables. (e.g. `theDog.name = ‘‘Fido’’;`). But sometimes, we write methods that expect instance variables to have certain kinds of values (for example, we often want integers to be positive). Encapsulation allows us to enforce these restrictions.
- The idea: instead of getting and setting instance variables with the dot operator, have *getter* and *setter* methods for access.
- How to force users to use these methods? *Access modifiers*

3 Access levels

- Java has 4 *access levels* and 3 *access modifiers*

  - can be applied to classes, constructors, instance variables, and methods

- `public` means anyone can access it
  - generally use this for classes, most constructors, and methods you want exposed to other code (like getters and setters)

- `private` means only code within the same class can access it
  - use this for instance variables, and for methods you want to hide

- default (what you get when you don't use a modifier) means only code in the same *package* can access it

- `protected` is like default, except that subclasses outside the package can *inherit* the protected thing (will learn what this means next week).

4 Packages

- A package is a collection of classes that "trust" each other.
- Also helps with name collisions.
  - Can type the full name of a class (e.g. `java.util.ArrayList`) every time you use it.
  - Or can use an import statement.
- Classes, methods, and variables with default protection are visible within a package.

5 Inner classes

- Defined *inside* another class
- Has access to even the private methods and variables of the outer class.
- An inner object is tied to a specific outer object.
  1. Make an instance of the outer class
  2. Use it to make an instance of the inner class.
  3. The two are now linked.