CS 4: Lecture 13
Monday, March 6, 2006

METHOD OVERLOADING

The _signature_ of a method consists of its name and a description of its parameters. Here's an example of a signature.

```
distance(Planet p)
```

The _prototype_ of a method consists of its signature along with its return value (plus "public" or "private", if present). Here's an example of a prototype.

```
private double distance(Planet p)
```

Two signatures are considered to be "a match" if they have the same method name and take the same number of parameters of the same types in the same order--even if the parameter names are different. For example, the following two signatures "match".

```
product(int x, double y)
product(int factor1, double factor2)
```

In Java, you can declare several methods that all have the same name--so long as they have different signatures. This is called _method_overloading_. For example, you could declare all of the following methods in one program.

```
void doIt(int x) { ... }
void doIt(String x) { ... }
int doIt(int x, double y) { ... }
void doIt(double x, int y) { ... }
```

Each of these four methods could do completely different things. Usually, though, you'll have them do very similar things, with minor variations. For example, they might all write the parameters to a disk file. If they really do completely different things, you should give them different names.

When you call a method, Java decides which method you want to call based not only on the method name, but also on the types of the parameters.

```
doIt("abc");   // Calls the second doIt.
doIt(3.0, 7);  // Calls the fourth doIt.
```

The principle is (remember this):

```
*** Java looks for a method whose signature   ***
*** matches the signature of the method call. ***
```

You've already used overloaded methods. For example, PrintStream objects (like System.out) have ten different methods called println. Here are some of their prototypes.

```
void println()
void println(double x)
void println(int x)
void println(String x)
```

When you write "System.out.println(3.0);", you are calling a different method than when you write "System.out.println("Hello");", because it takes different code to print a floating-point number than to print a String.

SOFTWARE ENGINEERING CONCEPTS

A _software_library_ is a set of classes and methods that somebody has written and made available for programs to build on. For example, Java's Math class includes methods for computing sines, cosines, and square roots. If you need to compute the square root of a number, you could write a method to do it yourself (if you know an algorithm for computing a square root). But usually it would be a waste of your time, because somebody has already written one, and you can just use it.

The _interface_ of a software library is a set of public classes, public fields, and prototypes for public methods, plus descriptions of the methods' behaviors.

For example, there's a Web page that gives the interface for Java's Math class. The interface includes two public fields and 32 public methods. It's worth your while to know about them, as they can be really handy in writing engineering programs.

The two public fields are both constants.

```
public static final double E                // base of the natural logarithm
public static final double PI               // pi
```

Here are prototypes for some of the public methods.

```
static double atan(double a)
// Returns the arc tangent of a, in the range -pi/2 through pi/2.

static double exp(double a)
// Returns e raised to the power of a.

static double log(double a)
// Returns the natural logarithm (base e) of a.

static double pow(double a, double b)
// Returns the value of a raised to the power of b.
```

There are 28 more of these. You can see them all at
http://java.sun.com/j2se/1.4.2/docs/api/java/lang/Math.html .

Each prototype and field includes a comment that describes its behavior.
The prototypes, comments, and fields are the _interface_ for the Math class.

Note that the code that implements the methods is _not_ part of the interface. Even the values of e and pi are not part of the interface. Why? Because the person who wrote the library reserves the right to change the implementation at any time in the future. This is called _information_hiding_.

When somebody provides a library for you to use, they usually make a deal with you, with two terms.

- They won't change the interface in future versions of the library. Well, they might add new features, but they won't remove the old ones or change what the methods do.
- In return, you promise to access the library only through the interface. "Information hiding" implies that you won't try to change fields or call methods that are not part of the official interface, which is why they are usually declared "private".

If both of you keep up your ends of the bargain, the person who wrote the library will be able to update their library to a new, faster, better version, and you'll be able to plug their new library into your code and have it work just like it did with the old library (except hopefully faster and with fewer bugs).

For example, Sun updates the Java libraries every few years, adding new features and improving old ones. But Java programs I wrote years ago still work with the new libraries.

An _application_ is a program that's used by a "user" to accomplish some task, like word processing or compiling Java programs or making a bank ATM work. Eclipse is an application, and so is your planetary motion simulation and your favorite video game. The difference between a library and an application is that a library is used to build applications (or other libraries), whereas an application is an end in itself.

An _application_program_interface_ (API) is a set of libraries and interfaces for building applications. A good API makes it easier to develop a program by providing all the building blocks, often including user interface libraries. A programmer puts the blocks together.

THE JAVA API

Java's libraries are grouped together into the "Java API". The Java API includes the Math class, the graphical user interface classes that have been a part of some of your labs, I/O (input and output) classes, and tons of other stuff.

The best way to learn about the API is to follow the "Java 1.4.2 standard libraries API" link from the CS 4 Web page, then select some classes that you've heard of. The most useful libraries to investigate are "java.lang", "java.math", and "java.io". By looking at the APIs for classes like String and PrintStream, you can glean information about the methods you can call on them. For example, here are some methods on Strings.

```
int length()
// Returns the length of a String.

String substring(int beginIndex, int endIndex)
// Returns a substring of this string, starting at index 'beginIndex' and
// ending _before_ index 'endIndex'.  Characters are indexed from zero.
```

Let's use them.

```
String s = "Lederhosen";
int c = s.length();                   // 10
String t = s.substring(5, 9);         // "hose"
```

THE "char" TYPE

Java has a primitive type called "char", which stores a character of text. Characters can be specified directly inside single right quotes (i.e. apostrophes).

```
char c = 'x';
```

Strings and characters are related, and there are methods that convert between them.

```
String s = "abcde";
char c = s.charAt(0);                 // 'a'
```

There is a String constructor that lets you create a String by putting together the chars in an array.

```
char word[] = { 'j', 'a', 'v', 'a'};
String s = new String(word);          // Constructs "java".
```