CS 4: Lecture 5
Wednesday, February 1, 2006

Why Write Methods?

(1) Suppose you have a complicated operation you want to do in many different places in your code, like computing the length of a vector. You can write one method to compute the length of a vector, and use it over and over again. This keeps your code much cleaner and easier to read than if you cut-and-paste the code over and over again.

```
lengthp = position.length();
lengthv = velocity.length();
lengtha = acceleration.length();
```

(2) Suppose you need to change the way you compute the length of a vector--for example, you were doing it wrong and need to correct your code. You only need to change your code in one place, instead of a hundred places.

(3) Very long programs are hard to read. You can make them much easier by dividing your code into big operations, each comprising many lines of code, and moving them into separate methods with meaningful names.

```
public void static main() {
  Data data = getData();                  // Was 500 lines of code.
  double energy = data.computeEnergy();   // Was 700 lines of code.
  data.outputResult(energy);              // Was 600 lines of code.
}
```

Then you can break "getData" and "computeEnergy" and "outputResult" into smaller methods, and those into smaller methods, and so on. This is how really large programs get written: by breaking them down into coherent pieces that can each be understood independently.

(4) By associating methods (code) with the objects they manipulate, you organize your code in a logical, more readable manner. You'll understand this better when you've had more programming experience.

Pass by Value

When you call a method and pass parameters to it, the parameters are passed _by_value_. This means that the method has a _copy_ of the actual parameter, and cannot change the original.

Let's look at an example.

```
public static void main(String[] args) {
  long c = 1;
  doNothing(c);
  System.out.println(c);
}

static void doNothing(long x) {
  x = 2;
}
```

When you run this program, Java begins execution with the main method, which declares a variable "c" and sets it to one:

```
  -----
c | 1 |
  -----
```

When "main" calls "doNothing(c)", the doNothing method declares a _new_ variable "x" to hold the value of the parameter, and _copies_ the value of c into x. That's what in means to pass parameters by value.

```
  -----       -----
c | 1 |     x | 1 |
  -----       -----
```

Next, the doNothing method assigns x a value of 2.

```
  -----       -----
c | 1 |     x | 2 |
  -----       -----
```

The important thing to notice is that the value of c is unchanged. In fact, there is _no_ way the doNothing method can change the value of c.

Next, the doNothing method ends. When a method ends, all the parameters and local variables declared inside that method disappear forever.

```
  -----
c | 1 |
  -----
```

To make it a little more confusing...we could have called doNothing's parameter "c".

```
static void doNothing(long c) {
  c = 2;
}
```

But things would have turned out exactly the same way. Confusingly, we would have two different variables named c.

```
  -----       -----
c | 1 |     c | 2 |
  -----       -----
```

But the left "c" can only be accessed from inside "main", and the right "c" can only be accessed from inside "doNothing". "Pass by value" still works the same way: main's c gets copied into doNothing's c, and then none of the changes doNothing makes to its c have any effect on main's c.

Constructors

A constructor is a method that constructs an object. Let's write a constructor that constructs a Quantity. The constructor won't actually contain code that does the creating; rather, Java provides a brand new object for us right at the beginning of the constructor, and all you have to write in the constructor is code to initialize the new object (or not even that).

```
class Quantity {
  // Include all the stuff from Lecture 4 here.

  public Quantity(String myUnit) {
    amount = 1.0;
    unit = myUnit;
  }
}
```

Notice that the constructor is named "Quantity", and it returns an object of class "Quantity". This constructor is called when we write "new Quantity(s)", where s is a String object. Now, we can shorten initializing a frequency.

```
  Quantity frequency = new Quantity("Hertz");
  frequency.contents();
```

The output is:

```
  I represent 1.0 Hertz.
```

In Lecture 4, we constructed a Quantity object without writing a constructor first. How did we do that? Java provides every class with a default constructor, which takes no parameters and does no initializing. Hence, when we wrote

```
  new Quantity()
```

we created a new, blank Quantity. If the default constructor were defined explicitly, it would look like this:

```
  public Quantity() {
  }
```

You can override the default constructor by explicitly writing your own constructor with no parameters.

```
  public Quantity() {
    amount = 0.0;
    unit = "thingies";
  }
```

Warning: if you write your own Quantity constructor, even if it takes parameters, the default constructor goes away. If you want to have the default Quantity constructor (with no parameters) _and_ another Quantity constructor (with parameters), you must define both explicitly.