Introduction to methods

In this lecture we give an introduction to methods – just enough so you can use the CodingBat site (http://codingbat.com). CodingBat is a website set up by a Stanford professor for students to run through programming drills, and you have to understand a little bit about methods before using the site. To fully use methods in your own programs we’ll have to cover some more details about classes. This will come in a later lecture.

A method is essentially a group of code that performs some task. It is sort of like a function in math. Some information goes in (variables), some computation is done on those variables, and then some result is normally sent back. For example, my calculator has a $y^x$ button. To use this function, I don’t have to know how it works. I only need to know what goes in and what comes out:

To someone that is using the $y^x$ button, the inner workings are like a black box. We can do the same thing for arbitrary code and it is called a method.

We have been using methods already. For example, when we used the Scanner class to read in an input value, such as

```java
i = keyboard.nextInt();
```

We are invoking the `nextInt` method that someone else already wrote for us. Any input parameters are provided inside the parentheses. In this case, there are no input values.
However, there is a return value, which is the integer typed in from the keyboard. Once again, we don’t have to worry about how the nextInt method works internally. All we need to know is what we are supposed to send in and what the method returns back. By putting code into a method it also increases its reusability and makes code easier to understand.

To write our own methods we use the following format. This would go inside the definition of a class:

```
Modifier  Return_type  methodName(type1  varName1,  type2  varName2, … )
{
    Instructions
    return (return_value);
}
```

- **Modifier** is either `public` or `private` to indicate if this method is available from outside the class. For now we will always use public.

- **Return_type** is a data type returned by the method. For example, `int`, `float`, `long`, another `Class`, or `void` if we have no data to return. If we are returning a value, we must specify what value the method returns with the return statement. Usually this is at the end of the method, but it could be anywhere inside the method. The data type of what is returned must match the data type specified by **Return_type**.

- **methodName** is an identifier selected by the user as the name for the method.

- The list of **parameters** are input variables that are passed to the method from the caller. This gives data for the method to operate on. To define these parameters, specify the type of the variable followed by the **variable name**. Multiple parameters are separated by commas.

For example, here is a method called `multiplyByTwo` that multiples an input integer by two:

```
public int multiplyByTwo(int num)
{
    return (num * 2);
}
```
Here is a method called isEven that returns true if the number is even and false if it is not:

```java
public boolean isEven(int num) {
    if (((num % 2)) == 0)
        return true;
    else
        return false;
}
```

If you remember the tax example of the previous lecture, we could embed this into a method and not have to worry about the details when we go to compute the taxes:

```java
public double computeTaxes(int income) {
    double tax = 0;

    if (income <= 15000) {
        tax = 0;
    } else if (income <= 25000) {
        tax = 0.05 * (income - 15000);
    } else {
        tax = 0.05 * (income - (25000 - 15000));
        tax += 0.10 * (income - 25000);
    }

    return tax;
}
```
Here is an example using three arguments. It finds the minimum of three doubles.

```java
public double findSmallest(double num1, double num2, double num3)
{
    double smallest = num1;  // Assume num1 is smallest
    if (num2 < smallest)
        smallest = num2;     // Smallest of num1 and num2
    if (num3 < smallest)
        smallest = num3;     // Smallest of num1, num2, and num3
    return smallest;
}
```

At this point we won’t be invoking methods ourselves. But if we were it would look something like this for the findSmallest method:

```java
double smallestNum;
smallestNum = findSmallest(10.5, 3.21, 78.9);
```

The order that we list values in parenthesis corresponds to the values sent in as variables. In this case, num1 gets set to 10.5, num2 gets set to 3.21, and num3 gets set to 78.9. The variable smallest gets set to 3.21 and returned. The result is that 3.21 is stored in the variable smallestNum.

Although we are getting a bit ahead of ourselves, the reason for discussing methods now is to use CodingBat. The website is set up with short programming exercises organized around a method. The website will give you a description of a problem and the structure of the method. You are left to fill in the rest. The website will test your submission by compiling it and testing your method with different input values then test the return result to see if it is correct.

Here is an example from CodingBat:

**Java > Warmup-1 > sumDouble**

Given two int values, return their sum. Unless the two values are the same, then return double their sum.

```java
sumDouble(1, 2) → 3
sumDouble(3, 2) → 5
sumDouble(2, 2) → 8
```

Here is an example from CodingBat:
You would type in code to implement the `sumDouble` method. This means you would be typing all the code that goes inside the curly braces but have to know what the input parameters are and how to return the result.

The nice thing about CodingBat is it will tell you if your solution is correct because it will test your function. Let's say that this is our attempt at implementing `sumDouble`:

```java
public int sumDouble(int a, int b) {
    int sumD = 0;
    if (a == b)
        sumD = 2*a+b;
    else
        sumD = a+b;
    return sumD;
}
```

When we try to run it, CodingBat tells us if it passes all its tests:

<table>
<thead>
<tr>
<th>Expected</th>
<th>This Run</th>
<th>Passed</th>
</tr>
</thead>
<tbody>
<tr>
<td>sumDouble(1, 2) → 3</td>
<td>3</td>
<td>OK</td>
</tr>
<tr>
<td>sumDouble(3, 2) → 5</td>
<td>5</td>
<td>OK</td>
</tr>
<tr>
<td>sumDouble(2, 2) → 8</td>
<td>6</td>
<td>X</td>
</tr>
<tr>
<td>sumDouble(-1, 0) → -1</td>
<td>-1</td>
<td>OK</td>
</tr>
<tr>
<td>sumDouble(0, 0) → 0</td>
<td>0</td>
<td>OK</td>
</tr>
<tr>
<td>sumDouble(0, 1) → 1</td>
<td>1</td>
<td>OK</td>
</tr>
</tbody>
</table>

Correct for more than half the tests

This tells us it is not working when the numbers are the same. A review of our code indicates a precedence error with arithmetic:

```java
if (a == b)
    sumD = 2*(a+b);
```

Running the program now gives us all green lights:
The CodingBat website is a nice place to go if you want to practice on small programming problems. I will have you complete some exercises for the in-class programming drills. I recommend you try other exercises on your own, independently, if you feel that you need more practice. Unfortunately it doesn’t work for other coding constructs, like creating classes, but we can use it for these topics:

- If-else statements, logic
- Loops
- Recursion
- Arrays

For the in-class drills I will need to know if you have completed a CodingBat problem. To do this, you must create an account. Then under “Prefs” type in kenrick@uaa.alaska.edu under the “Teacher Share” box. This will let me see what problems you have completed and when. You don’t need to have me come check you off in class because at the end of the day I can log in and see who has done what. Make sure you use your real name in creating your account so I can tell who you are.