Method Basics

Recall: methods are just collections of instructions with a name attached.

1. *identifier*: the method's name
2. *parameters*: zero or more variables the method gets *passed* by the *calling* method
3. *return type*: the type of result the method outputs

`static` methods

We've been calling external `static` methods with the form

```
<Class>.<method>(<params>);
```

When they're in your same class, you can take a shortcut and just call

```
<method>(<params>);
```

Note: there is **no** similar shortcut for not-`static` methods!

Writing `static` methods

Recall, a `static` method looks like this:

```
public static <return type> <id> (<params>) {
    <statements>
    return <return type>;
}
```

and it's good coding practice to only have one exit point from your methods (it also makes them less confusing).

Being careful about parameters

We're using the word "parameters" to mean two different things:

```
public static <ret> <name>(<params>)
```

and

```
<Class>.<method>(<params>)
```

When we write the method header, this gives us our parameter *type* and *name* (like when we *declare* a variable), and when we call the method, we give those parameters *values* (like when we *initialize* a variable).

Passing Parameters

Say we have a `static` method `average` which averages two `ints` and returns a `double`. Here's what happens:

```
double result = average(5,10);

  /* we jump into the method code */

double avg = (a+b)/2.0;
return avg;

  /* we jump out of the method code */

double result = <7.5>;
```

Using return values

If I have a method like this:

```
public static double average(int a, int b) {
    return (a+b)/2.0;
}
```

and call it like this:

```
average(5,10);
```

NOTHING HAPPENS. Treat method calls the same way you would literals. You wouldn't write a line that just said

```
7.5;
```

would you?

When methods return nothing

Maybe I want to implement a method that just, say, prints a line of a certain length and doesn't return any result at all. I still need to specify a return type in my method header, though - so I use `void`:

```
public static void writeLine() {...}
```

So what would be wrong with

```
System.out.println(writeLine()); // ?
```

Avoiding errors

Be very careful about modifying your parameters, regardless of your return type.

```
int[] a = {1,2,3,4,5};
test(a);

    /* jump into method code */

nums[0] += 5;

    /* jump out of method code */

System.out.println(a[0]); // what happens??
```

Avoiding errors

Compare that behavior to:

```
int a = 1;
test(a);

    /* jump into method code */

num += 5;

    /* jump out of method code */

System.out.println(a); // what happens??
```

Why is this behavior different?