Introduction to Object-Oriented Programming

Programs and Methods

Christopher Simpkins
chris.simpkins@gatech.edu

The Anatomy of a Java Program

It is customary for a programmer's first program in a new language to be "Hello, World." Here's our `HelloWorld.java` program:

```
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello, world!");
    }
}
```

- The first line declares our `HelloWorld` class. `class` is the syntax for declaring a class, and prepending with the `public` modifer means the class will be visible outside `HelloWorld`'s package.
- Because we didn't declare a package explicitly, `HelloWorld` is in the *default* package. More on packages in a future lectrue.
- The code between the curly braces, `{ ... }` define the contents of the `HelloWorld` class, in this case a single method, `main`

`public static void main(String[] args)`

In order to make a class executable with the `java` command, it must have a main method:

```
public static void main(String[] args) { ... }
```

- The `public` modifier means we can call this method from outside the class.
- The `static` modifer means the method can be called without instantiating an object of the class. Static methods (and variables) are sometimes called *class* methods.
- `void` is the return type. In particular, main returns nothing. Sometimes such subprograms are called *procedures* and distinguished from *functions*, which return values.
- After the method name, `main`, comes the parameter list. `main` takes a single parameter of type `String[]` - an array of `String`s. `args` is the name of the parameter, which we can refer to within the body of `main`

Methods

The `main` method is a special method that is used as the entry point for a Java program. We can define other methods as well. Consider this method from NameParser.java:

```
public static String extractLastName(String name) {
    int commaPos = name.indexOf(",");
    int len = name.length();
    String lastName = name.substring(0, commaPos).trim();
    return lastName;
}
```

Similar to our `main` method but:

- `return`s a `String` value
- takes a single parameter of type `String`

Method Parameters

In this method:

```
public static String extractLastName(String name) {
    int commaPos = name.indexOf(",");
    int len = name.length();
    String lastName = name.substring(0, commaPos).trim();
    return lastName;
}
```

`name` is a *parameter* (or *formal parameter*), a local scope variable within the `extractLastName` method. It is bound to a value when the method is called. In the statement:

```
String lastName = extractLastName(fullName);
```

the right-hand side, `extractLastName(fullName)`, is a *method invocation* (or *method call*). We say that `fullName` is the *argument* (or *actual parameter*) to this invocation of the `extractLastName` method.

Local Variables

Method parameters and variables declared inside the method are local to the method, invisible outside the method. Local variables "shadow" variables of the same name in an enclosing scope.

```
public class Methods {

    // Note that static variables cannot appear inside methods
    static String message = "Global message.";
    static int a = 5;

    public static int add(int a, int b) {
        String message = "Adding " + a + " and " + b;
        System.out.println(message);
        return a + b;
    }
...
```

In the `add` method, the parameter `a` shadows the `static` variable `a`, and the local variable `message` shadows the `static` variable `message`.

Methods as Expressions

Methods that return values are expressions which can be used anywhere a value of the method's return type can be used. Given:

```
public static int add(int a, int b) { ... }
```

which returns an `int`, this:

```
x + (x + y)
```

is equivalent to this:

```
x + add(x, y)
```

See Methods.java.

Closing Thoughts

Methods are subprograms with

- input (parameters),
- processing (a sequence of statements), and
- output (return value).

Methods are a powerful form of procedural abstraction, another step in the building of complex programs from simple parts.