The Java Tutorials have been written for JDK 8. Examples and practices described in this page don't take advantage of improvements introduced in later releases and might use technology no longer available.
See Java Language Changes for a summary of updated language features in Java SE 9 and subsequent releases.
See JDK Release Notes for information about new features, enhancements, and removed or deprecated options for all JDK releases.

Defining Methods

Here is an example of a typical method declaration:

```
public double calculateAnswer(double wingSpan, int numberOfEngines,
                              double length, double grossTons) {
    //do the calculation here
}
```

The only required elements of a method declaration are the method's return type, name, a pair of parentheses, `()`, and a body between braces, `{}`.

More generally, method declarations have six components, in order:

1. Modifiers—such as `public`, `private`, and others you will learn about later.
2. The return type—the data type of the value returned by the method, or `void` if the method does not return a value.
3. The method name—the rules for field names apply to method names as well, but the convention is a little different.
4. The parameter list in parenthesis—a comma-delimited list of input parameters, preceded by their data types, enclosed by parentheses, `()`. If there are no parameters, you must use empty parentheses.
5. An exception list—to be discussed later.
6. The method body, enclosed between braces—the method's code, including the declaration of local variables, goes here.

Modifiers, return types, and parameters will be discussed later in this lesson. Exceptions are discussed in a later lesson.

Definition: Two of the components of a method declaration comprise the *method signature*—the method's name and the parameter types.

The signature of the method declared above is:

```
calculateAnswer(double, int, double, double)
```

Naming a Method

Although a method name can be any legal identifier, code conventions restrict method names. By convention, method names should be a verb in lowercase or a multi-word name that begins with a verb in lowercase, followed by adjectives, nouns, etc. In multi-word names, the first letter of each of the second and following words should be capitalized. Here are some examples:

```
run
runFast
getBackground
getFinalData
compareTo
setX
isEmpty
```

Typically, a method has a unique name within its class. However, a method might have the same name as other methods due to *method overloading*.

Overloading Methods

The Java programming language supports *overloading* methods, and Java can distinguish between methods with different *method signatures*. This means that methods within a class can have the same name if they have different parameter lists (there are some qualifications to this that will be discussed in the lesson titled "Interfaces and Inheritance").

Suppose that you have a class that can use calligraphy to draw various types of data (strings, integers, and so on) and that contains a method for drawing each data type. It is cumbersome to use a new name for each method—for example, `drawString`, `drawInteger`, `drawFloat`, and so on. In the Java programming language, you can use the same name for all the drawing methods but pass a different argument list to each method. Thus, the data drawing class might declare four methods named `draw`, each of which has a different parameter list.

```
public class DataArtist {
    ...
    public void draw(String s) {
        ...
    }
    public void draw(int i) {
        ...
    }
    public void draw(double f) {
        ...
    }
    public void draw(int i, double f) {
        ...
    }
}
```

Overloaded methods are differentiated by the number and the type of the arguments passed into the method. In the code sample, `draw(String s)` and `draw(int i)` are distinct and unique methods because they require different argument types.

You cannot declare more than one method with the same name and the same number and type of arguments, because the compiler cannot tell them apart.

The compiler does not consider return type when differentiating methods, so you cannot declare two methods with the same signature even if they have a different return type.

Note: Overloaded methods should be used sparingly, as they can make code much less readable.
