Day 0: Hello, World.

by AllisonP

Terms you'll find helpful in completing today's challenge are outlined below, along with sample Java code (where appropriate).

Class

At its most basic level, a *class* is a collection of *variables* (*fields*) and functions called *methods*. A program is a collection of classes. The basic code for declaring a Java class is as follows:

```
class MyClass{
    // This is a single-line comment.

    /*  This is also a comment.
        This type of comment can span several lines
    */
}
```

When declaring a class, the name should always start with a *capital* letter; this signifies to certain compilers (and human readers of your code) that it is a class (or other similarly-behaved structure that you'll learn about later). If you wish to use a compound phrase (e.g.: "my class") as your class name, you should write it in CamelCase; this means you should capitalize each word and remove spaces between words (e.g.: "MyClass").

Note: Class names cannot *begin* with numbers or contain any spaces.

Variable

Think of this as a name (identifier) that points to (references) a location in memory where information associated with that name can be stored. In Java (and many other languages), it is a best practice to always start variable names with a *lowercase* letter and use CamelCase for variable names composed from compound phrases. Variable names cannot contain spaces or special characters (except underscores), though they can contain (but *not* begin with) numbers. A variable that is a member of a class is called a *field*.

Each variable has a *data type* associated with it, which essentially restricts what that variable is allowed to reference. This means your code will not work if you attempt to perform operations on your variables that aren't allowed for that data type. To *declare* a variable named $myVariable$ having the data type *DataType*, we write the following:

```
DataType myVariable;
```

If we want to declare a variable of type *DataType* named $myVar1$ and *initialize* it to be $value$ (here, we are using the word $value$ as a stand-in for a valid value of type *DataType*), we write:

```
DataType myVar1 = value;
```

In English, the above code is basically saying: "I'm creating a variable named $myVar1$; it refers to something of type *DataType*, and is assigned an initial value of $value$."

Note: The `=` operator is called the *assignment operator*, so you should interpret `=` as the English phrase "[left operand] is assigned the value of [right operand]".

A *String* is a data type that holds a sequence of characters. To create a *String* variable named $myString$ that stores the value "Hi!", write the following line of code:

```
String myString = "Hi!";
```

The compiler will interpret the characters between the two quotation marks as a *String*. Saving a reference to our it as variable $myString$ allows us to refer to it again and again by referencing our variable name, $myString$.

Note: Some coders use lowercase letters in conjunction with underscores to simulate spaces when declaring variables (e.g.: "my_variable"). This is a style called "lower snake case" and is not the naming convention used in Java, though there are many other languages where you might see this used frequently (e.g.: C, C++, Python, etc.); however, you may see some Java coders begin certain special variable names (e.g.: private class variables or constants) with an underscore to distinguish them from other variables used throughout their program.

Function

A sequence of packaged instructions that perform a task.

Method

In Object-Oriented programming, a *method* is a type of function that operates on the fields of a class.

```
int myMethod(){
        // ...does cool stuff.
}
void myMethod(int myInt){
        // ...does cool stuff.
}
```

Check out Oracle's Method documentation to learn more.

Object

An *Object* is an *instance* (or *variable*) of a class.

Stream

Think of this as the flow of data from one place to another. Most of our challenges require you to read input from *System.in* (also known as *stdin*, the standard input stream), and write output to *System.out* (also known as *stdout*, the standard output stream). In Java, the Scanner class is widely used to read input, but each language has its own mechanism for handling IO (input and output).

The syntax for reading from stdin using the *Scanner* class is as follows:

```
Scanner scan = new Scanner(System.in);
```

This creates a new *Scanner* object that reads from the *System.in* stream and can be accessed using the variable name $scan$. To read in information from stdin, you just need to apply Scanner's methods to your scanner object. Here are two basic examples:

```
scan.next(); // returns the next token of input
scan.hasNext(); // returns true if there is another token of input (false otherwise)
scan.nextLine() // returns the next LINE of input
scan.hasNextLine(); // returns true if there is another line of iput
```

Check out the comprehensive list of Scanner methods to learn more.

When you are finished reading from an input stream, you should *close* it to avoid a *resource leak*. The following line of code closes the *Scanner* object referenced by our $scan$ variable:

```
scan.close();
```

Let's say we want to assign a value received from stdin to some String that we'll name s, and then print it. We can accomplish this with the following code:

```
Scanner scan = new Scanner(System.in); // open scanner
String s = scan.next(); // read the next token and save it to 's'
scan.close(); // close scanner
System.out.println(s); // print 's' to System.out, followed by a new line
```

If the input token is `Hi!`, the above code will print `Hi!`.

You can also print text in quotes using *System.out.println*, or combine quoted text with a variable (e.g.: `System.out.println("Input received: " + s);`).

View Practice Challenge

Tutorial By

AllisonP

Video By

blondiebytes

View Practice Challenge