Java

Useful Links:

<https://www.tiobe.com/tiobe-index/>

<https://docs.oracle.com/javase/8/>

<https://spring.io/>

<https://www.oracle.com/java/technologies/downloads/>

<https://www.oracle.com/java/technologies/java-se-glance.html>

(a comprehensive Java reference site)

<https://www.javatpoint.com/java-tutorial>

(for the Spring section of the course.)

<https://www.baeldung.com/>

<https://www.w3schools.com/java/java_wrapper_classes.asp>

NOTES

Dynamic and Static

* Languages with *static typing* assign a type to declared variables at compile time and set aside the right amount of memory for that variable, while those with *dynamic typing* assign types only when the variable declaration code runs during execution and change the memory allocated for the variable as needed - this comes at the cost of program speed while it reorganizes things.
* **Java**, C, and Swift are statically typed languages. JavaScript, Python, and Ruby are dynamically typed.
* The biggest change you will see in your code resulting from this is that when we declare variables and functions in Java, there will be type declarations before the variable name.

Once a variable is declared to be a specific type, it can only store data of that type for its whole life cycle.

Text

Description automatically generated with medium confidence

1. We can declare variables without assigning a value because Java knows how much memory to allocate for that space
2. We can assign the value of a variable as we declare them too!
3. This will throw an error before we even run our code because the types do not match.

Default Values if Uninitialized

If a variable is declared but not initialized with a value, the following default values are assigned:

Graphical user interface, application

Description automatically generated

Naming Conventions

Some rules of thumb to follow when choosing a variable name:

* Variable names should *not* be reserved keywords like char, class or int, for example.
* Use *camel case*. Start the variable name with a lower-case letter and capitalize any words within the name, for example amountOfSale.
* Variable names cannot begin with number or a whitespace, and generally you should not use \_ (underscores).
* Make your variables readable. Rather than using a or b or other inscrutable names like amtColl, instead use something clear and descriptive like amountCollected, for example.

Text

Description automatically generated

Data Types

Primitive Types

Here are the most-used *primitive types* in Java:

* **int**: An int holds integers, like 4 or -51. It is limited to numbers between -2147483648 and 2147483647, or around 2 billion.
* **long**: These are for larger integers up to 2^63, or around 2 quintillion.
* **boolean**: A boolean represents one of two values: true or false.
* **double**: Doubles are for floating point numbers like 3.14159265. We will use *double* rather than *float* almost exclusively, because they are far more precise.
* **char**: The char data type is used to store a single character, like '$' or 'A'.

Object Types & "Wrapper Classes"

In addition to primitive data types, Java also has ***object types***. Some of these built-in object types in Java are what you might think of as ***wrapper classes.***A wrapper class is a more robust version that adds functionality to a primitive type and makes it easier to use in conjunction with other object types or data structures like ArrayList (which you'll learn about later in this chapter.) You may already have used something similar to Object types in Javascript or Python without knowing it! If you have ever used someString.length or someString.slice() in Javscript, these are added features to strings, that can only be accomplished in Java using object types.

**Some examples:**

* **Integer**:The Integer class wraps a value of the primitive type int into an object.
* **Long**: The Long class wraps a value of the primitive type long into an object.
* **Boolean**:The Boolean class wraps a value of the primitive type boolean into an object.
* **Double**: The Double class wraps a value of the primitive type double into an object.
* **Character**: The Character class wraps a value of the primitive type char into an object.

**Other object types built into Java are:**

* **String**: Represents a sequence of characters, much like a string in other languages.
* **BigInteger**: Represents an integer that can be any size at all.

**If Statements**

What if we have some code that we only want to execute under certain conditions? In computer programming, this is called a conditional statement. The syntax in Java looks like this:

Text

Description automatically generated

Text

Description automatically generated

A few things to remember:

* A condition must always evaluate to a boolean value, true or false
* If the condition is true, it will execute the block of code between the curly braces { ... } , then skip any other else if or else statements in that chain and keep going.
* If the condition is false, it will skip that block of code { ... } but continue to the code immediately following that block which may be another else if or else statement.

**Assignment Operator vs Comparison Operators**

As a programmer, you will often mistakenly type a = where you meant to type a == , but don't feel bad, seasoned developers do this from time to time, they just get faster and faster at catching it when things break. Let's review the assignment operator, = or what you might call the SET AS operator vs. all the comparison operators.

Graphical user interface

Description automatically generated with medium confidence

Comparison Operators

Remember, conditional statements evaluate to **boolean**values (true or false). Here is a table of the comparison operators you can use.

A picture containing graphical user interface

Description automatically generated

A picture containing text

Description automatically generated

Switch Statements

Text

Description automatically generated

We can see that the setup is very different from If Statements.

We start with the switch keyword and pass in the variable we are checking.

Inside of the switch statement, we specify what we want to happen for each possible case.

Using the example above, we can see that we print out the day of the week based on the value of our day variable.

Be sure to include the break keyword after each case. This will ensure that our program gets out of the switch block once the desired value has been found.

Including a default keyword at the bottom of our switch statement is equivalent to the else keyword in an if statement.

We do not need a break keyword after default as it is the last option in our switch block.

**Ternary Operators**

We will use the weather example from our previous lesson.

Text

Description automatically generated

Here is the same code using a ternary operator:



The ternary operator consists of three pieces; the condition, the return value if the condition is true, and the return value if the condition is false.

Methods of the String class

Text

Description automatically generated

Text

Description automatically generated

**When to use == and when to use .equals()**

When we use == in Java with objects, we are comparing if both variables point to the same address and the same object in memory. Much of the time this will have the desired outcome, but from time to time we might encounter some behavior that will seem different from what we've seen in other programming languages. For the assignments in this chapter, it will be fine just to use == to compare our basic built-in types. However, later as you work with more object types, you will often need to use .equals() instead of == to accomplish the desired comparison. In fact, when dealing with objects, this is convention.

Graphical user interface, text

Description automatically generated with medium confidence

We can still check if the two variables have the same value, we just need to use .equals() instead.

Text

Description automatically generated

Methods

A *method* is what we call a function in Java

Text

Description automatically generated

Now let's look at how methods are written in Java and compare. For the next assignment, we're going to be creating an Alfred Quote's library, a class that will have all of the functions associated with the Alfred bot's verbal responses. Again, because those functions belong to a class, they are called *methods*. Let's break down what methods look like in Java:

Text, website

Description automatically generated

Method Signatures:

Text

Description automatically generated

Methods Overloading

Method overloading is when we name two methods the same thing, even though they may behave completely differently. For example, we might have the following two methods in the same class:

Text

Description automatically generated

Which code would be invoked when we call the guestGreeting method? It depends on the argument type and the method signature! Due to Java's type system, if we try to call guestGreeting with a string and another string, it will know that we are trying to invoke the guestGreeting method with the signature guestGreeting(String, String). This allows us to handle different cases of method invocation based on our inputs. You may not use this in the beginning of your Java journey but knowing that Java allows this and identifies methods like this is extremely important.

The Main Method

Going forward you will only have one file that has this "main" method we've been using. Only files that contain this particular "main" method with the below signature can be run directly and executed. However, other files can be imported or injected into this file and will compile if they are referenced within the "main" method

Text

Description automatically generated

Importing

Java has many built in classes and methods that are readily available to developers. To tell Java that we want to use a special class for our program, all we have to do is use an import statement before our class definition. Let's say that we want to display the current date in our java program. All we have to do is import the Date class.

Text

Description automatically generated

Dependency Injection

Going forward, you will hear the term ***dependency injection*** quite a bit. Dependency injection is another way to use external code from another file. To do this, you create variable that will have access to all that class' methods. This is called an object instance. Object instances are used for other things, but this is one way to use methods contained in other files.

Text

Description automatically generated

Output:



Test File

From this point forward, we will be creating a test file that runs our Java classes. Our Greetings class contained only the information that belongs to the class itself, the functionality of our class. On the other hand, our TestGreetings class contained the main entry method and tested our Greetings class for functionality. We want to separate class information from testing/running information; this will allow us to modularize our code and reuse the same code across multiple programs. See the explanation of the TestGreetings code below.

Text

Description automatically generated

1. We are instantiating a new Greetings object. Now, all public methods of the Greetings class are available to that object.
2. Calling the getCurrentDate() method on the object.
3. Printing the currentDate string.

As long as both Greetings and TestGreetings files are in the same directory, you do not have to explicitly import one into another. Also, you can just run the javac compiler on TestGreetings, and it will compile both files for us.