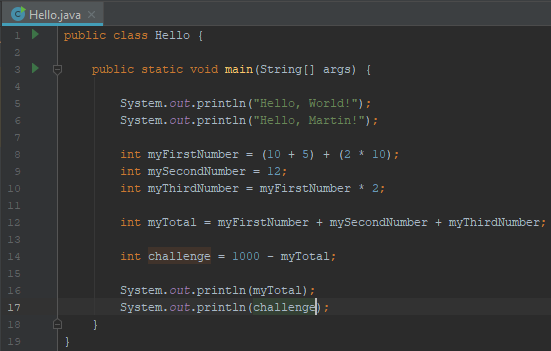
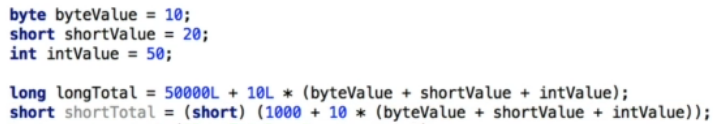
**Introduction**

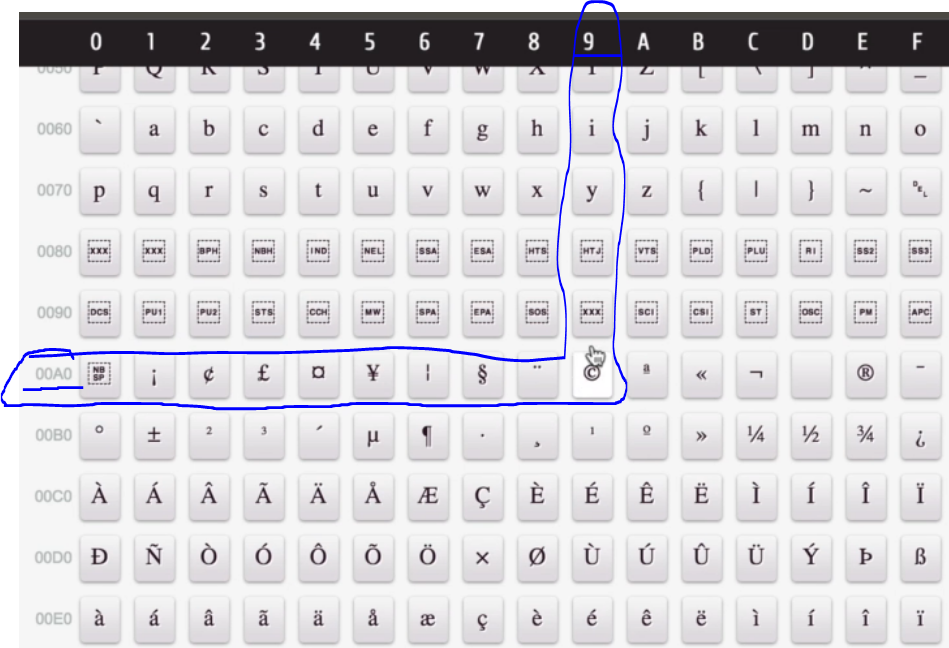
**What Are Variables?**  
\* A variable is something that makes a computer program very useful.  
\* A variable is a place in memory. Think of it as a box that you can store something in.  
\* What we can do is assign the type of information that we want to put into the box and we can also give it a name.  
\* Without the ability to store something, the computer wouldn’t be able to process anything and the programs you’d create would be very very basic and not really useful. So it stores information.  
\* `**int**` => short for integer.  
\* Integer in math terms is a whole number without any decimal points, just like in Java.  
\* If we want to use variables, we need to define what’s called the `**data type**` and then we need to assign a name to it, and optionally also define a value to it.  
\* It’s telling Java: please assign an area in memory that will be able to store an integer and call that memory location myFirstNumber and assign a value to it.  
\* A variable can be changed.  
\* You can type **sout** and have IntelliJ complete it as System.out.println().  
\* Case-sensitivity is very important.   


**Getting To Know Primitive Data Types - The Byte, Short, Int And Long**\* **To create a new project from a template**:  
=> Create New Project => Java => Next => Create project from template => Command Line App => Next => name => chance the Base package to: com.timbuchalka but you can change it to whatever you like => Finish  
=> Right-click the project name => Open Module Settings => make sure that this Language level is set to 8 - Lambdas, type annotations, etc. (we installed JDK 8)  
\* In computing terms normally there’s always a minimum/maximum for numbers, it’s in billions. If we use a number that’s smaller/greater than that range, we’d get an error and that’s because it physically won’t fit in the space that the computer allocated. In that scenario you’d use a different data type.   
\* If you assign an int variable to -2147483648, it’s the minimum value, if you try -2147483649, it’s going to give you an error saying that the number is too large. It’s telling you it tried to exceed the minimum/maximum boundaries. Same with 2147483647 being the maximum and 2147483648 being too large.  
\* I want to make it clear that any number that you type here that you say is a fixed number that you type the way we did when you define a variable, the part to the right (-2147483649) is called a **Literal**. So we’re typing in a Literal there to the right and that’s a fixed number. It wouldn’t be a Literal if you’ve done something like: (a \* b), that’s an **Expression**.  
\* You can include **underscores \_ in numbers to make them more readable** available since Java 7.  
\* 2\_147\_483\_647.  
\* If you want to use a number that’s outside of that range, you can use a different data type, Java gives us different data types for different types of things that we might wanna calculate.   
\* **byte**.  
\* This data type is allocated by the computer much smaller amount of space and consequently you can fit a much smaller number. A byte can store a number in range: -128 to 127. So why would we ever want to use a byte? A byte is more efficient because it takes about 1/4 of the space that integer would. So if you have a lot of calculations, it can be far more efficient to use a byte because the computer isn’t allocating as much space for a byte as it is for an integer. Particularly useful when the memory is very limited, such as a mobile device.  
\* **short**.  
\* The smallest number it can fit is: -32768 and the largest is 32767.  
\* Generally a short is allocated around 2x the amount of space that a byte is and about half the amount of an integer.  
\* **Long**.  
\* To process numbers that are larger than what int can store.  
\* When you’re assigning a long value, you want to **put the letter L at the end of it** and that tells the computer that this is a long value.   
\* The minimum value is: -9\_223\_372\_036\_854\_775\_808L.  
\* The maximum value is: 9\_223\_372\_036\_854\_775\_807L.

\* Those are the 4 Data Types that deal with whole numbers.  
\* In general you want to choose the integer by default. So probably 9-8/10 times you would use int when dealing with whole numbers.

\* When you type an expression such as: byte x = (y/2) you get an error: Incompatible types, required: byte, found: int.  
=> By default when you’re making an assignment in an expression, Java converts that to an int.  
=> What you need to do to get around that is you need to specifically tell Java that you want to treat it as a byte. To do that we do something called **Casting**.  
byte x = **(byte)** (y/2);  
\* Generally speaking an int is the way to go and you can see that that is the case because Java by default converts anything without specified Data Type to an Integer and that’s why I would suggest that you always work with an Integer unless you’ve got a good reason to not do that.  
\* If you do something like this:  
  
=> You don’t need to enter the (long) Cast there in front of it because this is converting automatically to type long. And long will happily accept an int. To be more precise we can put 5000L + 10L.  
\* The point I wanted to make here is that with variables of type `int` and `long` you really don’t need to do any Casting like you do for `short` and `byte`. If you put a value in a bracket for byte or short, it’ll automatically be converted to an int and a variable of type long will also accept an integer value.  
\* For a short you’d need to do:  
  
\* The point I wanna make is that using Integer is the best one to use generally but even if you’re using type long, Java handles a lot of the complexity for you and you don’t have to do this Casting because a long will happily accept an int when you’re defining a value to it as well.

**Getting To Know Primitive Data Types - Float And Double**  
\* **Float** and **Double** are the 2 Data Types that deal with floating point numbers.  
\* Float is single precision.  
\* Double is double precision.  
\* So essentially the Double data type has got twice as much detail as the Float.  
\* **The proper practice for entering a float is to enter an `f` after the number**.  
\* And because it is a single precision number, we can put a decimal point so we could enter 5.25f because a float actually handles that precision for us automatically, but we’re going to keep 5f now.  
\* **The proper practice for entering a double is to enter a `d` after the number**.  
\* Java will assume that a number with a decimal point is a Double by default and that’s similar to how a whole number was an Integer by default.  
\* So if you try: float x = 5.4, that would give you an error required: float, found: double.  
\* Using the `f` and `d` is better than using (Cast) if you can use it.  
\* If you print a float or a double, it’ll add a decimal .0 if you don’t have it there.  
\* **5 / 2 = 2**.  
\* **5f / 2f = 2.5**.  
\* **5d / 2d = 2.5**.  
\* Here you can see that the Double is much more precise:  
  
\* Double takes 2x the amount of memory that a float does.  
\* However, there are 2 compelling reasons to using a Double whenever you’re using a floating point in Java:  
1) Double is actually faster on many modern computers.  
2) Many in-built functions in Java (math functions like sin, cos) they actually use Double as their calculations.  
3) Double is far more precise, there’s more digits of precision. (16 vs 7 units of precision)  
\* You can use underscores in decimal numbers too.

**Getting To Know Primitive Data Types - Char And Boolean**  
\* **Char** and **Boolean**.  
\* **In a Char you can store one letter or number or special character in ‘’, you can also put what’s called Unicode characters, those are special characters that support multiple languages and also things like the copyright symbol and so on**.  
\* This is how to use them:  
  
\* The copyright symbol would be : “\u00A9”  
\* If you hover over the symbol on the site: unicode-table.com/en/#control-character, you can see what it should be.  
\* **Boolean can be either `true` or `false`**.

**Understanding Strings And Finishing Up Primitive Data Dypes**  
\* We’ve covered the following 8 Data Types:  
1. byte  
2. short  
3. int  
4. long  
5. float  
6. double  
7. char  
8. boolean  
\* They are called **Primitive Data Types**.  
\* When I say primitive type, that’s actually something that’s defined by the Java language and is named by special reserved keyword (byte, short, int, long, float, double, char, boolean).  
\* As you get more experienced in Java, you’ll learn to use the right Primitive Data Type for a given computing problem and that’s actually all part of becoming a programmer and I’m going to be teaching that as we go through this course.  
\* There’s a way to create **our own Data Types** also called **Objects**, the interesting thing about doing that in the future is you can actually combine Data Types. For example you can create sort of like a super data type that contains maybe a byte and an integer and a double. We’re getting into the concepts of creating classes and that’s actually an entire section of the course in it’s own right.  
\* **String**.  
\* You can’t say it’s a primitive data type but it’s something that’s going to be used so commonly and it’s been so integrated into the language that it really comes across as a ninth Data Type.  
\* String is a sequence of characters.  
\* We use single quotes: ‘’ for a char.  
\* We use double quotes: “” for a String.  
\* We can append to that string automatically by using the + operator.  
\* String x = “10” + 50 = “1050” so because we’re using a string on the left-hand side and there’s an int on the right-hand side, Java is actually smart enough to say: Okay I know you’re trying to add an int to a String so it actually converts or effectively looks at the contents of myInt and converts that into a String to add to the end.  
\* That holds true for other Data Types too.  
\* String x = “10” + 120.47 would also convert it to a String “10120.47”.  
\* Later we’re going to look into some of the advanced features of a String because we can do all sorts of things - we can delete characters out of it, we can insert characters at specific positions, there’s a lot of flexibility with Strings but we need to know a little bit more about Classes before we get into that in too much detail. I’m just saying that normally you would use other ways to add these things to a String than what we did, to make it a bit more clear what you’re trying to achieve.  
\* This was just an introduction to the **String Class**.  
\* String is not a primitive data type, it’s actually called a Class and it’s treated differently.

**Operators In Java**  
\* **Operators are actually special symbols that perform specific operations on one, two, or three operands, and then return a result**.  
\* **You can click Code => Reformat Code if you’re getting a bit disorganized**.  
\* Operands are the things in the right side of the operator.  
  
\* **=** **assignment** => whatever is evaluated on the right, its value is placed in the variable on the left (using a Literal or an expression - int result = 1; VS int result = 1 + 2;).  
\* **+** **addition**  
\* **-** **subtraction**  
\* **\*** **multiplication**  
\* **/** **division**  
\* **%** **remainder operator** => return the result of what the remainder is after dividing something.  
\* **++** **increment by one**  
\* **--** **deduct one**  
\* **==** **equal to**\* **!=** **not equal to**\* **>** **greater than**\* **>=** **greater than or equal to**  
\* **< less than**  
\* **<= less than or equal to**\* **&& AND**\* **|| OR**\* We can use these operators in combinations with = as a shortcut: +=, -=, %=, /=, \*=, etc.  
\* We’re now gonna start looking at a way of expanding this and using other types of operators. To do that, we need to talk firstly about the **`if/then` statement**, we’ll talk about it in more detail later but it’s a way to perform some conditional code, depending on whether a value is true or false.  
\* **if (condition) doSomething** you can indent it on the next line.

**More On Operators And Operator Precedence**  
\* Using parenthesis around operands makes it less ambiguous and more clear.  
\* **condition ? do : else** **Ternary Operator** is a way to set a value based on two conditions.

**Resources**  
**Unicode Table:**  
<https://unicode-table.com/en/#control-character>   
**Summary of Operators:**   
<https://docs.oracle.com/javase/tutorial/java/nutsandbolts/opsummary.html>   
**Java Operator Precedence:**  
<http://cs.bilkent.edu.tr/~guvenir/courses/CS101/op_precedence.html>