Lesson 1.3 Data Types and Variables

For computers to be able to work with data, they need to be able to know what type of data they're working with: if we say 2 + 3 do we want the computer to print those characters? or do we want it to perform that calculation?

We also need to be able to keep track of the data that they were working with in the computer's memory. Data stored in the computer's memory is referred to by a reference to that memory location called a variable.

Let's see how this all works.

Types and Variables

Examples of three most common types that we'll be working with:

Variables are references to memory

Declarations

So to store the value **`13`** in a variable called **`luckyNumber`**, you first have to declare the variable and the type of data that you'll be storing in it:

```
int luckyNumber;    // declare that luckyNumber will be "storing" an integer
```

Assigning a value to a variable

Once you've established a variable, you can then use the assignment operator—in Java this is the equals sign **=** to identify the primitive or object that the variable will refer to.

If you know that **`luckyNumber`** will start out with this value, you can combine these two steps into one step:

```
int luckyNumber = 13;
```

If you try this, though, you'll get an error:

```
int luckyNumber = “13";
```

If you want to change the value that a variable refers to, use the equals sign again:

```
luckyNumber = 12;     // variable is already declared, don't need to do it again!
```

Whatever value was stored in **`luckyNumber`** before is no longer available to us. We have overwritten its value with this new, updated value.

Identifiers

There are rules for naming identifiers, some imposed by Java, and some we'll impose on ourselves.

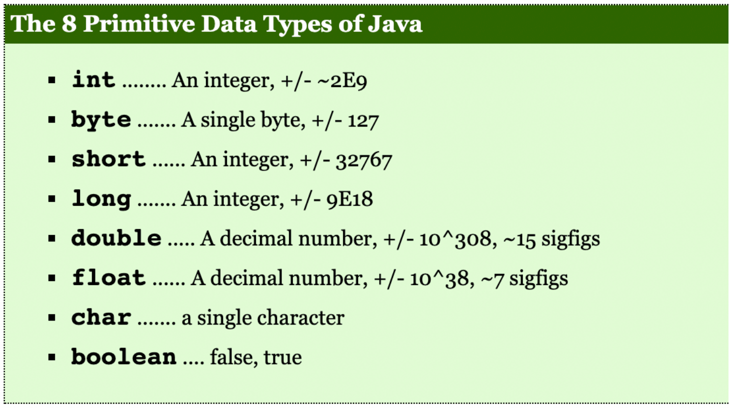
- can be made of letters, numbers, **`$`**, and **`_`**, but can't begin with a number
- thus can't have special characters like **`!`**, **`#`**, or spaces
- can't use *reserved words* that have special meaning in Java (**`public`**, **`class`**, **`main`**, etc.)
- are case-sensitive

Common conventions:

Intro to Number Types

We've began using data since the very first time we wrote a Java program, when we printed out the String "Hello, World!" Interestingly, the **`String`** is *not* one of the fundamental data types in Java, so... we've got a little bit still to learn in terms of our understanding of how values are represented.

There are 8 *primitive* data types in Java, types which are fundamentally different from *objects*. You should be aware of the list, although it's not necessary to memorize it.



The 8 Primitive Data Types of Java

- **`int`** .... An integer, +/- ~2E9
- **`byte`** .... A single byte, +/- 127
- **`short`** ...... An integer, +/- 32767
- **`long`** .... An integer, +/- 9E18
- **`double`** ..... A decimal number, +/- 10^308, ~15 sigfigs
- **`float`** ...... A decimal number, +/- 10^38, ~7 sigfigs
- **`char`** .... a single character
- **`boolean`** .... false, true

Arithmetic Operations and Math Functions

Order of Operations in Java

Java's Order of Operations follows the standard mathematical order of operations.

Please Excuse My Dear Aunt Sally

() → exponents? → *,/ → +,–

This can be very useful, except when you're not expecting it.

Examples of Division in Java

- Floating point division:

```
7.0 / 4 → 1.75 ( a double value)
```

- Integer division (whole number):

```
7 / 4 → 1 (an int value)
```

- Integer division (whole number remainder):

```
7 % 4 → 3 (an int value)
```

The `%` sign is called the *mod* operator, for "modulo"—it yields the remainder of an integer division.

Examples

Programming division

Which of the following Java statements will correctly calculate the average of 3, 5, and 6?

1. `double average = (3 + 5 + 6) / 3;`
2. `double average = (3 + 5 + 6) / 3.0;`
3. `double average = ((double) 3 + 5 + 6) / 3;`
4. `double average = (double) ((3 + 5 + 6) / 3);`
5. `double average = (double) (3 + 5 + 6) / 3;`