INTRODUCTION TO JAVA DATA TYPES AND OPERATIONS

Java has eight data types that are built into the Java language and identified by Java keywords; they are called the ***primitive data types***.

Java Built-In ("Primitive") Data Types	
Keyword	**Meaning**
`byte` **`short`** **`int`** **`long`**	Holds an ***integer*** value – that is, a whole number with no *fractional part.*
`char`	Holds a single keyboard character.
`boolean`	Holds either **`true`** or **`false`**.
`float` **`double`**	Holds a ***floating-point*** value – that is, one that may have a fractional part.

Java allows you to declare variables and write literals in any of these eight data types.

Examples

```
byte a = 127;
short b = 1_024;
int c = 65_368;
long d = 2_147_483_648L;
char e = 'A';
boolean f = true;
float g = 1E6F;
double h = 1E10;
```

To manipulate data of these primitive data types, Java provides a number of built-in operators. Below is a table that lists all but a handful of these operators. The purpose of showing all of these together is to convey the precedence and association between them. ***Precedence*** indicates when an operator is evaluated relative to operators of a different precedence. In the table, precedence is indicated by number – an operator of precedence 1 is evaluated first, one of precedence 2 is evaluated second and so on.

Example
Given the expression `x/y++`, the operator `++` has precedence 1 whereas `/` has precedence 3. Thus, `++` evaluates before `/`.

Association indicates when an operator is evaluated relative to operators of the same precedence when they are cascaded within the same expression.

Examples

Given the expression `x*y/z`, both operators `*` and `/` have precedence 3. Therefore, the association rule, left to right, dictates their order of evaluation. `*` evaluates before `/`.

On the other hand, `=` associates right to left so that the expression `x=y=z` evaluates `y=z` first.

The operator `++` "cannot be cascaded" even with parentheses so that both expressions `x++++` and `(x++)++` are syntactically incorrect.

On the other hand, `-` used as a unary minus operator can be cascaded using parentheses. The expression `---x` is not syntactically correct whereas `-(-(-x))` is correct.

Finally, the order that operators are evaluated can be changed by using parentheses.

Example
The expression `(x+y)/z` evaluates the `+` before the `/` even though it is of a lower precedence.

The Most Common Java Built-In Operators

Symbol	Operation	Precedence	Association
var`++`	Increment after fetching	1	Cannot be cascaded
var`--`	Decrement after fetching	1	Cannot be cascaded
`++`*var*	Increment before fetching	2	Cannot be cascaded
`--`*var*	Decrement before fetching	2	Cannot be cascaded
`+`	Unary plus	2	Cascade with ( ) only
`-`	Unary minus	2	Cascade with ( ) only
`!`	Logical NOT	2	Right to left

The Most Common Java Built-In Operators			
Symbol	**Operation**	**Precedence**	**Association**
`*`	Times	3	Left to Right
`/`	Divide	3	Left to Right
`%`	Remainder	3	Left to Right
`+`	Add	4	Left to Right
`-`	Subtract	4	Left to Right
`<`	Less than	6	Cannot be cascaded
`<=`	Less than or equal to	6	Cannot be cascaded
`>`	Greater than	6	Cannot be cascaded
`>=`	Greater than or equal to	6	Cannot be cascaded
`==`	Equal to	7	Left to right
`!=`	Not equal to	7	Left to right
`&&`	Logical AND	11	Left to right short-circuited
`||`	Logical OR	12	Left to right short-circuited
`? :`	Conditional	13	Innermost to outermost
var `=` *expr*	Store *expr* into *var*	14	Right to Left
var `*=` *expr*	*var* `=` *var* `*` *expr*	14	Right to Left
var `/=` *expr*	*var* `=` *var* `/` *expr*	14	Right to Left
var `%=` *expr*	*var* `=` *var* `%` *expr*	14	Right to Left
var `+=` *expr*	*var* `=` *var* `+` *expr*	14	Right to Left
var `-=` *expr*	*var* `=` *var* `-` *expr*	14	Right to Left

Because of their importance in computer programming, Java has built-in operators that manipulate character strings.

Example

Java has a built-in operator **+** that concatenates two strings (i.e. appends one onto another). For example, this statement prints “Herbert Hoover”:

```
System.out.println( "Herbert" + " " + "Hoover" );
```

In addition to these built-in data types and operators, the Java API contains support for manipulating strings and data of the eight primitive classes.

Java API Support for Strings and Primitive Data	
API Class Name	**Features**
`java.lang.String`	Allows you to build and manipulate objects that contain character strings
`java.util.Scanner`	Allows you to input and scan data of any of these data types, including `String` and excepting `char`

The topics that follow this one in the current collection discuss all of these Java features in detail. They are designed to be read in any order, or in isolation if you have need to learn one or two concepts before diving into something else.