Getting Started With Java – Data Types & Variables

The basic unit of information on a computer is called a *bit*, which has two states: *on* or *off* (1 or 0). Bits are generally grouped into units of 8, which is called a *byte*. A single byte can store all of the possible combinations of the 8 bits, or $2^8 = 256$ values.

In Java, as with most programming languages, there are various *data types*, which are used to represent and (as variables) store different types of data. Each data type has different demands on how much space it requires in the computer's memory, and this is measured in bytes.

Integer Data Types

We have already referred to a single byte representing 256 states of the 8 bits. Another way to look at this is to imagine each of those states representing a number. In the case of a single byte of data, Java uses the **byte** *integer data type* to represent the numbers from -128 to 127.

Type	Size (bits)	Size (bytes)	Range	Approximate Range
byte	8	1	-2^7 to 2^7-1	± 100
short	16	2	-2^{15} to $2^{15}-1$	± 30000
int	32	4	-2^{31} to $2^{31}-1$	$\pm 2\ 000\ 000\ 000$
long	64	8	-2^{63} to $2^{63}-1$	$\pm 9\times 10^{18}$

The **int** type is adequate for most tasks and will generally be your default integer data type.

Integer constants must not be written with a decimal point, and they cannot contain any separators between digits.

For example, each of the following is an illegal integer constant.

(a) 37.0 contains a decimal point
(b) -12 562 contains a blank space between digits
(c) 1,233,985 contains commas between digits

Real Data Types

Type	Size (bits)	Size (bytes)	Precision	Approximate Range
float	32	4	at least 6 decimal places	$\pm 3.4\times 10^{38}$
double	64	8	at least 15 decimal places	$\pm 1.8\times 10^{308}$

Due to memory and speed considerations, the **float** data type was traditionally used more frequently. With modern computers, however, the **double** data type does not result in the same performance issues, and it should be used for most calculations involving real numbers.

Each of the following is a valid floating point constant.

5.23 .3 2818. -0.0002 6.7

Floating point constants can also be written in a form similar to that used in scientific notation.

(a) 5.6e2 represents the value 5.6×10^{2} = 560
(b) 37E-4 represents the value 37×10^{-4} = 0.0037
(c) -0.667e-2 represents the value -0.667×10^{-2} = -0.00667

Character Data Type

Java follows a character representation called *Unicode*, which uses 16 bits, or 2 bytes, to represent characters. This allows for 65536 possible characters, so all of the world's characters and symbols, along with specialized symbols and shapes, can be represented in Unicode.

Characters in Java are of type **char**.

Identifiers

In order to create useful programs, we need the ability to store information and retrieve that information as needed. Information is stored in the memory of the computer in specially reserved areas known as *variables*. In order to keep track of these variables, we use *identifier names* (or simply *identifiers*) for each *variable*, *method*, or *class* in our program.

The rules for creating any type of identifier are:

1. Any character used in an identifier must be a letter or the alphabet, a digit (0, 1, ..., 9), or an underscore character (_).

2. The first character cannot be a digit.

3. By convention, *classes* are given identifiers using PascalCase, where each word begins with a capital letter.

4. By convention, *variables* and *methods* are given identifiers using camelCase, where the first letter is lowercase, and any other words in the identifier are uppercase.

5. You cannot use any of the following *reserved words*, each of which has a predefined meaning in the Java language.

abstract	**assert**	**boolean**	**break**	**byte**	**case**
catch	**char**	**class**	**const**	**continue**	**default**
double	**do**	**else**	**enum**	**extends**	**false**
final	**finally**	**float**	**for**	**goto**	**if**
implements	**import**	**instanceof**	**int**	**interface**	**long**
native	**new**	**null**	**package**	**private**	**protected**
public	**return**	**short**	**static**	**strictfp**	**super**
switch	**synchronized**	**this**	**throw**	**throws**	**transient**
true	**try**	**void**	**volatile**	**while**	

Declaring Variables

To reserve space in memory for variables, we need to write a *declaration statement*, where we specify the type of a variable and its identifier (i.e., name). A simple declaration might look like

```
<type> <identifier>;
```

Note the a declaration statement, like any other statement in Java is *terminated* (i.e., ended) by the semicolon character. It is possible to declare multiple variables of the same type using a single declaration statement of the form

```
<type> <identifier1>, <identifier2>, ..., <identifierk>;
```

Exercises

1. What is the smallest type of integer that can be represented by each of the following types?
   (a) 50 000
   (b) -3 000 000 000
   (c) -125
   (d) 128

2. State, with reasons, which of the following are not legal Java integer constants.
   (a) -47
   (b) 23.
   (c) -0
   (d) 22 900

3. Rewrite in standard decimal form.
   (a) 2.94e1
   (b) 0.0004e3
   (c) -2e-3
   (d) 26.77e-3
   (e) -54E-3
   (f) -.3e1

4. What is the difference between identifiers used for classes and those used for variables and methods?

5. Identify, with reasons, any identifiers that should not be used for Java variables.
   (a) digitSum
   (b) retail price
   (c) switch
   (d) heightPlusDepth
   (e) this&That
   (f) priceIn$
   (g) number-of-wins
   (h) ageDuGarcon
   (i) average_age
   (j) This

Assigning Values to Variables

Declaring a variable only reserves space, but does nothing to fill the space with useful values or information. Once variables are declared, there are many ways to give them values, the simplest of which is the *assignment statement*.

```
<identifier> = <value>;
```

It is possible to streamline this process by performing the *declaration* and the *assignment* statement at the same time.

```
<type> <identifier> = <value>;
```

For example,

```
int age = 16;    // declares age and initializes value to 16
```

String Variables

Java is an *object-oriented language*. There are eight primitive data types (byte, short, int, long, float, double, char, boolean), but an unlimited number of potential objects. One of the most useful objects, which we will use extensively, is the `String` object. We can declare a string variable and assign it a value in a manner which looks identical to the other data types.

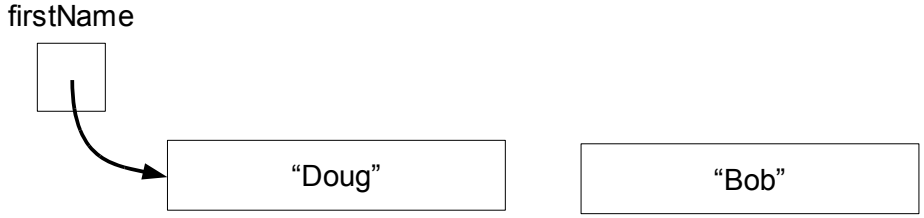
```
String firstName;     // declares a string variable called firstName
firstName = "Bob";    // assigns the value "Bob" to the string firstName
```

There is a subtle yet important difference between our String variable and variables containing the primitive data types. *The string variable doesn't actually contain a string*. Instead, it contains a reference to a string object, which is stored elsewhere in memory.

For example, consider the following code:

```
String firstName = "Bob";
firstName = "Doug";
```

The string variable is declared and initialized to refer to the string “Bob”. On the next line, the string variable refers to the string “Doug”. Both the strings “Bob” and “Doug” still exist in memory, but there is no way to get back to “Bob”.



Once we create a string, its value cannot be changed. Strings are called *immutable objects*. It is still possible to have the variable refer to a different string.

Constants

Java allows us to associate an identifier with a constant value through the use of the final modifier in the declaration of a variable. By convention, constants are given identifiers which are all upper case, which helps easily identify them when reading code.

```
final int CLASS_SIZE = 30;
final char TERMINATOR = '*';
```

Once an identifier has been declared to be final, its value can never be changed.

Exercises

1. Draw diagrams like those shown in the lesson to illustrate the result of executing the following statements.

(a)
```
int a = 1;
String b = "2";
```

(b)
```
String a = "first";
String b = "second";
```

(c)
```
String a = "one";
String b = a;
a = "two";
```

(d)
```
String a = "ein";
String b = "zwei";
a = b;
b = "drei";
```

2. What would be printed by the following fragment?

```
String c = "cat";
String d = "dog";
String s = c;
c = d;
d = s;
System.out.println("c is " + c);
System.out.println("d is " + d);
```