

Java Data Types

[< Previous](#)[Next >](#)

Java Data Types

As explained in the previous chapter, a variable in Java must be a specified data type:

Example

```
int myNum = 5;           // integer (whole number)
float myFloatNum = 5.99f; // floating point number
char myLetter = 'D';      // character
boolean myBool = true;    // boolean
String myText = "Hello";  // String
```

[Run example »](#)

Data types are divided into two groups:

- Primitive data types - includes `byte`, `short`, `int`, `long`, `float`, `double`, `boolean` and `char`
- Non-primitive data types - such as `String`, `Arrays` and `Classes` (you will learn more about these in a later chapter)

Primitive Data Types

A primitive data type specifies the size and type of variable values, and it has no additional methods.

There are eight primitive data types in Java:

Data Type	Size	Description
-----------	------	-------------

byte	1 byte	Stores whole numbers from -128 to 127
short	2 bytes	Stores whole numbers from -32,768 to 32,767
int	4 bytes	Stores whole numbers from -2,147,483,648 to 2,147,483,647
long	8 bytes	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,808
float	4 bytes	Stores fractional numbers from 3.4e−038 to 3.4e+038. Sufficient for storing 6 to 7 decimal digits
double	8 bytes	Stores fractional numbers from 1.7e−308 to 1.7e+038. Sufficient for storing 15 decimal digits
boolean	1 byte	Stores true or false values
char	2 bytes	Stores a single character/letter

Numbers

Primitive number types are divided into two groups:

Integer types stores whole numbers, positive or negative (such as 123 or -456), without decimals. Valid types are `byte`, `short`, `int` and `long`. Which type you should use, depends on the numeric value.

Floating point types represents numbers with a fractional part, containing one or more decimals. There are two types: `float` and `double`.

Even though there are many numeric types in Java, the most used for numbers are `int` (for whole numbers) and `double` (for floating point numbers). However, we will describe them all as you continue to read.

Integer Types

Byte

The `byte` data type can store whole numbers from -128 and 127. This can be used instead of `int` or other integer types to save memory when you are certain that the value will be within -128 and 127:

Example

```
byte myNum = 100;  
System.out.println(myNum);
```

[Run example »](#)

Short

The `short` data type can store whole numbers from -32768 to 32767:

Example

```
short myNum = 5000;  
System.out.println(myNum);
```

[Run example »](#)

Int

The `int` data type can store whole numbers from -2147483648 to 2147483647. In general, and in our tutorial, the `int` data type is the preferred data type when we create variables with a numeric value.

Example

```
int myNum = 100000;  
System.out.println(myNum);
```

[Run example »](#)

Long

The **long** data type can store whole numbers from -9223372036854775808 to 9223372036854775808. This is used when int is not large enough to store the value. Note that you should end the value with an "L":

Example

```
long myNum = 15000000000L;  
System.out.println(myNum);
```

[Run example »](#)

Floating Point Types

You should use a floating point type whenever you need a number with a decimal, such as 9.99 or 3.14515.

Float

The **float** data type can store fractional numbers from $3.4e-038$ to $3.4e+038$. Note that you should end the value with an "f":

Example

```
float myNum = 5.75f;  
System.out.println(myNum);
```

[Run example »](#)

Double

The **double** data type can store fractional numbers from $1.7e-308$ to $1.7e+038$. Note that you should end the value with a "d":

Example

```
double myNum = 19.99d;  
System.out.println(myNum);
```

[Run example »](#)

Use `float` or `double`?

The **precision** of a floating point value indicates how many digits the value can have after the decimal point. The precision of `float` is only six or seven decimal digits, while `double` variables have a precision of about 15 digits. Therefore it is safer to use `double` for most calculations.

Scientific Numbers

A floating point number can also be a scientific number with an "e" to indicate the power of 10:

Example

```
float f1 = 35e3f;  
double d1 = 12E4d;  
System.out.println(f1);  
System.out.println(d1);
```

[Run example »](#)

Booleans

A boolean data type is declared with the `boolean` keyword and can only take the values `true` or `false`:

Example

```
boolean isJavaFun = true;  
boolean isFishTasty = false;
```

```
System.out.println(isJavaFun);    // Outputs true
System.out.println(isFishTasty);  // Outputs false
```

[Run example »](#)

Boolean values are mostly used for conditional testing, which you will learn more about in a later chapter.

Characters

The **char** data type is used to store a **single** character. A char value must be surrounded by single quotes, like 'A' or 'c':

Example

```
char myGrade = 'B';
System.out.println(myGrade);
```

[Run example »](#)

Strings

The **String** data type is used to store a sequence of characters (text). String values must be surrounded by double quotes:

Example

```
String greeting = "Hello World";
System.out.println(greeting);
```

[Run example »](#)

The String data type is so much used and integrated in Java, that some call it "the special **ninth** type".

A String in Java is actually a non-primitive data type, because it refers to an object. The String object has methods that is used to perform certain operations on strings. **Don't worry if you don't understand the term "object" just yet.** We will learn more about strings and objects in a later chapter.

Non-Primitive Data Types

Non-primitive data types are called **reference types** because they refer to objects.

The main difference between primitive and non-primitive data types are:

- Primitive types are predefined (already defined) in Java. Non-primitive types are created by the programmer and not defined by Java (except for String).
- Non-primitive types can be used to call methods to perform certain operations, while primitive types cannot.
- A primitive type has always a value, while non-primitive types can be **null**.
- A primitive type starts with a lowercase letter, while non-primitive types starts with an uppercase letter.
- The size of a primitive type depends on the data type, while non-primitive types have all the same size.

Examples of non-primitive types are String, Arrays, Classes, Interface, etc. You will learn more about these in a later chapter.

[< Previous](#)[Next >](#)

Copyright 1999-2018 by Refsnes Data. All Rights Reserved.