**Data Types**

For easier and efficient processing, data is classified into different types (**Data Types**) such as **byte**, **char**, **boolean**, **int**, **float**, **String**, **Array**, **Enum** etc. These data types can be grouped as **Primitive** and **Composite (or Reference Type/non-Primitive)**.

As their names suggest, primitives form the primary building blocks(these usually have built-in support in the language) and composites are usually made up of one or more primitive type.

Java is a statically typed language, which means that the type imposes some constraints on the value it can hold and the operations that can be performed, and this is verified by the compiler during the compilation.

In Java, there are different numeric data types to represent whole numbers(integral) and fractions([floating-point](http://en.wikipedia.org/wiki/Floating_point)) of different sizes (magnitudes).

* Integral type: [signed](http://en.wikipedia.org/wiki/Signed_number_representations) [two's-complement](http://en.wikipedia.org/wiki/Two%27s_complement)
* byte: 8 bit (1 byte)  - 128 (-27) and 127 (27-1).
* short: 16 bits (2 byte) - 32,768 (-215) and 32,767 (215-1).
* int: 32 bits (4 byte) -2,147,483,648 (-231) and +2,147,483,647 (231-1).
* long: 64 bits (8 byte) -9,223,372,036,854,775,808 (-263) and 9,223,372,036,854,775,807 (263-1)
* Floating-point:
* float: 32 bit(4 byte) [single-precision 32-bit IEEE 754](http://en.wikipedia.org/wiki/Single-precision_floating-point_format) [floating point](http://en.wikipedia.org/wiki/Floating_point)
* double: 64 bit(8 byte) [double-precision 64-bit IEEE 754](http://en.wikipedia.org/wiki/Double-precision_floating-point_format) [floating point](http://en.wikipedia.org/wiki/Floating_point)
* Char: 16-bit unsigned integers representing UTF-16 code units ([#3.1](http://docs.oracle.com/javase/specs/jls/se8/html/jls-3.html#jls-3.1)).
* Boolean:

Note: The initial 256 characters of the [ASCII character set](https://en.wikipedia.org/wiki/ASCII#Code_chart) are the same in **Unicode** character set.

We can include an underscore ('\_') in numeric literals to show grouping of digits.  
For example: long mySalary = 900\_000L;// Which is equal to 900000 (Nine Hundred Thousand)

long magicNumberInHex = 0xCAFE\_BABE; // Want to know more about [Magic Number](http://en.wikipedia.org/wiki/Java_class_file#Magic_Number)?

Note: local variables are never assigned default value by compiler

***# int***

The primitive type int has a corresponding wrapper class called [Integer](https://docs.oracle.com/javase/8/docs/api/java/lang/Integer.html). Both int and Integer can be used interchangeably.

int num1 = 3;

Integer num2 = 4;

Integer total = num1 + num2;

int valueOfNum2 = total - num1;

both are correct.

The fixed values 3, 4 assigned to num1 and num2 respectively are called **integer literals**.

This class provides constants and methods that are useful while working with int(s).

The default value of a primitive int is 0, when not initialized.

default value of a reference of type Integer is null, when not initialized.

In Java 8 and later, int can also represent unsigned 32-bit integers, with a min value of 0 and a max value of 232-1. To use unsigned integers, use **Integer**

***# Long:***

The primitive type long has a corresponding wrapper class called Long. Both long and Long can be used interchangeably.

long num1 = 22222222223L;

Long num2 = 23L;

Long total = num1 + num2;

long valueOfNum2 = total - num1;

The fixed values 22222222223L, 23L assigned to num1 and num2 respectively are called long literals.

the long literal values are **suffixed** with char **'L'** (lowercase of 'L' can also be used, however the uppercase is more readable)

The **default value** of a **primitive** long is **0**L, when not initialized

the default value of a **reference of type** Long is **null**

In Java 8 and later, **long** can also represent **unsigned 64-bit integers**, with a min value of 0 and a max value of 264-1. To use unsigned long integers, use the Long class.

***# Short***

The primitive type short has a corresponding wrapper class called Short. Both short and Short can be used interchangeably

short num1 = 3;

Short num2 = 4;

Short total = (short)(num1 + num2);

short valueOfNum2 = (short)(total - num1);

The default value of a primitive short is 0, when not initialized.

the default value of a reference of type Short is null

We normally use int instead of short to hold integer values. However, when using large arrays if we know that the values stored will be within the range of a short, using an array of type short may help saving memory.

[ Type casting is required because ‘int’ is the default data type.]

***# Byte***

The primitive type byte has a corresponding wrapper class called Byte. Both byte and Byte can be used interchangeably. Which means, we can say:

byte num1 = 3;

Byte num2 = 4;

Byte total = (byte)(num1 + num2);

byte valueOfNum2 = (byte)(total - num1);

The default value of a primitive byte is 0, when not initialized.

default value of a reference of type Byte is null

We normally use int instead of byte to hold integer values. However (like in the case of short), when using large arrays if we know that the values stored will be within the range of a byte, using a byte array may help saving memory.  
A byte data type can also be used to indicate to a programmer that the value it will hold is small. More commonly they are used while manipulating raw data using byte arrays.

[ type casting is required]

***# Float***

float can represent

1. both positive and negative numbers
2. positive infinity (Float.POSITIVE\_INFINITY) and negative infinity (Float.NEGATIVE\_INFINITY)
3. NaN (a special Not-a-Number value, i.e. mathematically undefinable number).

A NaN value is used to represent the result of invalid operations such as dividing zero by zero, or square root of a negative number. Its value is represented by a constant in Float class (Float.NaN) etc.

The primitive type float has a corresponding wrapper class called Float. Both float and Float can be used interchangeably.

float num1 = 3.0f;

Float num2 = 4.2f;

Float total = num1 + num2;

float valueOfNum2 = total - num1;

The default value of a primitive float is 0.0f, when not initialized.

default value of a reference of type Float is null, when not initialized

***# Double***

double is more commonly used than a float in calculations like sine, cos, tan etc... Like in float a double can represent

1. both positive and negative numbers
2. positive infinity (Double.POSITIVE\_INFINITY) and negative infinity (Double.NEGATIVE\_INFINITY)
3. NaN (a special Not-a-Number value, i.e. mathematically undefinable number).  
   A NaN value is used to represent the result of invalid operations such as dividing zero by zero, or square root of a negative number. Its value is represented by a constant in Float class (Double.NaN) etc.

The **64 bits** of a **double** are divided into three parts.

- 1 Sign bit is used to mark the sign of the number.

- 11 bits are used to determine the value of the exponent.

- remaining 52 bits are used to determine the [mantissa](http://en.wikipedia.org/wiki/Significand) (significand).

*For more info* [*Double Precision Floating Point Numbers*](http://en.wikipedia.org/wiki/Double-precision_floating-point_format)

The primitive type double has a corresponding wrapper class called Double. Both double and Double can be used interchangeably. Which means, we can say:

double num1 = 3.4d;

Double num2 = 1.2d;

Double total = num1 + num2;

double valueOfNum2 = total - num1;

The default value of a primitive double is 0.0d, when not initialized

the default value of a reference of type Double is null, when not initialized.

// revise from here ?

***# BigDecimal***

double x = 0.2; double y = 0.1;

System.out.print(x + y); // Output: 0.30000000000000004

* The number of bits used for precision limits the set of rational numbers that can be represented exactly using a float or a double. This is why we get results like these when we use float or double in arithmetic operations.
* [how floating point numbers are represented in binary format(base-2)](http://en.wikipedia.org/wiki/Floating_point#Representable_numbers.2C_conversion_and_rounding).

**# Note:** Never use float or double in currency calculations which require a high degree of precision.

# BigDecimal is the right choice in Java for such calculations.

# Java provides a class called [BigDecimal](http://docs.oracle.com/javase/7/docs/api/java/math/BigDecimal.html" \t "_blank) for representing and computing decimal numbers with any required precision

Java provides a class called [BigDecimal](http://docs.oracle.com/javase/7/docs/api/java/math/BigDecimal.html" \t "_blank) for representing and computing decimal numbers with any required precision..

**Precision**is the total number of digits in a number**.** For example,

|  |  |
| --- | --- |
| **Number** | **Precision** |
| 246 | 3 |
| 56.256 | 5 |
| 1.0074256 | 8 |

When we apply a precision of 3 to the value 3.14159265358, using new MathContext(3), we get a result of 3.14.

When precision is applied to a **BigDecimal**, the number may undergo **rounding** (default being [RoundingMode.HALF\_UP](https://docs.oracle.com/javase/8/docs/api/java/math/RoundingMode.html" \l "HALF_UP)) and the value may change. Below table shows how **BigDecimal** rounds-off numbers to given precisions:

|  |  |  |
| --- | --- | --- |
| **Number** | **Precision** | **Rounded Value** |
| 2.34 | 2 | 2.3 |
| 2.36 | 2 | 2.4 |
| 24.6 | 2 | 25 |

In **BigDecimal**, **Scale is the number of digits to the right of the decimal point.** For example, in 3.14159265358, scale is 11

Scale is used to represent the number of digits after the decimal point.

The method **[setScale](https://docs.oracle.com/javase/8/docs/api/java/math/BigDecimal.html" \l "setScale-int-)**(int newScale) assumes that the value does not change after scale adjustment

[BigDecimal.setScale(int newScale)](https://docs.oracle.com/javase/8/docs/api/java/math/BigDecimal.html#setScale-int-)

[RoundingMode.HALF\_UP.](https://docs.oracle.com/javase/8/docs/api/java/math/RoundingMode.html#HALF_UP)

[setScale(int newScale, int roundingMode)](https://docs.oracle.com/javase/8/docs/api/java/math/BigDecimal.html#setScale-int-int-)

[ArithmeticException.](https://docs.oracle.com/javase/8/docs/api/java/lang/ArithmeticException.html)

[RoundingMode](http://docs.oracle.com/javase/7/docs/api/java/math/RoundingMode.html).

[BigDecimal](http://docs.oracle.com/javase/7/docs/api/java/math/BigDecimal.html" \t "_blank)

***# Boolean***

In Java, the boolean data type denotes a logical quantity with two possible values, indicated by the literals true or false.  
  
The primitive type boolean has a corresponding wrapper class called Boolean. Both booleanBoolean can be used interchangeably. Which means, we can say:

Boolean isHot = true;

boolean isSnowing = false;

boolean isSummer = isHot && (!isSnowing);

The fixed values true and false used above are called boolean literals.

The default value of a primitive boolean is false, when not initialized.

the default value of a reference of type Boolean is null, when not initialized.

***# Character***

In Java, the char data type denotes a 16-bit unsigned integer (between 0 and 65535), which represent the [Unicode](http://en.wikipedia.org/wiki/Unicode) values between '**\u0000**' and '**\uffff**' .

The primitive type char has a corresponding wrapper class called Character. Both char and Character can be used interchangeably. Which means, we can say:

char gender1 = 'M';

Character gender2 = gender1;

**A character literal must be wrapped in single quotes and it cannot span multiple lines.**

The fixed value 'M' assigned to gender1 is called character literal.  
The default value of a primitive char is 0, when not initialized. However, the default value of a reference of type Character is null, when not initialized.

**Since char is a 16-bit integer type, it can be interchangeably used with int.**

See and type the below code to understand the usage of most commonly used methods in Character class:

1. isLetter()
2. isDigit()
3. isLetterOrDigit()
4. isUpperCase()
5. isLowerCase()
6. toUpperCase()
7. toLowerCase()
8. isWhitespace()

***# STRING ( non-premetive/reference/object type)***

A String is a sequence of zero or more characters enclosed in double-quotes. For example,

String greetingText = "Hello Thames\nWelcome to London!";

We can embed escape characters (like \t, \n) also in a string literal.

--------------------- --------- ------- Methods --------------- --------- --------------------

The below table shows the **primitive types** and their corresponding **wrapper classes** in Java:

char    Character     **Number Types**

==================================

byte    Byte          Byte.valueOf(String s)

short   Short         Short.valueOf(String s)

int     Integer       Integer.valueOf(String s)

float   Float         Float.valueOf(String s)

double  Double        Double.valueOf(String s)

long    Long          Long.valueOf(String s)

All the above wrapper classes present for the **number types** contain a useful method called valueOf(), which converts a string to an instance of that number type.