The general rules for constructing names for variables (unique identifiers) are:

* Names can contain letters, digits, underscores, and dollar signs
* Names must begin with a letter
* Names should start with a lowercase letter and it cannot contain whitespace
* Names can also begin with $ and \_ (but we will not use it in this tutorial)
* Names are case sensitive ("myVar" and "myvar" are different variables)
* Reserved words (like Java keywords, such as int or boolean) cannot be used as names

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](https://www.w3schools.com/java/java_strings.asp), [Arrays](https://www.w3schools.com/java/java_arrays.asp) and [Classes](https://www.w3schools.com/java/java_classes.asp)

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 9,223,372,036,854,775,807 |
| float | 4 bytes | Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits |
| double | 8 bytes | Stores fractional numbers. Sufficient for storing 15 decimal digits |
| boolean | 1 bit | Stores true or false values |
| char | 2 bytes | Stores a single character/letter or ASCII values |

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.

Integer Types

**Byte**

The byte data type can store whole numbers from -128 to 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.

public class Main {

public static void main(String[] args) {

byte myNum = 100;

System.out.println(myNum);

}

}

**Short**

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

public class Main {

public static void main(String[] args) {

short myNum = 5000;

System.out.println(myNum);

}

}

Int

The int data type can store whole numbers from -2147483648 to 2147483647.  The int data type is the preferred data type when we create variables with a numeric value.

public class Main {

public static void main(String[] args) {

int myNum = 100000;

System.out.println(myNum);

}

}

**Long**

The long data type can store whole numbers from -9223372036854775808 to 9223372036854775807. 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;

public class Main {

public static void main(String[] args) {

long myNum = 15000000000L;

System.out.println(myNum);

}

}

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: public class Main {

public static void main(String[] args) {

float myNum = 5.75f;

System.out.println(myNum);

}

}

Double

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

Example: double myNum = 19.99d;

public class Main {

public static void main(String[] args) {

double myNum = 19.99d;

System.out.println(myNum);

}

}

Booleans

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

Example: public class Main {

public static void main(String[] args) {

boolean isJavaFun = true;

boolean isFishTasty = false;

System.out.println(isJavaFun);

System.out.println(isFishTasty);

}

}

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";

public class Main {

public static void main(String[] args) {

String greeting = "Hello World";

System.out.println(greeting);

}

}

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 is 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. Null is having or associated with the value zero.
* A primitive type starts with a lowercase letter, while non-primitive types start 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 Strings, Arrays, Classes, Interface, etc.

Java Type Casting

Type casting is when you assign a value of one primitive data type to another type.

In Java, there are two types of casting:

* **Widening Casting** (automatically) - converting a smaller type to a larger type size  
  byte -> short -> char -> int -> long -> float -> double

public class Main {

public static void main(String[] args) {

int myInt = 9;

double myDouble = myInt; // Automatic casting: int to double

System.out.println(myInt);

System.out.println(myDouble);

}

}

* **Narrowing Casting** (manually) - converting a larger type to a smaller size type  
  double -> float -> long -> int -> char -> short -> byte

public class Main {

public static void main(String[] args) {

double myDouble = 9.78;

int myInt = (int) myDouble; // Explicit casting: double to int

System.out.println(myDouble);

System.out.println(myInt);

}

}