
# Chapter One

# JAVA BASICS

# What is Java?

Java is a popular programming language.

Java is used to develop mobile apps, web apps, desktop apps, games and much more.

**Install Java:**

Download Java from the official Java web site:

https://www.oracle.com/java/technologies/downloads/#jdk20-windows

To check if you have Java installed on a Windows PC:

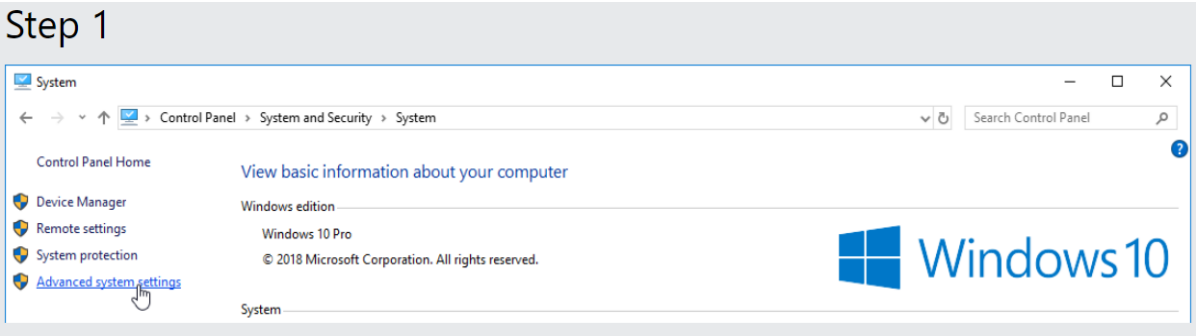
```
C:\Users>java -version
```

If Java is installed, you will see something like this (depending on version):

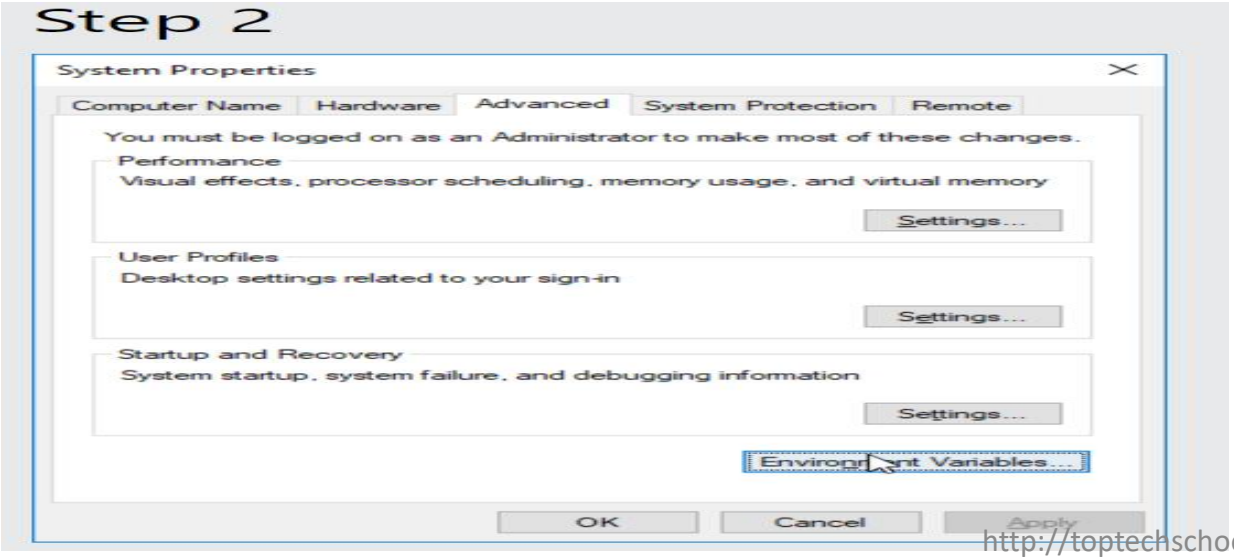
```
java version "11.0.1" 2018-10-16 LTS
Java(TM) SE Runtime Environment 18.9 (build 11.0.1+13-LTS)
Java HotSpot(TM) 64-Bit Server VM 18.9 (build 11.0.1+13-LTS, mixed mode)
```

# To set up Environment Variables on Windows:

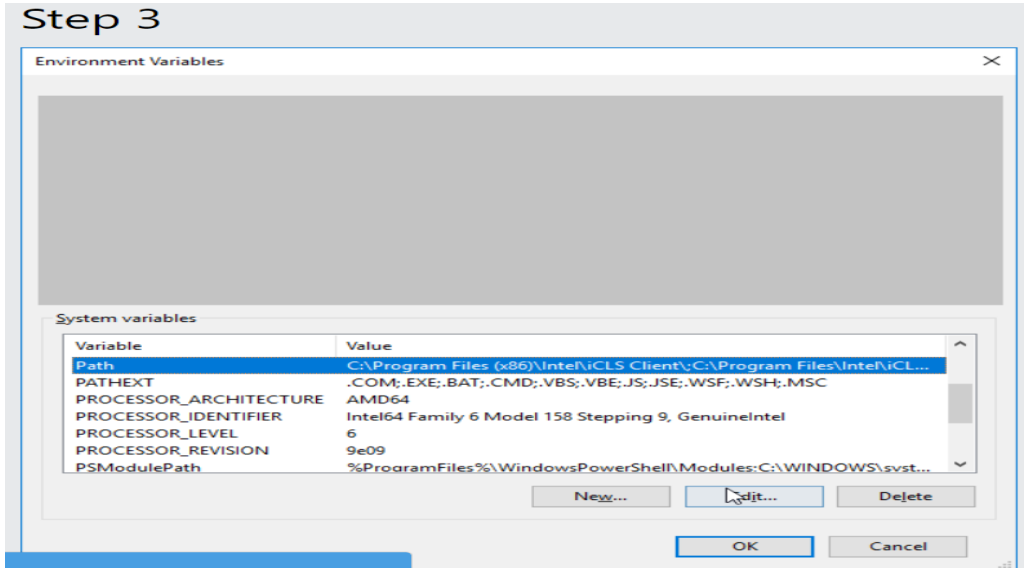
1.Go to "System Properties" (Can be found on Control Panel > System and Security > System > Advanced System Settings)



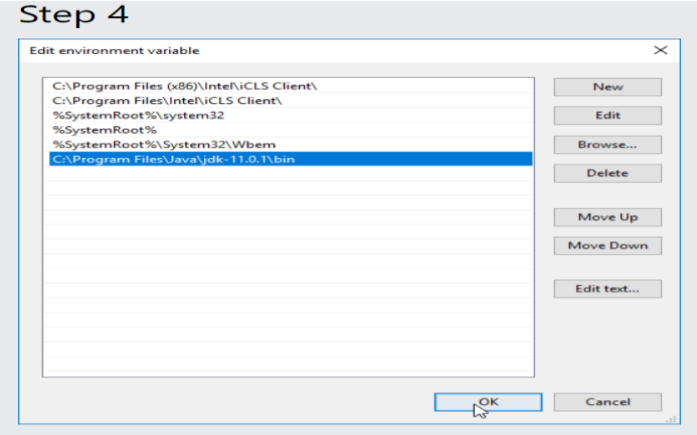

2. Click on the "Environment variables" button under the "Advanced" tab




3.Then, select the "Path" variable in System variables and click on the "Edit" button



4.Click on the "New" button and add the path where Java is installed, followed by **\bin**. By default, Java is installed in C:\Program Files\Java\jdk-11.0.1 (If nothing else was specified when you installed it). In that case, You will have to add a new path with: **C:\Program Files\Java\jdk-11.0.1\bin**
Then, click "OK", and save the settings




5. At last, open Command Prompt (cmd.exe) and type **java -version** to see if Java is running on your machine

```
C:\>echo %JAVA_HOME%
```

```
C:\>echo %PATH%
```

# What Is An Integrated Development Environment (IDE)?

- Integrated development environments (IDE) are applications that facilitates the development of other applications. Designed to encompass all programming tasks in one application, one of the main benefits of an IDE is that they offer a central interface with all the tools a developer needs, including:
- **Code editor:** Designed for writing and editing source code, these editors are distinguished from text editors because work to either simplify or enhance the process of writing and editing of code for developers
- **Compiler:** Compilers transform source code that is written in a human readable/writable language in a form that computers can execute.
- **Debugger:** Debuggers are used during testing and can help developers debug their application programs.
- **Build automation tools:** These can help automate developer tasks that are more common to save time.

**Common Types of IDE used in Software Development:**

**Eclipse**

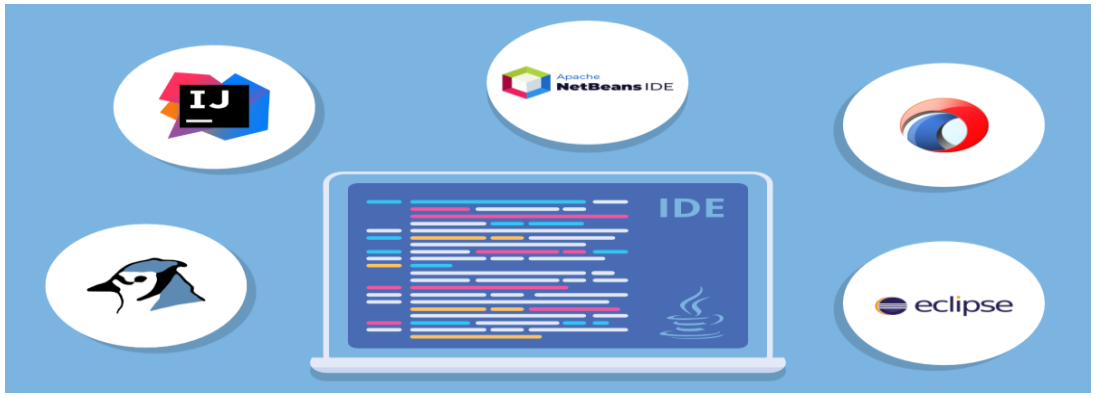
Eclipse is a Java IDE that is one of the 3 biggest and most popular IDE's in the world

**IntelliJ IDEA**

IntelliJ IDEA is a Java IDE that is one of the 3 biggest and most popular IDE's in the world

**NetBeans**

NetBeans is a Java IDE that is one of the 3 biggest and most popular IDE's in the world



**JAVA** was developed by James Gosling at **Sun Microsystems** Inc in the year **1995**, later acquired by Oracle Corporation. It is a simple programming language

- Java is a class-based, object-oriented programming language
- A general-purpose programming language made for developers to *write once run anywhere* that is compiled Java code can run on all platforms that support Java.
- Java applications are compiled to byte code that can run on any Java Virtual Machine.

**Java Terminology**

**Java Virtual Machine(JVM):** This is generally referred to as JVM. There are three execution phases of a program. They are written, compile and run the program.

1. Writing a program is done by a java programmer like you and me.

2. The compilation is done by the **JAVAC** compiler which is a primary Java compiler included in the Java development kit (JDK). It takes the Java program as input and generates bytecode as output.

3. In the Running phase of a program, **JVM** executes the bytecode generated by the compiler.

Every Operating System has a different JVM but the output they produce after the execution of bytecode is the same across all the operating systems. This is why Java is known as a **platform-independent language.**

- **Bytecode in** the **Development process:** As discussed, the Javac compiler of JDK compiles the java source code into bytecode so that it can be executed by JVM. It is saved as **.class** file by the compiler.
- **Java Development Kit(JDK):** While we were using the term JDK when we learn about bytecode and JVM. So, as the name suggests, it is a complete Java development kit that includes everything including compiler, Java Runtime Environment (JRE), java debuggers, java docs, etc. For the program to execute in java, we need to install JDK on our computer in order to create, compile and run the java program.

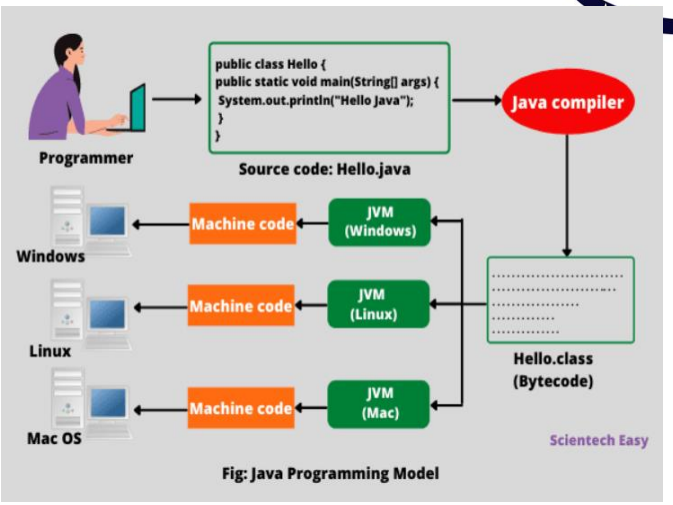
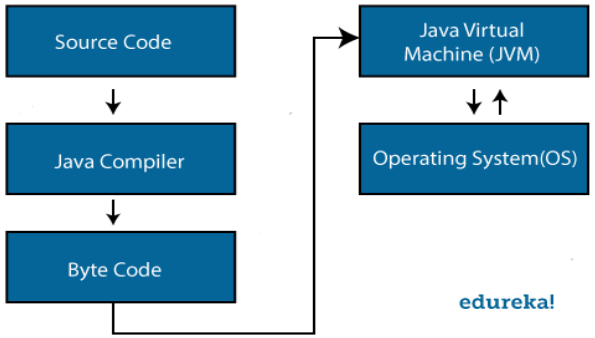
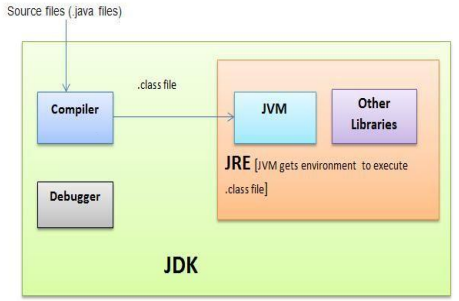


Fig: Java Programming Model



# Working with Java Programing Language

- Open your **Eclipse IDE**
- Create a **project**
- Then create a **package** inside your project
- Then create a **class** or classes inside your package
- Then Create **method** inside your class
- And then **start Coding** ☺!

•In Java, every application begins with a class name, and that class must match the filename.

•Let's create our first Java file, called Basics.java,

```
•public class Basics{
•  public static void main(String[] args) {
•    System.out.println("Hello World");
•  }
•}
```

•Every line of code that runs in Java must be inside a class. In our example, we named the class **Basics**.

•A class should always start with an uppercase first letter.

•**Note:** Java is case-sensitive: "MyClass" and "myclass" has different meaning.

•The name of the java file **must match** the class name. When saving the file, save it using the class name and add ".java" to the end of the filename.

# Package , Class & Methods/Functions in Java

- ***A package*** in Java is used to group related classes. Think of it as **a folder in a file directory**. We use packages to avoid name conflicts, and to write a better maintainable code. Packages are divided into two categories:

- Built-in Packages (packages from the Java API)
- User-defined Packages (create your own packages)

- ***A class*** in Java is a logical template to create **objects** that share common properties and methods.

- A **method** is a block of code which only runs when it is called.

You can pass data, known as parameters, into a method.

Methods are used to perform certain actions, and they are also known as **functions**.

**Why use methods?**

To reuse code: define the code once, and use it many times.

## The main Method

The main() method is required and you will see it in every Java program:

```
public static void main(String[] args)
```

Any code inside the main() method will be executed. Don't worry about the keywords before and after main. You will get to know them bit by bit while proceeding with further sessions.

For now, just remember that every Java program has a class name which must match the filename, and that every program must contain the main() method.

## System.out.println()

•Inside the main() method, we can use the println() method to print a line of text to the screen:

```
public static void main(String[] args) {

  System.out.println("Hello World");

}
```

•**Note:** The curly braces {} marks the beginning and the end of a block of code.

Java Output

•You learned from the previous chapter that you can use the println() method to output values or print text in Java:

**Example**

```
•System.out.println("Hello World!");
```

•You can add as many println() methods as you want. Note that it will add a new line for each method:

Example

```
•System.out.println("Hello World!");

•System.out.println("I am learning Java.");

•System.out.println("It is awesome!");
```

•

•You can also output numbers, and perform mathematical calculations:

**Example**

```
•System.out.println(3 + 3);
```

•Note that we don't use double quotes ("") inside println() to output numbers.

•The Print() Method

•There is also a print() method, which is similar to println().

•The only difference is that it does not insert a new line at the end of the output:

**Example**

```
•System.out.print("Hello World! ");

•System.out.print("I will print on the same line.");
```

•we will only use println() as it makes it easier to read the output of code.

**Java Comments**

Comments can be used to explain Java code, and to make it more readable. It can also be used to prevent execution when testing alternative code.

**Single-line Comments**

Single-line comments start with two forward slashes (//).

Any text between // and the end of the line is ignored by Java (will not be executed).

This example uses a single-line comment before a line of code:

Example

```
// This is a comment

System.out.println("Hello World");

System.out.println("Hello World"); // This is a comment
```

**Multi-line comments** start with /* and ends with */.

Any text between /* and */ will be ignored by Java.

This example uses a multi-line comment (a comment block) to explain the code:

Example

```
•/* The code below will print the words Hello World

•to the screen, and it is amazing */

•System.out.println("Hello World");
```

Single or multi-line comments?

•It is up to you which you want to use. Normally, we use // for short comments, and /* */ for longer.

```
  CTRL + SHIFT + F = code alignment

  CTRL + /  =  single comment (first select lines then use this
shortcut and used for            UNDO as well)

 CTRL + SHIFT + / = multiple lines and use CTRL+Z = to unde it
```

# Variables:

Variables are containers for storing data values.
In Java, there are different **types** of variables, for example:

- String - stores text, such as "Hello". String values are surrounded by double quotes
- int - stores integers (whole numbers), without decimals, such as 123 or -123
- float - stores floating point numbers, with decimals, such as 19.99 or -19.99
- char - stores single characters, such as 'a' or 'B'. Char values are surrounded by single quotes
- boolean - stores values with two states: true or false

**Declaring (Creating) an Initializing Variables**

**Declare a variable**
Int age ;
**Initialize a variable :**
**Int age = 25; or we can say age = 25;**

**Syntax**
*type variableName ;*
*VariableName = value;*

Where *type* is one of Java's types (such as int or String), and *variableName* is the name of the variable (such as **x** or **name**).
The **equal sign** is used to assign values to the variable.

To create a variable that should store text, look at the following example:

Example

Create a variable called **name** of type String and assign it the value "**John**":

```
String name = "John";
System.out.println(name);
```

To create a variable that should store a number, look at the following example:

Example

Create a variable called **myNum** of type int and assign it the value **15**:

```
int myNum = 15;    System.out.println(myNum);
```

You can also declare a variable without assigning the value, and assign the value later:

Example

```
int myNum;
myNum = 15;
System.out.println(myNum);
```

Note that if you assign a new value to an existing variable, it will overwrite the previous value:

Example

Change the value of myNum from 15 to 20:

```
int myNum = 15;
myNum = 20;

// myNum is now 20System.out.println(myNum);
```

## Final Variables

If you don't want others (or yourself) to overwrite existing values, use the final keyword (this will declare the variable as "final" or "constant", which means unchangeable and read-only):

Example

```
final int myNum = 15;


myNum = 20;  // will generate an error: cannot assign a value to a final variable
```

## Other Types

A demonstration of how to declare variables of other types:

Example

```
int myNum = 5;
float myFloatNum = 5.99f;
char myLetter = 'D';
boolean myBool = true;
String myText = "Hello";
```

**The println() method is often used to display variables.**

To combine both text and a variable, use the + character:

Example

```
String name = "John";
System.out.println("Hello " + name);
```

You can also use the + character to add a variable to another variable:

Example

```
String firstName = "John ";
String lastName = "Doe";
String fullName = firstName + lastName;

System.out.println(fullName);
```

For numeric values, the + character works as a mathematical operator (notice that we use int (integer) variables here):

**Example**

```
int x = 5;
int y = 6;
System.out.println(x + y); // Print the value of x + y
```

# Declare Many Variables

**Declare Many Variables**

To declare more than one variable of the **same type**, you can use a comma-separated list:

**Example**

Instead of writing:

```
int x = 5;
int y = 6;
int z = 50;
System.out.println(x + y + z);
```

You can simply write:

```
int x = 5, y = 6, z = 50;
System.out.println(x + y + z);
```

**One Value to Multiple Variables**

You can also assign the **same value** to multiple variables in one line:

**Example**

```
int x, y, z ;
x = y = z = 50;

System.out.println(x + y + z);
```

# Java Identifiers

**Java Identifiers**

All Java **variables** must be **identified** with **unique names**.

These unique names are called **identifiers**.

Identifiers can be short names (like x and y) or more descriptive names (age, sum, total Volume).

**Note:** It is recommended to use descriptive names in order to create understandable and maintainable code:

Example

```
// Good
int minutesPerHour = 60;   // it is good and descriptive.
int m = 60;  // it is right but not descriptive
```

**The general rules for naming variables 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

## 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
```

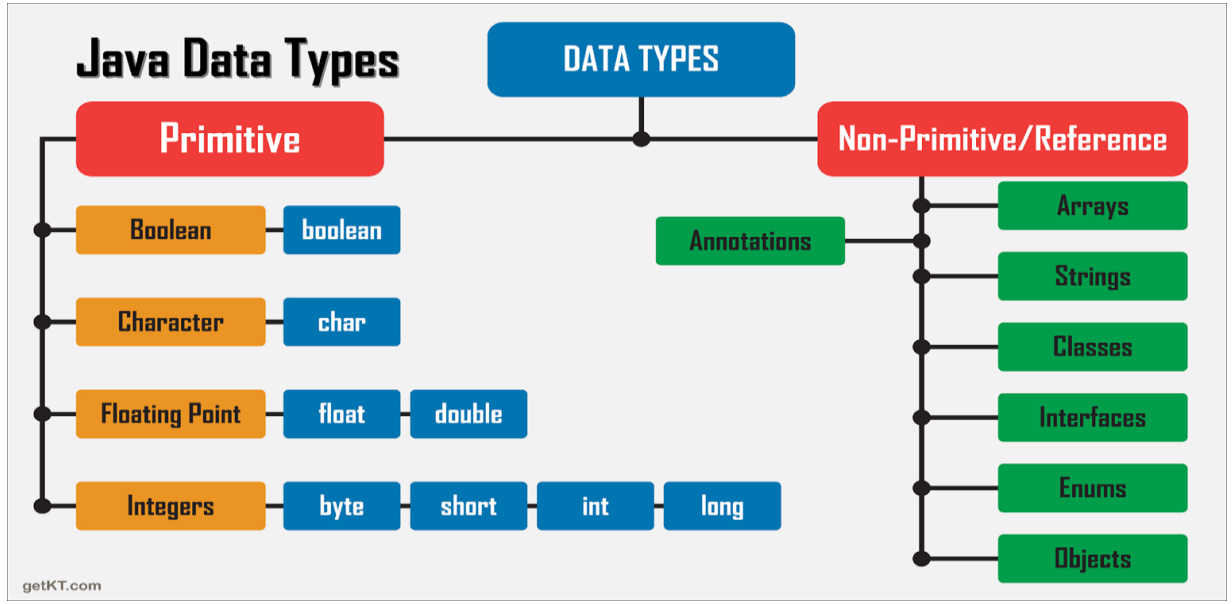
**Data types are divided into two groups:**

- **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 9,223,372,036,854,775,807 |
| 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 |

**Example**

**Byte:**

The byte data type can store whole numbers from -128 to 127. This can be used

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

**Short**

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

**Example**

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

**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);
```

**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;

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.

•The float and double data types can store fractional numbers. Note that you should end the value with an "f" for floats and "d" for doubles:

**Float Example**

```
•float myNum = 5.75f;

•System.out.println(myNum);
```

## Double Example

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

**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);
```

# Java Boolean Types

- 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

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

# Characters

**Characters**

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

Example

•char myGrade = 'B';

•System.out.println(myGrade);

•Alternatively, if you are familiar with ASCII values, you can use those to display certain characters:

Example

•char myVar1 = 65, myVar2 = 66, myVar3 = 67;

•System.out.println(myVar1);

•System.out.println(myVar2);

•System.out.println(myVar3);

## 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);
```

A String in Java is actually a **non-primitive** data type, because it refers to an object.

The String object has methods that are 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.

## Java Non-Primitive Data Types or Reference 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 as too.
- 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 Strings, Arrays, Classes, Interface, etc. You will learn more about these in a later chapter.

# Java Strings

A String variable contains a collection of characters surrounded by double quotes:

**Example**

Create a variable of type String and assign it a value:

```
String greeting = "Hello";
```

**String Length**

A String in Java is actually an object, which contain methods that can perform certain operations on strings.

For example, the length of a string can be found with the length() method:

**Example**

```
String txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
System.out.println("The length of the txt string is: " + txt.length());
```

There are many string methods available, for example toUpperCase() and toLowerCase():

```
String txt = "Hello World";System.out.println(txt.toUpperCase());// Outputs "HELLO WORLD”

System.out.println(txt.toLowerCase());    // Outputs "hello world" (Note: we’ll study more about String methods later)
```

(Note: we’ll study more about String methods later)

# String Concatenation

The + operator can be used between strings to combine them. This is called **concatenation**:

**Example**

```
String firstName = "John";
String lastName = "Doe";
System.out.println(firstName + " " + lastName);
```

Note that we have added an empty text (" ") to create a space between firstName and lastName on print.

You can also use the concat() method to concatenate two strings:

**Example**

```
String firstName = "John ";
String lastName = "Doe";
System.out.println(firstName.concat(lastName));
```

**Adding Numbers and Strings**

***WARNING!***

Java uses the + operator for both addition and conca

Numbers are added. Strings are concatenated.

If you add two numbers, the result will be a number:

**Example**

```
int x = 10;
int y = 20;
int z = x + y;  // z will be 30 (an integer/number)
```

If you add two strings, the result will be a string concatenation:

**Example**

```
String x = "10";  String y = "20";String z = x + y;  // z will be 1020 (a String)
```

If you add a number and a string, the result will be a string concatenation:

Example

```
String x = "10";  int y = 20;  String z = x + y;  // z will be 1020 (a String)
```

# Special Characters

Because strings must be written within quotes, Java will misunderstand this string, and generate an error:

```
String txt = "We are the so-called "Vikings" from the north.";
```

The solution to avoid this problem, is to use the **backslash escape character**.
The backslash (\) escape character turns special characters into string characters:

| Escape character | Result | Description |
|---|---|---|
| \' | ' | Single quote |
| \" | " | Double quote |
| \\ | \ | Backslash |
| \n | New Line | |
| \t | Tab | |
| \b | Backspace | |

The sequence \" inserts a double quote in a string:

**Example**

```
String txt = "We are the so-called \"Vikings\" from the north.";
```

The sequence \' inserts a single quote in a string:

**Example**

```
String txt = "It\'s alright.";
```

The sequence \\ inserts a single backslash in a string:

**Example**

```
String txt = "The character \\ is called backslash.";
```

# 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

```
// create int type variable
    int num = 10;
    System.out.println("The integer value: " + num);

    // convert into double type
    double data = num;
    System.out.println("The double value: " + data);
```

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

**Widening Casting**

Widening casting is done automatically when passing a smaller size type to a larger size type:
**Example**

```
int myInt = 9;
double myDouble = myInt; // Automatic casting: int to double

 System.out.println(myInt);      // Outputs 9
System.out.println(myDouble);   // Outputs 9.0
```

**Narrowing Casting**

Narrowing casting must be done manually by placing the type in parentheses in front of the value:

Example

```
double myDouble = 9.78d;

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

   System.out.println(myDouble);   // Outputs 9.78

   System.out.println(myInt);      // Outputs 9
```

# Java Math

The Java **Math class** has many methods that allows you to perform mathematical tasks on numbers.

Math.max(*x,y*)

The Math.max(*x,y*) method can be used to find the highest value of *x* and *y*:

**Example**

```
Math.max(5, 10);
```

Math.min(*x,y*)

The Math.min(*x,y*) method can be used to find the lowest value of *x* and *y*:

Example

```
Math.min(5, 10);
```

Math.sqrt(*x*)

The Math.sqrt(*x*) method returns the square root of *x*:

Example

```
Math.sqrt(64);
```

Math.abs(*x*)

The Math.abs(*x*) method returns the absolute (positive) value of *x*:

**Examples:**

```
int max =Math.max(40, 90);
  int min = Math.min(30, 50);

System.out.println( max );


System.out.println(min);
```

# Java Booleans

•Very often, in programming, you will need a data type that can only have one of two values, like:

- YES / NO
- ON / OFF
- TRUE / FALSE

•For this, Java has a boolean data type, which can take the values true or false.

•Boolean Values

•A boolean 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
```

However, it is more common to return boolean values from boolean expressions, for conditional testing (see below).

# Boolean Expression

A **Boolean expression** is a Java expression that returns a Boolean value: true or false.
You can use a comparison operator, such as the **greater than** (>) operator to find out if an expression (or a variable) is true:

**Example**

```
int x = 10;   int y = 9;        System.out.println(x > y); // returns true, because 10 is higher than 9
```

Or even easier:
**Example**

```
System.out.println(10 > 9); // returns true, because 10 is higher than 9
```

In the examples below, we use the **equal to** (==) operator to evaluate an expression:

**Example**

```
int x = 10;System.out.println(x == 10); // returns true, because the value of x is equal to 10
```

**Example**

```
System.out.println(10 == 15); // returns false, because 10 is not equal to 15
```

The Boolean value of an expression is the basis for all Java comparisons and conditions.
You will learn more about conditions in the next chapter.

## Java Operators

Operators are used to perform operations on variables and values.

In the example below, we use the + **operator** to add together two values:

**Example**

```
int x = 100 + 50;
```

Although the + operator is often used to add together two values, like in the example above, it can also be used to add together a variable and a value, or a variable and another variable:

**Example**

```
int sum1 = 100 + 50;        // 150 (100 + 50)
int sum2 = sum1 + 250;      // 400 (150 + 250)
int sum3 = sum2 + sum2;     // 800 (400 + 400)
```

*Java divides the operators into the following groups:*

- Arithmetic operators
- Assignment operators
- Comparison operators
- Logical operators
- Bitwise operators