Java

What is Java?

- It is an object-oriented language similar to C++, but with advanced and simplified features. Java is free to access and can run on all platforms.
- Java is a write-once, run-anywhere programming language developed by Sun Microsystems. It is similar to C and C++ but a lot easier. You can combine Java with a lot of technologies like Spring, node js, Android, Hadoop, J2EE, etc… to build robust, scalable, portable and distributed full- fledged applications. Java also promotes continuous integration and testing using tools like Selenium.

- **Object:** Object is Real world Physical entity.
- Anything which has Physical presence or Physical appearance can be considered as an **Object.** Object has **States** and **Behavior.**
- **State:** States of an Object is nothing but Property, Features, Data or an Information which describes what an Object is.
- State $\Rightarrow$ Information/Data $\Rightarrow$ Variable(Data-member)
- **Behavior:** Behavior of an Object represents **Action / Work** performed by an Object.
- Behavior $\Rightarrow$ Action/Work $\Rightarrow$ Method

History and features of Java

- Java was originally developed by James Gosling with his colleagues at Sun Microsystems during the early 1990s. Initially, it was called a project 'Oak' which had implementation similar to C and C++. The name Java has later selected after enough brainstorming and is based on the name of an espresso bean. Java 1.0, the first version was released in 1995 with the tagline of 'write once, run anywhere'. Later, Sun Microsystems was acquired by Oracle. From there, there has been no looking back.

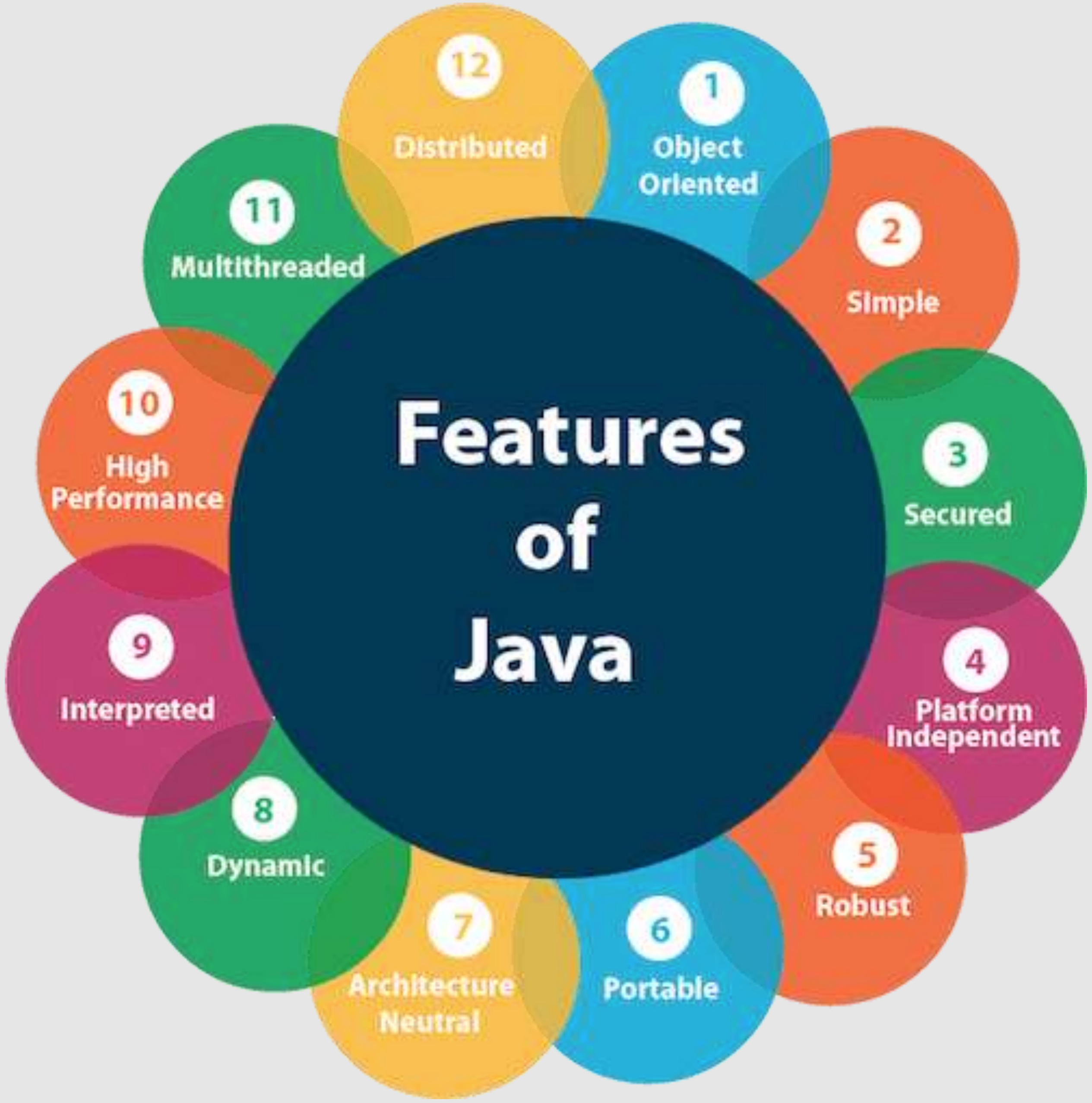
History and features of Java

- Oracle Corporation is the current owner of the official implementation of the Java SE platform, following their acquisition of Sun Microsystems on January 27, 2010. This implementation is based on the original implementation of Java by Sun. The Oracle implementation is available for Microsoft Windows, Mac OS X, Linux and Solaris.

Features of Java

- **Simple:** Java has made life easier by removing all the complexities such as pointers, operator overloading as you see in C++ or any other programming language.
- **Portable:** Java is platform independent which means that any application written on one platform can be easily ported to another platform.
- **Object-oriented:** Everything is considered to be an "object" which possess some state, behavior and all the operations are performed using these objects.
- **Dynamic:** It has the ability to adapt to an evolving environment which supports dynamic memory allocation due to which memory wastage is reduced and performance of the application is increased.

- **Robust:** Java has a strong memory management system. It helps in eliminating error as it checks the code during compile and runtime.
- **Interpreted:** Java is compiled to bytecodes, which are interpreted by a Java run-time environment.
- **Multithreaded:** The Java platform is designed with multithreading capabilities built into the language. That means you can build applications with many concurrent threads of activity, resulting in highly interactive and responsive applications.
- **Platform Independent:** Java code is compiled into intermediate format (bytecode), which can be executed on any systems for which Java virtual machine is ported. That means you can write a Java program once and run it on Windows, Mac, Linux or Solaris without re-compiling. Thus the slogan "Write once, run anywhere" of Java.

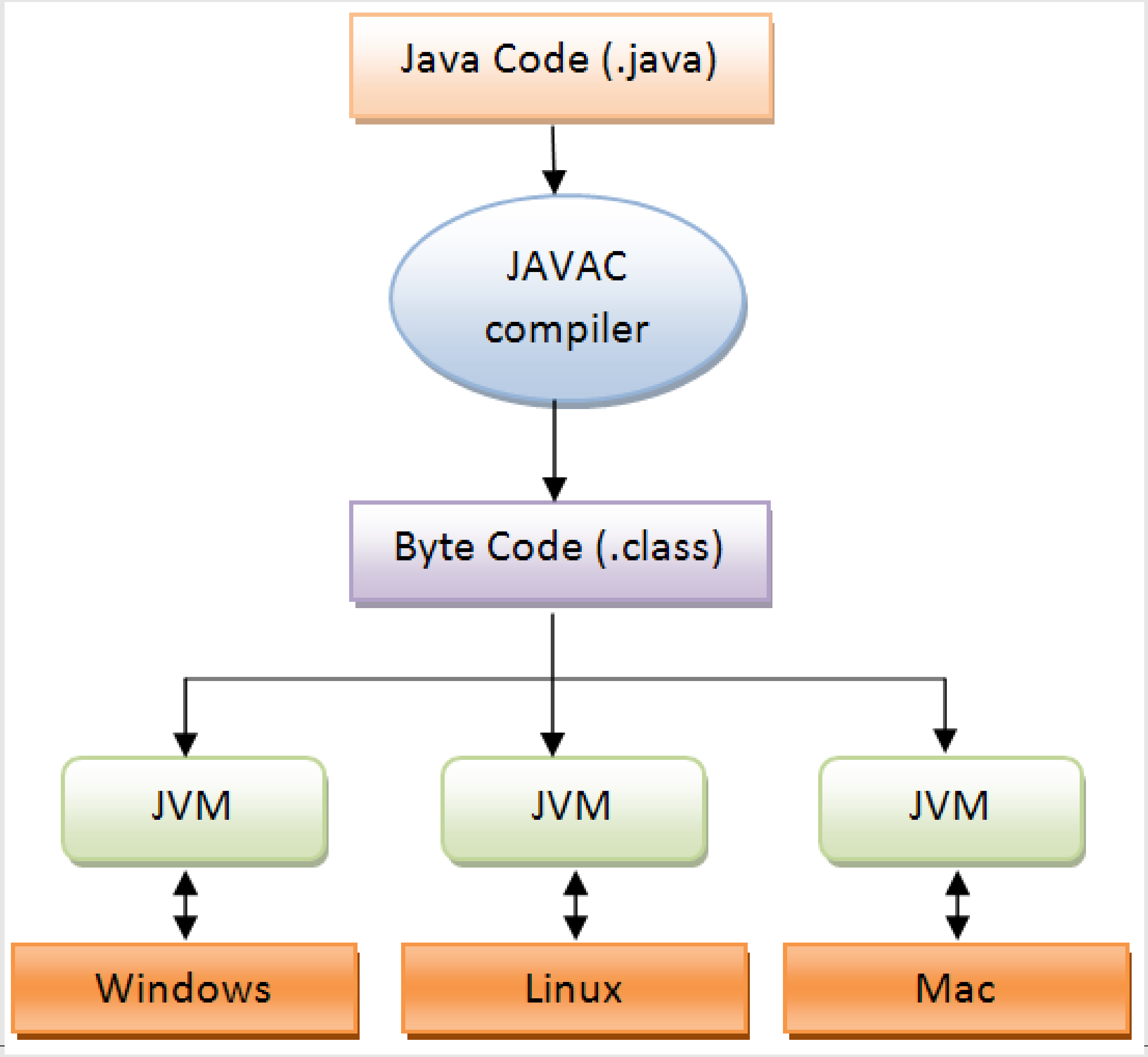

Features of Java
1 Object Oriented
2 Simple
3 Secured
4 Platform Independent
5 Robust
6 Portable
7 Architecture Neutral
8 Dynamic
9 Interpreted
10 High Performance
11 Multithreaded
12 Distributed

JVM(Java Virtual Machine)

- JVM (Java Virtual Machine) is an abstract machine. It is called a virtual machine because it doesn't physically exist. It is a specification that provides a runtime environment in which Java bytecode can be executed. It can also run those programs which are written in other languages and compiled to Java bytecode.

JVM(Java Virtual Machine)

- JVMs are available for many hardware and software platforms. JVM, JRE, and JDK are platform dependent because the configuration of each OS is different from each other. However, Java is platform independent. There are three notions of the JVM: specification, implementation, and instance.
- The JVM performs the following main tasks:
- Loads code
- Verifies code
- Executes code
- Provides runtime environment


Java Code (.java)
JAVAC compiler
Byte Code (.class)
JVM
JVM
JVM
Windows
Linux
Mac

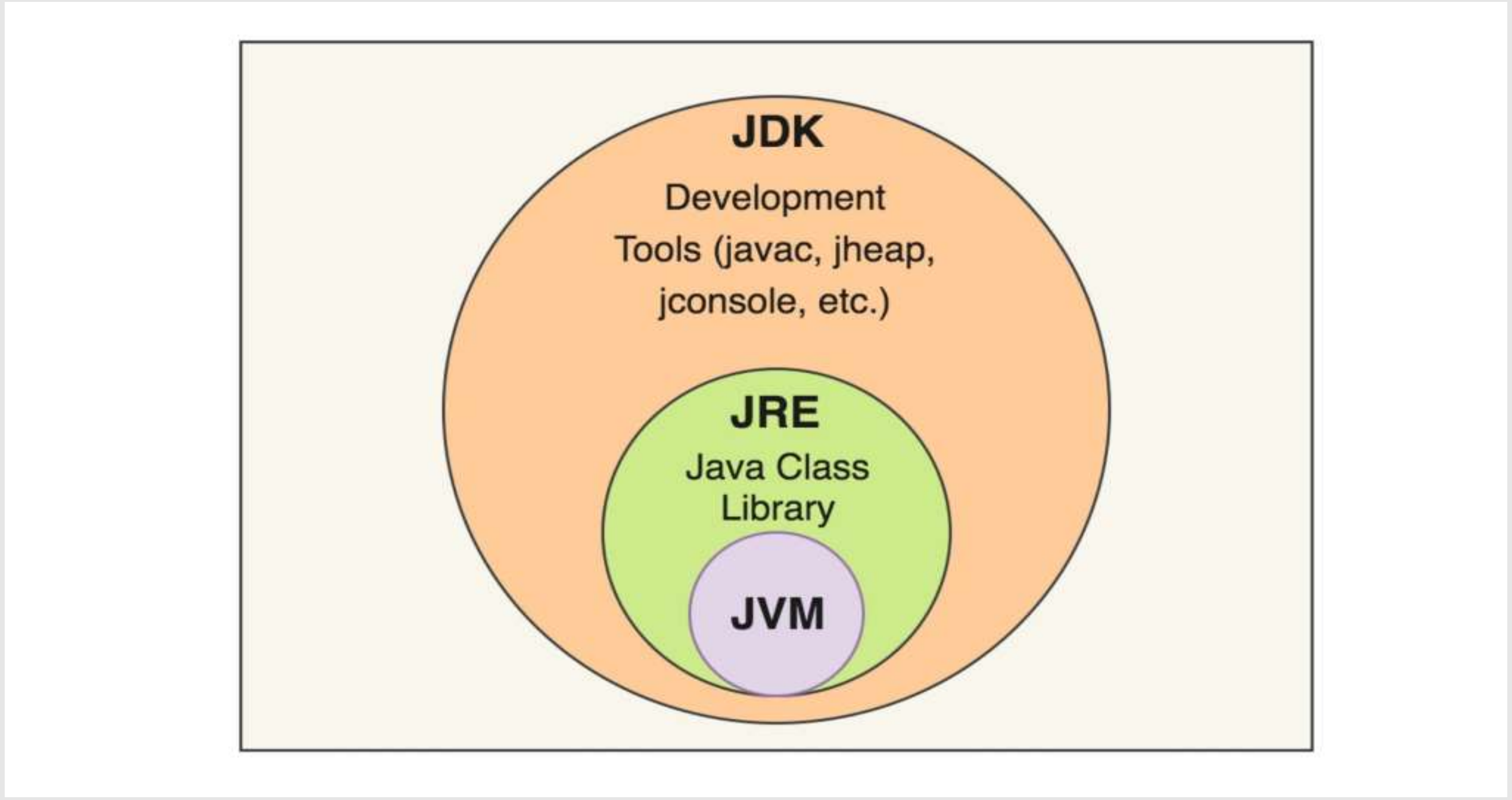
JRE(Java Runtime Environment)

- JRE is an acronym for Java Runtime Environment. It is also written as Java RTE. The Java Runtime Environment is a set of software tools which are used for developing Java applications. It is used to provide the runtime environment. It is the implementation of JVM. It physically exists. It contains a set of libraries + other files that JVM uses at runtime.
-
- The implementation of JVM is also actively released by other companies besides Sun Micro Systems.

JDK(Java Development Kit)

- JDK is an acronym for Java Development Kit. The Java Development Kit (JDK) is a software development environment which is used to develop Java applications and applets. It physically exists. It contains JRE + development tools.
- JDK is an implementation of any one of the below given Java Platforms released by Oracle Corporation:
  - Standard Edition Java Platform
  - Enterprise Edition Java Platform
  - Micro Edition Java Platform

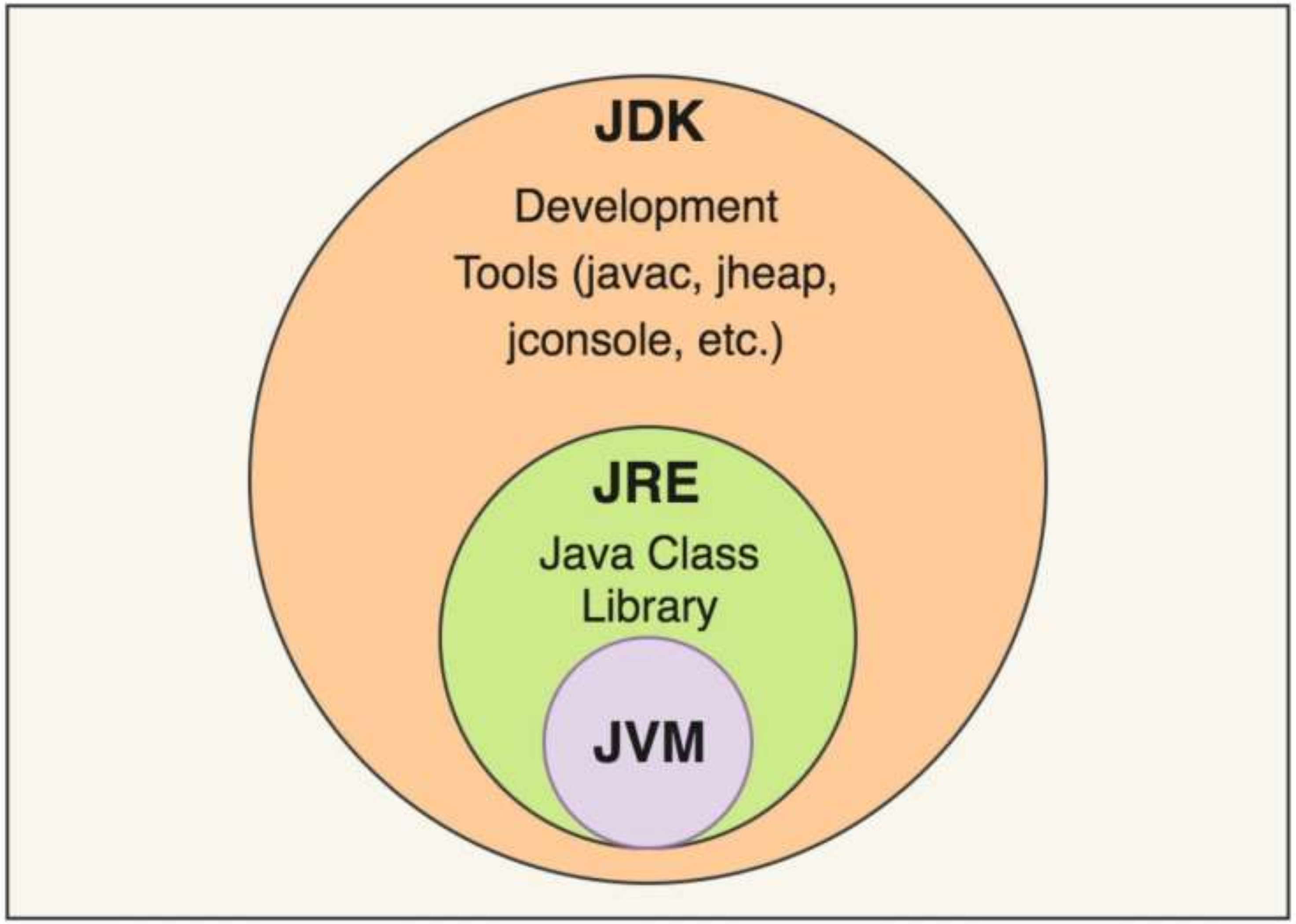
- The JDK contains a private Java Virtual Machine (JVM) and a few other resources such as an interpreter/loader (java), a compiler (javac), an archiver (jar), a documentation generator (Javadoc), etc. to complete the development of a Java Application.



- **Inside Java Development Kit**
- We can write the code.
- Inside jdk we have compiler.(javac)

 -When we save java program with (.java) file, compiler convert into (.class) file

- With the help of jdk we can debug our code as well.
- JDK = JRE+(Programming language, compiler, debugger etc.)


JDK
Development Tools (javac, jheap, jconsole, etc.)
JRE
Java Class Library
JVM

Setting UP Java Environment

Windows:

- Right-click on **This PC** or **My Computer**, then click **Properties**.
- Go to **Advanced System Settings → Environment Variables**.
- Under **System Variables**, click **New**.
- Enter JAVA_HOME as the variable name and the JDK installation path

 (e.g., C:\Program Files\Java\jdk-<version>) as the value.

- In the **Environment Variables** window, find
- the Path variable under **System Variables**.
- Click **Edit**, then **New**, and add the bin
- directory of the JDK
- (e.g., C:\Program Files\Java\jdk-<version>\bin).
- Open a terminal or command prompt.
- Type the command:

```
java -version
```

macOS/Linux

◦ Open a terminal.
◦ Edit the ~/.bashrc, ~/.zshrc, or ~/.bash_profile file.
◦ Add the line
 “xport JAVA_HOME=/path/to/jdk”
◦ Add the following line to your shell profile-
(~/.bashrc or ~/.zshrc)

“export PATH=$JAVA_HOME/bin:$PATH”

◦ Save and reload the profile using
(~/.bashrc or ~/.zshrc)
◦ Test the javac Compiler
 “javac –version”

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

- public: Makes the method accessible to the JVM.
- static: Allows the JVM to invoke it without creating an object.
- void: Indicates no return value.
- main: The name that the JVM recognizes as the entry point.
- Sring[] args: Lets the program accept and process command-line arguments.

- **Public**
- Access Modifier: The public keyword makes the method accessible from anywhere.
- Why is it public? The Java Virtual Machine (JVM) needs to access this method from outside the class to execute the program.
- **Static**
- Why is it static? The JVM calls the main method without creating an object of the class.
- Making it static ensures it can be invoked directly using the class name.
- **void**
- Why is it void ? The main method is designed to be the starting point of the program, not to return any value to the JVM.
- **Main**
- why is it main? It's a convention in Java for starting the execution of a program.
- **String[] args**
- String[]: Denotes an array of strings.
- args: The name of the array (it can be any valid identifier, but args is a convention).

Run Java Code in CMD

- Step-1: Go to your File Location __ C:\Users\Admin>cd desktop

- Step-2: Go inside your File__ C:\Users\Admin\Desktop>cd java

- Step-3 Write your Program File name__
  C:\Users\Admin\Desktop\java>javac Hello.java

- Step-4 then write java with Class Name___
  C:\Users\Admin\Desktop\java>java Hello
- Hello World

Command Line Arguments

- The java command-line argument is an argument i.e. passed at the time of running the java program.
- The arguments passed from the console can be received in the java program and it can be used as an input

```
class CommandLineExample{
public static void main(String args[]){
System.out.println("Your first argument is: "+args[0]);
 }
}
```

Identifiers in Java

- Identifiers is the one which is used to Identify out many class , We can Identify a class by it’s Name , Hence class name is called **Identifier.**
- Identifiers in Java are symbolic names used for identification. It can be a class name, variable name, method name, package name, constant name, and other names used in programming. However, in Java,
- some reserved words cannot be used as an identifier.

Importance of Identifiers

- **Programme Structure:** Classes, variables, methods, packages, and other things are named by identifiers that are the fundamental building blocks of Java programs.
- **Readability and Understanding:** When well-chosen identifiers improve the readability and comprehensibility of code, developers can more easily comprehend the function and goal of various program components.

- **Modifiability:** By offering a simple method of referencing and altering program elements, identifiers aid in code maintenance and change.
- **Communication:** Identifiers allow developers to communicate with each other by sharing details about the purpose and capabilities of certain program pieces.

Rules for Java Identifiers

- A valid identifier must have characters [A-Z] or [a-z] or numbers [0-9], and underscore (_) or a dollar sign ($). For example, @test is not a valid identifier because it contains a special character @.
- There should not be any space in an identifier. For example, "java test" is an invalid identifier.
- An identifier should not contain a number at the starting. For example, 123javatpoint is an invalid identifier.
- An identifier should be of length 4-15 letters only. However, there is no limit on its length. But, it is good to follow the standard conventions.

- We cannot use the Java reserved keywords as an identifier such as int, float, double, char, etc. For example, int double is an invalid identifier in Java.
-

  An identifier should not be any query language keywords such as SELECT, FROM, COUNT, DELETE, etc.

Naming Conventions

- **Classes:** Class names should start with an uppercase letter and follow the CamelCase convention. For example, Test.
- **Variables:** Variable names should start with a lowercase letter and follow the camelCase convention. For example, testVariable.
- **Methods:** Method names should start with a lowercase letter and follow the camelCase convention. For example, testMethod.

- **Packages:** Package names should be in lowercase letters and follow a reverse domain name convention. For example, com.example.package.

<u>Keywords or Reserved Word</u>

class , interface , abstract , import , package , static , public , private , protected , default , super, this , extends , implements , try , catch , finally , break , continue , int , double , float , if , else , switch , etc....

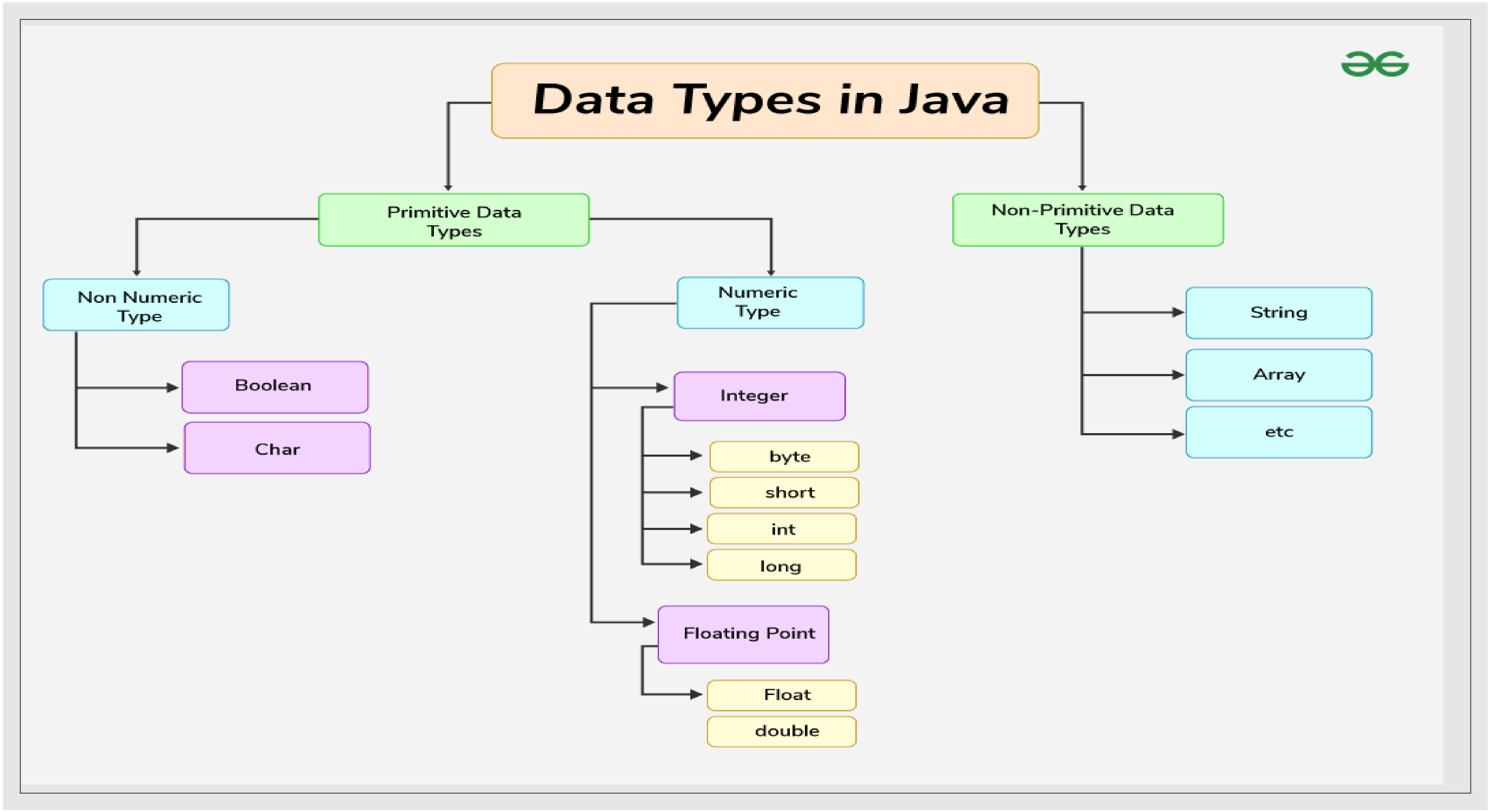
*Java identifiers are case-sensitive.

- Which of the following are valid declaration in java?
- A- int int=10;
- B- int Int=10;
- C- String string=“Group”;
- D- String String=“Group”;
- E- class class=Test.class;

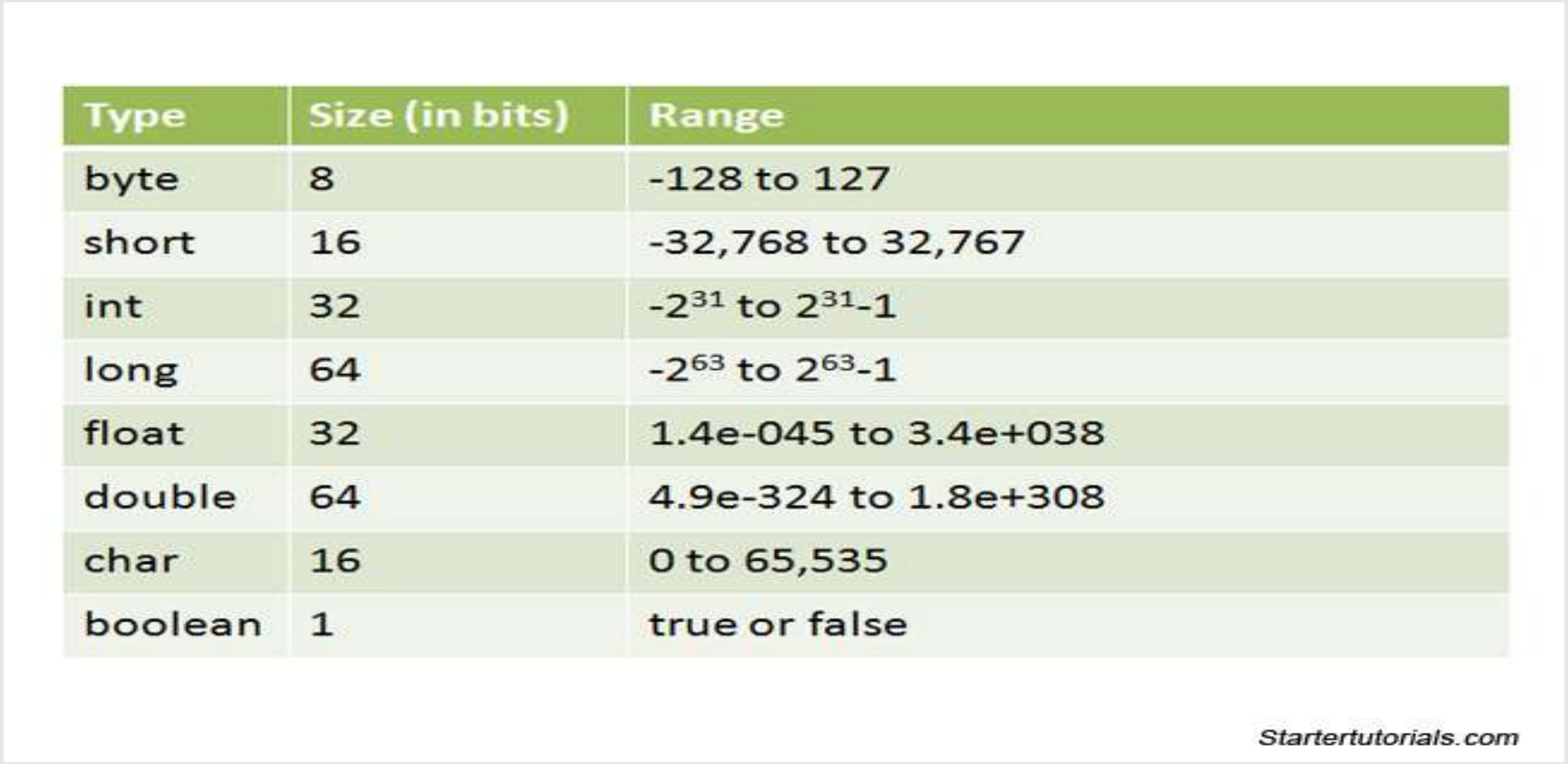
- Answer is B C D
- Explanation:
- Int int =10; ==➔ invalid because int is a keyword
- class class= Test.class: ==➔ invalid because class is a keyword.
- But all the predefine class names can be used as identifiers.

Data Types in Java

- Data types specify the different sizes and values that can be stored in the variable. There are two types of data types in Java:
- **Primitive data types:** The primitive data types include boolean, char, byte, short, int, long, float and double.
- **Non-primitive data types:** The non-primitive data types include Classes, Interfaces, and Arrays.
- **Java Primitive Data Types**
- In Java language, primitive data types are the building blocks of data manipulation. These are the most basic data types available in Java language.


Data Types in Java
Primitive Data Types
Non-Primitive Data Types
Non Numeric Type
Numeric Type
Boolean
Char
Integer
byte
short
int
long
Floating Point
Float
double
String
Array
etc

- **There are 8 types of primitive data types:**
- boolean data type
- byte data type
- char data type
- short data type
- int data type
- long data type
- float data type
- double data type



Type	Size (in bits)	Range
byte	8	-128 to 127
short	16	-32,768 to 32,767
int	32	-2^{31} to $2^{31}-1$
long	64	-2^{63} to $2^{63}-1$
float	32	1.4e-045 to 3.4e+038
double	64	4.9e-324 to 1.8e+308
char	16	0 to 65,535
boolean	1	true or false

Startertutorials.com

- **Boolean Data Type**
- The Boolean data type is used to store only two possible values: true and false. This data type is used for simple flags that track true/false conditions.
- The Boolean data type specifies one bit of information, but its "size" can't be defined precisely.
- **Example:** Boolean one = false.
- **Byte Data Type**
- The byte data type is an example of primitive data type. It is an 8-bit signed two's complement integer. Its value-range lies between -128 to 127 (inclusive). Its minimum value is -128 and maximum value is 127. Its default value is 0.
- The byte data type is used to save memory in large arrays where the memory savings is most required. It saves space because a byte is 4 times smaller than an integer. It can also be used in place of "int" data type.
- **Example:** byte a = 10, byte b = -20

- **Short Data Type**
- The short data type is a 16-bit signed two's complement integer. Its value-range lies between - 32,768 to 32,767 (inclusive). Its minimum value is -32,768 and maximum value is 32,767. Its default value is 0.
- The short data type can also be used to save memory just like byte data type. A short data type is 2 times smaller than an integer.
- **Example:** short s = 10000, short r = -5000
- **Int Data Type**
- The int data type is a 32-bit signed two's complement integer. Its value-range lies between - 2,147,483,648 (-2^31) to 2,147,483,647 (2^31 -1) (inclusive). Its minimum value is - 2,147,483,648and maximum value is 2,147,483,647. Its default value is 0.
- The int data type is generally used as a default data type for integral values unless if there is no problem about memory.
- **Example:** int a = 100000, int b = -200000

- The long data type is a 64-bit two's complement integer. Its value-range lies between -9,223,372,036,854,775,808(-2^63) to 9,223,372,036,854,775,807(2^63 -1)(inclusive). Its
- minimum value is - 9,223,372,036,854,775,808and maximum value is 9,223,372,036,854,775,807. Its default value is 0. The long data type is used when you need a range of values more than those provided by int.
- **Example:** long a = 100000L, long b = -200000L
- **Float Data Type**
- The float data type is a single-precision 32-bit IEEE 754 floating point.Its value range is unlimited. It is recommended to use a float (instead of double) if you need to save memory in large arrays of floating point numbers. The float data type should never be used for precise values, such as currency. Its default value is 0.0F.
- **Example:** float f1 = 234.5f

- **Double Data Type**
- The double data type is a double-precision 64-bit IEEE 754 floating point. Its value range is unlimited. The double data type is generally used for decimal values just like float. The double data type also should never be used for precise values, such as currency. Its default value is 0.0d.
- **Example:** double d1 = 12.3
- **Char Data Type**
- The char data type is a single 16-bit Unicode character. Its value-range lies between '\u0000' (or 0) to '\uffff' (or 65,535 inclusive).The char data type is used to store characters.
- **Example:** char letterA = ‘A’
- Primitive variable cannot store Object address , only a non primitive variable can store Object address.

- **Non-Primitive Datatypes**
- Non-Primitive data types refer to objects and hence they are called reference types. Examples of non-primitive types include Strings, Arrays, Classes, Interface,
- **Strings:** String is a sequence of characters. But in Java, a string is an object that represents a sequence of characters. The java.lang.String class is used to create a string object.
- **Arrays:** Arrays in Java are homogeneous data structures implemented in Java as objects. Arrays store one or more values of a specific data type and provide indexed access to store the same. A specific element in an array is accessed by its index.

- **Classes:** A class in Java is a blueprint which includes all your data. A class contains fields(variables) and methods to describe the behavior of an object.
- **Interface:** Like a class, an interface can have methods and variables, but the methods declared in interface are by default abstract (only method signature, no body).
- Example:
- class Bike {
- Bike b = new Bike ( );
- }
- Here Bike is non-primitive data type.

Operators in java

- **Operator** in Java is a symbol that is used to perform operations. For example: +, -, *, / etc.
- There are many types of operators in Java which are given below:

- **Arithmetic Operators**
- **Unary Operators**
- **Assignment Operator**
- **Relational Operators**
- **Logical Operators**
- **Ternary operator**
- **Bitwise Operators**
- **Shift Operators**

- **Arithmetic Operators**
- They are used to perform simple arithmetic operations on primitive and non-primitive data types.
- *** :** Multiplication
- **/ :** Division
- **% :** Modulo
- **+ :** Addition
- **– :** Subtraction

Unary Operators

- Unary operators need only one operand. They are used to increment, decrement, or negate a value.
- **– : Unary minus**, used for negating the values.
- **+ : Unary plus** indicates the positive value (numbers are positive without this, however).
- **++ : Increment operator**, used for incrementing the value by 1. There are two varieties of increment operators.
- **Post-Increment:** Value is first used for computing the result and then incremented.
- **Pre-Increment:** Value is incremented first, and then the result is computed.

- **– – : Decrement operator**, used for decrementing the value by 1. There are two varieties of decrement operators.
- **Post-decrement:** Value is first used for computing the result and then decremented.
- **Pre-Decrement: The value** is decremented first, and then the result is computed.
- **! : Logical not operator**, used for inverting a boolean value.

```
class Test{
public static void main(String[] args){
int a = 10;
int b = 10;
System.out.println("Postincrement : " + (a++));
        System.out.println("Preincrement : " + (++a));

        System.out.println("Postdecrement : " + (b--));
        System.out.println("Predecrement : " + (--b));
}
}
```

- **Post-Increment(++)**
- **Post-increment** means the value is used **first** and then incremented.
- System.out.println("Postincrement : " + (a++));
- a is currently 10 ,so the output is 10.
- After this statement, a become 11.
- **Pre-Increment (++a)**
- **Pre-increment** means the value is incremented **first**, then used.
- System.out.println("Preincrement : " + (++a));
- a was incremented earlier to 11.
- It is incremented again to 12.
- The output is 12.

Assignment Operator

- The assignment operator is used in Java to assign a value to a variable. It assigns the result of the right-hand side expression to the left-hand side variable.
- **Syntax-** variable = value/expression;
- instead of **a = a+5**, we can write **a += 5.**
- **+=**, for adding the left operand with the right operand and then assigning it to the variable on the left.
- **-=**, for subtracting the right operand from the left operand and then assigning it to the variable on the left.
- ***=**, for multiplying the left operand with the right operand and then assigning it to the variable on the left.
- **/=**, for dividing the left operand by the right operand and then assigning it to the variable on the left.
- **%=**, for assigning the modulo of the left operand by the right operand and then assigning it to the variable on the left.

- **Initial Value**:
- a=10;
- **a += 3**:
- Operation: a= a+3
- Calculation: 10+3=13(output)
- **a -= 2**
- Operation: a = a – 2
- Calculation: 13 - 2=11(output)
- **a *= 4**
- Operation: a = a * 4
- Calculation: 11 * 4=44(output)

Relational Operators

◦ Relational operators are used to compare two values or expressions. They return a boolean result (true or false)

based on the relationship between the operands.

List of Relational Operators

== Checks if two values are equal. (a==b).

!= Checks if two values are not equal. (a != b).

> Checks if the left operand is greater than the right operand. (a > b).

< Checks if the left operand is less than the right operand. (a < b).

>= Checks if the left operand is greater than or equal to the right operand. (a >= b).

<= Checks if the left operand is less than or equal to the right operand. (a <= b).

Logical Operators

- Logical operators are used to combine multiple boolean expressions or values into a single boolean result. They are primarily used in decision-making statements (e.g., If, while) to control the flow of a program.
- Conditional operators are**:**
- **&&, Logical AND:** returns true when both conditions are true.
- ||, **Logical OR:** returns true if at least one condition is true.
- **!, Logical NOT:** returns true when a condition is false and vice-versa

Ternary operator

- The ternary operator is a shorthand version of the if-else statement. It has three operands and hence the name Ternary.
- **Syntax** : condition ? value_if_true : value_if_false;
- Condition: A boolean expression that is evaluated.
- value_if_true: The value returned if the condition is true.
- value_if_false: The value returned if the condition is false.
- **How it Works**
- The ternary operator checks the condition.
- If condition is true, it evaluate and returns value_if_true.
- If condition is false, it evaluate and returns value_if_false.

Bitwise Operators

- These operators are used to perform the manipulation of individual bits of a number. They can be used with any of the integer types. They are used when performing update and query operations of the Binary indexed trees.
- **&, Bitwise AND operator:** returns bit by bit AND of input values.
- **|, Bitwise OR operator:** returns bit by bit OR of input values.
- **^, Bitwise XOR operator:** returns bit-by-bit XOR of input values.
- **~, Bitwise Complement Operator:** This is a unary operator which returns the one's complement representation of the input value, i.e., with all bits inverted.

```
◦ class Test{
◦ public static void main(String[] args)
◦     {
◦ int a = 5; // Binary: 0101
◦ int b = 3; // Binary: 0011
          System.out.println(" a & b : " + (a & b));
◦         System.out.println(" a | b : " + (a | b));
◦         System.out.println(" a ^ b : " + (a ^ b));
◦         System.out.println("~a: " + (~a));
◦ }
}
```

- **a** & **b** (Bitwise AND)
- Compares each bit of **a** and **b** he result is 1 f both bits are **1**, otherwise **0**
- **Calculation**: 0101 (a) & 0011 (b) **0001** (Result)- 1
- **a | b** (Bitwise OR)
- Compares each bit of **a** and **b** he result is **1** if at least one bit is **1**, otherwise 0.
- **Calculation**: 0101 (a) | 0011 (b) 0111 (Result)- 7.
- **a ^ b** (Bitwise XOR)
- Compares each bit of **a** and **b** he result is 1 if the bits are **different**, otherwise **0**.
- **Calculation**: 0101 (a) **^** 0011 (b) 0110 (Result)- 6.

- **~a** (Bitwise Complement)
- Inverts all bits of a (turns **1** to **0** and **0** to **1**).
- In Java, integers are stored as 32-bit signed numbers, so the result includes the sign bit (two's complement representation).
- **Calculation**: a=5.
- a = 5 in binary (32-bit): 00000000 00000000 00000000 00000101
- ~a (complement):   11111111 11111111 11111111 11111010.
- In two's complement, this represents -6

Shift Operators

- Shift operators in Java are used to shift bits of a number to the left or right. These operators work at the **bit level** and are used primarily for low-level programming, such as cryptography or optimizing computations.

Types of Shift Operators.

1. Left Shift(<<)
2. Right Shift(>>)
3. Unsigned Right Shift(>>>)

Left Shift(<<)

- Shifts bits to the left by the specified number of positions.
- Fills the empty bits on the **right with 0**.
- Effectively multiplies the number by 2^n (where n is the number of positions shifted).

Right Shift(>>)

- Shifts bits to the right by the specified number of positions.
- Preserves the **sign bit** (0 for positive, 1 for negative numbers).
- Fills the empty bits on the **left with the sign bit**.
- Effectively divides the number by ^{2n} (ignoring the remainder).

Unsigned Right Shift(>>>)

- Shifts bits to the right by the specified number of positions.
- Does **not preserve the sign bit** (always fills with **0** on the left).
- Used for unsigned numbers (works the same as **>>** for positive numbers).

Left Shift (a<<1)

- a = 00000000 00000000 00000000 00000101 // Binary for 5.
- a << 1 = 00000000 00000000 00000000 00001010 // Left shift by 1.

Right Shift(a>>1)

- a = 00000000 00000000 00000000 00000101 // Binary for 5
- a >> 1 = 00000000 00000000 00000000 00000010 // Right shift by 1

Unsigned Right Shift(b>>>1)

- b = 11111111 11111111 11111111 11110110 // Binary for -10
- b >>> 1 = 01111111 11111111 11111111 11111011 // Unsigned right shift by 1