AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

CHAPTER 1
INTRODUCTION TO JAVA AND ELEMENTARY PROGRAMMING

SUBJECT:OOP-I CODE:3140705	PREPARED BY: ASST.PROF.NENSI KANSAGARA (CSE DEPARTMENT,ACET)	AMIRAJ COLLEGE OF ENGINEERING & TECHNOLOGY

JAVA LANGUAGE SPECIFICATION API,JDK & IDE.

- *Java syntax is defined in the Java language specification, and the Java library is defined in the Java application program interface (API). The JDK is the software for compiling and running Java programs. An IDE is an integrated development environment for rapidly developing programs.*
- Computer languages have strict rules of usage. If you do not follow the rules when writing a program, the computer will not be able to understand it. The Java language specification and the Java API define the Java standards.
- The Java language specification is a technical definition of the Java programming language’s syntax and semantics. You can find the complete Java language specification at docs.oracle.com/javase/specs/.

JAVA LANGUAGE SPECIFICATION API,JDK & IDE.

- The *application program interface* (API), also known as library, contains predefined classes and interfaces for developing Java programs. The API is still expanding. You can view and download the latest version of the Java API at download.java.net/jdk8/docs/api/.
- Java is a full-fledged and powerful language that can be used in many ways. It comes in three editions:
  - Java Standard Edition (Java SE) to develop client-side applications. The applications can run on desktop.
  - Java Enterprise Edition (Java EE) to develop server-side applications, such as Java servlets, JavaServer Pages (JSP), and JavaServer Faces (JSF).
  - Java Micro Edition (Java ME) to develop applications for mobile devices, such as cell phones.

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

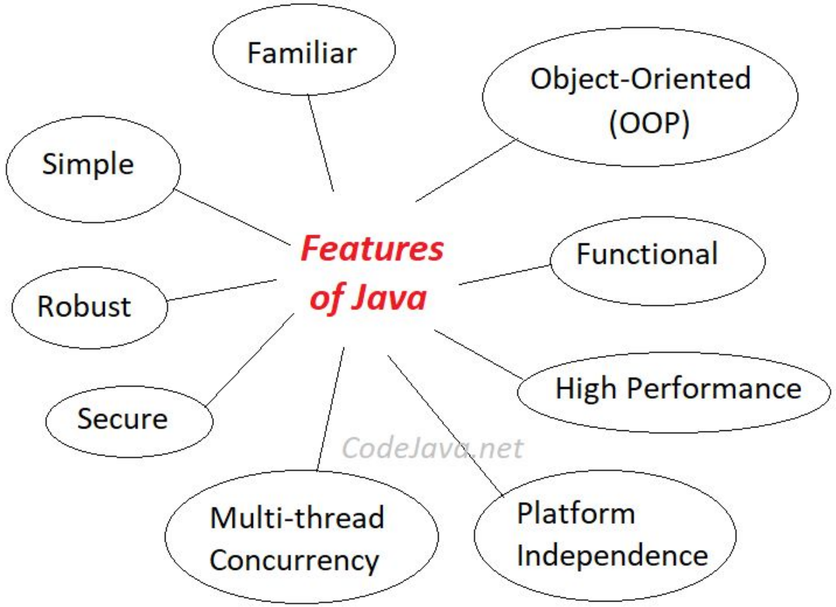
JAVA LANGUAGE SPECIFICATION API,JDK & IDE.

- Java SE is the foundation upon which all other Java technology is based. There are many versions of Java SE. Oracle releases each version with a Java Development Toolkit (JDK).
- The JDK consists of a set of separate programs, each invoked from a command line, for compiling, running, and testing Java programs. The program for running Java programs is known as JRE (Java Runtime Environment)
- Instead of using the JDK, you can use a Java development tool (e.g., NetBeans, Eclipse, and TextPad)—software that provides an integrated development environment (IDE) for developing Java programs quickly.

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

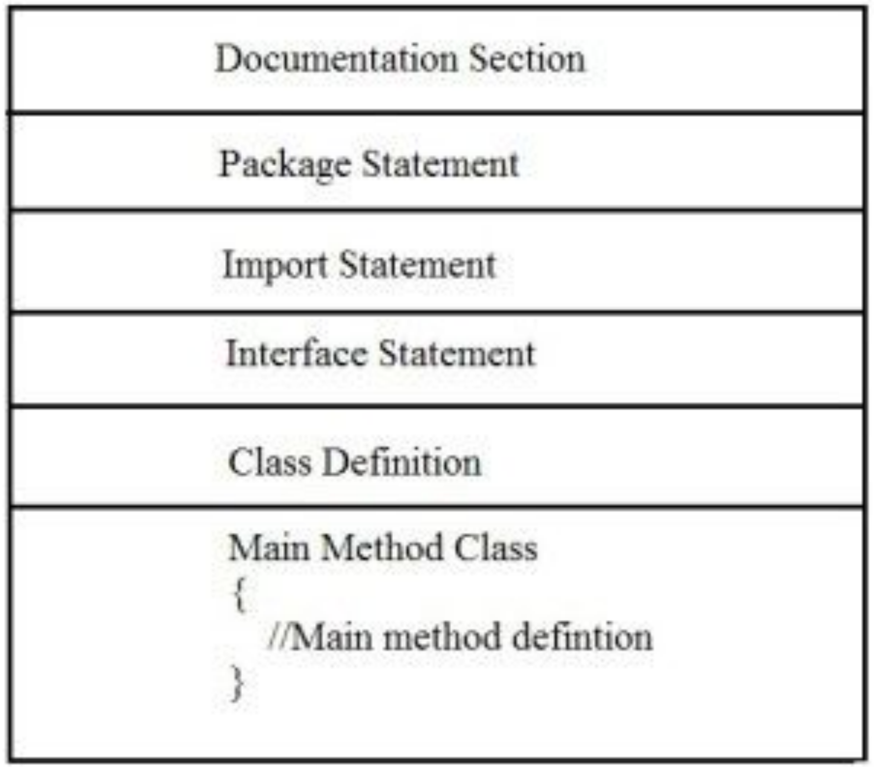
FEATURES OF JAVA

- ❖ Object Oriented
- ❖ Platform Independent.
- ❖ Simple.
- ❖ Secure.
- ❖ Architecture-neutral.
- ❖ Portable.
- ❖ Robust.
- ❖ Multithreaded.
- ❖ High Performance



AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

PROGRAMMING STRUCTURE



Documentation Section

Package Statement

Import Statement

Interface Statement

Class Definition

```
Main Method Class
{
   //Main method defintion
}
```

CREATING AND COMPILING AND EXECUTING A SIMPLE JAVA PROGRAM

CREATING JAVA PROGRAM:

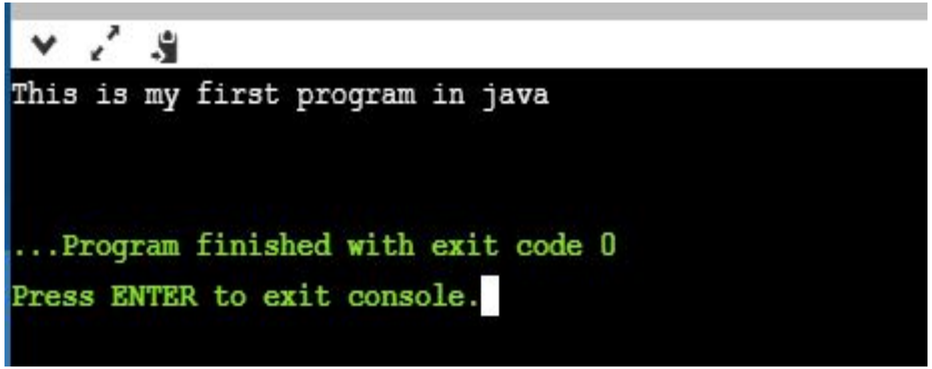
```
public class FirstJavaProgram {
  public static void main(String[] args){
    System.out.println("This is my first program in java");
  }//End of main
}//End of FirstJavaProgram Class
```

CREATING AND COMPILING AND EXECUTING A SIMPLE JAVA PROGRAM

COMPILING AND RUNNING ABOVE PROGRAM:

Step 1: Open a text editor, like Notepad on windows and TextEdit on Mac. Copy the above program and paste it in the text editor.

You can also use IDE like Eclipse to run the java program but we will cover that part later in the coming tutorials. For the sake of simplicity, I will only use text editor and command prompt (or terminal).



```
This is my first program in java


...Program finished with exit code 0
Press ENTER to exit console.
```

Step 2: Save the file as **FirstJavaProgram.java**. You may be wondering why we have named the file as FirstJavaProgram, the thing is that we should always name the file same as the public class name. In our program, the public class name is FirstJavaProgram, that’s why our file name should be **FirstJavaProgram.jav**

Step 3: In this step, we will compile the program. For this, open **command prompt (cmd) on Windows**, if you are **Mac OS then open Terminal**.

CONCEPT OF BYTECODE AND JVM

What is JVM?

Java virtual Machine (JVM) is a virtual Machine that provides runtime environment to execute java byte code.

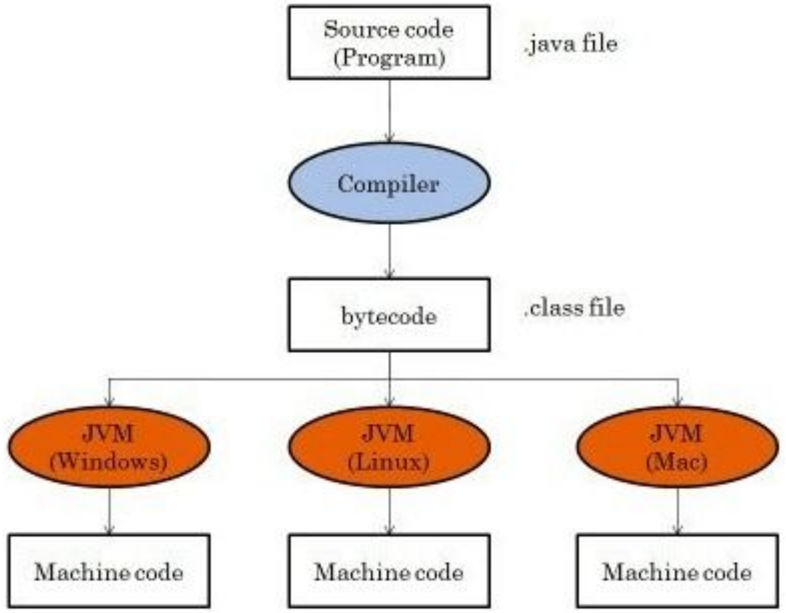
The JVM doesn't understand Java typo, that's why you compile your ***.java** files to obtain ***.class** files that contain the bytecodes understandable by the JVM.

JVM control execution of every Java program.

It enables features such as automated exception handling, Garbage-collected heap.

What is Bytecode?

Bytecode is nothing but the intermediate representation of Java source code which is produced by the Java compiler by compiling that source code. This byte code is an machine independent code.It is not an completely a compiled code but it is an intermediate code somewhere in the middle which is later interpreted and executed by JVM.Bytecode is a machine code for JVM.



AMIRAJ COLLEGE OF ENGINEERING & TECHNOLOGY

PROGRAMMING STYLE IN JAVA

- every class should be in a separate file;
- use meaningful names for the variables and for the file names;
- use comments; ...
- always use {} for blocks corresponding to "if", "while", and "for" statements, even if the corresponding block consists of only one statement;
- use indentation;

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

BLOCK STYLE

1) Next Line Style

```
Public class Test
{
        Public static void main(String[] args)
        {
                System.out.println("welcome");
        }
}
```

2)End of Line Style

```
Public class Test{
        Public static void main(String[] args)
        {
                System.out.println("welcome");
        }
}
```

AMIRAJ COLLEGE OF ENGINEERING & TECHNOLOGY

PROGRAMMING ERRORS

Programming errors can be categorized into three types:

1. syntax error
2. runtime errors
3. logic errors

1. Syntax Errors

Errors that are detected by the compiler are called **syntax errors** or **compile errors**. Syntax errors result from errors in code construction, such as mistyping a keyword, omitting some necessary punctuation, or using an opening brace without a corresponding closing brace.

These errors are usually easy to detect because the compiler tells you where they are and what caused them. For example, the following program has a syntax error:

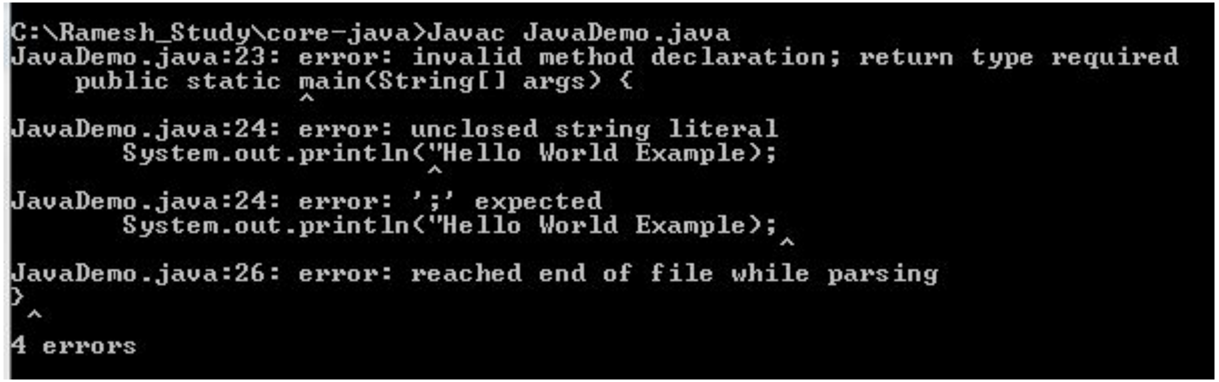
```
public class JavaDemo {

    public static main(String[] args) {

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

    }

}
```



```
C:\Ramesh_Study\core-java>Javac JavaDemo.java
JavaDemo.java:23: error: invalid method declaration; return type required
    public static main(String[] args) {
                  ^
JavaDemo.java:24: error: unclosed string literal
        System.out.println("Hello World Example);
                           ^
JavaDemo.java:24: error: ';' expected
        System.out.println("Hello World Example);
                                                 ^
JavaDemo.java:26: error: reached end of file while parsing
}
 ^
4 errors
```

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

2. Runtime Errors

Runtime errors are errors that cause a program to terminate abnormally. They occur while a program is running if the environment detects an operation that is impossible to carry out. Input mistakes typically cause runtime errors. An input error occurs when the program is waiting for the user to enter a value, but the user enters a value that the program cannot handle.

For instance, if the program expects to read in a number, but instead the user enters a string, this causes **data-type errors** to occur in the program.

Another example of runtime errors is **division by zero**. This happens when the divisor is zero for integer divisions. For instance, the following program would cause a runtime error:

```
public class JavaDemo {

    public static void main(String[] args) {

       System.out.println(10/0);

    }

}
```

Output:

```
Exception in thread "main" java.lang.ArithmeticException: / by zero

 at com.javaguides.JavaDemo.main(JavaDemo.java:24)
```

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

Logic Errors

Logic errors occur when a program does not perform the way it was intended to. Errors of this kind occur for many different reasons. For example, suppose you wrote the program to convert Celsius 35 degrees to a Fahrenheit degree:

```
public class JavaDemo {

    public static void main(String[] args) {

        System.out.println("Celsius 35 is Fahrenheit degree ");

        System.out.println((9 / 5) * 35 + 32);

    }

}
```

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

Output:

```
Celsius 35 is Fahrenheit degree

67
```

You will get Fahrenheit 67 degrees, which is wrong. It should be 95.0. In Java, the division for integers is the quotient—the fractional part is truncated—so in Java 9 / 5 is 1. To get the correct result, you need to use 9.0 / 5, which results in 1.8.

Common Errors

Missing a closing brace, missing a semicolon, missing quotation marks for strings, and misspelling names are common errors for new programmers.

IDENTIFIER AND VARIABLES

```
public class Test

{

    public static void main(String[] args)

    {

        int a = 20;

    }

}
```

In the above java code, we have 5 identifiers namely :

- **Test** : class name.
- **main** : method name.
- **String** : predefined class name.
- **args** : variable name.
- **a** : variable name.

Rules for defining Java Identifiers

There are certain rules for defining a valid java identifier. These rules must be followed, otherwise we get compile-time error. These rules are also valid for other languages like C,C++.

- The only allowed characters for identifiers are all alphanumeric characters([**A-Z**],[**a-z**],[**0-9**]), '**$**'(dollar sign) and '**_**' (underscore).For example "geek@" is not a valid java identifier as it contain '@' special character.
- Identifiers should **not** start with digits(**[0-9]**). For example "123geeks" is a not a valid java identifier.

- Java identifiers are **case-sensitive**.
- There is no limit on the length of the identifier but it is advisable to use an optimum length of 4 – 15 letters only.
- **Reserved Words** can’t be used as an identifier. For example “int while = 20;” is an invalid statement as while is a reserved word. There are **53** reserved words in Java.

Examples of valid identifiers :

```
MyVariable

MYVARIABLE

myvariable

x

i
```

i

x1

i1

_myvariable

$myvariable

sum_of_array

geeks123

Examples of invalid identifiers :

My Variable // contains a space

123geeks   // Begins with a digit

a+c // plus sign is not an alphanumeric character

variable-2 // hyphen is not an alphanumeric character

sum_&_difference // ampersand is not an alphanumeric character

VARIABLES IN JAVA

Variable is name of *reserved area allocated in memory*. In other words, it is a *name of memory location*. It is a combination of "vary + able" that means its value can be changed.

```
int data=50;//Here data is variable
```

Types of Variables

There are three types of variables in java:

- local variable
- instance variable
- static variable

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

1) Local Variable

A variable declared inside the body of the method is called local variable. You can use this variable only within that method and the other methods in the class aren't even aware that the variable exists.

A local variable cannot be defined with "static" keyword.

2) Instance Variable

A variable declared inside the class but outside the body of the method, is called instance variable. It is not declared as static.

It is called instance variable because its value is instance specific and is not shared among instances.

3) Static variable

A variable which is declared as static is called static variable. It cannot be local. You can create a single copy of static variable and share among all the instances of the class. Memory allocation for static variable happens only once when the class is loaded in the memory.

Example to understand the types of variables in java

```
class A{
int data=50;//instance variable
static int m=100;//static variable
void method(){
int n=90;//local variable
}
}//end of class
```

DATA TYPES

Data types specify the different sizes and values that can be stored in the variable. There are two types of data types in Java:

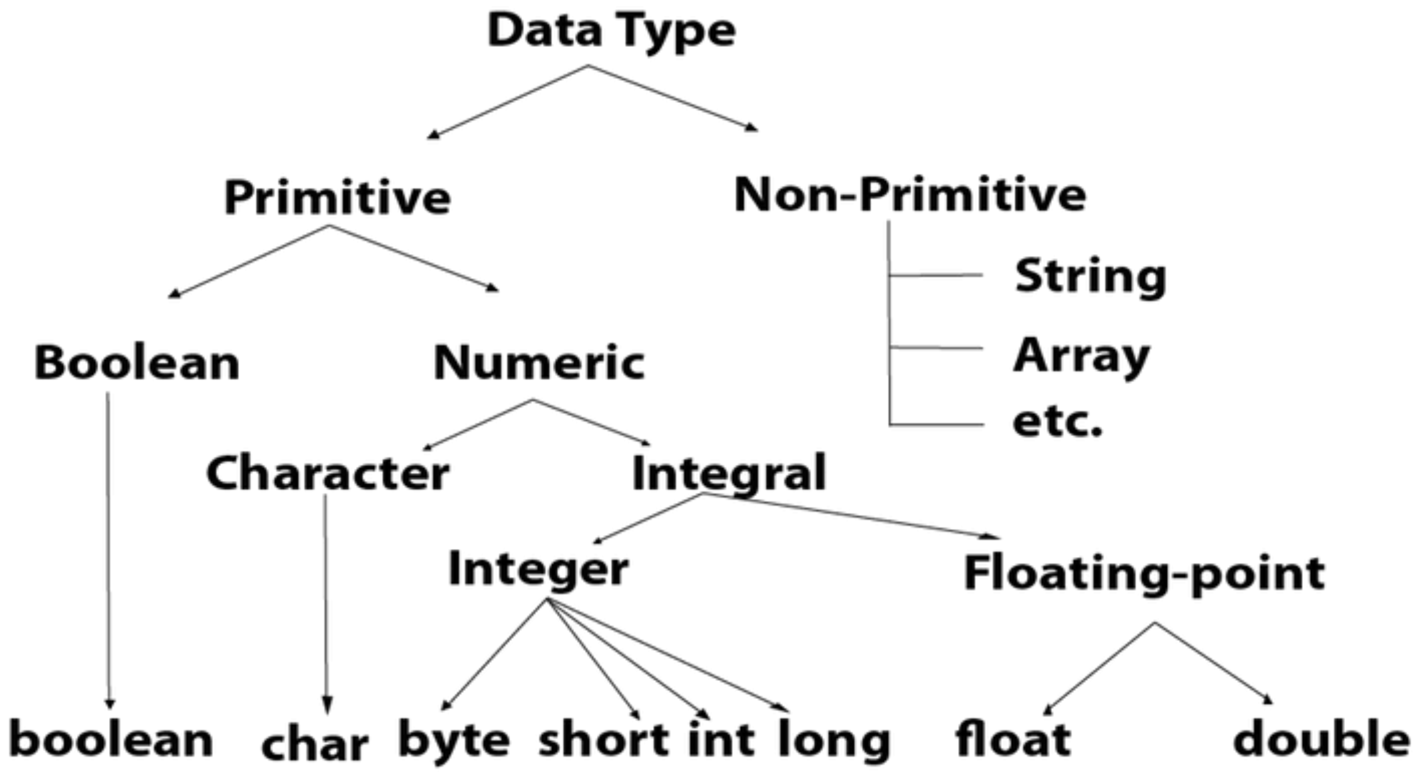
1. **Primitive data types:** The primitive data types include boolean, char, byte, short, int, long, float and double.
2. **Non-primitive data types:** The non-primitive data types include Classes, Interfaces, and Arrays.

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.

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



AMIRAJ COLLEGE OF ENGINEERING & TECHNOLOGY

Data Type	Default Value	Default size
boolean	false	1 bit
char	'\u0000'	2 byte
byte	0	1 byte
short	0	2 byte
int	0	4 byte

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

long	0L	8 byte
float	0.0f	4 byte
double	0.0d	8 byte

BOOLEAN DATATYPE

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

AMIRAJ COLLEGE OF ENGINEERING & TECHNOLOGY

BYTE DATATYPE

The byte data type is an example of primitive data type. It isan 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

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

SHORT DATATYPE

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 DATATYPE

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

LONG DATATYPE

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 DATATYPE

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 DATATYPE

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 DATATYPE

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'

NON-PRIMITIVE DATATYPE

The **Reference Data Types** will contain a memory address of variable value because the reference types won’t store the variable value directly in memory. They are **strings**, **objects**, arrays, etc.

String: Strings are defined as an array of characters. The difference between a character array and a string is the string is terminated with a special character ‘\0’.
Below is the basic syntax for declaring a string in Java programming language.
Syntax:
<String_Type> <string_variable> = “<sequence_of_string>”;
Example:
// Declare String without using new operator

String s = "GeeksforGeeks";

// Declare String using new operator

- String s1 = new String("GeeksforGeeks");

- **Interface**: Like a class, an interface can have methods and variables, but the methods declared in an interface are by default abstract (only method signature, no body).
  - Interfaces specify what a class must do and not how. It is the blueprint of the class.
  - An Interface is about capabilities like a Player may be an interface and any class implementing Player must be able to (or must implement) move(). So it specifies a set of methods that the class has to implement.
  - If a class implements an interface and does not provide method bodies for all functions specified in the interface, then class must be declared abstract.
  - A Java library example is, Comparator Interface. If a class implements this interface, then it can be used to sort a collection.

- **Array:** An array is a group of like-typed variables that are referred to by a common name.Arrays in Java work differently than they do in C/C++. Following are some important point about Java arrays.
  - In Java all arrays are dynamically allocated.(discussed below)
  - Since arrays are objects in Java, we can find their length using member length. This is different from C/C++ where we find length using sizeof.
  - A Java array variable can also be declared like other variables with [] after the data type.
  - The variables in the array are ordered and each have an index beginning from 0.
  - Java array can be also be used as a static field, a local variable or a method parameter.
  - The **size** of an array must be specified by an int value and not long or short.
  - The direct superclass of an array type is Object.
  - Every array type implements the interfaces Cloneable and java.io.Serializable.

TYPES OF OPERATOR

Java provides many types of operators which can be used according to the need. They are classified based on the functionality they provide. Some of the types are-

1. Arithmetic Operators
2. Unary Operators
3. Assignment Operator
4. Relational Operators
5. Logical Operators

Arithmetic Operators: They are used to perform simple arithmetic operations on primitive data types.

- *** :** Multiplication
- **/ :** Division
- **% :** Modulo
- **+ :** Addition
- **– :** Subtraction

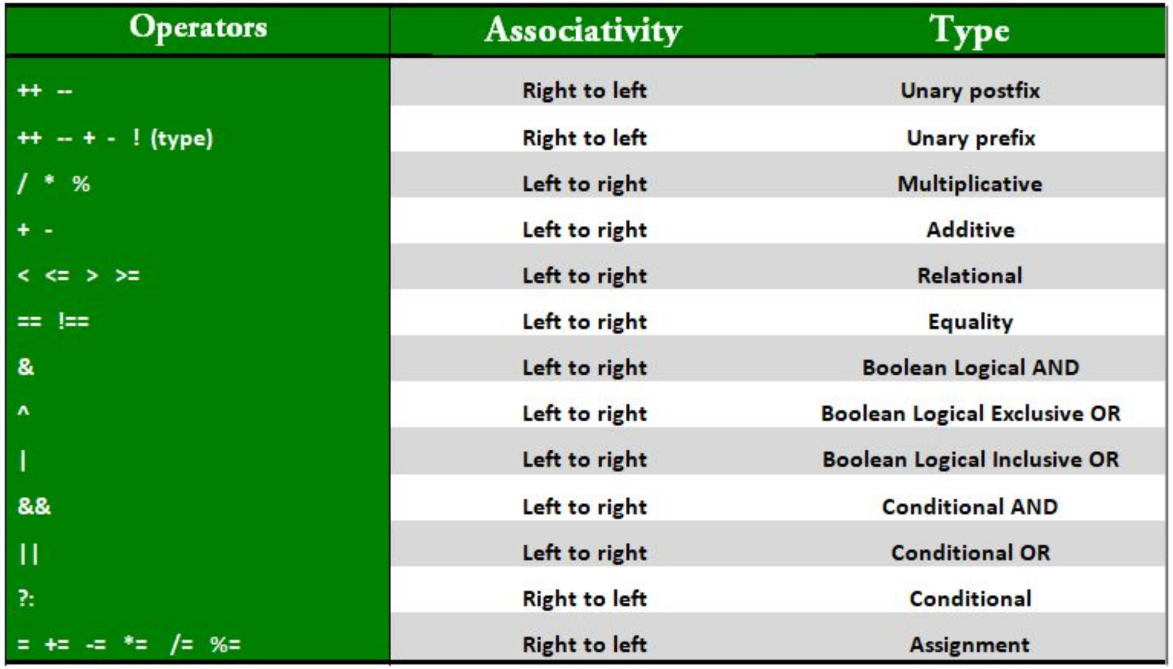
- **Unary Operators:** Unary operators need only one operand. They are used to increment, decrement or negate a value.
  - **– :Unary minus**, used for negating the values.
  - **+ :Unary plus**, used for giving positive values. Only used when deliberately converting a negative value to positive.
  - **++ :Increment operator**, used for incrementing the value by 1. There are two varieties of increment operator.
    - **Post-Increment :** Value is first used for computing the result and then incremented.
    - **Pre-Increment :** Value is incremented first and then result is computed.
  - **— : Decrement operator**, used for decrementing the value by 1. There are two varieties of decrement operator.
    - **Post-decrement :** Value is first used for computing the result and then decremented.
    - **Pre-Decrement :** Value is decremented first and then result is computed.
- **! : Logical not operator**, used for inverting a boolean value.

PRECEDENCE AND ASSOCIATIVITY OF OPERATORS

Precedence and associative rules are used when dealing with hybrid equations involving more than one type of operator. In such cases, these rules determine which part of the equation to consider first as there can be many different valuations for the same equation. The below table depicts the precedence of operators in decreasing order as magnitude with the top representing the highest precedence and bottom shows the lowest precedence.

Precedence and Associativity:

There is often a confusion when it comes to hybrid equations that is equations having multiple operators. The problem is which part to solve first. There is a golden rule to follow in these situations. If the operators have different precedence, solve the higher precedence first. If they have same precedence, solve according to associativity, that is either from right to left or from left to right.



Operators	Associativity	Type
++ --	Right to left	Unary postfix
++ -- + - ! (type)	Right to left	Unary prefix
/ * %	Left to right	Multiplicative
+ -	Left to right	Additive
< <= > >=	Left to right	Relational
== !==	Left to right	Equality
&	Left to right	Boolean Logical AND
^	Left to right	Boolean Logical Exclusive OR
|	Left to right	Boolean Logical Inclusive OR
&&	Left to right	Conditional AND
||	Left to right	Conditional OR
?:	Right to left	Conditional
= += -= *= /= %=	Right to left	Assignment

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`

- 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

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

```
public class MyClass {
  public static void main(String[] args) {
    double myDouble = 9.78;
    int myInt = (int) myDouble; // Manual casting: double to int

    System.out.println(myDouble);   // Outputs 9.78
    System.out.println(myInt);      // Outputs 9
}
}
```

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY

AMIRAJ
COLLEGE OF ENGINEERING & TECHNOLOGY