Chapter-2 Basics of Java

What is Java?

- Java is a programming language and a platform. Java is a high level, robust, object-oriented and secure programming language.
- Java was developed by Sun Microsystems (which is now the subsidiary of Oracle) in the year 1995. James Gosling is known as the father of Java. Before Java, its name was Oak. Since Oak was already a registered company, so James Gosling and his team changed the name from Oak to Java.
- **Platform:** Any hardware or software environment in which a program runs, is known as a platform. Since Java has a runtime environment (JRE) and API, it is called a platform.

Applications of Java

- Desktop Applications such as acrobat reader, media player, antivirus, etc.
- Web Applications such as irctc.co.in, javatpoint.com, etc.
- Enterprise Applications such as banking applications.
- Mobile
- Embedded System
- Smart Card
- Robotics
- Games, etc.

Features of Java

- **Compiled and Interpreted:** Basically, a computer programming language is either compiled or interpreted. Java has both these approaches thus making Java a two-stage system. Java compiler (javac) translates Java code (source code/ source file) to Bytecode and Java Interpreter generate machine code that can be directly executed by machine.
- **Platform Independent and portable:** Java language provided the portability features. Java programs can be easily run or moved from one computer system to another computer. Changes and upgrades in OS and resources will not force any alteration in Java programs. Java provided the portability in two ways. First way is that, Java compiler (javac) generates the bytecode and that bytecode can be executed on any machine. Second way is, size of primitive data types are machine independent.
- **Object-oriented:** Java is pure (truly) object-oriented language. Almost everything is an Object in java programming language. All program code and data are existing in objects and classes.
- **Robust and secure:** Java is a most strong and more powerful language which provides many securities to make more reliable code as compare to other programming languages. It is design as garbage collected language, which helps the programmers (user) virtually from all memory management problems. Java also support concept of exception handling, which detain serious errors and reduces all kind of threat of crashing the system.
- **Distributed:** Sometimes, Java language is also called Distributed language because we built applications on networks which can donate both data and programs. Java applications

can open and access from anywhere easily. That means multiple programmers can access multiple remote locations to work together on single task.

- **Familiar, simple and small:** Java is very familiar as well as small and simple language. Java does not use pointer and header files, go to statements, etc. It also eliminates operator overloading and multiple inheritance.
- **Multithreaded and Interactive:** Java also support multithreading concept. Multithreaded means manage multiple tasks simultaneously. That means we need not wait for the application to complete one task before starting next task. In other word, we can say that more than one tasks are executed in parallel. This feature is helpful for graphic applications.
- **High performance:** Performance of java language is actual unexpected for an interpreted language,
- due to the use of intermediate bytecode. Java architecture is also designed to shrink overheads during runtime. The multithreading improves the execution speed of program.
- **Dynamic and Extensible:** Java is also dynamic language. Java is capable of dynamically linking in new class, libraries, methods and objects. Java program is support functions written in other language such as C and C++, known as native methods.

Java OOPs Concepts

- The popular object-oriented languages are Java, C#, PHP, Python, C++, etc.
- Solving a problem by creating objects is called as object-oriented programming.
- The main aim of object-oriented programming is to implement real-world entities.
- Object oriented programming tries to map code instructions with real world making the code short and easier to understand.
- It is based on DRY (Do not Repeat Yourself) principle.
- Procedural programming is about writing procedures or methods that perform operations on the data, while object-oriented programming is about creating objects that contain both data and methods.

Object-oriented programming has several advantages over procedural programming:

- OOP is faster and easier to execute
- OOP provides a clear structure for the programs
- OOP helps to keep the Java code DRY "Don't Repeat Yourself", and makes the code easier to maintain, modify and debug
- OOP makes it possible to create full reusable applications with less code and shorter development time

Object:

- Any entity that has state and behavior is known as an object. For example, a chair, pen, table, keyboard, bike, etc. It can be physical or logical.
- An Object can be defined as an instance of a class. An object contains an address and takes up some space in memory. Objects can communicate without knowing the details of each other's data or code. The only necessary thing is the type of message accepted and the type of response returned by the objects.
- Example: A dog is an object because it has states like color, name, breed, etc. as well as behaviors like wagging the tail, barking, eating, etc.

Class:

- Collection of objects is called class. It is a logical entity.
- A class can also be defined as a blueprint from which you can create an individual object. Class doesn't consume any space.

Characteristics of OOP

- **Abstraction:** Hiding internal details and showing functionality is known as abstraction. For example, phone call, we don't know the internal processing. In Java, we use abstract class and interface to achieve abstraction.
- **Encapsulation:** Binding (or wrapping) code and data together into a single unit are known as encapsulation. For example, a capsule, it is wrapped with different medicines. A java class is the example of encapsulation. Java bean is the fully encapsulated class because all the data
- members are private here.
- **Inheritance:** When one object acquires all the properties and behaviors of a parent object, it is known as inheritance. It provides code reusability. It is used to achieve runtime polymorphism.
- **Polymorphism:** If one task is performed in different ways, it is known as polymorphism. For example: to convince the customer differently, to draw something, for example, shape, triangle, rectangle, etc. In Java, we use method overloading and method overriding to achieve polymorphism. Another example can be to speak something; for example, a cat speaks meow, dog barks woof, etc.

JVM, JDK, JRE & Java Bytecode

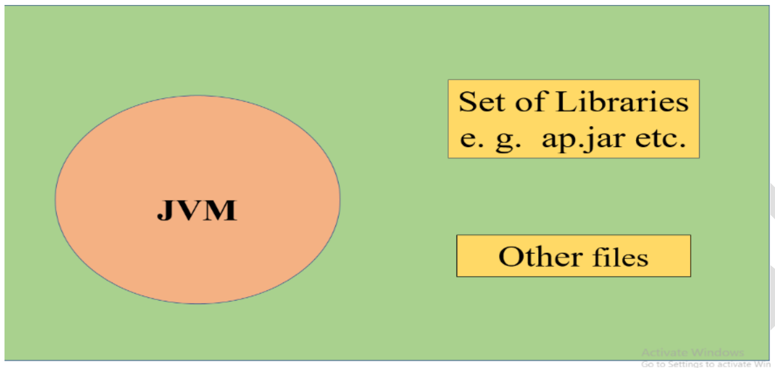
JVM

- 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.
- 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.
- The JVM performs the following main tasks:
  • Loads code
  • Verifies code
  • Executes code
  • Provides runtime environment.

JRE

- 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.

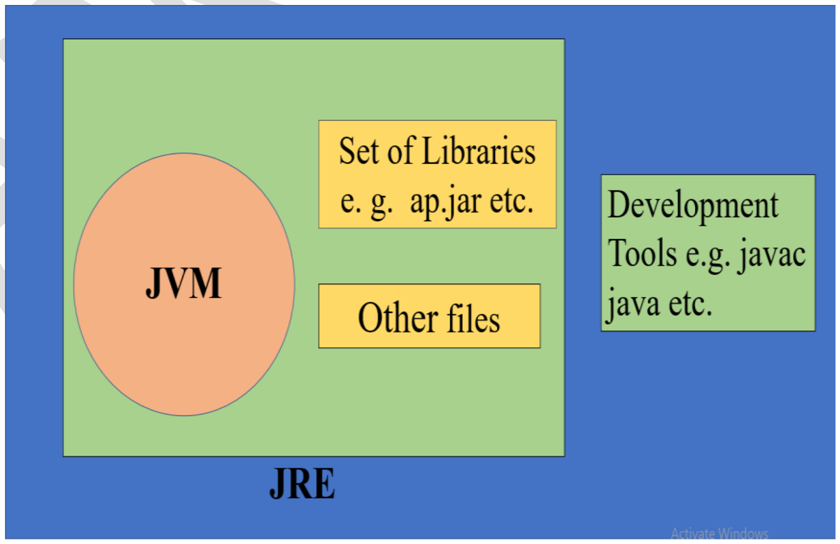
- It is the implementation of JVM. It physically exists. It contains a set of libraries + other files that JVM uses at runtime.
- The implementation of JVM is also actively released by other companies besides Sun Micro Systems.



JRE

JDK

- 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.



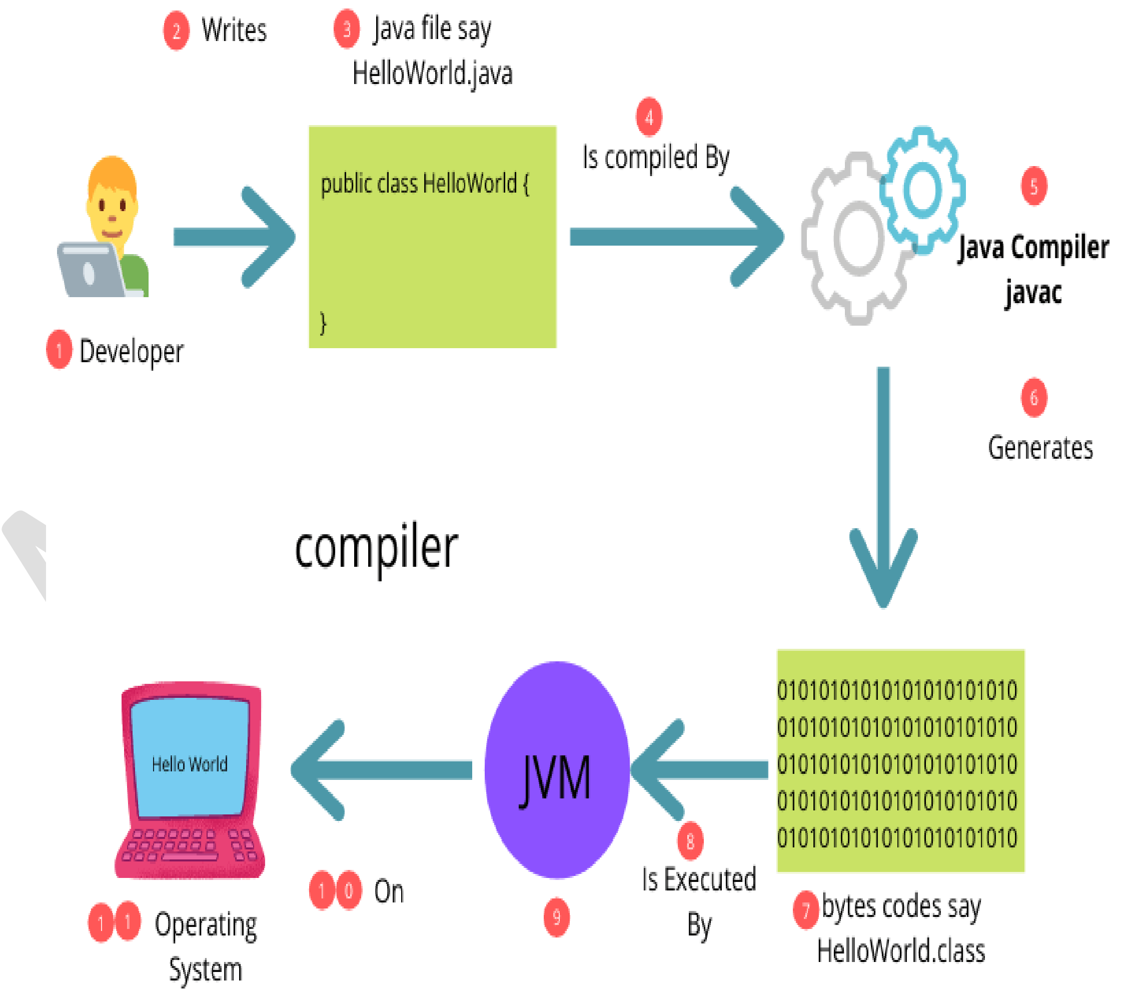
JDK

Java Bytecode

- Java bytecode is the instruction set for the Java Virtual Machine. It acts similar to an assembler which is an alias representation of a C++ code.
- As soon as a java program is compiled, java bytecode is generated. In more apt terms, java bytecode is the machine code in the form of a .class file.
- With the help of java bytecode, we achieve platform independence in java.

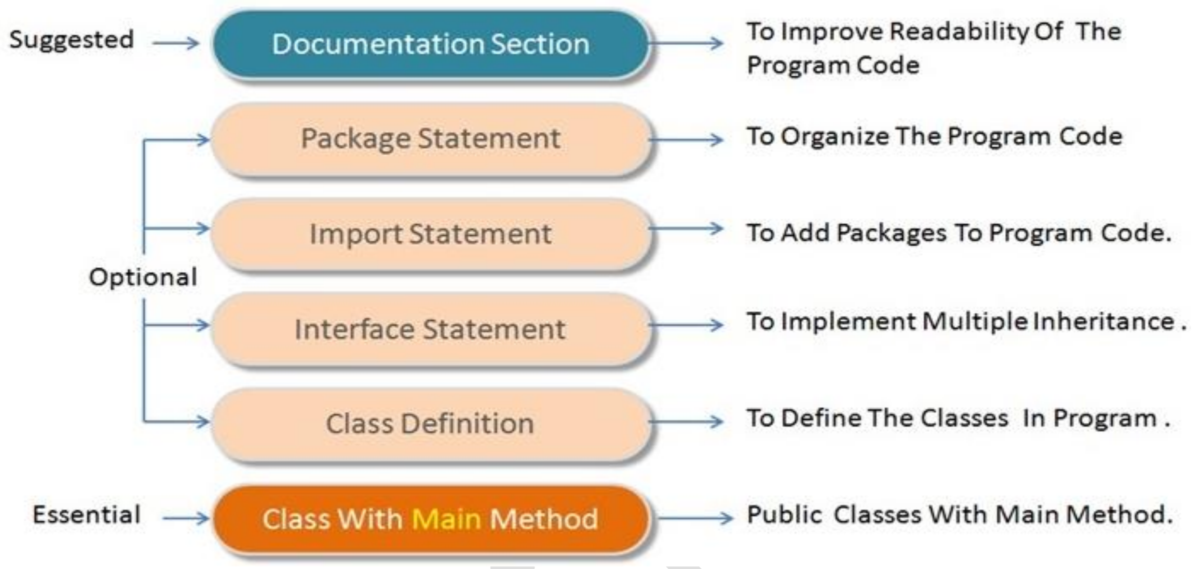
How does it work?

- When we write a program in Java, firstly, the compiler compiles that program and a bytecode is generated for that piece of code.
- When we wish to run this .**class file** on any other platform, we can do so.
- After the first compilation, the bytecode generated is now run by the Java Virtual Machine and not the processor in consideration. This essentially means that we only need to have basic java installation on any platforms that we want to run our code on.
- Resources required to run the bytecode are made available by the Java Virtual Machine, which calls the processor to allocate the required resources. JVM's are stack-based so they stack implementation to read the codes.



Basic Structure of Java Program

- Java is an object-oriented programming, platform independent, and secure programming language that makes it popular.
- Using the Java programming language, we can develop a wide variety of applications. So, before diving in depth, it is necessary to understand the basic structure of Java program in detail.


Suggested
Documentation Section
To Improve Readability Of The Program Code
Package Statement
To Organize The Program Code
Import Statement
To Add Packages To Program Code.
Optional
Interface Statement
To Implement Multiple Inheritance .
Class Definition
To Define The Classes In Program .
Essential
Class With Main Method
Public Classes With Main Method.

Documentation Section

- Documentation section is an important section but optional for a Java program.
- It includes **basic information** about a Java program. The information includes the **author's name, date of creation, version, program name, company name,** and **description** of the program.
- It improves the readability of the program. Whatever we write in the documentation section, the Java compiler ignores the statements during the execution of the program.
- To write the statements in the documentation section, we use **comments**. The comments may be **single-line, multi-line,** and **documentation** comments.
- **Single-line Comment:** It starts with a pair of forwarding slash **(//)**. For example:
  //First Java Program

- **Multi-line Comment:** It starts with a /* and ends with */**.** We write between these two symbols. For example:
  /*It is an example of
  multiline comment*/

- **Documentation Comment:** It starts with the delimiter **(/**)** and ends with ***/**. For example:
  /It is an example of documentation comment*/**

Package Declaration

- The package declaration is optional. It is placed just after the documentation section.
- In this section, we declare the package name in which the class is placed. Note that there can be only one package statement in a Java program. It must be defined before any class and interface declaration.
- It is necessary because a Java class can be placed in different packages and directories based on the module they are used. For all these classes package belongs to a single parent directory.
- We use the keyword package to declare the package name. For example:
    - **package arp;** //where arp is the package name
    - **package com.arp;** //where com is the root directory and arp is the subdirectory

Import Statements

- The package contains the many predefined classes and interfaces. If we want to use any class of a particular package, we need to import that class.
- The import statement represents the class stored in the other package. We use the **import** keyword to import the class. It is written before the class declaration and after the package statement.
- We use the import statement in two ways, either import a specific class or import all classes of a particular package. In a Java program, we can use multiple import statements. For example:
    - **import java.util.Scanner;** //it imports the Scanner class only
    - **import java.util.*;** //it imports all the class of the java.util package

Interface Section

- It is an optional section. We can create an **interface** in this section if required.
- We use the **interface** keyword to create an interface. An interface is a slightly different from the class. It contains only **constants** and **method** declarations. Another difference is that it cannot be instantiated.
- We can use interface in classes by using the **implements** keyword. An interface can also be used with other interfaces by using the **extends** keyword. For example:

```
interface car
{
    void start();
    void stop();
}
```

Class Definition

- In this section, we define the class. It is **vital** part of a Java program. Without the class, we cannot create any Java program.
- A Java program may conation more than one class definition. We use the **class** keyword to define the class.
- The class is a blueprint of a Java program. It contains information about user-defined methods, variables, and constants. Every Java program has at least one class that contains the main() method. For example:

```
class Student
{

}
```

Class Variables and Constants

- In this section, we define variables and **constants** that are to be used later in the program. In a Java program, the variables and constants are defined just after the class definition.
- The variables and constants store values of the parameters. It is used during the execution of the program. We can also decide and define the scope of variables by using the modifiers. It defines the life of the variables. For example:

```
class Student //class definition
{
    String sname; //variable
    int id;
    double percentage;
}
```

Main Method Class

- In this section, we define the **main() method.** It is essential for all Java programs. Because the execution of all Java programs starts from the main() method.
- In other words, it is an entry point of the class. It must be inside the class. Inside the main method, we create objects and call the methods. We use the following statement to define the main() method:

```
public class Student //class definition
{
    public static void main (String args[])
    {
        //statements
    }
}
```

***Note:**

- To run any program of java you must install JDK Java Development Kit.
- Open command prompt and enter “java –version”. If java is installed then, version number is displayed

Creating Hello World Example

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

- Save the above file as Hello.java. (You can also give different name if the class is not public)

  To compile:
  javac Hello.java

  To execute:
  java Hello

  Output:
  Hello Java

Parameters used in First Java Program

- Let's see what is the meaning of class, public, static, void, main, String[], System.out.println().
- **class** keyword is used to declare a class in Java.
- **public** keyword is an access modifier that represents visibility. It means it is visible to all.
- **static** is a keyword. If we declare any method as static, it is known as the static method. The core advantage of the static method is that there is no need to create an object to invoke the static method. The main() method is executed by the JVM, so it doesn't require creating an object to invoke the main() method. So, it saves memory.
- **void** is the return type of the method. It means it doesn't return any value.
- **main** represents the starting point of the program.
- **String[] args or String args[]** is used for command line argument
- **System.out.println() is** used to print statement. Here, System is a class, out is an object of the PrintStream class, println() is a method of the PrintStream class.

Identifiers in Java

- Identifiers in Java are symbolic names used for identification. They can be a class name, variable name, method name, package name, constant name, and more. However, In Java, There are some reserved words that cannot be used as an identifier.
- For every identifier there are some conventions that should be used before declaring them. Let's understand it with a simple Java program:

Rules for Identifiers in Java

- There are some rules and conventions for declaring the identifiers in Java. If the identifiers are not properly declared, we may get a compile-time error. Following are some rules and conventions for declaring identifiers:
- A valid identifier must have characters [A-Z] or [a-z] or numbers [0-9], and underscore(_) or a dollar sign ($). for example, @arp is not a valid identifier because it contains a special character which is @.
- There should not be any space in an identifier. For example, ar patel is an invalid identifier.
- An identifier should not contain a number at the starting. For example, 123arp 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 can't 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.

Java Reserved Keywords

- Java reserved keywords are predefined words, which are reserved for any functionality or meaning. We can not use these keywords as our identifier names, such as class name or method name. These keywords are used by the syntax of Java for some functionality. If we use a reserved word as our variable name, it will throw an error.
- In Java, every reserved word has a unique meaning and functionality.
- Below is the list of reserved keywords in Java:

abstract	continue	for	protected	transient
Assert	Default	Goto	public	Try
Boolean	Do	If	Static	throws
break	double	implements	strictfp	Package
byte	else	import	super	Private
case	enum	Interface	Short	switch
Catch	Extends	instanceof	return	void
Char	Final	Int	synchronized	volatile
class	finally	long	throw	Date
const	float	Native	This	while

- Although the const and goto are not part of the Java language; But, they are also considered keywords.

Example of Valid and Invalid Identifiers

- **Valid identifiers:**
  Following are some examples of valid identifiers in Java:
  - TestVariable
  - testvariable
  - a
  - i
  - Test_Variable
  - _testvariable
  - $testvariable
  - sum_of_array
  - TESTVARIABLE
  - arp123
  - JavaProgramming
  - Javatprogramming123
- **Invalid identifiers:**
  - Below are some examples of invalid identifiers:
  - Test Variable (We cannot include a space in an identifier)
  - 123arp (The identifier should not begin with numbers)
  - java+language (The plus (+) symbol cannot be used)
  - a-javatlang (Hyphen symbol is not allowed)
  - java_&_ProgLang (ampersand symbol is not allowed)
  - Java'language (we cannot use an apostrophe symbol in an identifier)

Java Naming Convention

- Java naming convention is a rule to follow as you decide what to name your identifiers such as class, package, variable, constant, method, etc.
- But, it is not forced to follow. So, it is known as convention not rule. These conventions are suggested by several Java communities such as Sun Microsystems and Netscape.
- All the classes, interfaces, packages, methods and fields of Java programming language are given according to the Java naming convention. If you fail to follow these conventions, it may generate confusion or erroneous code.

Advantage of Naming Conventions in Java

- By using standard Java naming conventions, you make your code easier to read for yourself and other programmers. Readability of Java program is very important. It indicates that less time is spent to figure out what the code does.

CamelCase in Java naming conventions

- Java follows camel-case syntax for naming the class, interface, method, and variable.

Naming Conventions of the Different Identifiers

- The following table shows the popular conventions used for the different identifiers.

Identifiers Type	**Naming Rules**	**Examples**

Class	It should start with the uppercase letter. It should be a noun such as Color, Button, System, Thread, etc.
	Use appropriate words, instead of acronyms.
		public class **Employee**
		{
		//code snippet
		}

Interface	It should start with the uppercase letter.
	It should be an adjective such as Runnable, Remote, ActionListener.
	Use appropriate words, instead of acronyms.
		interface **Printable**
		{
		//code snippet
		}

Method	It should start with lowercase letter.
	It should be a verb such as main(), print(), println().
	If the name contains multiple words, start it with a lowercase letter followed by an uppercase letter such as actionPerformed().
		class Employee
		{
		// method
		void draw()
		{
		//code snippet
		}
		}

Variable	It should start with a lowercase letter such as id, name.
	It should not start with the special characters like & (ampersand), $ (dollar), _ (underscore).
	If the name contains multiple words, start it with the lowercase letter followed by an uppercase letter such as firstName, lastName. Avoid using one-character variables such as x, y, z.
		classEmployee
		{
		//variable int id;
		//codesnippet
		}

Package	It should be a lowercase letter such as java, lang.
	If the name contains multiple words, it should
	be separated by dots (.) such as
	java.util, java.lang.
		//package
		package **com.java;**
		class Employee
		{
		//code snippet
		}

Constant	It should be in uppercase letters such as RED, YELLOW.
	If the name contains multiple words, it should be separated by an underscore(_) such as
	MAX_PRIORITY.
	It may contain digits but not as the first letter.
		class Employee
		{
		//constant
		static final int **MIN_AGE** = 18;
		//code snippet
		}

- If the name is combined with two words, the second word will start with uppercase letter always such as actionPerformed(), firstName, ActionEvent, ActionListener, etc.

Separators in java

Symbol	Name	Purpose
( )	Parentheses	used to contain a list of parameters in method definition and invocation. also used for defining precedence in expressions in control statements and surrounding cast types
{ }	Braces	Used to define a block of code, for classes, methods and local scopes Used to contain the value of automatically initialised array
[ ]	Brackets	declares array types and also used when dereferencing array values
;	Semicolon	Terminates statements
,	Comma	Used to separates package names from sub-package and class namesand also selects a field or method from an object
.	Period	separates consecutive identifiers in variable declarations also used tochains statements in the test, expression of a for loop
:	Colon	Used after labels

Variables

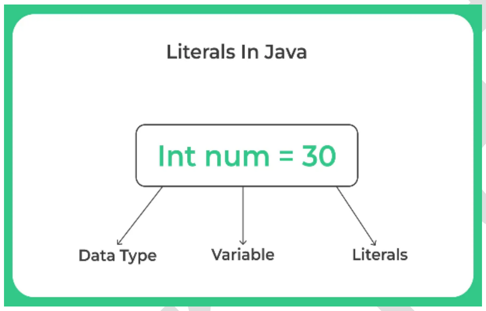
- A variable is the name of a reserved area allocated in memory. In other words, it is a name of the memory location.
- **int** data=50;//Here data is variable

Literals in Java

- In Java, **literal** is a notation that represents a fixed value in the source code. In lexical analysis, literals of a given type are generally known as **tokens**. In this section, we will discuss the term **literals in Java**.

Literals

- In Java, **literals** are the constant values that appear directly in the program. It can be assigned directly to a variable. Java has various types of literals. The following figure represents a literal.



Types of Literals in Java

- There are the majorly **four** types of literals in Java:
  - **Integer Literal**
  - **Character Literal**
  - **Boolean Literal**
  - **String Literal**

Integer Literals

- Integer literals are sequences of digits. There are three types of integer literals:

- **Decimal Integer:** These are the set of numbers that consist of digits from 0 to 9. It may have a positive (**+**) or negative (**-**) Note that between numbers commas and non-digit characters are not permitted. For example, **5678, +657, -89,** etc.

  - **int** decVal = 26;

- **Octal Integer:** It is a combination of number have digits from 0 to 7 with a leading 0. For example, **045, 026,**

  - **int** octVal = 067;

- **Hexa-Decimal:** The sequence of digits preceded by **0x** or **0X** is considered as hexadecimal integers. It may also include a character from **a** to **f** or **A** to **F** that represents numbers from **10** to **15**, respectively. For example, **0xd, 0xf,**

    - **int** hexVal = 0x1a;

- **Binary Integer:** Base 2, whose digits consists of the numbers 0 and 1 (you can create binary literals in Java SE 7 and later). Prefix 0b represents the Binary system. For example, 0b11010.

    - **int** binVal = 0b11010;

Real Literals

The numbers that contain fractional parts are known as real literals. We can also represent real literals in exponent form. For example, **879.90, 99E-3,** etc.

Backslash Literals

- Java supports some special backslash character literals known as backslash literals. They are used in formatted output. For example:
    - **\n:** It is used for a new line
    - **\t:** It is used for horizontal tab
    - **\b:** It is used for back space
    - **\v:** It is used for vertical tab
    - **\':** It is used for a single quote
    - **\":** It is used for double quotes

Character Literals

- A character literal is expressed as a character or an escape sequence, enclosed in a **single** quote ('') mark. It is always a type of char. For example, **'a', '%', '\u000d',** etc.

String Literals

- String literal is a sequence of characters that is enclosed between **double** quotes ("") marks. It may be alphabet, numbers, special characters, blank space, etc. For example, **"Jack", "12345", "\n",** etc.

Floating Point Literals

- The vales that contain decimal are floating literals. In Java, float and double primitive types fall into floating-point literals. Keep in mind while dealing with floating-point literals.
- Floating-point literals for float type end with F or f. For example, **6f, 8.354F**, etc. It is a **32**-bit float literal.
- Floating-point literals for double type end with D or d. It is optional to write D or d. For example, **6d, 8.354D,** etc. It is a **64**-bit double literal.

Floating:

- **float** length = 155.4f;

Decimal:

- **double** interest = 99658.445;

Boolean Literals

- Boolean literals are the value that is either true or false. It may also have values 0 and 1. For example, **true, 0,** etc.

- **boolean** isEven = **true**;

```
public class LiteralsExample
{
    public static void main(String args[])
    {
        int count = 987;
        float floatVal = 4534.99f;
        double cost = 19765.567;
        int hexaVal = 0x7e4;
        int binary = 0b11010;
        char alpha = 'p';
        String str = "Java";
        boolean boolVal = true;
        int octalVal = 067;
        String stuName = null;
        char ch1 = '\u0021';
        char ch2 = 1456;
        System.out.println(count);
        System.out.println(floatVal);
        System.out.println(cost);
        System.out.println(hexaVal);//2020
        System.out.println(binary);//26
        System.out.println(alpha);
        System.out.println(str);
        System.out.println(boolVal);
        System.out.println(octalVal);//55
        System.out.println(stuName);
        System.out.println(ch1);//!
        System.out.println("\t" +"backslash literal");
        System.out.println(ch2);//?
    }
}
```

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.

    - boolean data type
    - byte data type
    - char data type
    - short data type
    - int data type
    - long data type
    - float data type
    - double data type

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

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 = 256, **short** s1 = -256;

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 - 2,147,483,648** and **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 Data Type

- 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 - 9,223,372,036,854,775,808** and **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, **double** d2=12.3d;

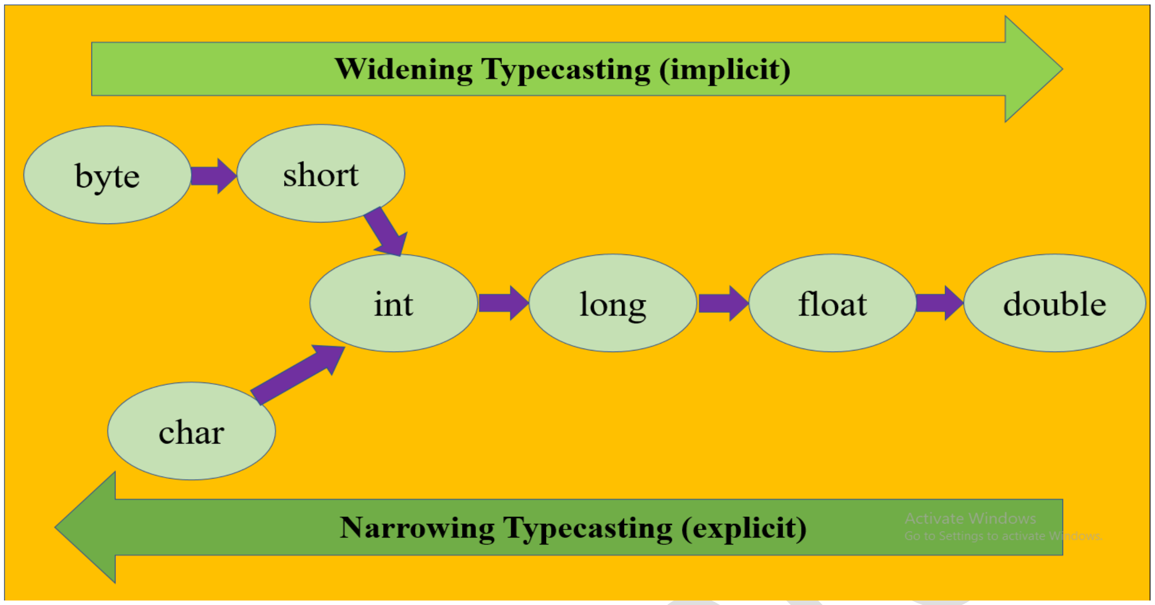
Char Data Type

- 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' ;

Type Casting in Java

- In Java, **type casting** is a method or process that converts a data type into another data type in both ways manually and automatically. The automatic conversion is done by the compiler and manual conversion performed by the programmer. In this section, we will discuss **type casting** and **its types** with proper examples.


Widening Typecasting (implicit)
byte
short
int
long
float
double
char
Narrowing Typecasting (explicit)

Type casting

- Convert a value from one data type to another data type is known as **type casting**.

Types of Type Casting

- There are two types of type casting:
    - Widening Type Casting
    - Narrowing Type Casting

Widening Type Casting

- Converting a lower data type into a higher one is called **widening** type casting. It is also known as **implicit conversion** or **casting down**. It is done automatically. It is safe because there is no chance to lose data. It takes place when:
- Both data types must be compatible with each other.
- The target type must be larger than the source type.
- For example, the conversion between numeric data type to char or Boolean is not done automatically. Also, the char and Boolean data types are not compatible with each other. Let's see an example.

```
public class WideningTypeCastingExample

{
    public static void main(String[] args)
    {
        int x = 7;
        //automatically converts the integer type into long type
        long y = x;
        //automatically converts the long type into float type
        float z = y;
        System.out.println("Before conversion, int value "+x);
```

```
            System.out.println("After conversion, long value "+y);
            System.out.println("After conversion, float value "+z);
        }
    }
    /*Before conversion, int value 7
    After conversion, long value 7
    After conversion, float value 7.0 */
```

- In the above example, we have taken an int variable x and converted it into a long type. After that, the long type is converted into the float type.

Narrowing Type Casting

- Converting a higher data type into a lower one is called **narrowing** type casting. It is also known as **explicit conversion** or **casting up**. It is done manually by the programmer. If we do not perform casting then the compiler reports a compile-time error.
- In the following example, we have performed the narrowing type casting two times. First, we have converted the double type into long data type after that long data type is converted into int type.

```
public class NarrowingTypeCastingExample
{
public static void main(String args[])
    {
        double d = 166.66;
        //converting double data type into long data type
        long l = (long)d;
        //converting long data type into int data type
        int i = (int)l;
        System.out.println("Before conversion: "+d);
        //fractional part lost
        System.out.println("After conversion into long type: 
"+l);
        //fractional part lost
        System.out.println("After conversion into int type: "+i);
    }
}
/*Before conversion: 166.66
After conversion into long type: 166
After conversion into int type: 166 */
```

Operators in java

- List of Operators see below:
    - Arithmetic operators:
    - Logical operators:

- Relational operators:
- Assignment operators:
- Bitwise operators:
- Conditional operators:
- Increment and decrement operators:

Arithmetic Operators:

Operator	Use	Description
+	op1 + op2	Adds op1 and op2; also used to concatenate strings
-	op1 - op2	Subtracts op2 from op1
*	op1 * op2	Multiplies op1 by op2
/	op1 / op2	Divides op1 by op2
%	op1 % op2	Computes the remainder of dividing op1 by op2

Example 1:

```
class ArithmeticOp
{
    public static void main(String[] args)
    {
        int a,b,add,sub,mul,div,mod,avg;
        a=16;
        b=3;
        add=a+b;
        sub=a-b;
        mul=a*b;
        div=a/b;
        mod=a%b;
        avg=(a+b)/2;
        System.out.println("addition =" +add);
        System.out.println("subtraction =" +sub);
        System.out.println("multiplication =" +mul);
        System.out.println("division =" +div);
        System.out.println("modules =" +mod);
        System.out.println("average =" +avg);
    }
}
/*addition =19
subtraction =13
multiplication =48
division =5
modules =1
average =9 */
```

Example 2:

```
            class ArithmeticOp2
            {
                public static void main(String[] args)
                {
                    int a=12;
                    int b=0;
                    System.out.println("Div="+(a/b));
                }
            }
/*Exception in thread "main" java.lang.ArithmeticException: / by zero
        at ArithmeticOp2.main(NarrowingTypeCastingExample.java:7) */
```

- Note: cannot divide any int by int 0 if will do so then get ArithmeticException

Example 3:

```
            class ArithmeticOp3
            {
                public static void main(String[] args)
                {
                    int a=12;
                    double b=0;
                    System.out.println("Div="+(a/b));
                }
            }
/*Div=Infinity */
```

- Note: when divide any int by double 0 or float 0 then output will be **Infinity.**

Logical Operator Or Short Circuit Operator:

- Logic operators **&& (and)** and || **(or)** are used when evaluating two expressions to obtain a single result.
- They correspond with Boolean logical operations AND and OR respectively. The result of them depends on the relation between its two operands:

Example: And Operator

```
if(condition1 && condition2)
{
    Statements;
}
```

```
…… Statement x;
```

- If **both conditions are become true** then after control goes inside body and execute statements.
- If **even any one condition becomes false** then skip statement and then direct execute statement x.
- If **condition1 becomes false** then compiler is **not going to check the condition2**
- See below table, Logical operator work like Boolean algebra And operator.

A	B	C
0	0	**0**
0	1	**0**
1	0	**0**
1	1	**1**

- Note: here **a and b** are considered as a condition and c is consider as an output action.
- Here **0 means condition becomes false and 1 means conditions becomes true**.
- And you can see **result (c)** in side table.

Example 1: find maximum number from given three number using AND (&&) operator

```
class DemoLogical
{
    public static void main(String[] args)
    {
        int a, b, c;
        a=16;
        b=3;
        c=20;
        if(a>b && a>c)
        {
            System.out.println("a is max" + a);
        }
        if(b>a && b>c)
        {
            System.out.println("b is max" + b);
        }
        if(c>a && c>b)
        {
            System.out.println("c is max" + c);
        }
    }
}
//Output:c is max20
```

Example: OR Operator (||)

```
if(condition 1 || condition 2)
{
    Statements;
}
…… Statement x;
```

- Here, either **Condition 1 becomes true or condition 2 becomes true, then statements will execute** then after statements x will execute.
- If **both conditions becomes false then skip statements which are inside the body** of if statements and then execute statements X.
- If **Condition 1** becomes **true** then compiler is not going to check **condition 2**
- See below table, OR operator work like Boolean algebra OR operator.

A	**B**	**C**
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**1**

- Note: here a and b are consider as a condition and c is consider as an output action.
- Here **0** means condition becomes **false** and **1** means conditions becomes **true.**
- And you can see result (c) in side table.

Relational Operators:

- Two expressions can be compared using relational and equality operators. For example, to know if two values are equal or if one is greater than the other.

operat	**description**
==	Equal to
!=	Not equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal

Example 1:demo of relational operator

```
class RelationalOp
{
    public static void main(String[] args)
    {
        int a, b; a=16; b=3;
```

```
            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<=b  " + (a<=b));
            System.out.println("a!=b  " + (a!=b));
            System.out.println("a==b  " + (a==b));
        }
    }
    /*Output:
        a>b   true
        a>=b  true
        a<b   false
        a<=b  false
        a!=b  true
        a==b  false
    */
```

Assignments Operator:

- The assignment operator assigns a value into a variable.

  int x = 5;

- This statement assigns the integer value 5 to the variable x

Example 1:demo of assignment operator

```
public class AssignOp
{
    public static void main(String[] args)
    {
        int a, b;                       // a=?,  b=?
        a = 10;                         // a=10, b=?
        b = 4;                          // a=10, b=4
        a = b;                          // a=4,  b=4
        b = 7;                          // a=4,  b=7
        System.out.println("Value of a=" +a);   //print value of a
        System.out.println("Value of b=" +b);  // print value of b
    }
}

/*
Output:
    Value of a=4
    Value of b=7
*/
```

Bitwise Operator:

- Bitwise operators modify variables considering the bit patterns that represent the values they store.

operator	description
&	Bitwise AND
|	Bitwise inclusive OR
^	Bitwise exclusive
~	Unary complement (bit inversion)
<<	Shift bits left
>>	Shift bits right

Example 1: bitwise AND

```
class DemobitwiseAnd
{
    public static void main(String[] args)
    {
        int a,b,c;
        a=10;
        b=7;
        c=a & b;
        System.out.println("ans is =" +c);

    }
}

/*Output:
    ans is =2
*/
```

Description:

- a=10 and b=7 first convert into binary value.

  00001010 this is binary value for 10
  00000111 this is binary value for 7

- Now apply And (&) operator to above binary value. And we will get value of c variable.

  00001010

```
          00000111
AND     00000010    convert this value into decimal
```
answer is c=2

Example 2: bitwise OR

```
class Demobitwiseor
{
    public static void main(String[] args)
    {
        int a,b,c;
        a=10;
        b=7;
        c=a | b;
        System.out.println("ans is =" +c);
    }
}
/*Output:
    ans is =15
*/
```

Description:

- a=10 and b=7 first convert into binary value.

  00001010   this is binary value for 10
  00000111   this is binary value for 7

- Now OR apply operator to above binary value. And we will get value of c variable.

```
         00001010
         00000111
OR      00001111    convert this value into decimal
```
answer is c=15

Example 3: bitwise XOR

```
class DemobitwiseXor
{
    public static void main(String[] args)
    {
        int a,b,c;
        a=10;
        b=7;
        c=a ^ b;
        System.out.println("ans is =" +c);
    }
}
/*Output:
    ans is =13
*/
```

Description:

- a=10 and b=7 first convert into binary value.

  00001010 this is binary value for 10
  00000111 this is binary value for 7

- Now apply XOR operator to above binary value. And we will get value of c variable.

```
a          00001010
b          00000111
XOR (c)    00001101   convert this value into decimal
```
answer is c=13

Example 4: bitwise shift left

```
class demobitwiseshiftleft
{
    public static void main(String[] args)
    {
        int a,b;
        a=20;
        b=a<<1;
        System.out.println("ans is =" +b);
    }
}

/*Output:
    ans is =40
*/
```

Description:

- a =20; first convert this into binary value.

  00010100. Binary value for 20

- Now shift each bit left side.

  00101000. After left shift we will get bit position like this.

- Now convert above bit patent into decimal and get result b=40;

Example 5: bitwise shift right

```
class demobitwiseshiftright
{
    public static void main(String[] args)
    {
        int a,b;
        a=20;
        b=a>>1;
        System.out.println("ans is =" +b);
    }
}
/*Output:
    ans is =10
*/
```

Description:

- a =20; first convert this into binary value.

  00010100. Binary value for 20

- Now shift each bit right side.

  00001010. After right shift we will get bit position like this.

- Now convert above bit patent into decimal and get result b=10;

Example 5: Unary complement (bit inversion)

```
class demoUnarycomplement
{
    public static void main(String[] args)
    {
        int a,b,c,d;
        a=20;
        b=-20;
        c=~a;
        d=~b;
        System.out.println("ans  c=" +c);
        System.out.println("ans  d=" +d);
    }
}
/*Output:
    ans c=-21
    ans d=19
*/
```

- Here unary compliment of a=-(a+1) and b=-(b+1)

Conditional Operator Or Ternary Operator:

- The conditional operator evaluates an expression, returning one value if that expression evaluates to true, and a different one if the expression evaluates as false

  Syntax:

  condition? result1: result2;

Example 1: demo of conditional operator

```
class ConditinalOP
{
    public static void main(String[] args)
    {
        int a,b,c;
        a=10;
        b=7;
        c = (a>b)? a: b;
        System.out.println("maximum number is=" +c);
    }
}
/*Output:
    maximum number is=10
*/
```

Increment or Decrement Operator:

- **Increment Operator (++):**The increment operator in Java is denoted by ++. It adds 1 to the current value of a variable. There are two ways to use the increment operator:
  int x = 5;
  x++;   // Equivalent to x = x + 1;

  or

  int x = 5;
  ++x;   // Equivalent to x = x + 1;

- Both of these statements increase the value of x by 1. However, there is a subtle difference between the two when used as part of a larger expression.

- **Prefix Increment Operator:**

  - When the increment operator is placed before the variable (++x), it increments the value of the variable before the value is used in the expression. For example:

    int a = 3;
    int b = ++a; // Now, a is 4 and b is also 4
    In this case, a is incremented before its value is assigned to b.

- **Postfix Increment Operator:**

  - When the increment operator is placed after the variable (x++), it increments the value of the variable after its current value is used in the expression. For example:

    int c = 3;
    int d = c++; // Now, c is 4, but d is 3

  Here, d is assigned the current value of c before c is incremented.

- **Decrement Operator (--):**The decrement operator in Java is denoted by --. It subtracts 1 from the current value of a variable. Similar to the increment operator, there are two ways to use the decrement operator:
  int y = 8;
  y--;   // Equivalent to y = y - 1;

  or

  int y = 8;
  --y;   // Equivalent to y = y - 1;

- Both of these statements decrease the value of y by 1.

- **Prefix Decrement Operator:**

  - Similar to the increment operator, the prefix decrement operator (--x) decrements the value of the variable before its value is used in the expression.

    int m = 7;
    int n = --m; // Now, m is 6 and n is also 6

- **Postfix Decrement Operator:**

  - The postfix decrement operator (x--) decrements the value of the variable after its current value is used in the expression.

    int p = 7;
    int q = p--; // Now, p is 6, but q is 7

Operator precedence and Associativity

Precedence	Operator	Type	Associativity
15	() [] .	Parentheses Array subscript Member selection	Left to Right
14	++ --	Unary post-increment Unary post-decrement	Right to left
13	++ -- + - ! ~ (type)	Unary pre-increment Unary pre-decrement Unary plus Unary minus Unary logical negation Unary bitwise complement Unary type cast	Right to left
12	* / %	Multiplication Division Modulus	Left to right
11	+ -	Addition Subtraction	Left to right
10	<< >> >>>	Bitwise left shift Bitwise right shift with sign extension Bitwise right shift with zero extension	Left to right
9	< <= > >= instanceof	Relational less than Relational less than or equal Relational greater than Relational greater than or equal Type comparison (objects only)	Left to right
8	== !=	Relational is equal to Relational is not equal to	Left to right
7	&	Bitwise AND	Left to right
6	^	Bitwise exclusive OR	Left to right
5	|	Bitwise inclusive OR	Left to right
4	&&	Logical AND	Left to right
3	||	Logical OR	Left to right
2	? :	Ternary conditional	Right to left
1	= += -= *= /= %=	Assignment Addition assignment Subtraction assignment Multiplication assignment Division assignment Modulus assignment	Right to left

Note: Larger the number higher the precedence

Command Line Arguments In Java

- Command line arguments are the arguments passed at run time to java program. Arguments are used as input for the program. Any number of command line arguments can be passed to the program.
- We need to pass the arguments as space-separated values. We can pass both strings and primitive data types(int, double, float, char, etc) as command-line arguments. These arguments convert into a string array and are provided to the main() function as a string array argument.
- When command-line arguments are supplied to JVM, JVM wraps these and supplies them to args[]. It can be confirmed that they are wrapped up in an args array by checking the length of args using args.length.
- Internally, JVM wraps up these command-line arguments into the args[ ] array that we pass into the main() function. We can check these arguments using args.length method. JVM stores the first command-line argument at args[0], the second at args[1], the third at args[2], and so on.
-

Example 1:CommandLine Argument

```
class CommandLine
{
    public static void main(String[] args)
    {
        String s=args[0];
        char ch=args[1].charAt(0);
        int i=Integer.parseInt(args[2]);
        long l=Long.parseLong(args[3]);
        float f=Float.parseFloat(args[4]);
        double d=Double.parseDouble(args[5]);
        boolean b=Boolean.parseBoolean(args[6]);
        System.out.println("value of s="+s);
        System.out.println("value of ch="+ch);
        System.out.println("value of i="+i);
        System.out.println("value of l="+l);
        System.out.println("value of f="+f);
        System.out.println("value of d="+d);
        System.out.println("value of b="+b);

    }
}
```

```
PS E:\exx\exx\src> javac AssignOp.java
PS E:\exx\exx\src> java CommandLine hello A 25 9898915690 22.5 22.5 true
```

```
/*Output:
value of s=hello
value of ch=A
value of i=25
value of l=9898915690
value of f=22.5
value of d=22.5
value of b=true
*/
```

Scanner Class in Java

- Scanner is a class in **java.util package** used for obtaining the input of the primitive types like int, double, etc. and strings. It is the easiest way to read input in a Java program, though not very efficient if you want an input method for scenarios where time is a constraint like in competitive programming.
- To create an object of Scanner class, we usually pass the predefined object System.in, which represents the standard input stream. We may pass an object of class File if we want to read input from a file.

Example:1 Java program to read data of various types using Scanner class.

```
import java.util.Scanner;
public class ScannerDemo1
{
    public static void main(String[] ars)

    {
        // Declare the object and initialize with
        // predefined standard input object
        Scanner sc = new Scanner(System.in);
        // String input
        System.out.print("Enter Name:");
        String name = sc.nextLine();

        // Character input
        System.out.print("Enter Gender (M/F):");
        char gender = sc.next().charAt(0);

        // Numerical data input
        // byte, short and float can be read
        // using similar-named functions.
        System.out.print("Enter Age:");
        int age = sc.nextInt();
```

```
            System.out.print("Enter Mobile.No:");
            long mobileNo = sc.nextLong();

            System.out.print("Enter CGPA:");
            double cgpa = sc.nextDouble();

             // Print the values to check if the input was correctly
    obtained.
            System.out.println("Name: "+name);
            System.out.println("Gender: "+gender);
            System.out.println("Age: "+age);
            System.out.println("Mobile Number: "+mobileNo);
            System.out.println("CGPA: "+cgpa);
        }
    }

    /* Input:
    Enter Name: ARP
    Enter Gender (M/F):M
    Enter Age:37
    Enter Mobile.No:9898915690
    Enter CGPA:98.98

    Output:
    Name: ARP
    Gender: M
    Age: 37
    Mobile Number: 9898915690
    CGPA: 98.98
    */
```

Math class and its Methods

```
class MathFun

{
    public static void main(String[] args)
    {
        int a=10;
        int b=20;
    //(1)public static int min(int a, int b)

        //Returns the smaller of two values.If the arguments have the same
        //value, the result is that same value.
        System.out.println(Math.min(a, b));//10

    //(2)public static int max(int a, int b)

        //Returns the greater of two values
        //If the arguments have the same value, the result is that same
value.
        System.out.println(Math.max(a, b));//20

        int c=2;
        int d=2;
    //(3)public static double pow(double a, double b)
        //Returns the value of the first argument raised to the power of the
second argument

        System.out.println(Math.pow(c,d));//4.0
        System.out.println((int)Math.pow(0,0));//1
        System.out.println(Math.pow(0,1));//0.0
        System.out.println(Math.pow(2,-1));//0.5
        System.out.println((int)Math.pow(2,-1));//0
        System.out.println(Math.pow(2,-1.5252));//0

    //(4)public static double sqrt(double a)
        //Returns the correctly rounded positive square root of a value.
        int e=16;
        System.out.println(Math.sqrt(e));//4.0
        System.out.println(Math.sqrt(9));//3.0
        System.out.println(Math.sqrt(-9));//NaN
        System.out.println(Math.sqrt(-0));//0.0

        System.out.println(Math.pow(Math.sqrt(-9),2));//NaN
        System.out.println(Math.pow(Math.sqrt(-9),0));//1.0

    //(5) public static double cbrt(double a)
    /* Returns the cube root of a value.the cube root of a negative value is
     the negative of the cube root of that value's magnitude. */

        int f=8;
        System.out.println(Math.cbrt(f));//2.0
        System.out.println(Math.cbrt(27));//3.0
        System.out.println(Math.cbrt(-27));//-3.0

    //(6) public static double ceil(double a)
        /*Returns the smallest value that is greater than or equal to the
        argument and is equal to a mathematical integer */
        System.out.println(Math.ceil(2));//2.0
        System.out.println(Math.ceil(2.1));//3.0
        System.out.println(Math.ceil(2.2));//3.0
        System.out.println(Math.ceil(2.9));//3.0
        System.out.println(Math.ceil(-2.9));//-2.0

    //(7) public static double floor(double a)
        /*Returns the largest value that is less than or equal to the
        argument and is equal to a mathematical integer. */
        System.out.println(Math.floor(2));//2.0
        System.out.println(Math.floor(2.1));//2.0
        System.out.println(Math.floor(2.9));//2.0
        System.out.println(Math.floor(-2.9));//-3.0

    //(8)public static final double PI = 3.14159265358979323846;
        // PI is the static final variable of Math class

        System.out.println(Math.PI);//3.141592653589793

    //(9) public static final double E = 2.7182818284590452354;
        // E is the static final variable of Math class

        System.out.println(Math.E);//2.718281828459045

    //(10) public static double abs(double a)
        /*Returns the absolute value of a value.
        If the argument is not negative, the argument is returned.
        If the argument is negative, the negation of the argument is
returned.*/
        System.out.println(Math.abs(10.0));//10.0
        System.out.println(Math.abs(-10.0));//10.0
        System.out.println(Math.abs(0.0));//0.0
        System.out.println(Math.abs(-0.0));//0.0

    //(11) public static double random()
    /*Returns a value with a positive sign, greater
     * than or equal to 0 and less than 1.*/

    System.out.println(Math.random());
    System.out.println((int)(Math.random()*10));
    System.out.println((int)(Math.random()*100));

    }
}
```