UNIT-1

➢ Procedural and object Oriented Programming

OOP	POP
1. Object oriented.	1. Structure oriented.
2. Program is divided into objects.	2. Program is divided into functions.
3. Bottom-up approach.	3. Top-down approach.
4. Inheritance property is used.	4. Inheritance is not allowed.
5. It uses access specifier.	5. It doesn't use access specifier.
6. Encapsulation is used to hide the data.	6. No data hiding.
7. Concept of virtual function.	7. No virtual function.
8. Object functions are linked through message passing.	8. Parts of program are linked through parameter passing.
9. Adding new data and functions is easy	9. Expanding new data and functions is not easy.
10. The existing code can be reused.	10. No code reusability.
11. use for solving big problems.	11. Not suitable for solving big problems.
12. C++, Java.	12. C, Pascal.

Benefits of OOPs

1. **Modularity**
   - OOP promotes the division of a software program into distinct modules or classes, each representing a specific component or functionality.
   - This modularity makes the code easier to manage, maintain, and understand.
2. **Reusability**
   - OOP encourages code reuse through inheritance and composition.
   - By creating new classes from existing ones, developers can leverage pre-existing code, reducing redundancy and effort.

3. **Encapsulation**
   - Encapsulation hides the internal implementation details of classes and exposes only the necessary interfaces.
   - This leads to better data protection and reduces the likelihood of unintended interference or misuse of data.
4. **Inheritance**
   - Inheritance allows new classes to inherit properties and methods from existing classes.
   - This promotes code reuse, establishes a natural hierarchy, and enables polymorphism.
5. **Polymorphism**
   - Polymorphism allows methods to behave differently based on the object invoking them, enabling flexibility and scalability in code.
   - It simplifies code maintenance and enhances the ability to extend the system with new functionalities without modifying existing code.
6. **Abstraction**
   - Abstraction simplifies complex systems by modeling classes appropriate to the problem domain.
   - It allows focusing on the essential features of an object, reducing complexity and improving code clarity.
7. **Improved Maintainability**
   - OOP's modularity and encapsulation make updating, modifying, and debugging code easier.
   - Changes in one part of the system have minimal impact on other parts, enhancing maintainability.
8. **Ease of Troubleshooting**
   - OOP simplifies the process of debugging and troubleshooting by isolating functionality into separate classes.
   - Problems can be localized and resolved more efficiently.
9. **Real-World Modeling**
   - OOP helps in the modeling of real-world entities and relationships in software.
   - This makes the code more intuitive and aligned with human understanding of the problem domain.
10. **Extensibility**
    - OOP's principles make it easier to extend and scale software systems.
    - New features and functionalities can be added with minimal disruption to existing code.

Applications of OOPs

Application Area	Description	Examples
Real-World Modeling	Modeling real-world entities as objects.	Simulation systems, games (e.g., flight simulators, RPG games)
Enterprise Applications	Structuring business logic and data processing.	Banking systems, CRM systems
GUI Applications	Creating graphical user interfaces.	JavaFX, Swing applications (e.g., desktop apps)
Web Applications	Building dynamic and interactive web applications.	Servlets, JSP, Spring Framework-based applications
Distributed Systems	Developing systems that run on multiple networked computers.	Java RMI, microservices architectures

Mobile Applications	Developing applications for mobile devices.	Android applications
Software Development Tools	Creating tools and environments for software development.	Integrated Development Environments (IDEs) like IntelliJ IDEA, Eclipse, NetBeans
Scientific Applications	Performing scientific simulations and data analysis.	Simulation and modeling software, scientific computing libraries
Embedded Systems	Developing applications for embedded devices.	Java ME (Micro Edition) applications
Middleware Products	Building middleware that provides services to applications beyond those available from the operating system.	Application servers like Apache Tomcat, JBoss, WebLogic

➢ OOPs Concept in JAVA

- As the name suggests, Object-Oriented Programming or Java OOPs concept refers to languages that use objects (Objects - real-world entities like book, vehicle, tree, etc..)in programming
- **Object-Oriented Programming** is a methodology or paradigm to design a program using classes and objects. It simplifies software development and maintenance by providing some concepts:
    - Object
    - Class
    - Abstraction
    - Inheritance
    - Encapsulation
    - Polymorphism

Object: An Object can be defined as an instance of a class. Or Any entity that has state and behavior is known as an object. An object contains an address and takes up some space in memory.

- **State:** represents the data (value) of an object.
- **Behavior:** represents the behavior (functionality) of an object such as deposit, withdraw, etc.
  Ex: 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 or template from which you can create an individual object. Class doesn't consume any space.

```
class Sample
{
        //data members (or)
                variables

        //member functions (or)
                methods
}
```

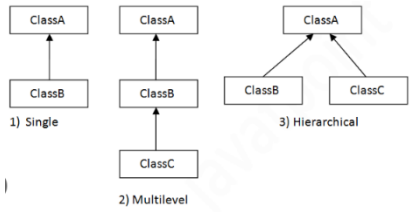
Principles or pillars of oops

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.

```
class Sample
{
        void msg();
        void display();
}

interface Demo
{
        void msg1();
        void display1();
}
```

Inheritance: Inheritance in java is a mechanism in which one object acquires all the properties and behaviours of a parent object. It is an important part of OOPs (Object Oriented programming system).



Encapsulation: Encapsulation in Java is a process of wrapping code and data together into a single unit, for example, a capsule which is mixed of several medicines. Java Class is an Example of Encapsulation.

```
class Sample
{
        //variables...
        //methods.....
}
```

Polymorphism: Polymorphism in Java is a concept by which we can perform a single action in different ways. Polymorphism is derived from 2 Greek words: poly and morphs. The word "poly" means many and "morphs" means forms. So polymorphism means many forms. There are two types of polymorphism in Java: compile-time polymorphism(method overloading) and runtime polymorphism(method overiding).

```
class Sample
{
        void add(arguments list) { }
        int add(arguments list)  { }
}
```

➢ Overview of java:

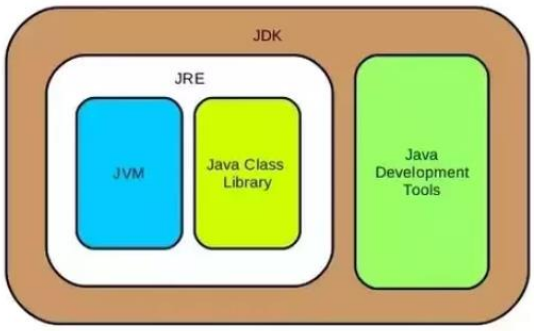
Java was originally designed for interactive television, but it was too advanced technology for the digital cable television industry at the time. The history of Java starts with the Green Team. Java team members (also known as Green Team), initiated this project to develop a language for digital devices such as set-top boxes, televisions, etc. However, it was best suited for internet programming.

Currently java mainly used in Mobile Applications, Gaming, Desktop Applications, Internet programming, Web applications, robotics, Enterprise applications etc

History of Java

- James Gosling, Mike Sheridan, and Patrick Naughton initiated the Java language project at Sun Microsystems in June 1991. The small team of sun engineers called Green Team.
- Firstly, it was called "Greentalk" by James Gosling, and the file extension was .gt.
- After that, it was called Oak. Oak is a symbol of strength and chosen as a national tree of many countries like the U.S.A., France, Germany, etc.
- In 1995, Oak was renamed as "Java".

➢ Java Environment



JDK:
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.
Components of jdk:

- Applet viewer: This tool is used to run and debug Java applets without a web browser.
- Javac: It specifies the Java compiler, which converts source code into Java bytecode.
- java: This tool is an interpreter and can interpret the class files generated by the javac compiler.

JRE:
Java Run-time Environment (JRE) is the part of the Java Development Kit (JDK). It has Java Class Library, specific tools, and JVM. It is the most common environment available on devices to run java programs. The source Java code gets compiled and converted to Java bytecode. If you wish to run this bytecode on any platform, you require JRE. The JRE loads classes, verify access to memory, and retrieves the system resources. JRE acts as a layer on the top of the operating system.
Working of JRE with JVM:
JRE has an instance of JVM with it, library classes and development tools.
Once you write your program, you have to save it with .java extension. Compile your program. The output of the Java compiler is a byte-code which is platform independent. After compiling, the compiler generates a .class file which has the bytecode. The bytecode is platform independent and runs on any device having the JRE.
JVM:
Java Virtual Machine is an abstract machine in which java bytecode can be executed.Java applications are WORA(Write Once Run Anywhere) this means a programmer can develop a java code on one system and can expect it to run on any other system.

➢ Features/Buzz words of Java

The primary objective of Java programming language creation was to make it portable, simple and secure programming language. Apart from this, there are also some excellent features as " Robust, Platform-independent, High Performance, Multithreaded, Architecture Neutral, Object-Oriented,

Interpreted, and Dynamic" which play an important role in the popularity of this language. The features of Java are also known as Java buzzwords.

- Simple
  - ✓ Java is very easy to learn, and its syntax is simple, clean and easy to understand.
  - ✓ Java has removed many complicated features, for example, pointers, operator overloading, Const, goto etc.
- Object-Oriented
  - ✓ Almost everything written in Java is object and class, making it a true object-oriented programming (OOP) language. The basic concept of OOPs are Class, object, Abstrction, Inheritance, Encapsulation and Polymorphism.
- Portable
  - ✓ Java is portable because it facilitates you to carry the Java bytecode to any platform. It doesn't require any implementation.
- Platform independent
  - ✓ Java is platform independent because it is different from other languages like C, C++, etc. which are compiled into platform specific machines while Java is a write once, run anywhere language (Java code is compiled by the compiler and converted into bytecode. This bytecode is a platform-independent code because it can be run on multiple platforms). A platform is the hardware or software environment in which a program runs.
- Secured
  - ✓ Java is best known for its security. With Java, we can develop virus-free systems. Java is secured because:
    - No explicit pointer
    - Java Programs run inside a virtual machine
- Robust
  - ✓ It uses strong memory management.
  - ✓ There is a lack of pointers that avoids security problems.
  - ✓ Java provides automatic garbage collection which runs on the Java Virtual Machine to get rid of objects which are not being used by a Java application anymore.
  - ✓ There are exception handling and the type checking mechanism in Java.
- Architecture neutral
  - ✓ Java is architecture neutral because there are no implementation dependent features, for example, the size of primitive types is fixed.
  - ✓ In C programming, int data type occupies 2 bytes of memory for 32-bit architecture and 4 bytes of memory for 64-bit architecture. However, it occupies 4 bytes of memory for both 32 and 64-bit architectures in Java.
- Interpreted
  - ✓ Java is a self Interpreted Language
- High Performance
  - ✓ Java is faster than other traditional interpreted programming languages because Java bytecode is "close" to native code.
- Multithreaded
  - ✓ A thread is like a separate program, executing concurrently. We can write Java programs that deal with many tasks at once by defining multiple threads.
  - ✓ The main advantage of multi-threading is that it doesn't occupy memory for each thread. It shares a common memory area.

- Distributed
  - ✓ Java is distributed because it facilitates users to create distributed applications in Java.
  - ✓ Enterprise Java Beans (EJB) are used for creating distributed applications.
- Dynamic
  - ✓ Java is a dynamic language. It supports the dynamic loading of classes. It means classes are loaded on demand.

Simple java program:

- Step1: open notepad or text editor and write the code

```
class SimpleProgram
{
        public static void main(String args[])
        {
                System.out.println("Welcome to JAVA programming");
        }
}
```

- Step2: Save the file with .java extension and the file name must be the name of the class
  Ex: SimpleProgram.java
- Step3: For compilation open the command prompt then type javac filename.java
  Ex: javac SimpleProgram.java
- Step4: For execution type java filename
  Ex: java SimpleProgram
  Output:

```
E:\JAVA>javac SimpleProgram.java

E:\JAVA>java SimpleProgram
Welcome to JAVA programming
```

Closer look to java program

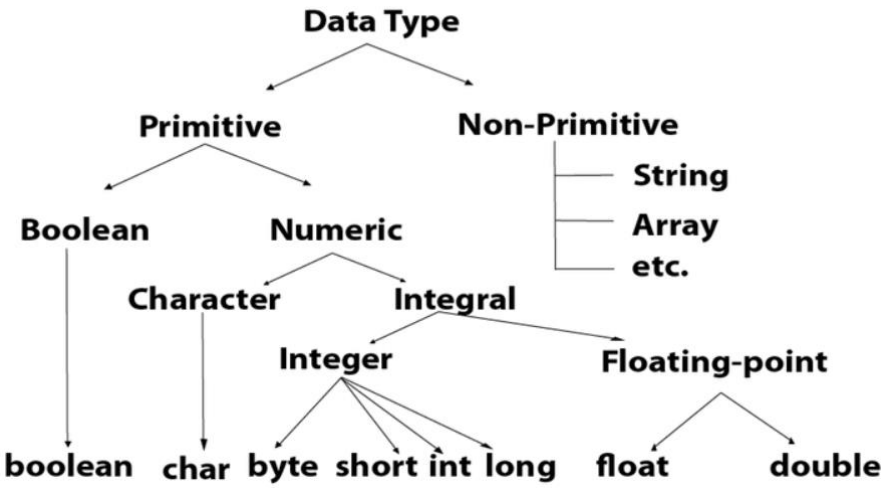
- **class ClassName**: Here class is a keyword used to declare a class in java. Every java program must have atleast one class definition. A class is a group of objects
- **public**: It makes the main method public and we can call the methd from outside of the class. Public is a keyword and access specifier it means visibility to all.
- **static**: A static is a keyword. If we declare any method or variable as static we need not to create any objects to those because we can directly access them by using classname itself.
- **void**: It is a return type. void means it does not return any value.
- **main()**: main() is a heart of any program. It represents starting point of the program
- **String args[]**: It can be used for command line arguments that are passed as strings at runtime.
- **System**.out.println(): This method prints the content inside the double quotes. It is available in java.lang.System package.
- **new:** new is a keyword which is used to create objects in java
  Syntax: ClassName obj_name = new ClassName()
  Ex: SimpleProgram sp = new SimpleProgram()

Data types: A data type is a keyword which can be used to identify the type of data

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.



Primitive data types:

These are the most basic data types available in Java language. There are 8 types of primitive data types

byte:

- Used to store integer values
- Size is 1 byte (8 bit representation)
- Minimum value is -128 (-2^7), Maximum value is +127 ((2^7)-1)
- Default value is 0

Ex: byte a=100, b=-50;
byte c =-147 //Error

short:

- Used to store integer values beyond byte
- Size is 2 bytes (16 bit representation)
- Minimum value is -32,768 (-2^15), Maximum value is +32,767 ((2^15)-1)
- Default value is 0

Ex: short a=2156;
short b= 40513 //Error

int:

- Used to store integer values beyond byte and short
- Size is 4 bytes (32 bit representation)
- Minimum value is - 2,147,483,648 (-2^31), Maximum value is +2,147,483,647 ((2^31)-1)
- Default value is 0
  Ex: int a=25679;

long:

- Used to store integer values beyond byte, short and int.
- Size is 8 bytes (64 bit representation)
- Minimum value is -9,223,372,036,854,775,808(-2^63), Maximum value is -9,223,372,036,854,775,807 ((2^63)-1)
- Default value is 0
  Ex: int a=2147483649;

float:

- Used to store floating point numbers.

- Size is 4 bytes
- Default value is 0.0f
  Ex: int a=21.936f;

double:

- Used to store floating point numbers.
- Size is 8 bytes
- Default value is 0.0d
  Ex: int a=11.36d;

boolean:

- used to store only two possible values: true and false
- Size is 1 bit
- Default value is false
  Ex: boolean flag =true;

char:

- used to store a single character
- Size is 2 bytes
- Default value is Null or \0
  Ex: char ch= ‘a’ , b= ’#’;

Java Program on data types

```
class DataTypes
{
        public static void main(String args[])
        {
                byte a=126;
                short b=1000;
                int c=873527;
                long d=100000000;
                float e=45.67f;
                double f=510.643;
                boolean g=true;
                char h='H';
                System.out.println("byte : "+a);
                System.out.println("short : "+b);
                System.out.println("int : "+c);
                System.out.println("long : "+d);
                System.out.println("float : "+e);
                System.out.println("double : "+f);
                System.out.println("boolean : "+g);
                System.out.println("char : "+h);


        }
}
```

```
E:\JAVA>javac DataTypes.java

E:\JAVA>java DataTypes
byte : 126
short : 1000
int : 873527
long : 100000000
float : 45.67
double : 510.643
boolean : true
char : H
```

Variables

- Variables are the data containers that save the data values during Java program execution. Every Variable in Java is assigned a data type that designates the type and quantity of value it can hold.
- Variables in Java are only a name given to a memory location.
- It is a combination of "vary + able" which means its value can be changed.
- There are three types of variables in java: local variables, instance variables and static variables.

Declaration : datatype variable_name;

Ex : int a;

Initialization : datatype variable_name = value;

Ex : int a = 20;

Local Variable:

- A variable which is declared inside the block or method or constructor is called local variable.
- Local variable memory is created in stack memory.
- Scope of a local variable is within the block in which it is declared and the lifetime of a local variable is until the control leaves the block in which it is declared.
- We cannot access the local variables of one method to another method.

Instance Variable:

- Instance variable is a non static variable which is declared inside the class and outside the block or method or constructor.
- The scope of an instance variable is throughout the class except in static methods. The lifetime of an instance variable is until the object stays in memory and are created when an object of a class is created.
- Unlike local variables we can use access specifiers for instance variables. If we don't specify any access specifier, the default access specifier is used.

Static variable

- The static variables also called as class variables these are similar to instance variables the only difference is that static variables are declared by using **static** keyword inside the class and outside the block or method or constructor.
- The scope of a static variable is throughout the class and the lifetime of a static variable is until the end of the program and are created while the execution of the program (.class file is loading into memory).
- To access static variables we need not to create any object for the class we can simply access the variables directly or by using the class name as follows
  Ex: ClassName.variable_name
  Ex: static int a=10;

Java Program on types of variables

```
class Variables
{
        int a=20;               //instance variable
        static int b=30;        //static variable
        void msg()
        {
                int c=40;       //local variable
                System.out.println(c);
        }
        public static void main(String args[])
        {
                Variables v=new Variables();

                System.out.println(v.a);
                System.out.println(Variables.b);
                v.msg();
        }
}
```

```
E:\JAVA>javac Variables.java

E:\JAVA>java Variables
20
30
40
```

Scope of a variable: The variable's scope refers to the region where they are created and accessed in a given program or function. The variable scope also refers to its lifetime.

Lifetime of a variable: The lifetime of a variable indicates how long the variable stays alive in the memory.

Constants in java:

- A constant is an entity in programming that is immutable.
- Constant is a value that cannot be changed after assigning it. Java does not directly support the constants. There is an alternative way to define the constants in Java by using the non-access modifiers static and final.
- to declare any variable as constant, we use static and final modifiers. It is also known as non-access modifiers. According to the Java naming convention the identifier name must be in capital letters.

Static and Final Modifiers

- The purpose to use the static modifier is to manage the memory.
- It also allows the variable to be available without loading any instance of the class in which it is defined.
- The final modifier represents that the value of the variable cannot be changed. It also makes the primitive data type immutable or unchangeable.

  Syntax: static final datatype constant_var_name = value;

  Ex: static final int MRP = 999;

Program:

```
class Constant
{
        static final int MRP = 499;
        // MRP=999;  - ERROR
        public static void main(String args[])
        {
                System.out.println("The cost is "+MRP);
        }
}
```

```
E:\JAVA>javac Constant.java

E:\JAVA>java Constant
The cost is 499
```

Java Tokens

In Java, the program contains classes and methods. Further, the methods contain the expressions and statements required to perform a specific operation. These statements and expressions are made up of tokens. In other words, we can say that the expression and statement is a set of tokens. The tokens are the small building blocks of a Java program that are meaningful to the Java compiler. Further, these two components contain variables, constants, and operators.

What is token?

The Java compiler breaks the line of code into text (words) is called Java tokens. These are the smallest element of the Java program. The Java compiler identified these words as tokens. These tokens are separated by the delimiters. It is useful for compilers to detect errors. Remember that the delimiters are not part of the Java tokens.

Types of Tokens

- Keywords
- Identifiers
- Literals
- Operators
- Separators
- Comments

- ✓ **Keywords:** These are the pre-defined reserved words of any programming language. Each keyword has a special meaning. It is always written in lower case. Java provides the following keywords:

01. abstract	02. boolean	03. byte	04. break	05. class
06. case	07. catch	08. char	09. continue	10. default
11. do	12. double	13. else	14. extends	15. final
16. finally	17. float	18. for	19. if	20. implements
21. import	22. instanceof	23. int	24. interface	25. long
26. native	27. new	28. package	29. private	30. protected
31. public	32. return	33. short	34. static	35. super
36. switch	37. synchronized	38. this	39. thro	40. throws
41. transient	42. try	43. void	44. volatile	45. while
46. assert	47. const	48. enum	49. goto	50. strictfp

- ✓ **Identifier**: Identifiers are used to name a variable, constant, function, class, and array. It usually defined by the user. It uses letters, underscores, or a dollar sign as the first character. Remember that the identifier name must be different from the reserved keywords. There are some rules to declare identifiers are:
  - The first letter of an identifier must be a letter, underscore or a dollar sign. It cannot start with digits but may contain digits.
  - The whitespace cannot be included in the identifier.
  - Identifiers are case sensitive.

    Ex: a, a1, PRICE, _name, $email, firstName

    1a //invalid

- ✓ **Literals**: literal is a fixed value (constant). Assigning a value to the variable is called literal. It can be categorized as an integer literal, string literal, Boolean literal, etc. It is defined by the programmer. Java provides five types of literals are as follows:
  - **Integer**: Assigning an integer value to the variable is called integer literal. These are 4 types such as byte, short, int and long.

    Ex: byte a =123

    short b =243;

    int c = 837;

    long d =38267;
  - **Floating Point**: Assigning a floating point value to the variable is called floating point literal. These are 2 types such as float and double.

    Ex: float f = 3.14f

double d = 365.98

- **Character**: Assigning a single character to the variable is called character literal.
  Ex: char c ='s'
  char ch= '#'
- **String**: Assigning a group of charecters to the variable is called string literal.
  Ex: String str = "hello"
- **Boolean:** Assigning true or false value to the variable is called boolean literal.
  Ex**:** boolean b = true

✓ **Operators:** In programming, operators are the special symbol that tells the compiler to perform a specific operation. Java provides different types of operators that can be classified according to the functionality they provide. There are eight types of operators in Java, are as follows:

- Arithmetic Operators
- Assignment Operators
- Relational Operators
- Unary Operators
- Logical Operators
- Ternary Operators
- Bitwise Operators
- Shift Operators

Operator	Symbols
Arithmetic	+ , - , / , * , %
Unary	++ , - -
Assignment	= , += , -= , *= , /= , %= , ^=
Relational	==, != , < , >, <= , >=
Logical	&& , || , !
Ternary	(Condition) ? (Statement1) : (Statement2);
Bitwise	& , | , ^ , ~ , << , >> , >>>

Java Arithmetic Operators:
Arithmetic operators are used to perform arithmetic operations on variables and data.
Ex: int a=4,b=2

Operator	Example	Output
+	a+b	6
-	a-b	2
*	a*b	8
/	a/b	2
%	a%b	0

Program:

```
class Arithmatic
{
        public static void main (String[] args)
        {
                int a = 10;
                int 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));
        }
}
```

```
E:\JAVA>javac Arithmatic.java

E:\JAVA>java Arithmatic
a + b = 13
a - b = 7
a * b = 30
a / b = 3
a % b = 1
```

Unary operators

++ : 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.

```
class Unary
{
    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));
    }
}
```

```
E:\JAVA>javac Unary.java

E:\JAVA>java Unary
Postincrement : 10
Preincrement : 12
Postdecrement : 10
Predecrement : 8
```

Assignment operators

- ‘=’ Assignment operator is used to assign a value to any variable.
- The assignment operator can be combined with other operators to build a shorter version of the statement called a Compound Statement. For example, instead of a = a+5, we can write a += 5. The following are assignment operators.

  = , += , -= , *= , /= , %=

```
class Assignment
{
    public static void main(String[] args)
    {
        int f = 7;
        System.out.println("f += 3: " + (f += 3));
        System.out.println("f -= 2: " + (f -= 2));
        System.out.println("f *= 4: " + (f *= 4));
        System.out.println("f /= 3: " + (f /= 3));
        System.out.println("f %= 2: " + (f %= 2));
    }
}
```

```
E:\JAVA>javac Assignment.java

E:\JAVA>java Assignment
f += 3: 10
f -= 2: 8
f *= 4: 32
f /= 3: 10
f %= 2: 0
```

Relational Operators

These operators are used to check for relations like equality, greater than, and less than. They return boolean results after the comparison and are extensively used in looping statements as well as conditional if-else statements. The following are relational operators.

== Equal to
!= Not Equal
< Less than
> Greater than
<= Less than Equal to
>= Greater than Equal to

```
class Relational
{
    public static void main(String[] args)
    {
        int a = 10;
        int b = 3;
        int c = 5;
        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 == c: " + (a == c));
        System.out.println("a != c: " + (a != c));
    }
}
```

```
E:\JAVA>javac Relational.java

E:\JAVA>java Relational
a > b: true
a < b: false
a >= b: true
a <= b: false
a == c: false
a != c: true
```

Logical Operators

These operators are used to perform "logical AND" and "logical OR" operations, i.e., a function similar to AND gate and OR gate in digital electronics.

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

```
class Logical
{
    public static void main(String[] args)
    {
        boolean x = true;
        boolean y = false;
        System.out.println("x && y: " + (x && y));
        System.out.println("x || y: " + (x || y));
        System.out.println("!x: " + (!x));
    }
}
```

```
E:\JAVA>javac Logical.java

E:\JAVA>java Logical
x && y: false
x || y: true
!x: false
```

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 ? true : false

```
class Ternary
{
    public static void main(String[] args)
    {
        int a = 20,b = 30,max;
        max= (a > b)? a : b;
        System.out.println("Max number is = "+max);
    }
}
```

```
E:\JAVA>javac Ternary.java

E:\JAVA>java Ternary
Max number is = 30
```

Bitwise Operators

These operators are used to perform the manipulation of individual bits of a number.

- ✓ &, 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 Bitwise
{
    public static void main(String[] args)
    {
        int d = 0b1010;
        int e = 0b1100;
        System.out.println("d & e: " + (d & e));
        System.out.println("d | e: " + (d | e));
        System.out.println("d ^ e: " + (d ^ e));
        System.out.println("~d: " + (~d));
        System.out.println("d << 2: " + (d << 2));
        System.out.println("e >> 1: " + (e >> 1));
        System.out.println("e >>> 1: " + (e >>> 1));
    }
}
```

```
E:\JAVA>javac Bitwise.java

E:\JAVA>java Bitwise
d & e: 8
d | e: 14
d ^ e: 6
~d: -11
d << 2: 40
e >> 1: 6
e >>> 1: 6
```

- ✓ **Separators**: The separators in Java is also known as punctuators. There are nine separators in Java, are as follows:

  Separator <= ; | , | . | (|) | { | } | [|]

- ✓ **Comments**: Comments can be used to explain Java code, and to make it more readable. The Java compiler treats comments as whitespaces. Java provides the following 3 types of comments:
  - Single line comment: The single-line comment is used to comment only one line of the code. It is the widely used and easiest way of commenting the statements. Single-line comments start with two forward slashes (//).
  - Multi line comment: The multi-line comment is used to comment multiple lines of code. It can be used to explain a complex code snippet or to comment multiple lines of code at a time. Multi-line comments are placed between /* and */.
  - Documentation comment: Documentation comments are usually used to write large programs for a project or software application as it helps to create documentation API. The documentation comments are placed between /** and */.

  Program on comments:

```
class Comment
{
        public static void main(String args[])
        {
                //this is SINGLE LINE COMMENT
                System.out.println("SINGLE LINE COMMENT");

                /*
                line 1
                line 2
                line 3
                */
                System.out.println("MULTI LINE COMMENT");

                /**
                *line 1
                *line 2
                *line 3
                */
                System.out.println("DOCUMENTATION COMMENT");
        }
}
```

```
E:\JAVA>javac Comment.java

E:\JAVA>java Comment
SINGLE LINE COMMENT
MULTI LINE COMMENT
DOCUMENTATION COMMENT
```

Type casting:

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.

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.
- A boolean value cannot be assigned to any other data type. Except boolean, all the remaining 7 data types can be assigned to one another either implicitly or explicitly. but boolean cannot.

byte -> short -> char -> int -> long -> float -> double

```
class TypeConv
{
        public static void main(String args[])
        {
                int a=10;
                long b=a;
                float c=b;
                System.out.println("before conversion = "+a);
                System.out.println("int -> long = "+b);
                System.out.println("long -> float = "+c);
        }
}
```

```
E:\JAVA>javac TypeConv.java

E:\JAVA>java TypeConv
before conversion = 10
int -> long = 10
long -> float = 10.0
```

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.

double -> float -> long -> int -> char -> short -> byte

string to numeric : Integer.valueOf(), Integer.parseInt()

Float.valueOf(),Float.parseFloat()

Numeric to string: String.valueOf(),Integer.toString()

program:

```
class TypeConv
{
        public static void main(String args[])
        {
                double a=199203.988;
                int b=(int)a;

                System.out.println("before conversion = "+a);
                System.out.println("double -> int = "+b);
        }
}
```

```
E:\JAVA>javac TypeConv.java

E:\JAVA>java TypeConv
before conversion = 199203.988
float -> int = 199203
```

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.
- So, it provides a convenient way to check the behavior of the program for the different values. You can pass N (1,2,3 and so on) numbers of arguments from the command prompt.
  Program

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

```
E:\JAVA>javac CommandLine.java

E:\JAVA>java CommandLine hiiiii...
first argument is: hiiiii...
```

- In the above example, we are receiving only one argument and printing it. To run this java program, we must pass at least one argument from the command prompt.

```
class MulCommandLine
{
        public static void main(String args[])
        {
                for(int i=0;i<args.length;i++)
                        System.out.println(args[i]);
        }
}
```

```
E:\JAVA>javac MulCommandLine.java

E:\JAVA>java MulCommandLine hey hi 12 34.8
hey
hi
12
34.8
```

- In this example, we are printing all the arguments passed from the command-line. For this purpose, we have traversed the array using for loop.

Java Scanner

- Scanner class in Java is found in the java.util package. Java provides various ways to read input from the keyboard, the java.util.Scanner class is one of them.
- The Java Scanner class is widely used to parse text for strings and primitive types using a regular expression. It is the simplest way to get input in Java.

- By the help of Scanner in Java, we can get input from the user in primitive types such as int, long, double, byte, float, short, etc.
- The Java Scanner class provides nextXXXXX() methods to return the type of value such as nextInt(), nextByte(), nextShort(), next(), nextLine(), nextDouble(), nextFloat(), nextBoolean(), etc. To get a single character from the scanner, you can call next().charAt(0) method which returns a single character.

```
import java.util.Scanner;
class StreamScanner
{
        public static void main(String args[])
        {
                Scanner sc = new Scanner(System.in);
                System.out.print("Enter your name: ");
                String name = sc.nextLine();
                System.out.println("Name is: " + name);
        }
}
```

```
E:\JAVA>javac StreamScanner.java

E:\JAVA>java StreamScanner
Enter your name: mike
Name is: mike
```

Evaluation of expressions

- Java expression is a combination of variables, constants, literals, operators, and method invocations that can be evaluated to produce a single value.
- Expressions are the building blocks of Java code, and they are used to perform calculations, comparisons, assignments, and other operations.
- **Types of Expressions**:
  Infix Expression – operators are used between operands (2+3)
  Postfix Expression – operators are used after operands (2 3 +)
  Prefix Expression – operators are used before operands (+ 2 3)

operator precedence

- In an expression, it determines the grouping of operators with operands and decides how an expression will evaluate.
- While solving an expression two things must be kept in mind the first is a **precedence** and the second is **associativity**.

Precedence

- Precedence is the priority for grouping different types of operators with their operands. It is meaningful only if an expression has more than one operator with higher or lower precedence. The operators having higher precedence are evaluated first. If we want to evaluate lower precedence operators first, we must group operands by using parentheses and then evaluate.

Associativity

We must follow associativity if an expression has more than two operators of the same precedence.In such a case, an expression can be solved either **left-to-right** or **right-to-left,** accordingly.

Precedence	Operator	Associativity
3	()	Left to right
2	* / %	Left to right
1	+ -	Left to right

Ex: 1 + 5 * 3 ->1+15 -> 16
4-2+5 ->2+5 ->7
2*7*3 -> 14*3 ->42