UNIT-1

INTRODUCTION TO JAVA PROGRAMMING LANGUAGE

- **JAVA** is an **Object-Oriented & High-Level Programming Language.**
- **JAVA** is **Platform Independent Programming Language.** Means it can run on multiple platforms like:- Windows; Mac; Linux; Unix.
- **JAVA** is from **IDE:- Integrated Development Environment.**
- In this 3 types of programs are there. They are:-

➲ **Stand-aline Application Programs:-** These are made & run on users pc's.

➲ **Applet Programs:**- These are meant to run in a web page on browser.

➲ **Java Servlets:**- These can run in pc's that provide web services. They are also called as **Server side programs** or **Servlets.**

WRITING SIMPLE PROGRAMS IN JAVA:-

In java every program **imports packages** which are provides necessary **classes & interfaces.**

Example Program:-

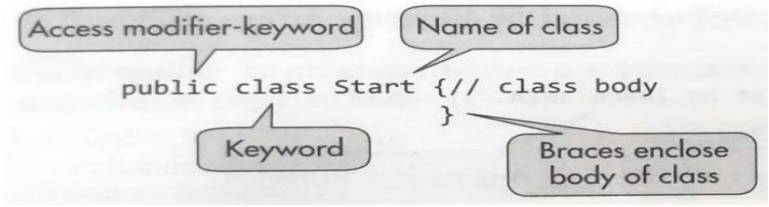
```
import java.util.*;

import java.io.*;
```

After import statement, every java program starts with the declaration of the class. A program may have one or more classes.

A class declaration starts with the keyword class, followed by the identifier or name of the class. Giving the name of a package at the top is optional.

Class declaration contains name of the class and body of the class. The body of the class may consist of several statements and is enclosed between the braces {}.

DECLARATION OF SAMPLE CLASS:-



Here:-

Public is access **specifier.** This **class is accessible to any outside code**. **Otherwise** the **class is accessibe to only same package classes**.

Class is a **keyword of java** which is **used to declare the class**. The class body starts with "**{"** & ends with **"}"**. **"//"** are **comments** which are **neglected by the compiler**.

A class body may comprise statements for declaration of variables, constants, expressions & methods. Sample java program have to save as:- **"Start.java".**

```
public class Start()
{
   public static void main()
   {
         System.out.println("Hello World");
   }
}
```

Here:-

Class is a **Keyword.**

Start is a **Name of Class.** Here class is declared as public, So it is applicable to all classes.

Main() is method which initiate & terminate the program. In java Functions are called as Methods.

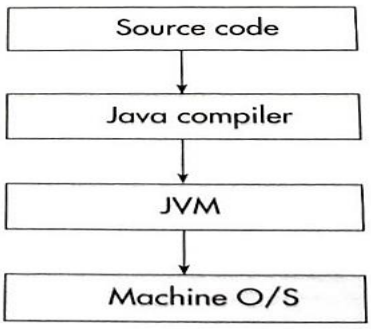
Println() is method of object **"out". "out"** is the object of class **"system".**

Println() prints the string **"Hello World".**

COMPILING & RUNNING JAVA PROGRAM:-

Java compiler 1st converts the source code into Byte or Virtual Machine Code.

To run the byte code, we need the **"Java Virtual Machine (JVM)".** JVM exists only inside the computer memory and runs on top of the Operating System. The byte code is not machine specific. The Java interpreter converts the byte code into Machine code.



For compiling the program, the Java compiler javac is run, specifying the name of the source file on the command line as depicted here:

javac Start.java. The Java compiler creates a file called **"Start.class"** containing the byte code version of the program. The java interpreter in JVM executes the instructions contained in this intermediate Java byte code version of the program. The Java interpreter is called with **"java"** at the command prompt.

C:\> java Start

Output: ***Hello World***

Here java command calls the Java interpreter which executes the Start.class (byte code of Start.java).

ELEMENTS OR TOKENS IN JAVA:-

It is time to more formally describe the atomic elements (Tokens) of Java. Java programs are a collection of **"identifiers, literals, comments, operators, separators & keywords".**

Identifiers:- Identifier is the name of variables, methods, classes etc.

- ♣ It should be a single word which contains **alphabets a to z or A to Z, digits 0 to 9, underscore "_" & dollar "$".**
- ♣ It should not contain **white spaces & special symbols.**
- ♣ It should not be a **keyword of Java.**
- ♣ It should not be **Boolean literal** (true or false) or **not a null literal.**
- ♣ It should **not start with a digit** but it **can start with an underscore.**
- ♣ Names of packages are completely in lower-case letters such as mypackage, java.lang.
- ♣ Names of classes and interfaces start with an upper-case letter.
- ♣ Names of methods & variables start with a lower-case character.

Literals:- Literal (constant) These are values represented by a set of character, digits, etc.

A literal represents a value which may be of primitive type, String type, or null type.

Types of literals:- **1.** Integer literals **2.** Floating point literal **3.** Boolean literal **4.** Character literal **5.** String literal **6.** Null literal.

Comments:-

- Comments are Line of Text which is not a part of the compiled program.
- These are used for documentation to explain source code & are added to the source code of the program.
- These are the statements that are ignored by compiler or interpreter. It is used to explore about variables, methods, classes or any statements.
- It can also be used to hide program code for a specific time.

There are 3 types of comments in java. **1.** Single line comments **2.** Multi line comments **3.** Documentation comments.

- **Single line Comments:-** These comments are started with two front slash characters **//**.
- **Multi-line Comments:-** These comments are enclosed with **/* and */.**

- **Documentation Comments:-** It can extracted by javadoc utility to generate an HTML document for the program. These comments are enclosed with **/** and */**.

Operators:- Operators are mostly represented with special symbols that performs specific operations on 1,2 or 3 operands & then returns a result.

The Operators are:-

- **Arithmetic Operators:-** These are used to perform arithmetic operators on variables & data types.

Operator	Name	Description	Example
+	Addition	Adds together two values	x + y
-	Subtraction	Subtracts one value from another	x - y
*	Multiplication	Multiplies two values	x * y
/	Division	Divides one value by another	x / y
%	Modulus	Returns the division remainder	x % y

Example:-

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

- **Relational Operators:-** These are used to check the relationship between two operands.

Operator	Name	Example
==	Equal to	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Example:-

```
public class Main {
      public static void main(String[] args)
    {
          int x = 5;
          int y = 3;
          System.out.println(x == y);   // returns false because 5 is not equal to 3
    }
}
```

- **Logical Operators:-** It is a symbol or word used to connect two or more expressions.

Operator	Description	Example
&& (and)	True if both the operands is true	a<10 && a<20
|| (or)	True if either of the operands is true	a<10 || a<20
! (not)	True if an operand is false (complements the operand)	!(x<10 && a<20)

Example:-

```
public class JavaOperators {
      public static void main(String[] args) {
            int a = 10;
            boolean b=true;
        System.out.println(a++);  //returns 11
        System.out.println(++a);
        System.out.println(a--);
        System.out.println(--a);
        System.out.println(!b); // returns false
          }
}
```

- **Assignment Operators:-** These are used to assign values to variables.

Operator	Description	Example
=	Assigns values from right side operands to left side operand	c = a + b
+=	It adds right operand to the left operand and assigns the result to left operand	c += a
-=	It subtracts right operand from the left operand and assigns the result to left operand	c -= a
*=	It multiplies right operand with the left operand and assigns the result to left operand	c *= a
/=	It divides left operand with the right operand and assigns the result to left operand	c /= a
%=	It takes modulus using two operands and assigns the result to left operand	c %= a
^=	Performs exponential (power) calculation on operators and assign value to the left operand	c ^= a

Examples:-

```
public class JavaOperators {
	public static void main(String[] args) {
		int a = 10;
		int b=20;
		int c;
		System.out.println(c = a); // Output =10
		System.out.println(b += a);// Output=30
		System.out.println(b -= a);// Output=20
		System.out.println(b *= a);// Output=200
		System.out.println(b /= a);// Output=2
		System.out.println(b %= a);// Output=0
		System.out.println(b ^= a);// Output=0
	}

}
```

- **Increment or Decrement Operators:-** These are known as **unary operators** to add or subtract from their operand. In this there are 2 types **1.** Pre Increment & Decrement **2.** Post Increment & Decrement.

Operator	Description	Example
++	increments the value by 1. There is post-increment and pre-increment operators	a++ and ++a
—	decrements the value by 1. There is post decrement and pre decrement operators	a– or –a
!	invert a boolean value	!a

Example:-

```
public class JavaOperators {
	public static void main(String[] args) {
		int a = 10;
		boolean b=true;
	  System.out.println(a++);  //returns 11
	  System.out.println(++a);
	  System.out.println(a--);
	  System.out.println(--a);
	  System.out.println(!b); // returns false
	    }
}
```

- **Bitwise Operators:-** These operators compares the values in binary format of each operand & yields a value whose bit pattern shows which bits in either of operands has the value 1. If both of the bits are 0, the results of that bit is 0; otherwise, the result is 1.

Operator	Description	Example
& (AND)	returns bit by bit AND of input	a&b
| (OR)	returns OR of input values	a|b
^ (XOR)	returns XOR of input values	a^b
~ (Complement)	returns the one's complement. (all bits reversed)	~a

Example:-

```
public class JavaOperators {
      public static void main(String[] args) {
            int a = 58; //111010
            int b=13; //1101
          System.out.println(a&b);  //returns 8 = 1000
          System.out.println(a|b);  //63=111111
          System.out.println(a^b);  //55=11011
          System.out.println(~a);  //-59
            }
}
```

- **Conditional Operators:-** These are also known as the **ternary operator.** This operator consists of 3 operands & is used to evaluate Boolean expressions. Goal of this is a value should be assigned to a variable.

Operator	Description
?:	Used to construct Conditional expression

Example:-

```
num1 = 10;
num2 = 20;
res=(num1>num2) ? (num1+num2):(num1-num2)

Since num1<num2,
the second operation is performed
res =  num1 - num2  = -10
```

- **Separators:-** These includes some symbols like (), , :, ;, [], {}

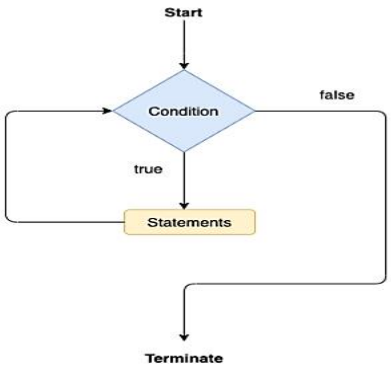
Symbol	Name	Purpose
( )	Parentheses	Used to contain lists of parameters in method definition and invocation. Also used for defining precedence in expressions, containing expressions in control statements, and surrounding cast types.
{ }	Braces	Used to contain the values of automatically initialized arrays. Also used to define a block of code, for classes, methods, and local scopes.
[ ]	Brackets	Used to declare array types. Also used when dereferencing array values.
;	Semicolon	Terminates statements.
,	Comma	Separates consecutive identifiers in a variable declaration. Also used to chain statements together inside a for statement.
.	Period	Used to separate package names from subpackages and classes. Also used to separate a variable or method from a reference variable.

STATEMENTS IN JAVA:- In java statements means instructions to program. These specify the actions to be performed when some method or constructor is invoked. The details of these statements would be provided as when is required.

1. **Empty Statement:-** These are used during program development.
2. **Variable Declaration Statement:**- It defines a variable that can be used to store the values.
3. **Labelled Statement:**- It is used to prefix a lable to an identifier.
4. **Expression Statement:**- It is a one kind of statement that usually created to produce some new values.
5. **Control Statements:**- These are used to control the statements. This comprises **selection**, **iteration** & **jump statements.**

Selection Statements:- Used to select one of the various control flows is selected when a certain condition is true. In this there are 3 types. **If, if-else, nested if, if-else-if, switch.**

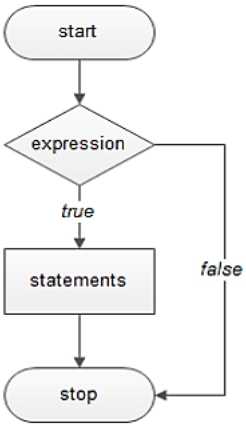
If statement:- The most simple decision-making statement



If Syntax:

```
if(condition)
{
statements;
}
```

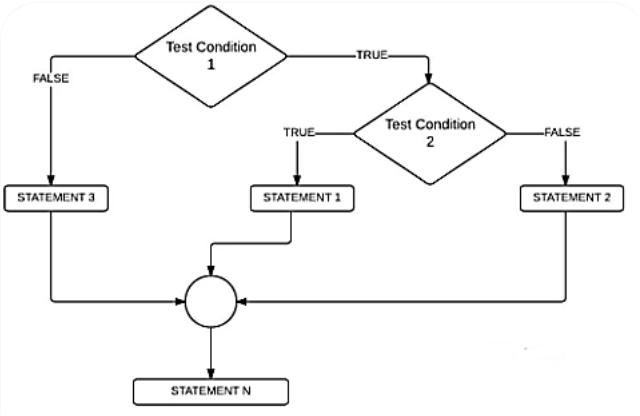
If-else statement:- It is a control flow statement that allows you to execute a block of code only if a certain condition is met.



if – else Syntax:

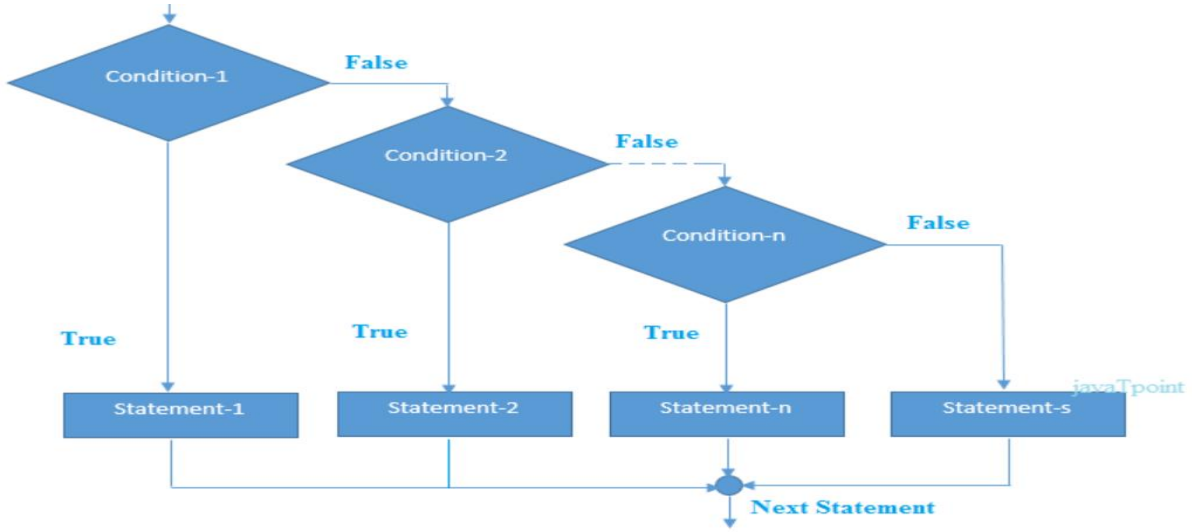
```
if(condition)
{
  true-block statements;
}
else
{
      false-block statements;
 }
```

Nested if:- If any statement is inside an if statement.




```
if condition1 {
   // code to be executed if condition1 is true
  if condition2 {
     // code to be executed if both condition1 and condition2 are true
  }
}
```

If-else-if or if-else ladder:- This executes one condition from multiple statements.




```
if(condition1){
//code to be executed if condition1 is true
}else if(condition2){
//code to be executed if condition2 is true
}
else if(condition3){
//code to be executed if condition3 is true
}
...
else{
//code to be executed if all the conditions are false
}
```

Switch:- It executes one statement from multiple conditions like if-else-if only.

```
switch(expression){
case value1:
 //code to be executed;
 break;  //optional
case value2:
 //code to be executed;
 break;  //optional
.....

default:
  code to be executed if all cases are not matched;
}
```

6. **Iteration Statements:-** These involve the use of loops until some condition for the termination of loop is satisfied. In this there are 3 statements. For, While, Do-while.
7. **Jump Statement:-** In this the control is transferred to the beginning or end of the current block or to a labelled statement. There are 4 types including break, continue, return & throw.
8. **Synchronization statement:-** These are used with multi-threading.
9. **Guarding statement:-** These are used to carry out the code safely that may cause exceptions (such as division by zero & so on). These statements make use of try & catch block & finally.

COMMAND LINE ARGUMENTS:-

- Java can accept any No.of arguments from the command line.
- This is accomplished by passing command-line arguments to main( ).
- It directly follows the program's name on the command line when it is executed.
- To access these inside a Java program is quite easy they are stored as strings in a String array passed to the args parameter of main( ). The 1st command-line argument is stored at args[0], the second at args[1] & so on.

Example:-

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

Output:-

```
C:\> Java vehicle "Car Cycle Motorbike"
    Car
    Cycle
    Motorbike
```

PROGRAMMING STYLE IN JAVA:-

Java is a free form language. We need not have to indent any lines to make the program work properly. Java system does not care when on the line we begin typing. While this may be a license for bad programming style. We should try to use this fact to our advantage for producing readable programs.
Although several alternate styles are possible, we should select one and try to use it with total consistency.
For example, the statement *System.out.println("Java is Wonderful!");*
can be written as

```
System.out.println
("Java is Wonderful!");
```

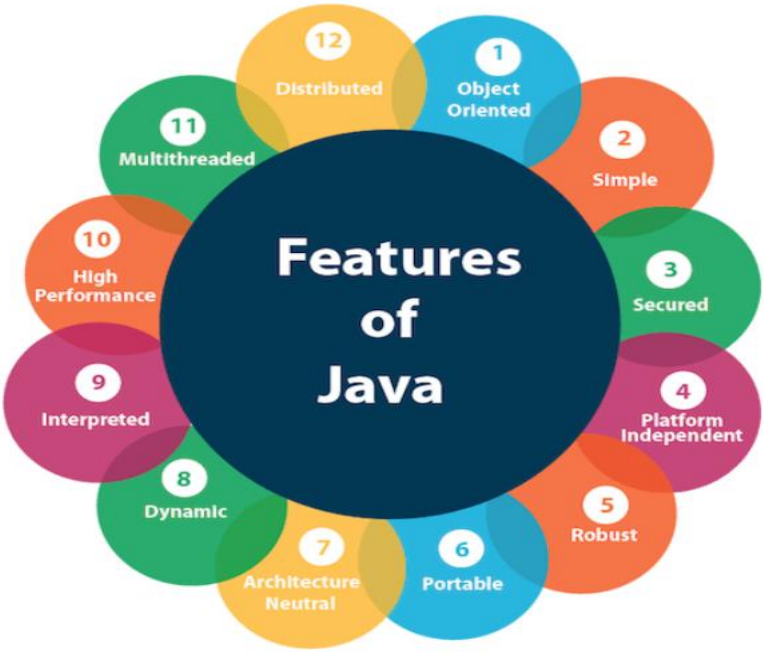
or, even as

```
System.
out.println
("Java is Wonderful!"
);
```

Practise for good programming style:

- ✓ Appropriate comments
- ✓ Naming conventions
- ✓ Proper indentation and spacing lines
- ✓ Block styles

FEATURES OF JAVA PROGRAMMING LANGUAGE:-



1. **Object-Oriented:-**

- Java programming language is **"Object-Oriented Programming Language".**
- Like C++ java provides most object features.
- Java is pure **OOP Language.**

2. **Simple:-**

- Java is designed to be easy for the professional programmer to learn & use.
- It has a concise, cohesive set of features that makes it easy to learn & use.
- Most of the concepts are drew from C++ i.e; making java learning simpler.

3. **Secured:-**

- Java program cannot harm other systems. That's why it is secured.
- It provides secure means of creating internet applications.
- It provides secure way to access web applications.

4. **Platform Independent:-**

- Java programs can run on any platform like:- windows, Mac, Linux.
- It doesn't require a particular platform to run or execute the programs.

5. **Robust:-**

- Java encourages error-free programming by being strictly types & performing run-time checks.

6. **Portable:-**

- Java programs can execute in any environment for which there is a **JVM-Java Virtual Machine.**
- Java programs can be transferred over World Wide Web E.g; Applets.

7. **Architecture-Neutral:-**

- Java is not tied to a specific machine or OS architecture.
- Machine Independent i.e; Java is independent of hardware.

8. **Dynamic:-**

- Java programs carry with them substantial amounts of run-time type information that used to verify & resolve access to objects at run-time.

9. **Interpreted:-**

- Java supports cross-platform code through the use of java byte code.
- Bytecode can be interpreted on any platform by JVM.

- It can also translate into the native machine code for efficiency.

10. High Performance:-

- Java programs can be compiled into an intermediate representation byte code can be interpreted by any JVM.

11. Multi-Threaded:-

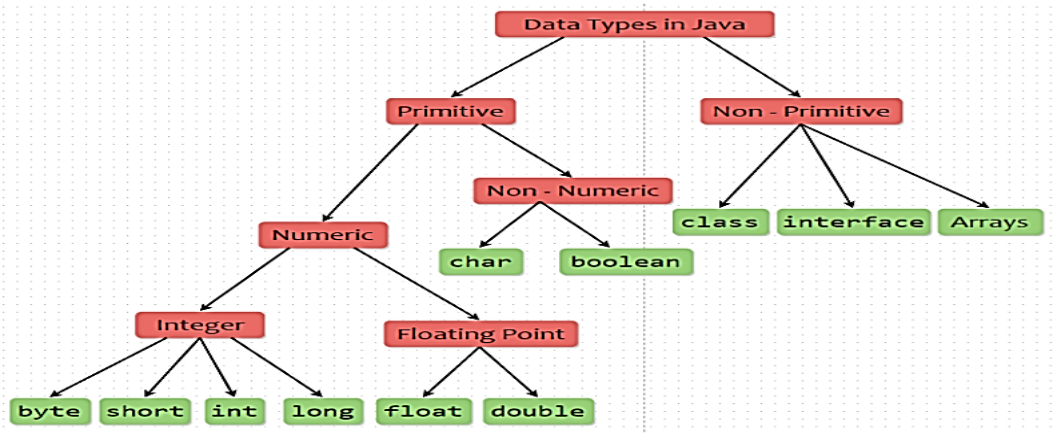
- Supports multi-threaded programming for writing program that perform concurrent computations.

12. Distributed:-

- Java handles TCP/IP protocols, accessing a resource through its URL much like accessing a local file.

DATA TYPES & VARIABLES & OPERATORS IN JAVA

DATA TYPES IN JAVA:-



Type	Size in bytes	Range	Default Value
byte	1 byte	-128 to 127	0
short	2 bytes	-32,768 to 32,767	0
int	4 bytes	-2,147,483,648 to 2,147,483, 647	0
long	8 bytes	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	0
float	4 bytes	approximately ±3.40282347E+38F (6-7 significant decimal digits) Java implements IEEE 754 standard	0.0f
double	8 bytes	approximately ±1.79769313486231570E+308 (15 significant decimal digits)	0.0d
char	2 bytes	0 to 65,536 (unsigned)	'\u0000'
boolean	Not precisely defined"	true or false	false

→Data type means the data which can accepts by the computer. Every variable & expression has a data type.

→Java is an object-oriented programming language based on classes and interfaces.

The declaration of data type with a variable or expression limits the types of values that a variable can have or the expression it can evaluate.

Here we have 2 main types of data types; 4 sub-main types & 11 sub-data types.

The 2 main types are **Primitive & Non-primitive Data Types.**

The 4 sub-main data types are **Numeric, Non-Numeric, Integer & Floating Point.**

The 11 sub data types are **Class, Interface, Arrays, Boolean, Char, Byte, Short, Int, Long, Float & Double.**

Primitive Data Types:-

- ♠ Primitive means basic. The java defines some primitive (basic) data types that are not reference types of any class or interface & these are hard coded into the compiler to recognize when the program is executed. Hence these are called as **"Basic/Primitive Data Types"**.

In primitive we have 2 types. **Numeric & Non-Numeric.**

Non-Numeric Data Types:-

- ♠ In this data that cannot be manipulated mathematically using Standard Arithmetic Operators & it comprises string & boolean data types. Hence it is called as **"Non-Numeric Data Types".**

In this we have 2 data types. They are:-

1. **Char:-** It is the name suggests is useful for storing single value characters. It's default value is '\uoooo' with the max. value being '\uffff' & has a size of 2 bytes.

   Ex:- char a='D';

```
// char can be handled like integers
public class CharClass
{
  public static void main(String args[])
  {
    char myChar1 = 'A';
    char myChar2 = 'B';

    System.out.println("myChar1: " +myChar1);
    System.out.println("myChar2: " +myChar2);
    myChar2++; // valid increment operation
    System.out.println("The Incremented value of myChar2: "
+myChar2);
  }
}
```

Output:-

```
myChar1: A
myChar2: B
The incremented value of myChar2: C
```

2. **Boolean:-** It is a special datatype which can have only two values ‘true’ & ‘false’. It has a default value of ‘false’& a size of 1 byte. It comes in use for storing flag values.

Ex:- boolean flag=true;

```
public class BooleanDataType
{
  public static void main(String args[])
  {
    boolean myBool = true;
    if(myBool == true)
      System.out.println("I am using a Boolean data type");
      System.out.println(myBool);
  }
}
```

Output:-

```
I am using a Boolean data type
true
```

Numeric Data Types:-

♠ These are the numbers stored in Database Columns & it has the fixed-point numeric data. Hence it is called as **“Numeric Data Types”.**

In this we have 2 sub-main data types. They are:-

1. **Integer:-** These can store whole numbers from -2,147,483,647 to 2,147,483,647 for 9 to 10 digits of precision. The integer value is stored as a signed binary integer & is typically used to store quantities e.t.c. In this we have 4 sub-data types. They are:-

I. **Byte:-** It's an 8 bit signed two's complement. The range of values are -128 to 127. It is space efficient because it is smaller than integer datatype. It can be a replacement for int datatype usage but it doesn't have the size range as the integer datatype.

Ex:- byte a=10;

```
public class ByteDataType
{
  public static void main(String args[])
  {
    byte myByte1,myByte2;
    myByte1 = 127;
    myByte2 = -48;
    System.out.println("Byte 1: " +myByte1);
    System.out.println("Byte 2: " +myByte2);
    myByte1++; // Looping back within the range
    System.out.println("Incremented Value of myByte1: " +myByte1);
  }
}
```

Output:-

```
Byte 1: 127
Byte 2: -48
Incremented Value of myByte1: -128
```

II. **Short:-** This is also similar to the integer datatype. However it's 2 times smaller than the integer datatype. It's min. range is -32,768 & max. range is 32,767.

Ex:- short a=54;

```
public class ShortDataType
{
  public static void main(String args[])
  {
    short myShort = 6000;
    System.out.println("myShort: " + myShort);
  }
}
```

Output:-

```
myShort: 6000
```

III. **Int:-** It is used for storing integer values. It's size is 4 bytes & has a default value of "0". The max. value is 2^31 & min. value is -2^31. It can be used to store integer values unless there is a need for storing numbers larger or smaller than the limits.

Ex:- int a=56;

```
public class IntDataType
{
  public static void main(String args[])
  {
    int myNum1, myNum2, result;
    myNum1 = -7000;
    myNum2 = 90000;
    result = myNum1 + myNum2;
    System.out.println("Number 1: " +myNum1);
    System.out.println("Number 2: " +myNum2);
    System.out.println("Number 1 + Number 2: " +result);
  }
}
```

Output:-

```
Number 1: -7000
Number 2: 90000
Number 1 + Number 2: 83000
```

IV. **Long:-** This primarily stores huge sized numeric data. It is a 64 bit integer & ranges from -2^63 to +(2^63)-1. It has a size of 8 bytes & is useful when you need to store data which is longer than int datatype.

Ex:- long a=1273762;

```
public class LongDataType
{
  public static void main(String args[])
  {
    long myLong1, myLong2, result;
    myLong1 = 100000000L;
    myLong2 = 200L;
    result = myLong1 * myLong2;
    System.out.println("Number 1: " +myLong1);
    System.out.println("Number 2: " +myLong2);
    System.out.println("Number 1 * Number 2: " +result);
  }
}
```

Output:-

```
Number 1: 100000000
Number 2: 200
Number 1 * Number 2: 20000000000
```

2. **Floating Point:**- These represents numbers with a fractional part, containing one or more decimals. In this we have 2 sub datatypes. They are:-
   a. **Float:-** It is used for sorting decimal values. It's default value is o.of & has a size of 4 bytes. It has a infinite value range. However its always advised to use float in place of double if there is a memory constraint. Currency should also never be stored in float datatype.

Ex:- float a=98.7f;

```
public class FloatDataType
{
      public static void main(String args[])
      {
            float myFloat1,myFloat2,result;
            myFloat1=1000.666f;
            myFloat2=110.77f;
            result=myFloat1-myFloat2;
            System.out.println("Number1: "+myFloat1);
            System.out.println("Number2: "+myFloat2);
            System.out.println("Number1-Number2: "+result);
      }
}
```

Output:-

```
Number1: 1000.666
Number2: 110.77
Number1-Number2: 889.896
```

b. Double:- This is similar to the float datatype. However it has one advantage over float datatype i,e; it has two bit precision over the float datatype which has one bit precision. It has the range from -2^31 to (2^31)-1.

Ex:- double DataFlair=99.987d;

```
public class DoubleDataType
{
     public static void main(String args[])
     {
          double myDouble1, myDouble2, result;
          myDouble1 = 48976.8987;
          myDouble2 = 29513.7812d;
          result = myDouble1 + myDouble2;
          System.out.println("Number 1: " +myDouble1);
          System.out.println("Number 2: " +myDouble2);
          System.out.println("Number 1 + Number 2: " +result);
     }
}
```

Output:-

```
Number 1: 48976.8987
Number 2: 29513.7812
Number 1 + Number 2: 78490.6799
```

Non-Primitive Data Types:-

- ♠ These are created by the programmer & they are not pre-defined. Since these are referred to as objects they can be assigned with null; which is not a case in primitive type. This type must be start with an upper case letter. These are the class and interface types. The name of a class or interface is the name of type. Hence it is called as **"Non-Primitive Data Type".**

In this we have 3 sub-datatypes. They are:-

1. **Class:-** In java class is a blueprint from which we can create an individual object. Java provides a keyword named class by which we can declare a class. We can also refer a class as a user-defined data type.

Ex:-

```
public class Student
{
  int marks = 76;
  public static void main(String[] args)
  {
    Student student1 = new Student();
    // creating object of the class by using new operator
    System.out.println("Marks of student: " +student1.marks);
    Accessing the property "marks" of the class with the help of an
object.
  }
}
```

Output:-

```
Marks of student: 76
```

2. **Interface:-** It is a reference type in java. It is similar to class. It is a collection of abstract methods. It specifies which objects can used as values of which variables or parameters.

Ex:-

```
//creating an interface
interface Runnable
{
  public void run(); //an abstract method
}
 // implementing the interface
public class Person implements Runnable
{   public void run()
  {
    System.out.println("This person can run");
  }

  public static void main(String args[])
  {
    Person person1 = new Person();
    person1.run();
  }
}
```

Output:-

```
This person can run
```

DECLARATION OF VARIABLES & DATA TYPES:-

Variables:- The variable is the basic unit of storage in a Java program. A variable is defined by the combination of an identifier, a type & an optional initializer. In addition, all variables have a scope, which defines their visibility & a lifetime.

1) Local Variable

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

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

2) Instance Variable

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

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

3) Static variable

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

Declaration:- In Java, all variables must be declared before they can be used. The basic form of a variable declaration is:-

type identifier;

type identifier = value;

Ex:-

```
byte n; // declares a variable of type byte.
short m = 67;  // declares and initiates a short number
int length;  //declares length-a variable of type int
length - 50; // value is re-assigned after declaration
char ch = 'A'; // declares a character variable ch.
```

TYPE CASTING:-

In java type casting is the method is the method or process that can converts a data type into another data type in both ways in manually & automatically. The automatic conversion is done by the compiler & manual conversion performed by the programmer.

In this we have 2 types. They are:-

1.Widening Casting (Automatically)---------Implicit Type Casting.

2.Narrowing Casting (Manually)----------Explicit Type Casting.

1. Implicit, Widening, Automatic Type Casting:-

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

➲ It is the process of conversion of a lower data type to a higher data type.

➲ Java automatically performs this type of casting without any explicit code writing, which is why this type of casting is also known as **"Automatic/Widening/Implicit Type Casting".**

```
public class Main {
  public static void main(String[] args) {
    int myInt = 9;
    double myDouble = myInt; // Automatic casting: int to double

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

Output:- 9

9.0

2. Explicit, Narrowing, Manual Type Casting:-

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

➲ It is the process of conversion of Higher Data Type to Lower Data Type.

➲ It is not done automatically by java but needs to be explicitly done by the programmer, which is why it is also called as **"Explicit/Narrowing/Manual Type Casting".**

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

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

Output:- 9.78

9

SCOPE OF VARIABLE IDENTIFIERS:-

➲ Scope is that area of the program where the variable is visible to a program & can be used.

- The scope of variable determines its accessibilities for other part of the program. Java allows declaring variables within any block.

Basic Variable Declaration

- datatype identifier [=value];
- datatype must be

1. A simple datatype
2. User defined datatype (class type)

- Identifier is a recognizable name confirm to identifier rules
- Value is an optional initial value.

Variable Declaration

- We can declare several variables at the same time:
- type identifier [=value][, identifier [=value] ...];
- int a, b, c;
- int d = 3, e, f = 5;
- double pi = 3.14159;

Variable Scope

- Scope determines the visibility of program elements with respect to other program elements.

In Java, scope is defined separately for classes and methods:

1) variables defined by a class have a global scope
2) variables defined by a method have a local scope

- A scope is defined by a block:

```
{
  ..
}
```

- A variable declared inside the scope is not visible outside:

Variable Lifetime

- Variables are created when their scope is entered by control flow and destroyed when their scope is left
- A variable declared in a method will not hold its value between different invocations of this method.
- A variable declared in a block looses its value when the block is left.
- Initialized in a block, a variable will be re-initialized with every re-entry. Variables lifetime is confined to its scope!

LITERAL CONSTANTS:-

- Any constant value which can be assigned to the variable is called as “Literal Constant”.
- Here constant means the value which can’t/cannot be changed.
- In java literals are the constants values that appear directly in the program. It can be assigned directly to a variable.

- **Constants** refer to fixed values that the program may not alter during its execution.
- These fixed values are also called **literals**.
- **Constants** can be of any of the basic data types like an
integer **constant**,
floating **constant**,
character **constant**,
string **literal**.
- There are enumeration **constants** as well.

Here are different types of literals in Java.

1. Boolean Literals
In Java, Boolean literals are used to initialize Boolean data types. They can store two values: true and false.
2.Integer Literals
An integer literal is a numeric value(associated with numbers) without any fractional or exponential part.
3. Floating-point Literals
A floating-point literal is a numeric literal that has either a fractional form or an exponential form.
4. Character Literals
Character literals are Unicode character enclosed inside single quotes.
5.String literals
A string literal is a sequence of characters enclosed inside double-quotes.

SYMBOLIC CONSTANTS:-

→ A symbolic constant is a lable(name)that represents a fixed value that never changes during the course of a program.

→ It means a name given to some numeric constant/a character constant or a string constant or any other constants.

→ Symbolic constants names are also known as constant identifiers. Pre-processor directive #define is used for defining symbolic constants.

Public static final PI = 3.145926535;

Public static final PI = 299792458; //speed of light.

Formatted Output with printf() Method:-

It is the method of java printstream class is a convenience method which is used to write a string which is formatted to this output stream. It uses the specified format string & arguments to write the string.

The syntax of the method printf () method is as follows

```
System.out.printf("Formatting string" variables separated by comma);
```

- The formatting string specifics the output format for each variable that consists of percent (%) sign followed by a conversion letter.
- Thus, the format string for output of an integer and character is "X" and or respectively.
- The order of variables in variable list should match with the list of formats in formatting string.

STATIC VARIABLES & METHODS:-

- ✓ The static variable is a class level variable & it is common to all the class objects .i.e; a single copy of the static variable is shared among all the class objects.
- ✓ A static method manipulates the static variables in a class. These blocks are only executed once when the class is loaded.

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

 System.out.println("The Square root root of 16 = "+ Math.sqrt(16));
 System.out.println("The cubroot root of 27 = "+ Math.cbrt(27));
 //printing five random variables
 for(int i =1; i<=5;i++)
      System.out.println("Random Number "  + i + " = " +
                         (int)(100 *Math.random()));
      }
}
```

```
E:\>javac StaticMethods.java

E:\>java StaticMethods
The Square  root root of 16 = 4.0
The cubroot root of 27 = 3.0
Random Number 1 = 77
Random Number 2 = 69
Random Number 3 = 83
Random Number 4 = 2
Random Number 5 = 66

E:\>java StaticMethods
The Square root root of 16 = 4.0
The cubroot root of 27 = 3.0
Random Number 1 = 67
Random Number 2 = 31
Random Number 3 = 10
Random Number 4 = 13
Random Number 5 = 40
```

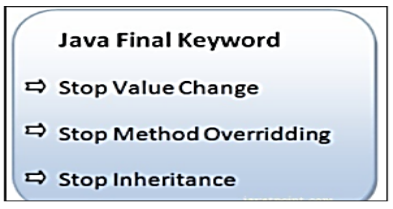
Attribute final

- There are many things which requires repeatedly and if we want to make changes then we have to make these changes in whole program where this variable is used .
- Java provides **"final"** keyword to declare the value of variable as follows:

Syntax:
final type Symbolic_name= value;

Ex: If I want to declare the value of 'PI' then :
final float PI=3.1459:

- The condition is, Symbolic_name will be in capital letter .
- The final keyword is called a "modifier".



- The final keyword is a non-access modifier used for classes, attributes and methods, which makes them non-changeable (impossible to inherit or override).
- The final keyword is useful when you want a variable to always store the same value, like PI (3.14159...).

Ex:

```
public class Main {
final int x = 10;
public static void main(String[] args) {
  Main myObj = new Main();
  myObj.x = 25; // will generate an error: cannot assign a value to a final variable
  System.out.println(myObj.x);
 }
}
```

Precedence and Associativity of Operators:-

Java operators have 2 properties those are precedence & associativity. Precedence is the priority order of an operator, if there are 2 or more operators in an expression then the operator of highest priority will be executed 1st then higher & then high.

Category	Operator	Associativity
Postfix	() [] -> . ++ - -	Left to right
Unary	+ - ! ~ ++ - - (type)* & sizeof	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	<< >>	Left to right
Relational	< <= > >=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right
Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %=>>= <<= &= ^= |=	Right to left
Comma	,	Left to right

****HAPPY LEARNING****

****THANK YOU****