UNIT-1

UNIT I: Object Oriented Programming: Basic concepts, Principles, Program Structure in Java: Introduction, Writing Simple Java Programs, Elements or Tokens in Java Programs, Java Statements, Command Line Arguments, User Input to Programs, Escape Sequences Comments, Programming Style.

Data Types, Variables, and Operators :Introduction, Data Types in Java, Declaration of Variables, Data Types, Type Casting, Scope of Variable Identifier, Literal Constants, Symbolic Constants, Static Variables and Methods, **Introduction to Operators**, Precedence and Associativity of Operators, Assignment Operator (=), Basic Arithmetic Operators, Increment (++) and Decrement (- -) Operators, Ternary Operator, Relational Operators, Boolean Logical Operators, Bitwise Logical Operators.

Control Statements: Introduction, if Expression, Nested if Expressions, if–else Expressions, Switch Statement, Iteration Statements, while Expression, do–while Loop, for Loop, Nested for Loop, Break Statement, Continue Statement

Object Oriented Programming

- Object Oriented Programming is a programming concept that works on the principle that objects are the most important part of your program.
- It allows users create the objects that they want and then create methods to handle those objects.
- Manipulating these objects to get results is the goal of Object Oriented Programming. Object Oriented Programming popularly known as OOP, is used in a modern programming languages like Java.

Object :

Any real world entity that has state and behaviour is called as Object .(or)

Objects have state and behaviour. Example: Apple, Orange, Bat, Table, etc..

In Java, An Object is an Instance of class.

Class :

Collection of similar objects is called Class . For Example, Apple, orange, Papaya are grouped into a class called "Fruits" where as Apple, Table, Bat cannot be grouped as class because they are not similar groups. It is only an logical component not as physical entity.

Inheritance :

One object acquires all properties and behaviour of the parent object.
It's creating a parent-child relationship between two classes. It offers robust and natural,mechanism for organizing and structure of any software.

Polymorphism :

It refers as " one interface and many forms" (or) the ability of a variable , object or function to take on multiple forms.
Ex:- In English, the verb "run" has a different meaning if you see it with a "laptop", and "a foot race".

Abstraction :

Abstraction is a process of hiding the implementation details from the user.
Ex:- while driving a car, you do not have to be concerned with its internal working.
Abstraction can be achieved using Abstract Class and Abstract Method in Java.

Encapsulation :

Encapsulation is a principle of wrapping data (Variables) and code together as a single unit. In this OOPS concept, variables of a class are always hidden from other classes. It can only be accessed using the methods of their current classes.

Program Structure in Java

Elements or Tokens in Java Programs

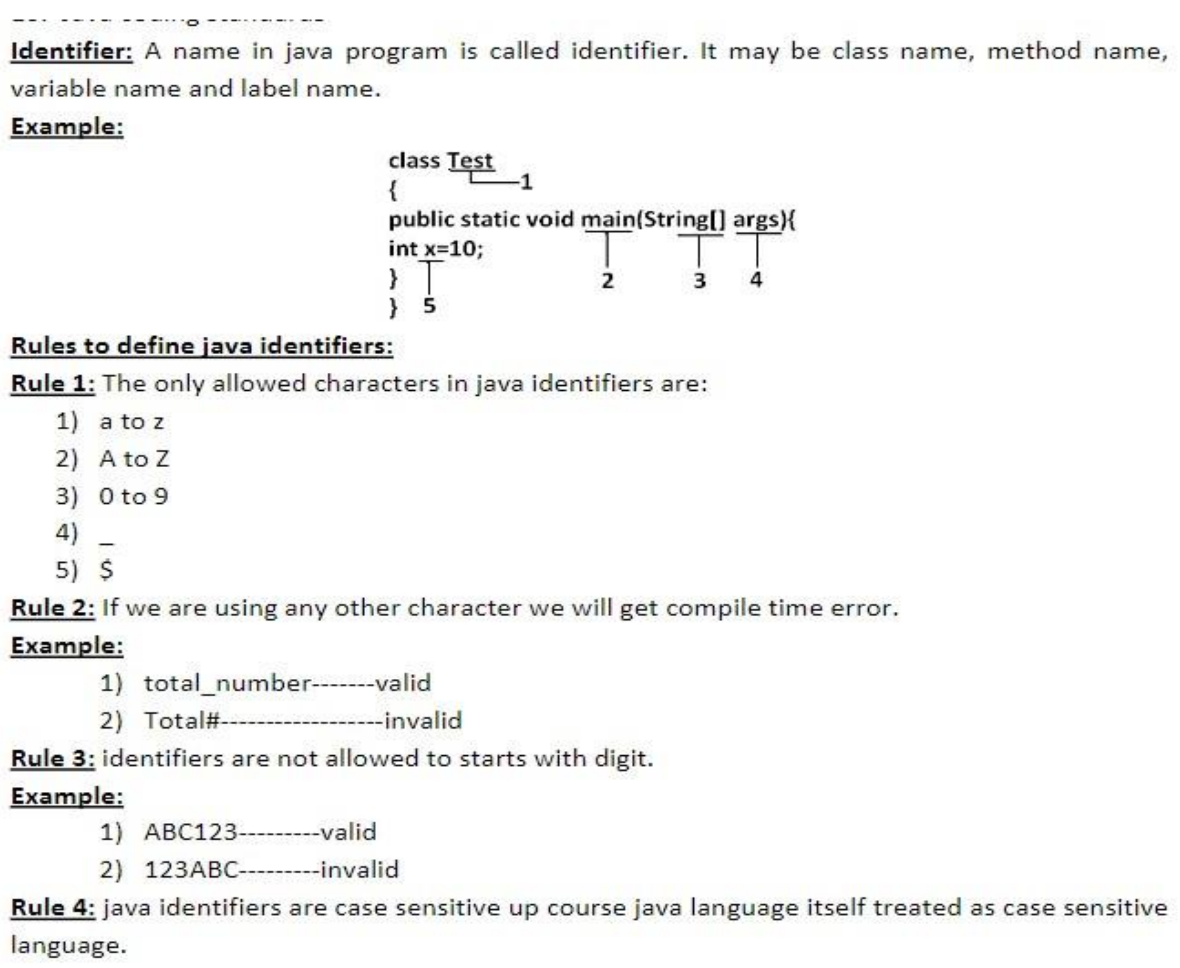
In Java programming, elements or tokens are the smallest individual units in a program. These tokens are the building blocks of Java code and are used to construct statements and expressions. Here are the main types of tokens in Java

1.Keywords
2.Identifiers
3.Literals
4.seperators
5.comments
6.operators

1. Keywords

Keywords are reserved words in Java that have a predefined meaning and cannot be used as identifiers (names for variables, classes, methods, etc.). Examples of keywords include class, public, static, void, int, if, else, for, while, return, etc.

2.Identifiers



Identifier: A name in java program is called identifier. It may be class name, method name, variable name and label name.

Example:

```
class Test
       └──1
{
public static void main(String[] args){
int x=10;             |         |      |
}  |                  2         3      4
}  5
```

Rules to define java identifiers:

Rule 1: The only allowed characters in java identifiers are:

1) a to z
2) A to Z
3) 0 to 9
4) _
5) $

Rule 2: If we are using any other character we will get compile time error.

Example:

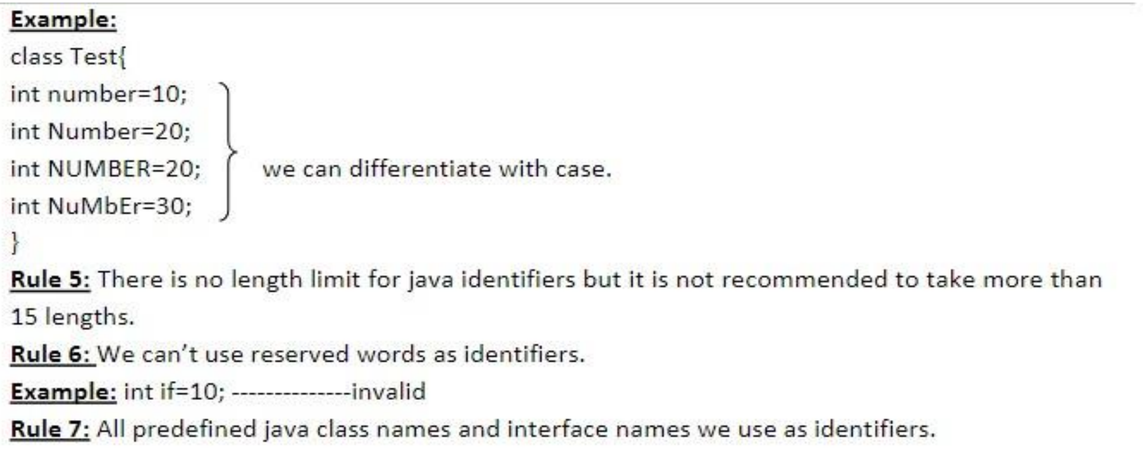
1) total_number-------valid
2) Total#------------------invalid

Rule 3: identifiers are not allowed to starts with digit.

Example:

1) ABC123---------valid
2) 123ABC---------invalid

Rule 4: java identifiers are case sensitive up course java language itself treated as case sensitive language.



Example:

```
class Test{
int number=10;   ⎫
int Number=20;   ⎪
int NUMBER=20;   ⎬  we can differentiate with case.
int NuMbEr=30;   ⎭
}
```

Rule 5: There is no length limit for java identifiers but it is not recommended to take more than 15 lengths.

Rule 6: We can't use reserved words as identifiers.

Example: int if=10; --------------invalid

Rule 7: All predefined java class names and interface names we use as identifiers.

3. Constants or Literals

- Entities that do not change their values in a program are called Constants or Literals.
- Java Literals are classified into 5 types:
    1. Integer Literals
    2. Floating Point Literals
    3. Character Literals
    4. Boolean Literals
    5. String Literals

1) Integer Literals :

- A whole number is called an integer. Eg: 25,27 etc…are integers
- Java supports 3 types of integer literals Decimal, Octal, Hexadecimal.
- 25, 27 are decimal integers
- Octal stats from 0 and followed by 0 to 7 . Eg: 0.037, 0.08656 are octal integer
- Hexadecimal start with OX and followed by digits 0 to 9 , A to F. Eg: 0*29, 0*2AB9 are hexadecimal integer literals .

2. Floating Point Literals :

- Numbers with decimal point and fractional values are called floating point literals.
- They can be expressed in either standard or scientific notation.
- Standard notation consists of a whole number component followed by a decimal point followed by a fractional component.
- A Floating point number followed by letter E (or) and a signed integer. Eg: 6.237E-35 stands for 6.237*10^-35.
- Floating point literals in java defaults to double precision.

3. Boolean literals :

- In java, Boolean literals take two values false or true.
- These two values are not related to any numeric value as in C or C++.
- The Boolean value true is not equal to 1 and false is not value is not equal to 0.

4. Character literals :

- Single characters in java are called character literals.

- In java characters belong to 16-bit character set called Unicode.
- Java characters literals are written within a pair of single quote. Eg: ‘a’, ‘z’, represent character literals.
- To represent such characters, java provides a set of character literals called escape sequence.

5. **String Literals :**
   - A sequence of characters written within a pair of double quote is called String Literal.
     Eg: “This is String”.
   - String Literals are to be started and ended in one line only.

4.Separators

Separators (or delimiters) are symbols that separate elements of the code. Common separators in Java include:

- **Parentheses:** ()
- **Braces:** {}
- **Brackets:** []
- **Semicolon:** ;
- **Comma:** ,
- **Dot:** .

5.Comments

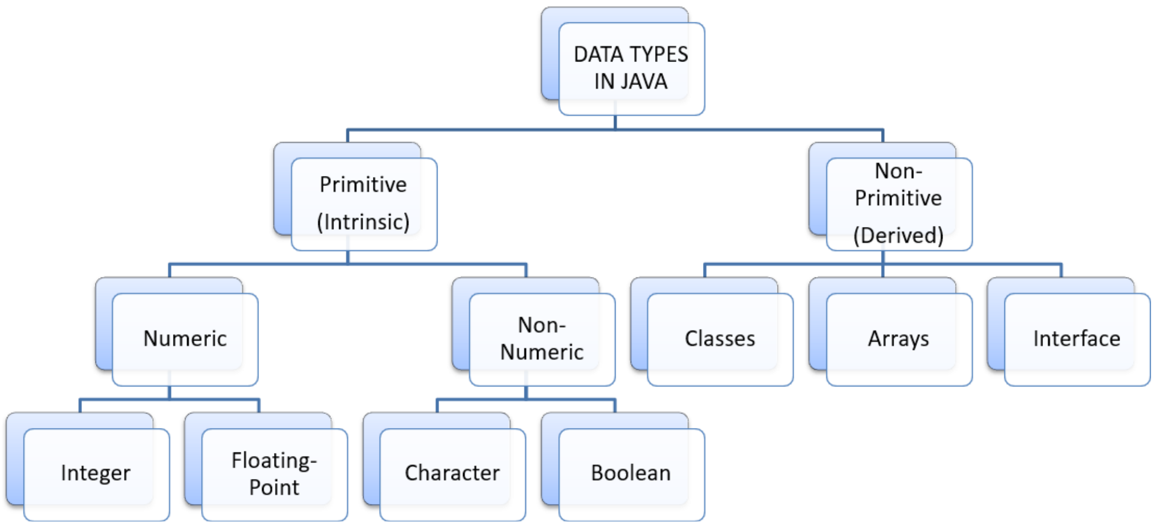
Comments are non-executable parts of the code that are used to describe or explain the code. They are ignored by the Java compiler. There are two types of comments in Java:

- **Single-line comments:** Start with //
- **Multi-line comments:** Enclosed between /* and */

Data Types, Variables, and Operators

Data Types

- Every variable in java has a data type.
- Data type specify the size and type of values that can be stored.
- Data type in java under various categories are shown as:



A. Primitive data types :

Primitive data types are whose variables allows us to store only one value but they never allow us to store multiple values of same type. This is a data type whose variables can hold maximum one value at a time.

Example:

```
int a;
a=10;//valid
a=10,20,30;//invalid
```

B. Non Primitive Data Types or Derived

Derived data types are those which are developed by programmers by making use of appropriate features of the language. User defined data types related variables allow us to store multiple values either of same type or different type or both.

Example:

```
Student s=new Student();
```

Java defines some primitive types of data. They are:

Integer types

Character types

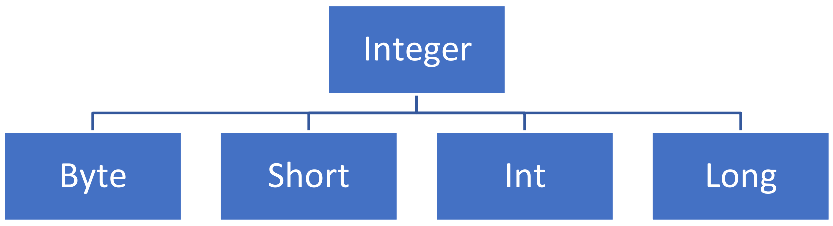
Floating type

Boolean type

Integer Types :

This type indicates byte, short, int, long which are for whole-valued signed numbers.

The width and ranges of these integer types vary widely as shown in below :



The width and ranges of these integer types vary widely as shown in below :

Name	**Width**	**Range**
Long	64	-9,223,372,036,854,775,808 TO 9,223,372,036,854,775,807
Int	32	-2.147,483,648 to 2.147,483,647
Short	16	-32,768 to 32,767
Byte	8	-128 to 127

Byte :

- Smallest integer type is byte.
- This is signed 8-bit type that has range from -128 to 127
- It is declared by byte keyword

Short :

- Short is signed as 16 bit type
- It has range from -32,768 to 32,767
- It is declared by short keyword.

Int :

- The most commonly used type is integer type as int.
- It is signed as 32 bit type and has range from -2,147,483,648 to 2,147,483,647

Long :

- long is signed as 64-bit type and is useful for those occasions where as int.
- The range of long is quite large.
- This makes it useful when big, whole numbers are needed.

Floating Point Types :

- This group includes float and double which represented numbers in fractional precision.
- They are two types of floating point types ,float and double, which represents single and double precision numbers

Name	**Width in bits**	**Approximate range**
Double	64	4.9e-324 to 1.8e+308
Float	32	1.4e-045 to 3.4e+038

3. **Characters :**
   - In Java, the data type is used to store characters is char.
   - Char in java is not same as C or C++
   - In C/C++ char is 8 bit type whereas in java char is 16-bit type
4. **Boolean Type :**
   - Java has a primitive data type called Booleans, for logical values.
   - It can have only one of two possible values true or false.

Variables

- A Variable is an identifier that denote s a storage location used to store a data value . (or) Variables are the names of storage locations.
- Variable names may consist of alphabets, digits, the underscore and dollar characters.
  - ❑ They must not begin with a digit.
  - ❑ Uppercase and Lowercase are distinct. This means that the variable Total is not same as total or TOTAL.
  - ❑ It should not be a keyword.
  - ❑ White space is not allowed.
  - ❑ Variable names can be any length.

Declaration of Variables :

- A Variable must be declared before it is used in the program. The general form of declaration of a variable is

  Type variable1, variable2,………. variable
- Variables are separated by commas. A declaration statement must end with a semicolon. Some valid declarations are:

Int count;
float x, y;

Giving values to Variables :

A Variable must be given a value after it has been declared but before it is used in an expression. This can be in two ways:

1. By using an Assignment statement

   VariableName=value

2. By using a read statement

   we may also give values to variables interactively through the keyword using the readLine().

Scope of the variable :

- the scope refers to validity across the java program.
- The scope of a variable is limited to the block defined within the braces { and }
- It means a variable cannot be accessed outside the scope (Or) The scope or a particular variable is the range within a program 's source code in which that variable is recognized by the compiler.

Type Conversion and Casting

Assigning a value of one type to a variable of another type is known as **Type Casting.**

Example:

```
int x=10;
byte y=(byte)x;
```

In Java, type casting is classified into two types.

Widening Casting (Implicit) : Process of Converting lower data type into higher data type

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

----------------------------->

Widening

Narrowing Casting (Explicitly done) : Process of converting Higher Data type into LowerData Type

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

------------------------------>

Example: Converting int to double

```
class Main {
  public static void main(String[] args) {
    // create int type variable
    int num = 10;
    System.out.println("The integer value: " + num);

    // convert into double type
    double data = num;
    System.out.println("The double value: " + data);
  }
}
```

Output

```
The integer value: 10
The double value: 10.0
```

Example: Converting double into an int

```
class Main {
  public static void main(String[] args) {
    // create double type variable
    double num = 10.99;
    System.out.println("The double value: " + num);

    // convert into int type
    int data = (int)num;
    System.out.println("The integer value: " + data);
  }
}
```

Output

```
The double value: 10.99
The integer value: 10
```

-

Types of Variables

- Based the type of value represented by the variable all variables are divided into 2 types. They are:
  1) Primitive variables
  2) Reference variables

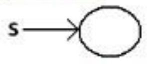
Primitive variables: Primitive variables can be used to represent primitive values.

Example: int x=10;

Reference variables: Reference variables can be used to refer objects.

Example: Student s=new Student();

Diagram:



- Based on the purpose and position of declaration all variables are divided into the following 3 types.
  1) Instance variables
  2) Static variables
  3) Local variables

Instance variables

- If the value of a variable is varied from object to object such type of variables are called instance variables.
- For every object a separate copy of instance variables will be created.
- Instance variables will be created at the time of object creation and destroyed at the time of object destruction hence the scope of instance variables is exactly same as scope of objects.
- Instance variables will be stored on the heap as the part of object.
- Instance variables should be declared with in the class directly but outside of any method or block or constructor.
- Instance variables can be accessed directly from Instance area. But cannot be accessed directly from static area.
- But by using object reference we can access instance variables from static area.

Example:

```
class Test
{
        int i=10;
        public static void main(String[] args)
        {
                //System.out.println(i);//C.E:non-static variable i cannot be referenced from a
static context(invalid)
                Test t=new Test();
                System.out.println(t.i);//10(valid)
                t.methodOne();
        }
        public void methodOne()
        {
                System.out.println(i);//10(valid)
        }
}
```

- For the instance variables it is not required to perform initialization JVM will always provide default values.

Static variables:

- If the value of a variable is not varied from object to object such type of variables is not recommended to declare as instance variables. We have to declare such type of variables at class level by using static modifier.
- In the case of instance variables for every object a separate copy will be created but in the case of static variables for entire class only one copy will be created and shared by every object of that class.
- Static variables will be crated at the time of class loading and destroyed at the time of class unloading hence the scope of the static variable is exactly same as the scope of the **.class** file.
- Static variables will be stored in method area. Static variables should be declared with in the class directly but outside of any method or block or constructor.
- Static variables can be accessed from both instance and static areas directly.
- We can access static variables either by class name or by object reference but usage of class name is recommended.
- But within the same class it is not required to use class name we can access directly. jk

- For the static variables it is not required to perform initialization explicitly, JVM will always provide default values.

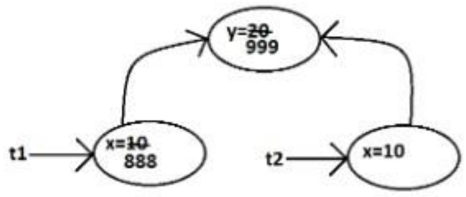
Example:

```
class Test
{
        static String s;
        public static void main(String[] args)
        {
                System.out.println(s);//null
        }
}
```

Example:

```
class Test
{
        int x=10;
        static int y=20;
        public static void main(String[] args)
        {
                Test t1=new Test();
                t1.x=888;
                t1.y=999;
                Test t2=new Test();
                System.out.println(t2.x+"----"+t2.y);//10----999
        }
}
```

Diagram:



- Static variables also known as class level variables or fields.

Local variables:

- Some time to meet temporary requirements of the programmer we can declare variables inside a method or block or constructors such type of variables are called local variables or automatic variables or temporary variables or stack variables.

- The local variables will be created as part of the block execution in which it is declared and destroyed once that block execution completes. Hence the scope of the local variables is exactly same as scope of the block in which we declared.

Example 1:

```
class Test
{
	public static void main(String[] args)
	{
		int i=0;
		for(int j=0;j<3;j++)
		{
			i=i+j;
		}
		System.out.println(i+"----"+j);
```

C.E

```
javac Test.java
Test.java:10: cannot find symbol
symbol  : variable j
location: class Test
```

```
	}
}
```

Example 2:

```
class Test
{
	public static void main(String[] args)
	{
		try
		{
			int i=Integer.parseInt("ten");
		}
		catch(NullPointerException e)
		{
			System.out.println(i);
```

C.E

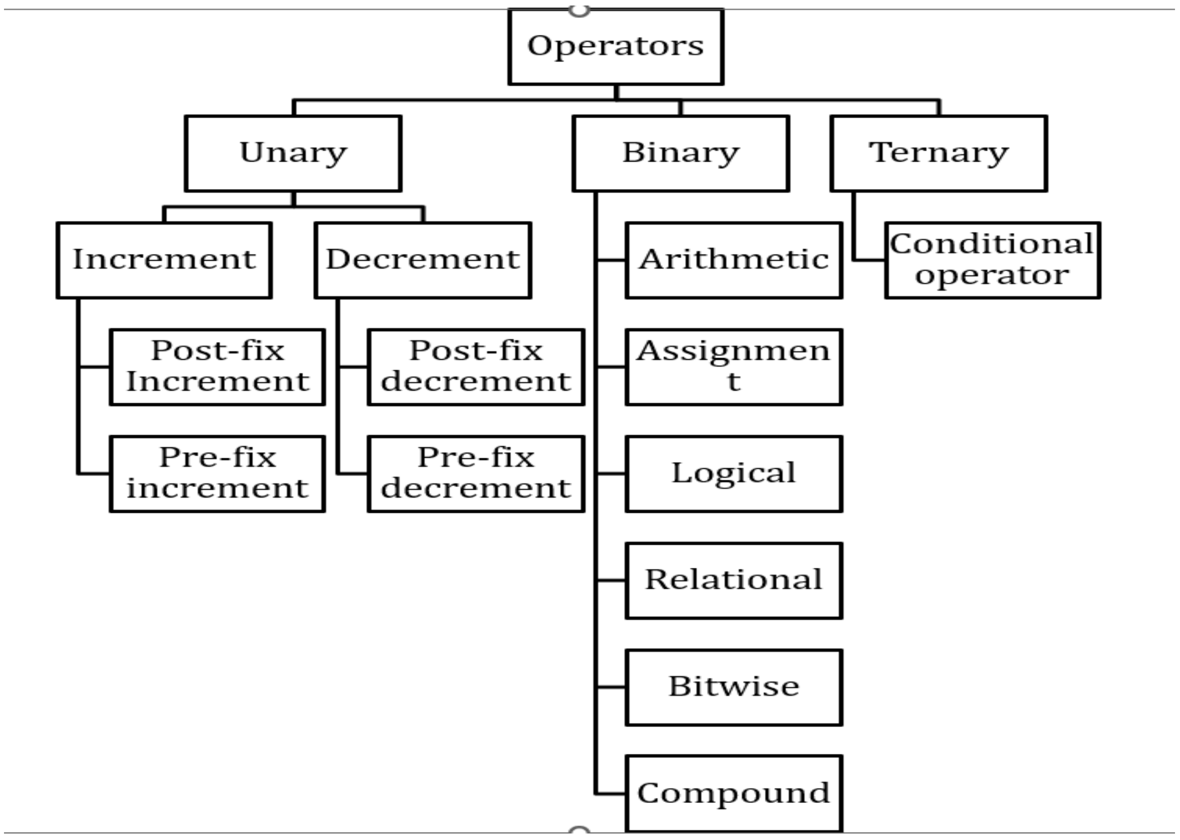
```
javac Test.java
Test.java:11: cannot find symbol
symbol  : variable i
location: class Test
```

```
		}
	}
}
```

OPERATORS AND EXPRESSIONS

- ❑ In java Operators are symbols that are used to perform some operations on the operands.
- ❑ Combination of operands and operators are known as Expressions.
- ❑ Java provides, a rich set of operators to manipulate the variables. There are three types of operators in java.

  1. Unary operators
  2. Binary operators
  3. Ternary operators



1. UNARY OPERATORS:

In which we use one operand is called unary operator. It has two types:

1.1 Increment Unary operator

1.2 Decrement Unary operator

1.1 INCREMENT UNAY OPERATOR:

This is used to increase the value one by one. It has two types:

* Post-fix Increment operator

* pre-fix Increment operator

1.1.1 POST-FIX INCREMENT OPERATOR:

"++" symbol is used to represent Post-fix Increment operator. This symbol is used after the operand.

In this operator, value is first assign to a variable and then incremented the value.

```
EX:         int a , b;
              a=10;
              b=a++;
```

In the above example first the value of "a" is assign to the variable "b", then I Increment the value, so the value of b variable is "10".

1.1.2 PRE-FIX INCREMENT OPERATOR:

"++" symbol is used to represent Pre-Fix operator, this symbol is used after the operand. In this operator value is incremented first and then assigned to a variable.

```
EX:           int a,b;
               a=10;
                b=++a;
```

In the above example first the increment is done then the value of "a" variable is assigned to the variable "b", so the value of "b" variable is "11".

1.2 DECREMENT UNARY OPERATOR:

"-" symbol is used to decrease the value by one. It has two types:

1.Post-fix decrement operator

2.pre-fix decrement operator

1.2.1 POST_FIX DECREMENT OPEATOR:

"-" symbol is used to represent post-fix decrement operator, this symbol is used after the operand. In this operator, value is first assigned to a variable and then decrement the value.

```
EX:       int a , b;
          a=10;
           b=a--;
```

In the above example first the value of "a" is assign to the variable " b", then decrement the value. So the value of "b" variable is "10".

1.2.2 PRE-FIX DECREMENT OPERATOR:

"-" Symbol is used to represent the pre-fix decrement operator. This symbol is used after the operand. In this operator, value is decremented first and then decremented value is used in expression.

EX:
```
            int a,b;
    a=10;
          b=--a;
```

In the above example first the value of "a" is decrement then assign to the variable "b". So the value of b variable is "9".

2. BINARY OPERATOR:

In which we use two operand is called Binary operator. Java supports many types of Binary operators:

* Assignment operator
* Arithmetic operator
* Logical operator
* Comparison operator

2.1 ASSIGNMENT OPERATOR:

This operator is used to assign the value. This symbol "=" is used to assign the value .

EX:
```
int a=12;
```

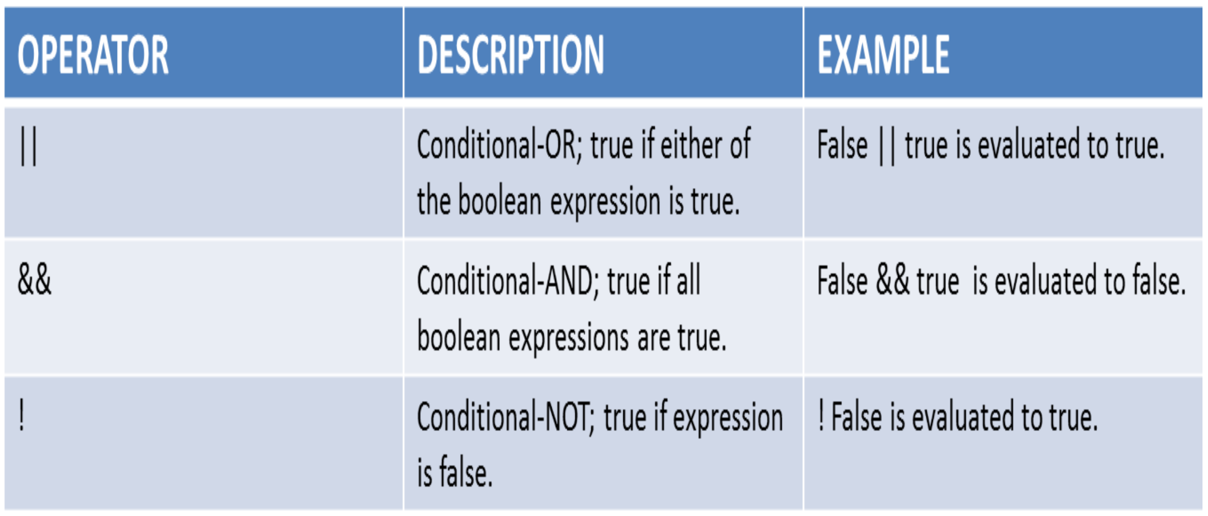
2.2 ARITHMETIC OPERATOR:

This operator is used to perform mathematical operand. Arithmetic operator are:

	Operators	Description	Use
1.	Additional operator (“+”) :	Used to add the value of two operand.	a+b
2.	Subtract operator (“-“) :	Used to subtract the value of two operand.	a-b
3.	Multiply operator (“*”) :	Used to multiply the value of two operand.	a*b
4.	Division operator (“/”) :	Used to divide the value of two operand.	a/b
5.	Modulus operator(“%”) :	Used returns the remainder of a division operation.	a%b

LOGICAL OPERATOR:

The logical operator ||(conditional-OR),&&(conditional-AND),!(conditional-NOT) operates on boolean expressions, here’s how they work:



OPERATOR	DESCRIPTION	EXAMPLE
||	Conditional-OR; true if either of the boolean expression is true.	False || true is evaluated to true.
&&	Conditional-AND; true if all boolean expressions are true.	False && true is evaluated to false.
!	Conditional-NOT; true if expression is false.	! False is evaluated to true.

LOGICAL NOT OPERATOR:

- Logical NOT operator is used to reverse the logical state of its operand. If a condition is true then

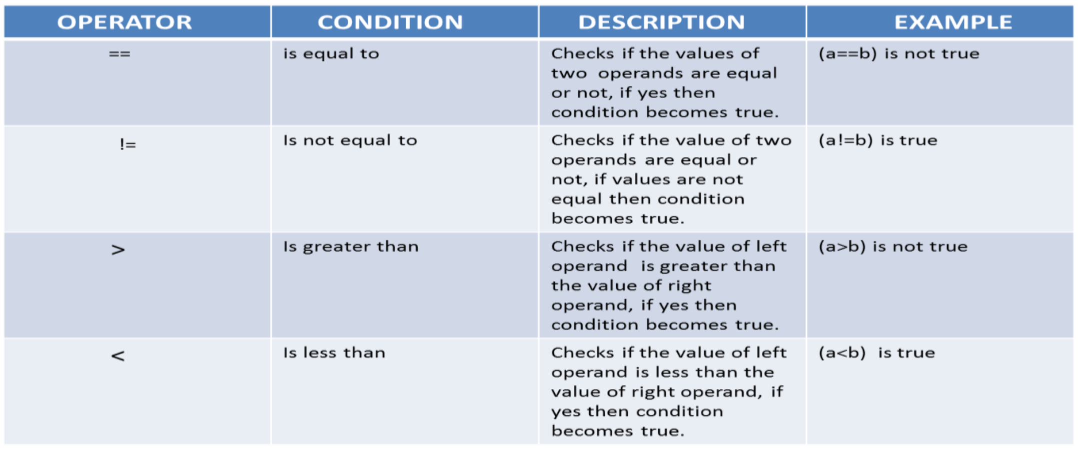
  logical NOT operator will make false. If a condition is false then Logical NOT operator will make true.
- Then NOT operator is probably the easiest to understand. It is simply the opposite of what the condition says.

```
EX:             boolean a=true;
          if(!a)
                  System.out.println(“u r win”);
                 else
         System.out.println(“u r not win”);
```

- In above example “if not true” is asking if the variable “a” variable is not true, otherwise known as false.
- If “a” variable is false, java will displays ”u r win” “a” variable is not true , so that code will not execute, then the else part is execute shown in output.

RELATIONAL OPERATOR:

- This operator is used to compare the two values, so this operator is also known as “comparison operator”
- Conditional symbols and their meanings for comparison operator are below:



OPERATOR	CONDITION	DESCRIPTION	EXAMPLE
==	is equal to	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(a==b) is not true
!=	Is not equal to	Checks if the value of two operands are equal or not, if values are not equal then condition becomes true.	(a!=b) is true
>	Is greater than	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(a>b) is not true
<	Is less than	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(a<b) is true

TERNARY OPERATOR:

In ternary operator use three operands. It is also called Conditional assignment statement because the value assigned to a variable depends upon a logical expression.

SYNTAX:

```
        variable=(test expression)?Expression 1: Expression 2
```

EX:

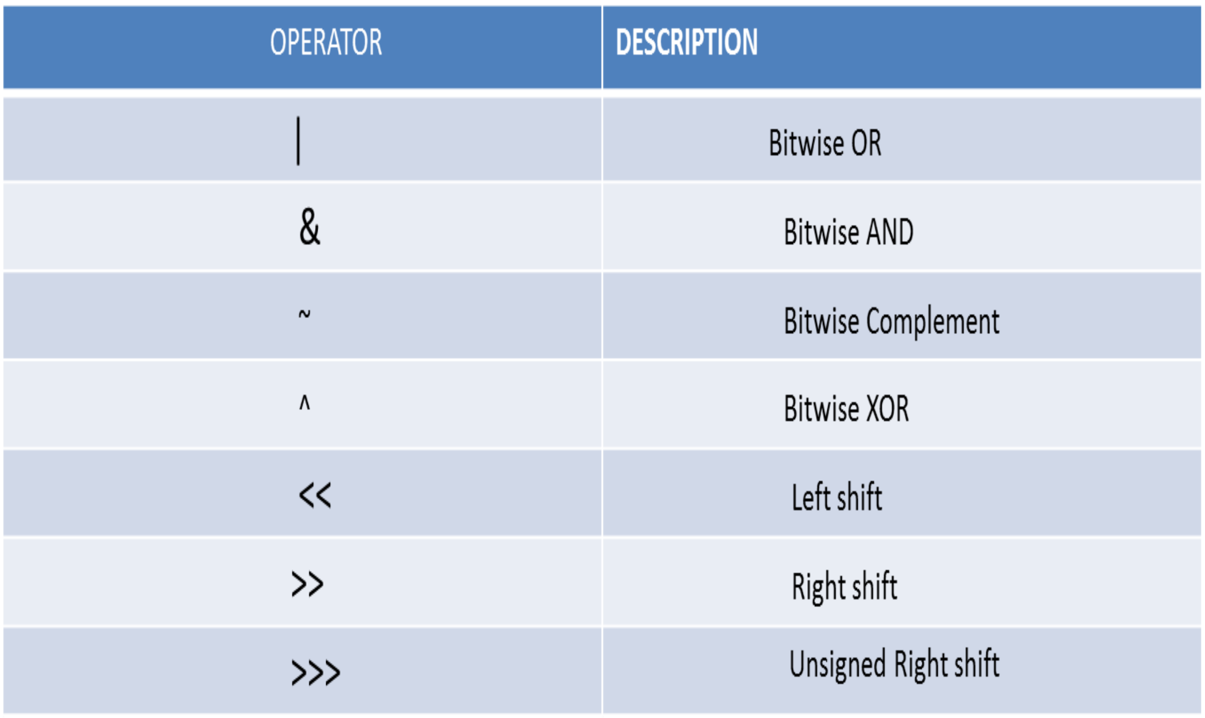
```
                c=(a>b)?a:b:
               c= (a>b) ? a:b;
                Test condition  ? Expression1 : Expression2;
```

BITWISE OPERATORS:

Java provides 4 bitwise and 3 bit shift operators to perform bit operations.

* **|** Bitwise OR
* **&** Bitwise AND
* **-** Bitwise Complement
* **^** Bitwise XOR
* **<<** Left shift
* **>>** Right shift
* **>>>** Unsigned Right Shift

Bitwise and bit shift operators are used on integral types (byte, short, int and long) to perform bit-level operations.



OPERATOR	DESCRIPTION
|	Bitwise OR
&	Bitwise AND
~	Bitwise Complement
^	Bitwise XOR
<<	Left shift
>>	Right shift
>>>	Unsigned Right shift

BITWISE OR:

Bitwise OR is a binary operator(operates on two operands). It's denoted by |. The | operator compares corresponding bits of two operands. If their of the bits is 1. If not, it gives 0.

EX:

```
12= 00001100
25= 00011001
     Bitwise OR Operation of 12 and 25
                  00001100
                   00011001
                   ----------------
                    00011101     =29(In decimal)
                   ----------------
```

BITWISE AND :

Bitwise AND is a binary operator (operates on two operands). It's denoted by &. The & operator compares corresponding bits of two operands. If both bits are 1. If either of the bits is not 1, it gives 0.

```
EX:          12=  00001100
             25=00011001
                Bit operation of 12 and 25
                      00001100
                      00011001
                     ----------------
                      00001000  = 8(in decimal)
                      ----------------
```

BITWISE COMPLIMENT :

Bitwise compliment is an unary operator(works on only one operand). It is denoted by ~. The ~ operator inverts the bit pattern. It makes every 0 to 1, and every 1 to 0.

```
EX:              35=  00100011(in binary)
                              bitwise complement of 35
                                  ~ 00100011
                                   ---------------------
                                  11011100  =    220(in decimal)
                                  -----------------------
```

BITWISE XOR :

Bitwise XOR is a binary operator(operates on two operands). It's denoted by "^" . The operator compares corresponding bits of two operands. If corresponding bits are different, It gives 1. If corresponding bits are same, it gives 0.

```
EX:                    12=00001100
                        25=00011001
                          Bitwise XOR operation of 12 and 25 is:
                   00001100
                 1 00011001
                ---------------------------
                  00010101    =21(in decimal)
                ---------------------------
```

Control Statements

Causes the flow of execution to advance and branch based on changes to the state of program.

In Java, control statements can be divided into the following three catego

1) Selection Statements
2) Iteration Statements
3) Jump Statements

1) Selection Statements

Selection statements allow you to control the f
outcome of an expression or state of a va
can be divided into the following ca

a) The if and if-else statemen
b) The if-else statemen
c) The if-else-if statement
d) The switch

The if statements :

The first contained statement (that can be a block) of an if statement only executes when the specified condition is true. If the condition is false and there is not else keyword then the first contained statement will be skipped and execution continues with the rest of the program. The condition is an expression that returns a boolean value.

General form of simple if statement is

```
if<expression>
{
        Statement-block;
}
```

The statement-block may be single statement or a group of statements .

if the expression is true, the statement block will be executed, otherwise the statement block will be skipped to the statement-x.

if else statement:-

if else statement is an extension of the simple if statement. The general form is

```
if(expression)
{
        True-block statements
}
Else
{
        False-block statements
}
```

- if the test expression is true, then the true-block statements immediately following the if statement are executed. Otherwise, the false-block statements are executed.
- In either case, Either true-block or false-block will be executed, not both.
- In both the cases, the control is transferred subsequently to the statement-x Diagram

Nested if else statement :-

- A nested if is an if statement that is the target of another if or else.
- Nested ifs are very common in programming
- General form of Nested if looks like
- Nested if else statement is made by placing one if else in another if else statement.
- Nested if else statement helps to select one out of many chooses.
- General form of Nested if else is

```
if<cond1>
    {
            if<cond2>
            {
                    if<cond3>
                            stmt 4
                    else
                            stmt3
                    }
                    else
                            stmt2
                    }
                    else
                            stmt 1
```

- In the nested if else statement, the outermost if is evaluated first.
- If the condition1 is false, the statement is the outermost else is evaluated and if else ends.
- If the conditon1 is true, the control goes to execute the next inner if statement.
- If conditon2 is false, statement2 is executed otherwise conditon3 is evaluated
- If condition3 is false statement3 is executed. Otherwise statement is executed.

else if ladder:-

- ❑ A common programming construct that is based a sequence of nested is based upon a sequence of nested ifs is the if else if ladder.
- ❑ • General form of if else ladder

```
if<condition>
        stmt
else if<condition>
        stmt;
else if<condition>
        stmt;
else
        stmt;
```

- ❑ The if statements are Executed from the top down. As soon as one of the conditions controlling the if is true, the stmt associated with that if is Executed, and the rest of the ladder is bypassed.
- ❑ If none of the condition is true, then the final else stmt will be executed
- ❑ The final else acts as a default condition; i.e if all other conditional tests fail, then the last else stmt is performed.
- ❑ If there is no final else and all other condition are false.

Switch statement:-

- The switch statement helps to select one out of many chooses.
- It often provides a better alternative than a large d=series of if else if statements
- General form of switch statement is

```
Switch(expression)
{
        Case value 1:stmt1;
                Break;
        Case value 2:stmt2;
                Break;
                .
                .
        Case value N: stmt N;
                Break;
        Default:    stmt;
}
```

- The expression must be of type byte, short, int or char.
- Each of the values specified in the case stmts must be of a type compatible with the expression.
- Each case value must be unique literal.
- Duplicate case value are not allowed.
- The switch stmt works like this

while:-

The while loop is java's most fundamental loop stmt

- It repeats a stmt or block while its controlling expression is true.
- The general form of while stmt is

```
While <condition>
{
Body of the loop
}
```

The condition can be any Boolean expression.

The body of the loop will be executed as long as the conditional expression is true

When condition becomes false, control passes to the next line of code immediately following the loop.

The curly braces are unnecessary if only a single stmt is being repeated.

Do-while:-

- ❑ If the conditional expression controlling a while loop is initially false, then the body of the loop will not executed at all
- ❑ However, it is desirable to execute the of a loop at least once even if condition expression is false to begin with
- ❑ Fortunately, java supplies a loop that does just that : the do while
- ❑ The do while loop always execute its body at least once, because its conditional expression is at bottom of loop
- ❑ The general form of do while is

```
do
{
        Body of the loop
}
While<condition>
```

For statement

- General form of traditional for statement

```
isfor(initialization; condition;
iteration)
        {
         Body of the loop
        }
```

- It is important to understand that initialization expression is only executed once. Next, condition is evaluated. This must be a Boolean expression .i.e the loop controlvariable against a target value.

- If this expression is true, then the body of the loop is executed.
- If it is false, the loop terminates.
- Next, the iteration portion of the loop executed
- This is usually an expression that increments or decrements the loop controlsvariable.
- This loop then ITERATES
- First evaluating the conditional expression , then executing the body of the loop ,and then executing the iteration expression with each pass.
- This process repeats until the controlling expression is false.

Nested loop:-

Like all other programming languages, java allows loops to be nested.
i.e one loop may be inside
anotherEg:-

```
For(i=0 ; i<10 ; i++)
{
        For(j=I ; j<10 ; j++)
        {
                statement block
        }
}
```

Jump stmts:_

Java supports 3 jump stmts

1. break
2. continue
3. return.

Break stmt:-

It has 3 uses.

1. It terminates a stmts sequence in a switch stmt.
2. If can be used to exit a loop.
3. If can be used as a "civilized" form of goto.

When a break stmt is encountered inside a loop. The loop is terminated and program control resumes at the next stmt following the loop.

i.e by using break, we can force immediate termination of a loop, by passing the conditional expression (eg: i<=10) and any Remaining code in the body of the loop.

continue:-

sometimes, you might want to continue running the loop but stop continue running the

remainder of the code in its body for this particular iteration

the continue stmt performs such as an action

Return:-

Return stmt is used to explicitly return from a method

i.e it causes program control to transfer back to the caller of the method

return stmt can be used to cause execution to branch back to the caller at the method.