SNS COLLEGE OF TECHNOLOGY

Coimbatore-35
An Autonomous Institution

Accredited by NBA – AICTE and Accredited by NAAC – UGC with 'A+' Grade
Approved by AICTE, New Delhi & Affiliated to Anna University, Chennai

DEPARTMENT OF INFORMATION TECHNOLOGY

OBJECT ORIENTED PROGRAMMING USING JAVA

I YEAR - II SEM

UNIT 1 – Introduction to Object Oriented Programming

TOPIC 3 – JAVA DataTypes and Constructors

Data Types

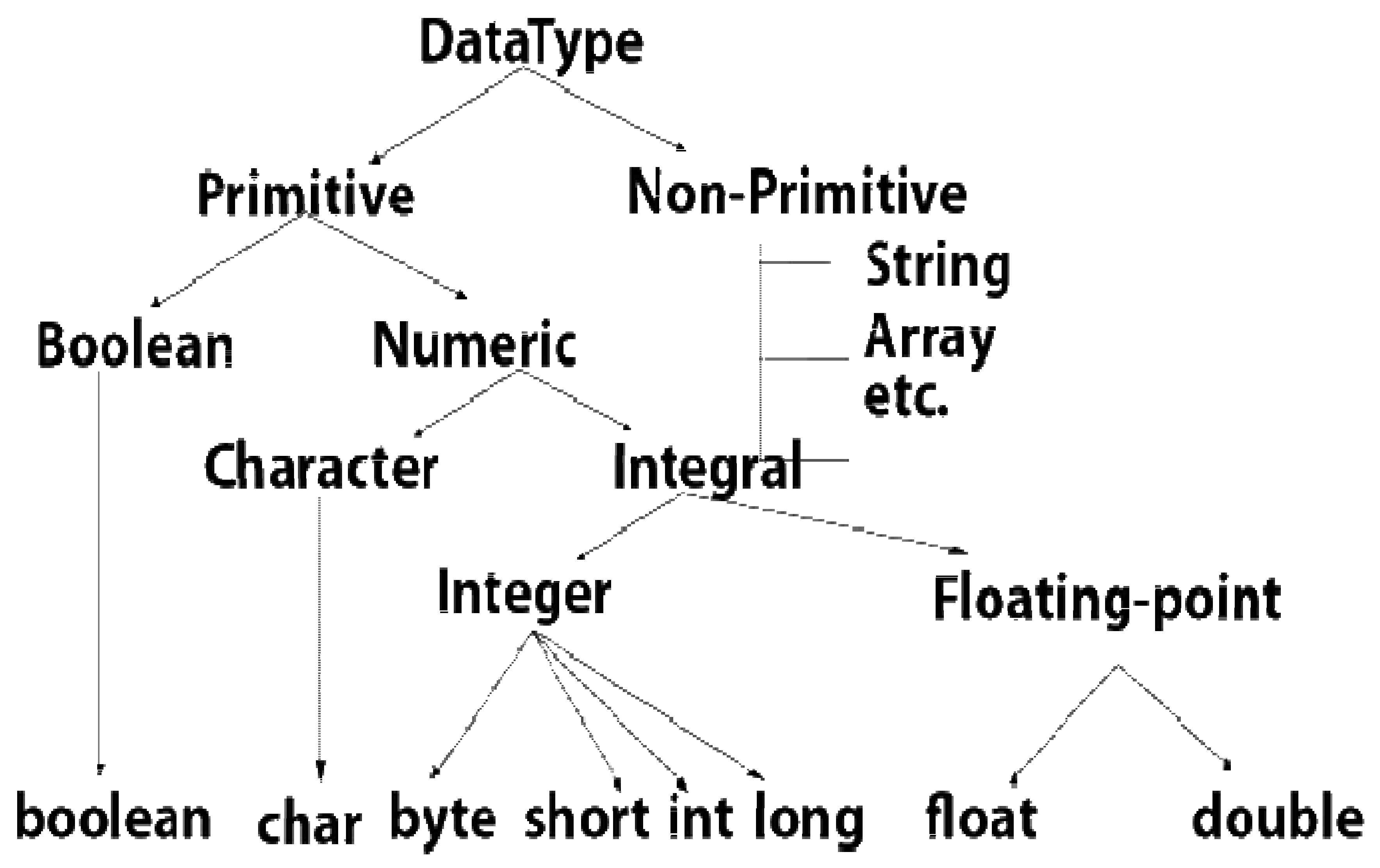
- Data types specify the different sizes and values that can be stored in the variable.

- There are two types of data types in Java:

- <u>Primitive data types: </u>The primitive data types include boolean, char, byte, short, int, long, float and double.

- <u>Non-primitive data types: </u>The non-primitive data types include Classes, Interfaces, and Arrays.

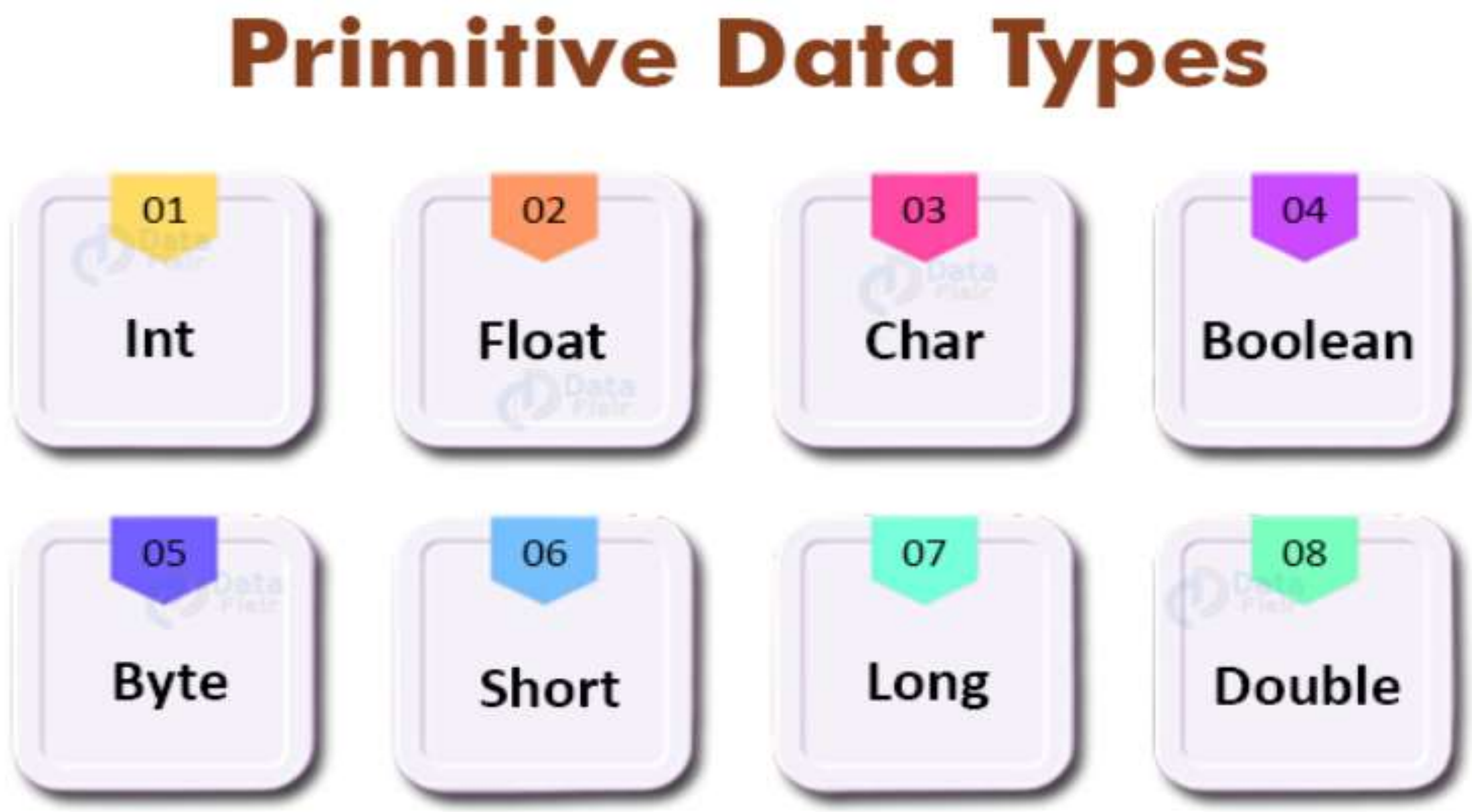
Data Types


DataType
Primitive
Non-Primitive
String
Array
etc.
Boolean
Numeric
Character
Integral
Integer
Floating-point
boolean
char
byte
short
int
long
float
double

Data Types & its size

Data Type	Default Value	Default size
boolean	false	1 bit
char	'\u0000'	2 byte
byte	0	1 byte
short	0	2 byte
int	0	4 byte
long	0L	8 byte
float	0.0f	4 byte
double	0.0d	8 byte

Data Types & its size


Primitive Data Types

01 Int

02 Float

03 Char

04 Boolean

05 Byte

06 Short

07 Long

08 Double

Data Types & its size

Numbers

Primitive number types are divided into two groups:

Integer types stores whole numbers, positive or negative (such as 123 or -456), without decimals. Valid types are `byte`, `short`, `int` and `long`. Which type you should use, depends on the numeric value.

Floating point types represents numbers with a fractional part, containing one or more decimals. There are two types: `float` and `double`.

Example

```
byte myNum = 100;
System.out.println(myNum);
```

Integer Types

Byte

The `byte` data type can store whole numbers from -128 to 127. This can be used instead of `int` or other integer types to save memory when you are certain that the value will be within -128 and 127:

Data Types & its size

Short

The `short` data type can store whole numbers from -32768 to 32767:

Example

```
short myNum = 5000;
System.out.println(myNum);
```

Int

The `int` data type can store whole numbers from -2147483648 to 2147483647. In general, and in our tutorial, the `int` data type is the preferred data type when we create variables with a numeric value.

Example

```
int myNum = 100000;
System.out.println(myNum);
```

Long

The `long` data type can store whole numbers from -9223372036854775808 to 9223372036854775807. This is used when int is not large enough to store the value. Note that you should end the value with an "L":

Example

```
long myNum = 15000000000L;
System.out.println(myNum);
```

Data Types & its size

Floating Point Types

You should use a floating point type whenever you need a number with a decimal, such as 9.99 or 3.14515.

Float

The `float` data type can store fractional numbers from 3.4e−038 to 3.4e+038. Note that you should end the value with an "f":

Example

```
float myNum = 5.75f;
System.out.println(myNum);
```

Data Types & its size

There are eight primitive data types in Java:

Data Type	Size	Description
byte	1 byte	Stores whole numbers from -128 to 127
short	2 bytes	Stores whole numbers from -32,768 to 32,767
int	4 bytes	Stores whole numbers from -2,147,483,648 to 2,147,483,647
long	8 bytes	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4 bytes	Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits
double	8 bytes	Stores fractional numbers. Sufficient for storing 15 decimal digits
boolean	1 bit	Stores true or false values
char	2 bytes	Stores a single character/letter or ASCII values

Variable

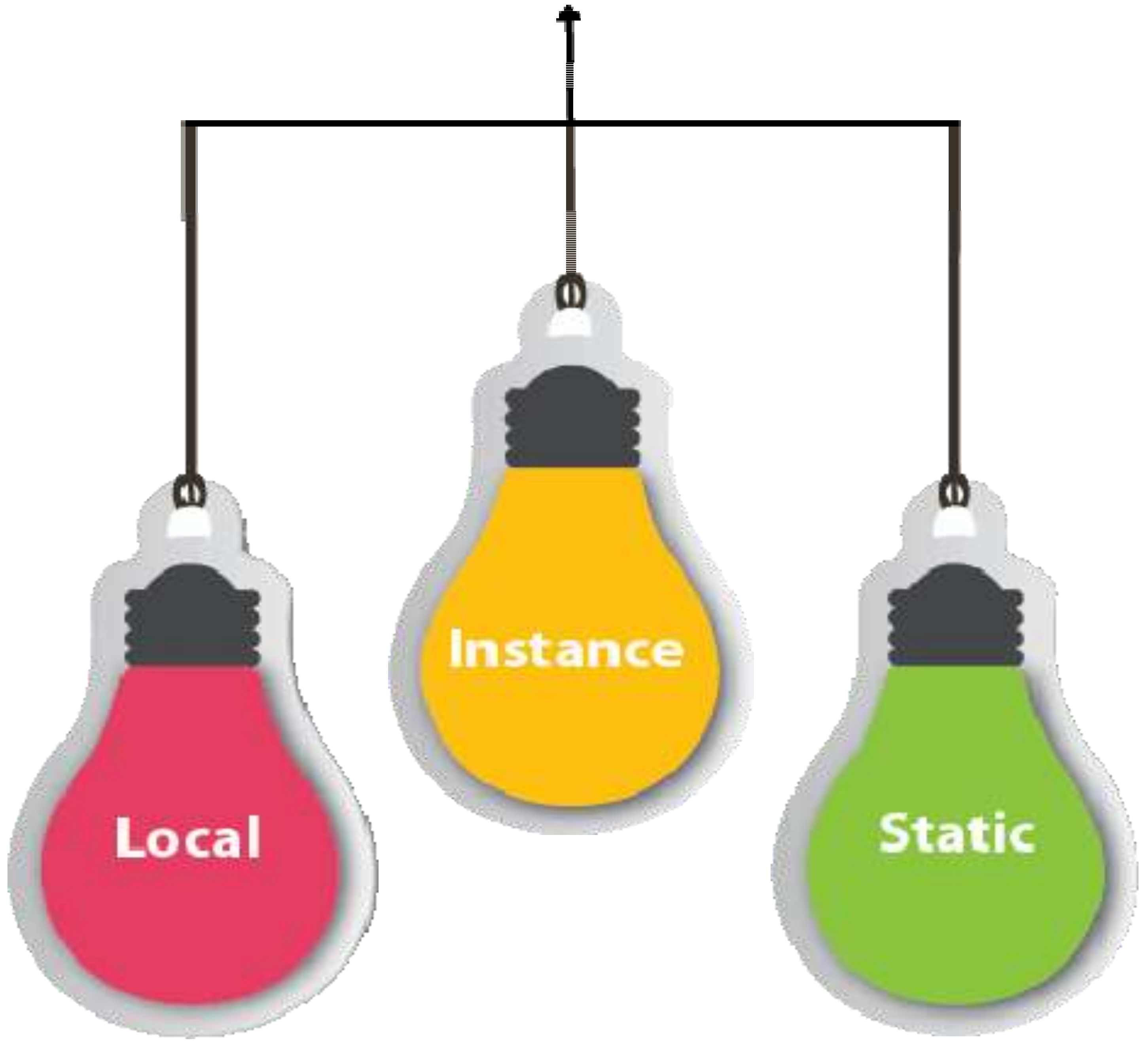
- ➢A variable is a container which holds the value while the Java program is executed.

- ➢A variable is assigned with a data type.

- ➢Variable is a name of memory location.

- ➢There are three types of variables in java:
  - ➢1. local variable
  - ➢2. instance variable
  - ➢3.Static variable

Types of Variables



Local Variable

- A variable declared inside the body of the method is called local variable.

- It could be used within that method

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.

Static Variable

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

Example

```
class A
{
int data=50;//instance variable
static int m=100;//static variable
Public static void main(String args[])
{
int n=90;//local variable
}
}//end of class
```

Operators

- ➢Operator in Java is a symbol which is used to perform operations.
- ➢For example: +, -, *, / etc.
- ➢Types of operators
  - ➢Unary Operator,
  - ➢ Arithmetic Operator,
  - ➢Shift Operator,
  - ➢Relational Operator,
  - ➢Bitwise Operator,
  - ➢Logical Operator,
  - ➢Ternary Operator and
  - ➢Assignment Operator

Arithmetic
Relational
Logical
Ternary
Shift
Bitwise
Assignment
Unary

Operator Precedence

Operator Type	Category	Precedence
Unary	Postfix	*expr++ expr--*
	Prefix	++*expr* --*expr* +*expr* -*expr* ~ !
Arithmetic	Multiplicative	* / %
	Additive	+ -
Shift	Shift	<< >> >>>
Relational	Comparison	< > <= >= instanceof
	Equality	== !=
Bitwise	bitwise AND	&
	bitwise exclusive OR	^
	bitwise inclusive OR	|
Logical	logical AND	&&
	logical OR	||

Arithmetic Operators

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

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

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

int a = 20, b = 10, c = 0, d = 20, e = 40, f =  30;
String x = "Thank", y = "You";
// + and - operator  System.out.println("a + b = "
+ (a + b));  System.out.println("a - b = " + (a -
b));
// + operator if used with strings
// concatenates the given strings.
System.out.println("x + y = " + x + y);
// * and / operator  System.out.println("a * b = "
+ (a * b));  System.out.println("a / b = " + (a /
b));
// modulo operator gives remainder
// on dividing first operand with second
System.out.println("a % b = " + (a % b));
} }
```

Unary Operators

++ :Increment operator, used for incrementing the value by 1.
Post-Increment : Value is first used for computing the result and then incremented.
Pre-Increment : Value is incremented first and then result is computed.
—: Decrement operator, used for decrementing the value by 1.
Post-decrement : Value is first used for computing the result and then decremented.
Pre-Decrement : Value is decremented first and then result is computed.

```
public class operators {
  public static void main(String[] args)
  {
    int a = 20, b = 10, c = 0, d = 20, e = 40, f = 30;
    boolean condition = true;
     //  pre-increment operator a = a+1 and then c = a;
    c = ++a;
    System.out.println("Value of c (++a) = " + c);
     // post increment operator   c=b then b=b+1
    c = b++;
    System.out.println("Value of c (b++) = " + c);
     //  pre-decrement operator d=d-1 then c=d
    c = --d;
    System.out.println("Value of c (--d) = " + c);
     // post-decrement operator   c=e then e=e-1
    c = e--;
    System.out.println("Value of c (e--) = " + c);
     //  Logical not operator
    System.out.println("Value of !condition="
                + !condition);
  }}
```

Logical Operators

&&, Logical AND : returns true when both conditions are true.
||, Logical OR : returns true if at least one condition is true.

```
import java.util.*;
 public class operators {
  public static void main(String[] args)
  {
     String x = "Sher";
     String y = "Locked";

     Scanner s = new Scanner(System.in);
     System.out.print("Enter username:");
     String uuid = s.next();
     System.out.print("Enter password:");
     String upwd = s.next();
         if ((uuid.equals(x) && upwd.equals(y))
       || (uuid.equals(y) && upwd.equals(x))) {
       System.out.println("Welcome user.");
     }
     else {
       System.out.println("Wrong uid or password");
     }
  }
}
```

Shift Operators

<<, Left shift operator
>>, Signed Right shift operator

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

      int a = 10;
      int b = 20;

            System.out.println("a<<2 = " + (a<<2));

      System.out.println("a>>2 = " + (a>> 2));


      // right shift operator
      System.out.println("b>>>2 = " + (b >>2));
   }
}
```

Bitwise Operators

&, Bitwise ANDoperator: returns bit by bit AND of input values.
|, Bitwise ORoperator: returns bit by bit OR of input values.
^, Bitwise XORoperator: returns bit by bit XOR of input values.
~, Bitwise Complement Operator

```
public class operators {
   public static void main(String[] args)
   {
         int b =0x0007;

          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  = " + ~a);
      a &= b;
      System.out.println("a=  " + a);
   }
}
```