Data Type

=> Programming Language Fundamentals / Terminologies :-

-> Data Types
-> Variables
-> Tokens
    -> Literals
    -> Operators
    -> Separators
    -> Punctuators
    -> Comments
    -> Keywords / Reserved Words
    -> Identifiers

=> Data Types :-

-> The type of data that we are specifing to java is known as Data Type.

-> For example :-

10 - int
'a' - char
"deepak" - String
true - boolean

-> According to data type, languages are divided into 2 categories :-

1. Statically Type Langauges :- In this type of languages we have to specify the type of each data and thus compiler known which type of data we have provided. For examples C, C++, Java, FORTRAIN, Pascal etc

2. Dynamically Typed Langauges :- In this type of languages we dont need to specify the type of data that we have provided.
For examples : Python, JavaScript, Objective C, Ruby etc

-> Types of Data Types :-

1. Primitive Data Types (Predefined Data Type) :-

- The data types which are already provided by java and whose size are fixed are know as primitive data type.
- Examples :- There are 8 primitive data types :-
  boolean, char, byte, short, int, long, float, double
- To find the range of Integer primitive data type we can use the formula i.e. -2^(n-1) to 2^((n-1) - 1) (where n is no of bits)
- To find the range (minimum and maximum value) of primitive data types (excluding boolean) we can use static int variables i.e. MIN_VALUE & MAX_VALUE

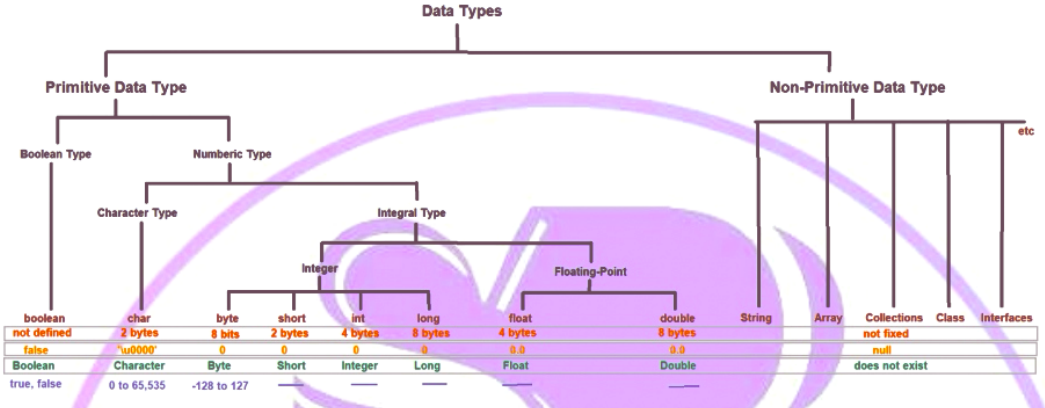
```java
package DataType;

public class ImplicitTypeCasting {
    public static void main(String[] args) {
        System.out.println("min value: " + Integer.MIN_VALUE);  //min value: -2147483648
        System.out.println("max value: " + Integer.MAX_VALUE);  //max value: 2147483647


        System.out.println("min value: " + Byte.MIN_VALUE);  //min value: -128
        System.out.println("max value: " + Byte.MAX_VALUE);  //max value: 127


        System.out.println("min value: " + Long.MIN_VALUE);  //min value: -9223372036854775808
        System.out.println("max value: " + Long.MAX_VALUE);  //max value: 9223372036854775807

    }
}
```



Data Types with default size, default values, range & their corresponding wrapper class

Data Type	Default Size	Default Value	Range	Corresponding Wrapper Class
boolean	Preciously Not Defined	false	Only true & false	Boolean
char	2 bytes (16 bits)	0 (represents blank space)	0 to 65535	Byte
byte	1 byte (8 bits)	0	-2^7 to 2^7-1 (-128 to 127)	Character
short	2 bytes (16 bits)	0	-2^{15} to $2^{15}-1$ (-32768 to 32767)	Short
int	4 bytes (32 bits)	0	-2^{31} to $2^{31}-1$ (-2147483648 to 2147483647)	Integer
long	8 bytes (64 bits)	0	-2^{63} to $2^{63}-1$ (-9223372036854775808 to 9223372036854775807)	Long
float	4 bytes (32 bits)	0.0	-3.4e38 to 3.4e38	Float
double	8 bytes (32 bits)	0.0	-1.7e308 to 1.7e308	Double

2. Non-Primitive Data Types (User Defined Data Type or Derived Data Type) :-

- Non-Primitive data types are not predefined data types but are created by the programmer. These are sometimes know as "reference variable" or "object reference"
- The size of non-primitive data type is not fixed
- Examples :- String, Array, Collection, Class, Abstract Class, Interface etc

What are Wrapper Classes

-> The classes which are used to convert primitive into objects and objects into primitive

-> There are 8 wrapper classes :- Boolean, Character, Byte, Short, Integer, Long, Float & Double

-> Java introduced autoboxing and unboxing in J2SE 5.0 version which coverts primitive into object and object into primtive automatically

What is Autoboxing & Unboxing ?

-> Autoboxing is the automatic conversion of primitive data type into its corresponding wrapper classes by java compiler not by JVM.

-> Unboxing is the automatic conversion of an object of wrapper type to its corresponding primitive value by java compiler

Main.java

```java
package DataType;

import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        int a = 10 ;
        Integer i = new Integer(a) ;  // Explicit
        Integer ii = a ;              // autoboxing
        System.out.println(a);        // 10
        System.out.println(ii);       // 10
    }
}
```

```java
package DataType;

import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        int a = 10 ;
        //Integer i = new Integer(a) ;  // Explicit
        Integer ii = a ;              // autoboxing--> internally compiler will convert to Integer iii = Integer.valueOf(a) ;
        Integer iii = Integer.valueOf(a) ;
        System.out.println(a);        // 10
        System.out.println(ii);       // 10
        System.out.println(iii);      // 10
    }
}
```

Autoboxing

```java
package DataType;

import java.util.Scanner;

public class Main {
    public static void main(String[] args) {
        Integer i = 10;
        int a = i.intValue() ;    // explicitly
        int aa = i ;              // unboxing-->internally --> (aa = i.intValue(10))
        System.out.println(i);
        System.out.println(a);
        System.out.println(aa);


    }
}

```

Unboxing

==> Why java is not purely OOP's Langauge :-

-> Java is not purely OOP's Langauge because:-

1. Usage of Primitive Data Types
2. Usage of Static members

*Static member Store in method area and that area is for class not for the object

==>How primitive variables passed to methods - by value or by reference :-

-> Java supports only pass by value

```java
package DataType;

import java.util.Scanner;

public class Main {

    public int  sum(int a, int b){  1 usage
        return a+b ;
    }
    public static void main(String[] args) {
        Main main = new Main() ;
        main.sum( a: 10,  b: 20) ;   // --> Pass by value
    }
}

```

=> Type Casting :-

-> The process of converting the data from one data type to another data type is

-> There are 2 types of Type-Casting in Java :-

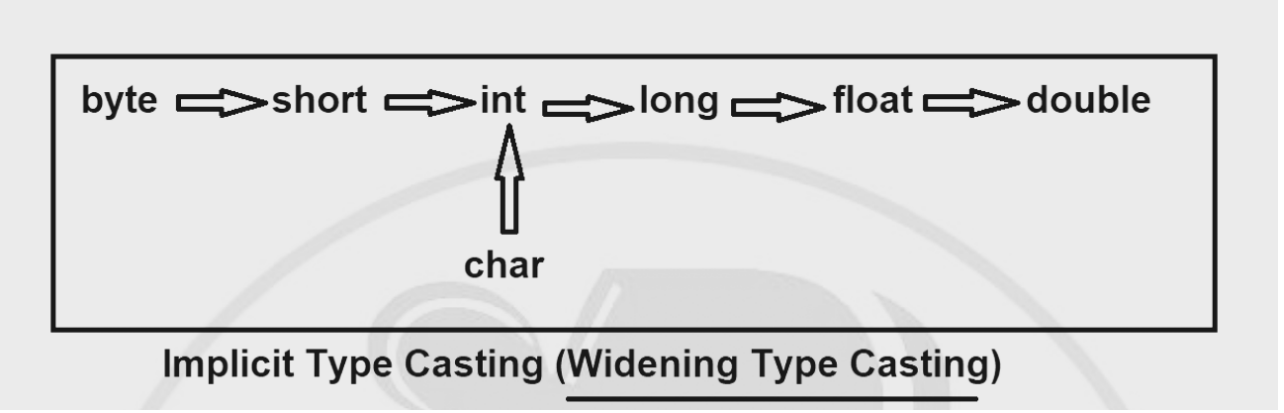
1. "Primitive Data Type" Type Casting
    1.1 Widening Type Casting (Implicit Type Casting)
    1.2 Narrowing Type Casting (Explicit Type Casting)

*boolean cannot be typecast

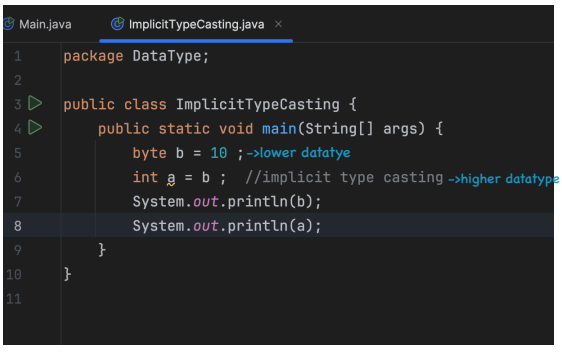
1. "Primitive Data Type" Type Casting

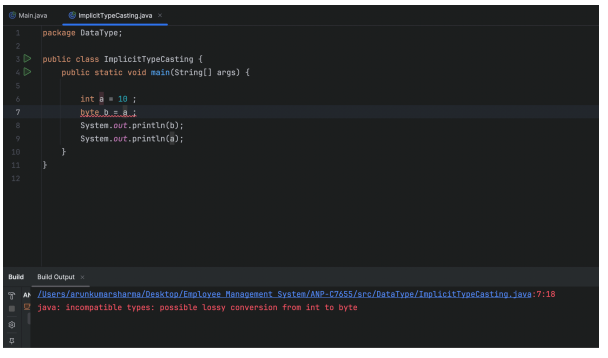
1.1 Widening Type Casting (Implicit Type Casting)

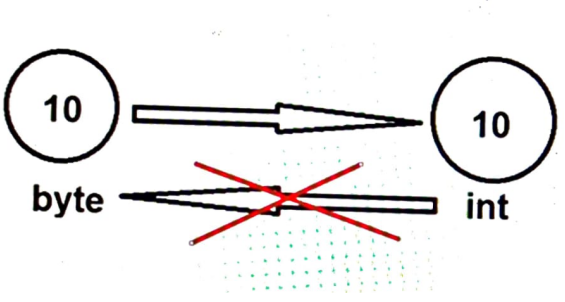
-> It is the process of converting data from lower data type to higher data type

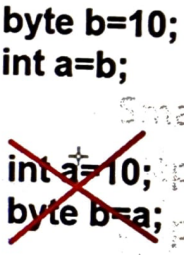


int can be converted to long, float, double, similarly long can be converted to float or









Why java is not purely OOP's Langauge :-

-> Java is not purely OOP's Langauge because:-

1. Usage of Primitive Data Types
2. Usage of Static members

What is Type Checking & Type Casting ?

-> Type Checking : Type Checking is the responsibility of compiler. It checkes whether the syntax is correct or not and whether we are assigning lower data type to higher data type.

-> Type Casting : Type Casting is the responsibility of JVM. In this phase lower data type (for eg byte) will convert into higher data type (for eg int) and the value will be copied in higher data type

Main.java ImplicitTypeCasting.java

```java
package DataType;

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

        byte b1 = 127 ;
        System.out.println(b1);  // no error since 127 is in the range of byte

        byte b2 = 128 ;   //error since 128 is not in the range of byte


    }
}
```

Main.java ImplicitTypeCasting.java

```java
package DataType;

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

        byte b1 = 10 ;
        byte b2 = 20 ;
        byte b = b1 + b2 ;

        /*
        error bcoz if no1 and no2 are in byte, short, int then the result will be in int only
         */
    }
}
```

==>

Main.java ImplicitTypeCasting.java

```java
package DataType;

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

        byte b1 = 10 ;
        byte b2 = 20 ;
        int b = b1 + b2 ;

        /*
        error bcoz if no1 and no2 are in byte, short, int then the result will be in int only
         */
    }
}
```

=> Note :if we have no1 & no2 , result

no1 + no2 = result

1. If no1 & no2 is byte, short or int then result will be always in int
2. If no1 & no2 (anyone of them) is long, float, double etc then result will be in higher data type

```java
class ItcDemo1
{
        public static void main(String[] args)
        {
                //byte b1=10;
                //byte b2=20;
                //int res=b1+b1;
                //------------------------
                //byte b1=10;
                //short s1=20;
                //int res=b1+s1;
                //------------------------
                int i1=10;
                short s1=30;
                int res=i1+s1;
        }
}
```

Narrowing Type Casting (Explicit Type Casting)

-> In this case we convert higher data type into lower data type

-> Narrowing Type Casting can be achieved by using "cast operator"

Main.java ImplicitTypeCasting.java

```java
package DataType;

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

        int no1 = 10 ;
        byte no2 = (byte)no1 ;  // (byte) --> cast Operator

        // ====================

        int n1 = 20 ;
        short n2 = (short)n1 ;


        // ====================

        int a1 = 20 ;
        short a2 = (byte)n1 ;  //short se chote me bhi kar sakte hai, i.e., byte me

    }
}
```

Main.java ImplicitTypeCasting.java

```java
package DataType;

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

        int no1 = 130 ;
        byte no2 = (byte)no1 ;   // the code will compile but answer will be wrong since 130 is out of range for byte
        System.out.println(no2);

    }
}
```

"User Defined Data Type" Type Casting :-

-> It is the process of converting data from one user defined data type to another user defined data type

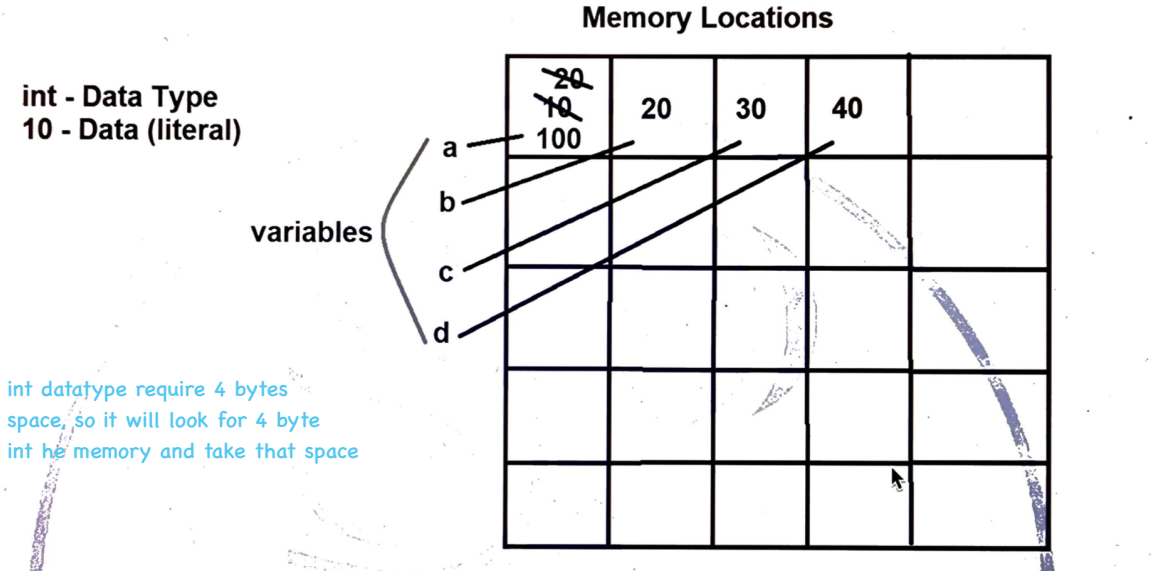
-> For "user defined data type" type casting, both data types should have relation (either extends or implements)

```
class Object
{

}
class String extends Object
{

}
class Test
{
        String name="deepak";
        Object o=(Object)name;
}
```

Variables



=> Variable :

-> Variable is the name of memory location that contains the data

-> The variable value can changed according to programming logic

-> Every variable has its data type

-> Examples :

int a=10; (a is the variable)

char c='a'; (c is the variable)

-> Types of variables :-

1. Local Variables
2. Instance Variables
3. Static Variables

1. LOCAL VARIABLES :-

-> Declaration : Local variables are declared within the body of methods, constructors or blocks

-> Scope : Local variables can be used within the methods or constructors or blocks but not outside them

-> When local variables gets memory allocated : Local variables gets allocated when the methods or constructors or blocks are executed and get get deleted from memory when that method or block or constructor execution completes

-> Stored Memory Area : Local variables gets memory allocated in "STACK AREA"

-> Default Values : Local variables does not have any default value, if we dont provide the value for local variables and use them, it will provide compile time error saying "variable variable_name might not have been initialized"

-> Access Modifiers : We cannot use access modifiers i.e. public, protected and private with local variables

-> How to access Local Variables :

1. directly

2. INSTANCE VARIABLES :-

-> Declaration : Instance variables are declared within the class but outside the methods or constructors or blocks

-> Scope : Instance variables can be used within the class and every method or block or constructor but not inside the static methods or static blocks

-> When instance variables gets memory allocated : Whenever new object is created, instance variables gets memory allocated and when that object is destroyed instance variables also gets deleted

-> Stored Memory Area : Instance variables are stored in "HEAP AREA"

-> Default Values : Instance variables have default values for ex int - 0; boolean - false; float - 0.0; etc

-> Access Modifiers : We can use access modifiers i.e. public, protected and private with instance variables

-> How to access Instance Variables :

1. directly
2. by using object name

3. STATIC VARIABLES :-

-> Declaration : Static variables are also declared within the class but outside the methods or constructors or blocks and we also use "static" keyword with them

-> Scope : Static variables can be used in every methods or static methods or blocks or static blocks or constructors

-> When static variables gets memory allocated : When we run java program, .class file or byte code gets loaded in JVM and at that time only static variables also gets memory allocated. And when the .class file gets unloaded from JVM static variables gets destroyed from the memory

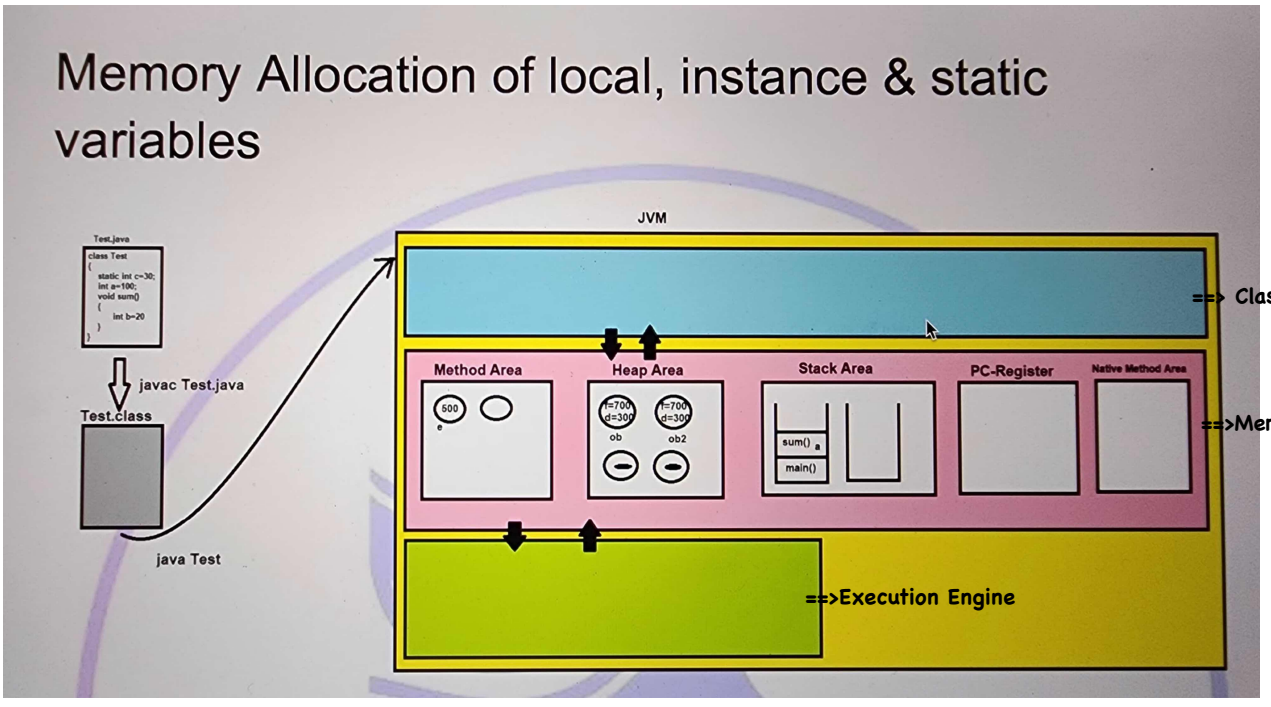
-> Stored Memory Area : Static variables are stored in "METHOD AREA"

-> Default Values : Static variables have default values for ex int - 0; boolean - false; float - 0.0; etc

-> Access Modifiers : We can use access modifiers i.e. public, protected and private with static variables

-> How to access Static Variables :

1. directly
2. by object name
3. by class name

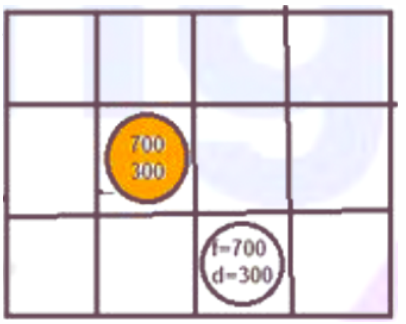


*details of Which line to execute in main method is stored in PC Register

Program flow

```
class VariablesDemo2
{
    int f=700;
    int d=300;          //instance variable
    static int e=500; //static variable
    void sum()
    {
        int a=100;  //local variable
    }
    void mul()
    {
        int b=200;  //local variable
        //int c=a+b;        //error because we cannot use local variable a inside mul() method
        int c=d+b+e;
    }

    static void divide()
    {
        int rollno=101;   //local variable
        //int res=d; //error because d is not static variable
    }
    public static void main(String[] args)
    {
        VariablesDemo2 ob=new VariablesDemo2();
        ob.sum();
        VariableDemo2 ob2=new VariableDemo2();
    }
}
```



=> Tokens :-

-> Tokens are the smallest unit or say small building blocks of java program that are meaningful to the java compiler

-> For example :System.out.println("hello");

Tokens are : System, ., out, println, (,), hello, ;

-> Our java program converts into tokens and then java compiler converts these tokens into java bytecode

-> Different types of tokens are :-

1. Literals
2. Operators
3. Separators
4. Punctuators
5. Comments
6. Keywords/Reserved Words

1. Literals :

-> Any contant value assigned to the variable is known as literal

-> For example :-

```
int a=10; //10 is literal
char c='x'; //x is literal
String name="deepak"; //deepak is literal
boolean b=true;
```

-> Types of literals :-

1. Intergal Literals (byte, short, int, long)
2. Floating-Point Literals (float, double)
3. Character Literals (Single Quotes ['], Char literal in Integer literal, unicode representation,

2. Operators :-

-> Operators are the special symbols which are used to perform any operation on one or more operands

-> For example :-

c=a+b; (+ is operator and a,b are operands)

-> Types of operators :-

1. Arithmetic Operators

   +, -, *, /, %

2. Unary Operators

   posfix : no++, no--

   prefix : ++no, --no, +no, -no, ~no, !no

```
public class ImplicitTypeCasting {
    public static void main(String[] args) {
        int no = 10 ;
        System.out.println(no);        // 10
        System.out.println(no++);     // 10
        System.out.println(++no);    // 12
        System.out.println(no);     // 12
    }
}
```

```
public class ImplicitTypeCasting {
    public static void main(String[] args) {
        int no = 10 ;
        System.out.println(no+++no);   //21
    }
}
```

```
public class ImplicitTypeCasting {
    public static void main(String[] args) {
        int no = 10 ;
        System.out.println(++no+++no);   // error--> confuse in post increment and pre increemement
        System.out.println(++no-++no);    // -1
    }
}
```

3. Assignment Operators

   =, +=, -=, *=, /=, &=, ^=, |=, <<=, >>=, >>>=

```
public class ImplicitTypeCasting {
    public static void main(String[] args) {
        int no = 10 ;
        no += 5 ;
        System.out.println(no);
    }
}
```

4. Relational Operators

   ==, !=, <, >, <=, >=, instanceof (provides the output in true or false)

5. Bitwise Operators

   -> Binary Bitwise Operators

      1. Bitwise Logical Operators

         & (bitwise AND), |(bitwise OR), ^(bitwise exclusive OR)

      2. Shift Operators

         >> (right shift), << (left shift), >>> (zero fill right shift)

   -> Unary Bitwise Operators

      One's compliment Operator

6. Logical Operators

   && (logical AND), || (logical OR)

7. Ternary Operator

   ?: (variable = condition ? expression 1 : expression2)

```
public class TernaryDemo {
    public static void main(String[] args) {
        int no1 = 10 ;
        int res = (no1==10)? no1+10 : no1-10 ;
        System.out.println(res);
    }
}
```

Bitwise Operator Table

no1	no2	no1 & no2	no1 | no2	no1 ^ no2
0	0	0	0	0
1	0	0	1	1
0	1	0	1	1
1	1	1	1	0

What is instanceof operator ?

-> This is used to verify if the specified object is the instance of specified class

==> check karwane ke liye kaam aata hai ki jo object hai wo proper class humne specify ki hui hai usi ka instance hai ki nahi

```java
public class TestDemo {
    public static void main(String[] args) {
        TestDemo td = new TestDemo() ;
        System.out.println(td instanceof TestDemo);  //true
    }
}
```

```java
public class TestDemo {
    public static void main(String[] args) {
        TestDemo td = new TestDemo() ;
        System.out.println(td instanceof Object);  //true
    }
}
```

What is difference between & and && operator?

-> & - Bitwise AND
&& - Logical AND
-> & - Operates on bit values
&& - Operates on boolean values
-> & - In this case both sides will get evaluated because there is no true or false case here
&& - In this case if first expression is false,then it will not evaluate the second condition and it will return false directly

=> Separators :-

-> A separator is a symbol that is used to separate a group of code from one another.
-> Separator help to define the java program structure
-> Types of separators :-
1. () - Parentheses :
> Encloses the arguments in method defination and calling
> Adujst the precedence in arithmetic expressions
> Used in testing the expressions in control statements
> Used in casting
2. {} - Braces :
> Used to define the block of class or methods or blocks
> Used to put values in arrays
3. [] - Brackets :
> Used to declare an arrays
> Used when dereferencing array values
4. ; - Semicolon :
> Used to separate or terminate the statements
5. , - Comma :
> Used to separate the variables declaration
> Used to chain statements together inside a for loop
6. . - Period
> Used to select methods or fields from an object

=> Punctuators :-

-> Punctuators are the symbols or tokens that has semantic meaning to the compiler

-> Types of punctuators :-

1. ? - Question Mark

> Used in ternary operator

2. : - Colon

> Used after loop labels

3. :: - Double Colon

> Used to create method or constructor references

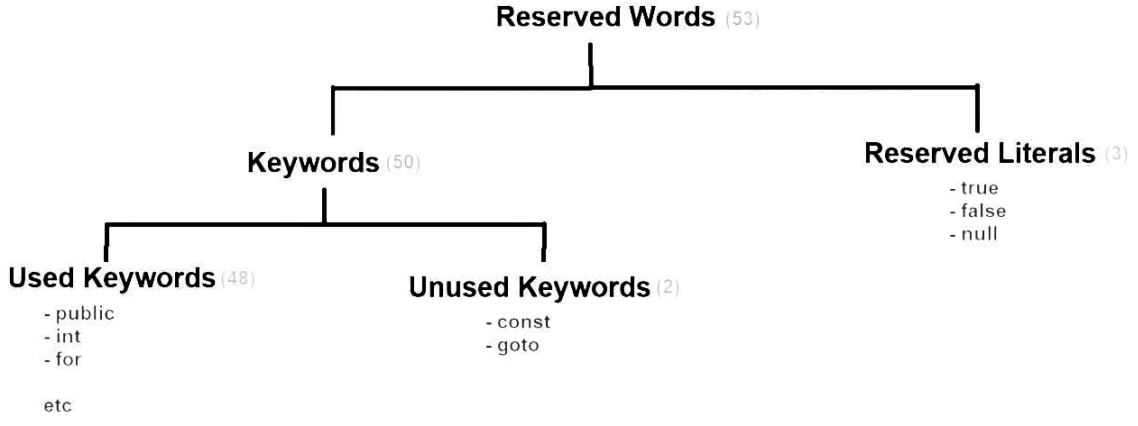
=> Comments :-

-> There are three types of comments :-

1. Single Line Comment - //
2. Multiline Comment - /* ------ */

=> Keywords/Reserved Words :-

-> Keywords are the predefined words having any specific meaning



List of Keywords in Java

Primitive Keywords	Control Statements Keywords	Keywords used in OOP			Return Type Keyword	Others	Exception Handling Keywords
		Class Related Keywords	Object Related Keywords	Access Modifiers			
- boolean - char - byte - short - int - long - float - double	- if - else - switch - case -default - for - while - do - break - continue - return	- class - interface - package - import - extends - implements	- new - instanceof - this - super	- public - protected - private - abstract - static - final - synchronized - volatile - strictfp - native	- void	- enum	- try - catch - finally - throw - throws - assert

=> Identifiers :-

-> Identifier is any name, it can be variable name or class name or interface name or method name or package name etc.

-> Rules for identifiers :-

1. Spaces cannot be used
2. Only two symbols i.e. _ and $ can be used in identifiers name
3. Integer values cannot be used at first position but can be used after first character
4. Reserved words cannot be used as an identifier name

Which is correct

1. import or imports
2. implement or implements
3. extend or extends
4. instanceof or instanceOf