UNIT 2 Data type and Variable

- Data type

↳ A data type is a classification of data.
↳ It represent the type and size of data values that can be stored in a variable.
↳ In other words, a java data type is a set of values and operations defined on those values data types are divided into two groups:-

1) Primitive data type
2) Non-primitive data type

1) Primitive data type

↳ Primitive data type are basic data type that specifies size and type of data and can not be further divided into simpler data types.

1) Integer types

↳ It stores whole number positive and negative without decimals. Valid types are bytes, short, int and long.

Data type	Size	Description
byte	1 byte	stores whole numbers Froms. -128 to 127
short	2 byte	stores whole numbers from -32,768 to 32,767
int	4 byte	Stores whole number from -2147483,648 to 2,147,483,647

long	8 bytes	Stores whole number from -9223372,036,854,775,808 to 9,223,372,036,854,775,807

2. Floating point types

↳ It stores numbers with a fractional part containing one or more decimals two types Float and double.

Data type	Size	Description
Float	4 bytes	Stores fractional number sufficient for storing 6 to 7 decimal digits.
double	8 bytes	stores fractional number sufficient for ~~sh~~ storing 15 decimal digits

3. Character type

↳ The char data type is used to store single character.

↳ Occupies 2 bytes memory

↳ The character value must be surrounded by single quotes like A or (a).

4. Boolean type

↳ The boolean data type is used to store logical value either true or false.

↳ It occupied 1 bit memory space.

Stringe types

↳ The string type is used to store a sequence of characters (text). string value must be surrounded by double qootes like "Ram"

Ex:

```
String name = "RAM"
System.out.println(name);
```

• Variables

↳ Variable are containers for storing data values.

↳ They are names of memory location that stores different types of data specified by data types.

Declaration of variable

Syntax:

```
datatype variableName = value;
```

where,

datatype : Specifies data type such as int, float, double, char etc.

Variable_Name : specifies name of variable that must be valid identifier.

"=" is assigment operator that assigns data value to variable

Value : is data value to be assigned to variable.

Example:

```
int num = 5;
float marks = 45.5f;
char grade = 'A'
boolean mybool = true;
String name = "Bipana";
```

// Program to add numbers

```
class sum
{
public static void main(String [] args)
{
int a = 5;
int b = 6;
int sum = a+b;
System.out.println("The sum "=+sum);
}
}
```

- Constant

↳ Constant is a value that can not be changed

after assigning it.

↳ Java does not support the constant directly.

↳ There is alternative way to depline constant in java using non-access modifiers static and final.

Static and final modifiers

↳ Static modifiers is used to manage memory. It allows the variable without loading any instance of the class in which it is defined.

↳ final modifiers represents that the value of the variable can not be changed. It also makes primitive data types immutable

Syntax for creating constants

Staic final type identifier_name = value;
eg:- Static final double pi = 3.1416;

- Identifier

↳ Identifiers are symbolic names used for identification.

↳ They can be class name, variable name, method name, package name, constant name and more. These unique name are called identifiers.

Rules for naming identifiers:

1) Valid identifiers must have characters [A - Z] or [a - z], numbers [0 - 9] & underscore (_).

2) Should not start with number.

3) Should not be white space in an identifier.

4) Should be length 4 - 15 letters only However, ~~they is not~~ there is not limit on its length.

5) Keywords can be not used as identifier.

6) Query language keywords such as SELECT, FROM, WHERE, DELETE etc. can not be identifier.

7) Case sensative

Some valid identifiers:

a, sum, num1, num_1, pi, calculateArea1, RectArea etc.

- Keywords

↳ Java keywords are predefined or reserved word used for any functionality or meaning.

↳ keywords can not be used for identifiers.

↳ List of Java keywords :-

- abstract
- boolean
- break
- byte
- case
- cath
- char
- class
- continue
- default
- do
- While
- for
- if
- else

• Access Modifiers

↳ The access modifier specifies the accessibility or scope of a field, method, class or constructor.

↳ Access level can be changed by applying the access modifier on field, method, class or constructor.

↳ 4 types of Java access modifiers.

① Private
② default
③ Protected
④ Public

1) Private :- The access level of Private modifier is only within the class. It can not be accessed from outside of the class.

2) Default :- The access level of default modifier is only within the package. It can not be accessed from outside the package.

3) Protected : The access level is within package and outside the package through child class. If we do not make child class, it can't be accessed from outside the package.

4) Public : The access level of it is anywhere. It can be accessed from within the class, outside the class, within the Package and outside the package.

Example of private access modifier

```
class A
{
  private int data = 40;
  private void msg ()
{
  System.out.println ("Hello Java");
}
}

public class Test
{
  public static void main (String [] args)
{
  A obj = new A ()
System.out.println (obj.data);
  obj.msg ();
}
}
```

- Escape Sequence in Java

↳ A character preceded by a back slash (\) is an escape sequence and has a special meaning to the compiler.

↳ Following types of escape sequences in Java:

Escape sequence	Description
\t	-Inserts a tab space in the text at this point.
\b	-Inserts backspace in the text at this point.
\n	-Insert a new line in the text at this point.
\r	-Inserts carriage return in the text at this point. It is used to bring cursor to the starting of the line without changing the line
\f	-Inserts form feed in the text. It is old strategy to show page break.
\'	-Inserts single quote characters in the text at this point.
\"	-Inserts double quote characters in the text at this point
\\	-Inserts backslash character in this text at this point

- Operators in Java

↳ Operators are special symbols that performs certain operation on operands.

↳ Used to perform operation on variables and values.

↳ In java operators are divided into following categories.

1. Arithmetic operator
2. Assigment operator
3. Logical operator
4. Relational operator
5. Bitwise operator
6. ~~Shift operator~~
7. Unary operator
8. Tenary Operator
9. Arithmetic operator

↳ Used to perform basic arithmetic operations such as addition, subtraction etc.

Operator	Description	Example (let A = 20, B = 5)
(+) addition	adds values of operand	A + B (20+5) = 25
(-) Subtraction	Subtract the operand	A - B (20-5) = 15
(*) Multiplication	Multiplication the operand	A * B (20 * 5) = 100
(/) Division	Divide the operand	A / B (20/5) = 4

(%) Modulus division	divides left hand operand with right hand operand and gives remainder	A%B(20%5)=0

Example:

```
public class OperatorTest
{
    public static void main(String[] args)
    {
        int a = 20;
        int b = 5;
        System.out.println(a+b);
        System.out.println(a-b);
        System.out.println(a*b);
        System.out.println(a/b);
        System.out.println(a%b);
    }
}
```

② Assignment operator:

↳ Assignment operators are used to assign/ store value of right hand operand to right hand operand or variable.

Operators	Description	Example
=	used to assign the value on right to the operand on the left.	a=b
+=	Used to add right operand to the	

	left operand and assigns the result to left operand	a+=b ⇒ a=a+b
-=	Used to subtract right operand from left operand and assign the result to left operand.	a-=b ⇒ a=a-b
*=	Used to Multiply right operand from left operand and assign the result to left operand.	a*=b ⇒ a=a*b
/=	Used to Divide right operand from left operand and assign the result to left operand.	a/=b ⇒ a=a/b
^=	Used to perform exponential calculation on operands and assigns the result to left operand.	a^=b ⇒ a=a^b
%=	Used to Divide left operand with right operand and assign remainder to the left operand	a%=b ⇒ a=a%b

3) Logical Operator

↳ Logical operator are used to combine two or more conditions.

↳ It returns logical value either true or false.

Operators	Description	Examples
&& (Logical AND)	- Performs logical AND operation. It returns true if all inputs are true; otherwise returns false	a<5 && a>b20
|| (logical OR)	- Performs logical OR operation. It returns true if at least one input is true; otherwise returns false.	a<5 || a>20
! (logical NOT)	- Performs complement operation. It returns true if input is false and vice-versa	!(a<5 && a>20)

4) Relational Operators

↳ Relational operators are used to check the relationship between two operands

↳ If relationship (condition) is true, it returns true

value otherwise; it returns false value.

Operators	Operator name	Examples (let A=10, B=15)
==	Equal to	A==B ⇒ 10==5 ⇒ Returns false
>	greater than	A>B ⇒ 10>5 ⇒ Returns true
>=	greater than of equal to	A>=B ⇒ 10>=5 ⇒ Returns true
<	less than	A<B ⇒ 10<5 ⇒ Return false
<=	less than of equal to	A<=B ⇒ 10<=5 ⇒ Return false
!=	Not equal to	A!=B ⇒ 10!5 ⇒ Returns true

Example

```
public class RelationalOP
{
    public static void main(String [] args)
    {
        int a=10, b=5;
    System.out.println(a==b);
    System.out.println(a>b);
    System.out.println(a>=b);
    System.out.println(a<b);
    System.out.println(a<=b);
    System.out.println(a!=b);
    }
}
```

5) Bitwise Operators:

↳ These operators are used to perform bit level operations on integer and boolean data.

↳ Following types:

2 | 10 - 0
2 | 5 - 1
2 | 2 - 0
2 | 1 - 1
0

Operators	Description	Example (let a=10 b=3)
& (Bitwise AND)	- Returns 1 if all bits are 1.	a&b = 2 a&b = 0010
| (Bitwise OR)	- Returns 1 if any or all bits are 1.	a|b = 11 a|b = 1011
^ (Bitwise XOR)	- Returns 1, if only one bit is 1.	a^b = 9 a^b = 1001
~ (Bitwise NOT)	- Returns 1's complement of the operand. It is unary operator.	~a = -11
<< (Bitwise left shift)	- Moves the number of bits to the left.	a<<2 = 40
>> (Bitwise Right shift)	- Moves the number of bits to the right	a>>2 = 2

```
public class BitwiseOP
{
  public static void main (String [] args)
{
   int A = 10, B = 3
  System.out.println (A&B);
  System.out.println (A|B);
  System.out.println (A^B);
  System.out.println (A~B);
  System.out.println (A<<2);
  System.out.println (A>>2);
}
}
```

Bitwise left shift (a<<2)

A = 10 = 1010 (8 4 2 1)
1 0 1 0 0 0 (32 16 8 4 2 1)
= 40

Bitwise Right Shift (a >> 2)

A = 10 = 1 0 1 0 (8 4 2 1)
0 0 1 0 (0 0 1 0)
= 2

(6) Unary operates

↳ It takes only a single operand.

↳ Used to increament or decrement the value, negating an expression or inverting a boolean value.

(I) Increament Operator (++):

↳ Used to increament the value of operand by 1.

↳ Two types: post-increment (a++) and pre-increment (++a).

↳ In post-increment, value is first used for computing the result and then, incremented.

↳ In pre-increment, value is incremented first, and then result is computed.

(II) Decrement operator (--)

↳ Used for decrementing the value of operand by 1.

↳ Also two types: post decrement (a--) and pre-decrement (--a).

(III) Logical NOT operator (!)

↳ Used for inverting a boolean value (!b).

Sample example:

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

10	true
a	b

a++ = 10, 11

++a = 11

a-- = 10, 9

--a = 9

!b = False

7) Tenary Operator

↳ It is a shorthand version of the if-else Statement

↳ It has 3 operands and hence name Tenary

↳ The general format is:-

(Condition ? true-value : False-value)

↳ If condition is true, then execute the statement after '?' else executes the statement after ':'

Sample example

```
public class OpTest
{
public static void main (String [] args)
{
    int a = 20, b = 10;
    int result;
      result = (a>b) ? a : b;
    System.out.println ("Greater no is = " +
      result);
  }
}
```

3: If (condition)

```
{
    Statement-1;
}
else
{
    Statement-2;
}
```