

JAVA @11

Language Fundamentals and
Operators

Objective

After completing this session you will be able to understand,

- Java Keywords
- Primitive data types
- Variables & Literals
- Casting primitives
- Operators

Keywords

What are Java Keywords?

Keywords are reserved identifiers that are predefined in java language.

Keywords cannot be used as names for variables, methods and classes.

Few Points on Java Keywords

- Keywords use letters only, they do not use special characters (such as \$, _, etc.) or digits.
- All keywords are in lowercase letters and incorrect usage results in compilation errors.

Table of Keywords

abstract	default	implements	protected	throw
assert	do	import	public	throws
boolean	double	instanceof	return	transient
break	else	int	short	try
byte	extends	interface	static	void
case	final	long	strictfp	volatile
catch	finally	native	super	while
char	float	new	switch	null
class	for	package	synchronized	true
continue	if	private	this	false

Primitive Data Types

Java, like other programming languages such as C and C++, supports basic built-in data types, which are also called primitive data types.

There are eight primitive data types in Java they are,

Data Type	Size In Bits	Range	Default Value
boolean	1	true or false	false
byte	8	-2^7 to $2^7 - 1$	0
short	16	-2^{15} to $2^{15} - 1$	0
int	32	-2^{31} to $2^{31} - 1$	0
long	64	-2^{63} to $2^{63} - 1$	0L
float	32	-2^{31} to $2^{31} - 1$	0.0f
double	64	-2^{63} to $2^{63} - 1$	0.0d
char	16	0 to $2^{16} - 1$	'\u0000'

Variables in Java

Dictionary Definition of Variables:

- **A quantity that can assume any of a set of values.**
- **Something that is likely to vary; something that is subject to variation.**

Variable analogy:

The box is used as a container to hold white papers.



What is a Java variable?

Similar to the box, Java variables are the basic unit of storage in a Java program. It is used to store the state of objects.

How are variables declared?

Variables declaration:

```
<data type> <name> [=initial value];
```

Where,

Data Type – Indicates the type of value that variable can hold.

Name – Name by which the variable is identified.

Values enclosed in <> are mandatory attributes

Values enclosed in [] are optional attributes

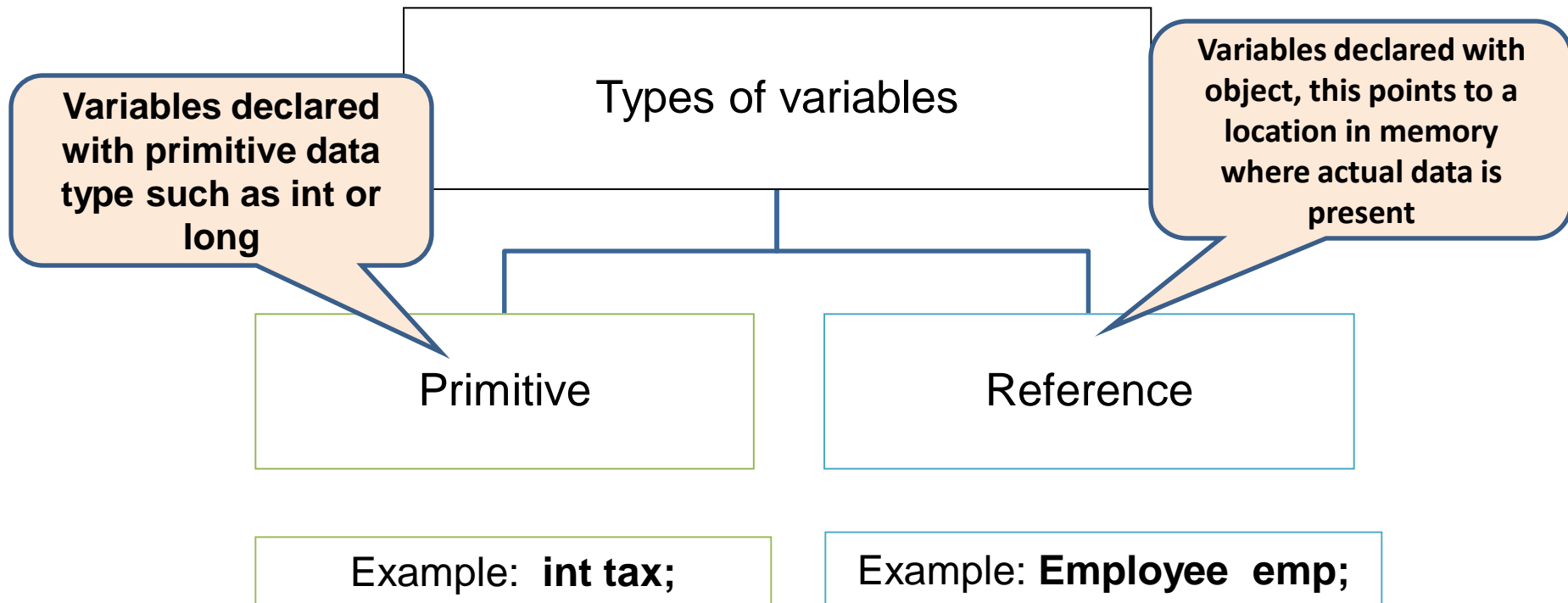
Example:

```
int salary; // Declared a variable salary with data type int.
```

(or)

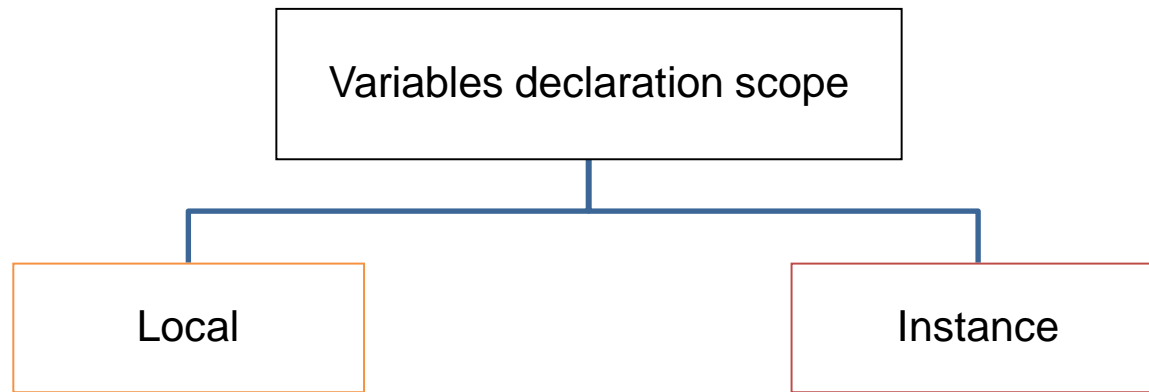
```
int salary =18000; // Salary declared as int with initial value 18000
```

Types of Variables in Java



NOTE: A primitive variable holds the value of the data item, while a reference variable holds the memory address where the object values are stored.

Scope of the Variables



- Variables declared inside a method.
- Can be accessed only inside the method in which they are declared.
- They lose their values stored when the method execution completes.

- Variables declared inside a class and not inside any method.
- This variable can be accessed by all the methods of that class.
- This variable can also be accessed by other classes based on access modifiers.

You will learn more about variable scope and accessibility in the Access modifier session.

Variable Declaration - Example

Example:

```
public class Student {  
    int regId=1201;  
    void makeMarkList() {  
        int deptId=2011;  
        //code block  
    }  
}
```

} Instance Variable – Declared inside the body of the class.

} Local Variable – Declared inside the body of the method.

Naming Rules

- Variable names are **case-sensitive**.
- A variable's name **can be any legal identifier**.
- Length of Variable name can be any number.
- Its necessary to use Alphabet at the start (however we can use underscore , but do not use it)
- Some auto generated variables may contain '\$' sign. But try to avoid using Dollar Sign.
- **White space** is not permitted.
- **Special Characters** are not allowed.
- **Digit at start** is not allowed.
- Variable name must not be a **keyword or reserved word**.

Naming Rules

If the variable name has only one word, then spell that word in all lowercase letters. If it consists of more than one word, then capitalize the first letter of each subsequent word.

Example: `employeeName`, `employeeAccountNo`

- Avoid abbreviations, the variable name should be meaningful.

Don't: `empSal`, `empAdd`.

Do: `empSalary`, `empAddress`.

Java Literals

What is a literal?

- A literal is a value assigned to a variable in a class.

A literal is a value and not a variable

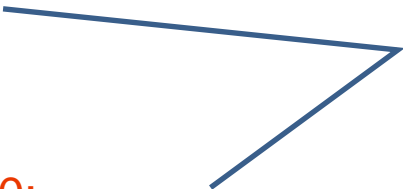
- Literals appears on the right hand side of the variable declaration.
- Literals value can be assigned for any of the primitive data type.

Example:

`int tax = 20;`

(or)

`float salary = 45000.0;`



**Where “20” and “45000.0”
are int and float literals
respectively.**

Java Literals Types

Literal Type	Description	Formats and Example
Integer Literals	Integral literals may be represented by decimal, octal, or hexadecimal numbers. The octal number is prefixed with 0, and hexadecimal with 0x or 0X.	Decimal :12 Hexadecimal:0xC Octal:014
Floating Literals	A floating-point literal is represented by a floating-point number, and is used for both float and double data types.	Float:1234.5f Double:12.65, 3.56E2, 23d
Boolean Literals	A variable of boolean type can have only one of two possible values, true or false. Therefore, these two values are the only available boolean literals	true, false

Java Literals Types

Literal Type	Description	Formats and Example
Char Literals	A char literal may be represented by a single character enclosed in single quotes.	'a', '\u4567'
String Literals	String literals represent multiple characters and are enclosed by double quotes.	"Hello World"
Escape Sequences	Java supports the escape sequences for denoting special characters.	\n :a new line \r :a return \t :a tab \b :a backspace \f :a form feed ' :a single quote " :a double quote \\ :a backslash

Example – Variables Declaration

1. Create a java class “**Employee**” with the following variables created,

Variable Name	Data Type	Default value
empld	long	1025
empSalary	double	45000
empTax	float	14.5
empDaysOfWork	int	24

2. Create a method named **calculatePF()** which will have a local float variable named “**pfRate**” with value as 10.5 and the following message should be printed.

“The PF Rate Of The Employee is <pfRate>”

3. Create a java class “**VariableDemo**” add a main method which will Create a object instance of the Employee. Access the instance variables and display the default values of variable as follows.

“The Id of the Employee is <empld> ”

“The Salary of The Employee is <empSal> ”

“The Percentage of Tax The Employee needs to Pay is <empTax>”

“The Total Number of Working Days is <daysOfWork>”

4. Invoke the calculatePF() method .

Variables Declaration - Solution

```
class Employee{
    long empId = 1025L;
    double empSalary = 45000d;
    float empTax = 14.5f;
    int empDaysOfWork = 24;
    void calculatePF() {
        float pfRate = 10.5f;
        System.out.println("The PF rate of the Employee is "+pfRate);
    }
}

public class VariableDemo {
    public static void main(String[] args) {
        Employee e = new Employee();
        System.out.println("Id of the Employee is "+e.empId);
        System.out.println("The Salary of the Employee is "+e.empSalary);
        System.out.println("The Percentage of Tax the Employee needs to pay is "+e.empTax);
        System.out.println("The Total number of Working Days is "+e.empDaysOfWork);
        e.calculatePF();
    }
}
```

Casting Primitives

What is casting?



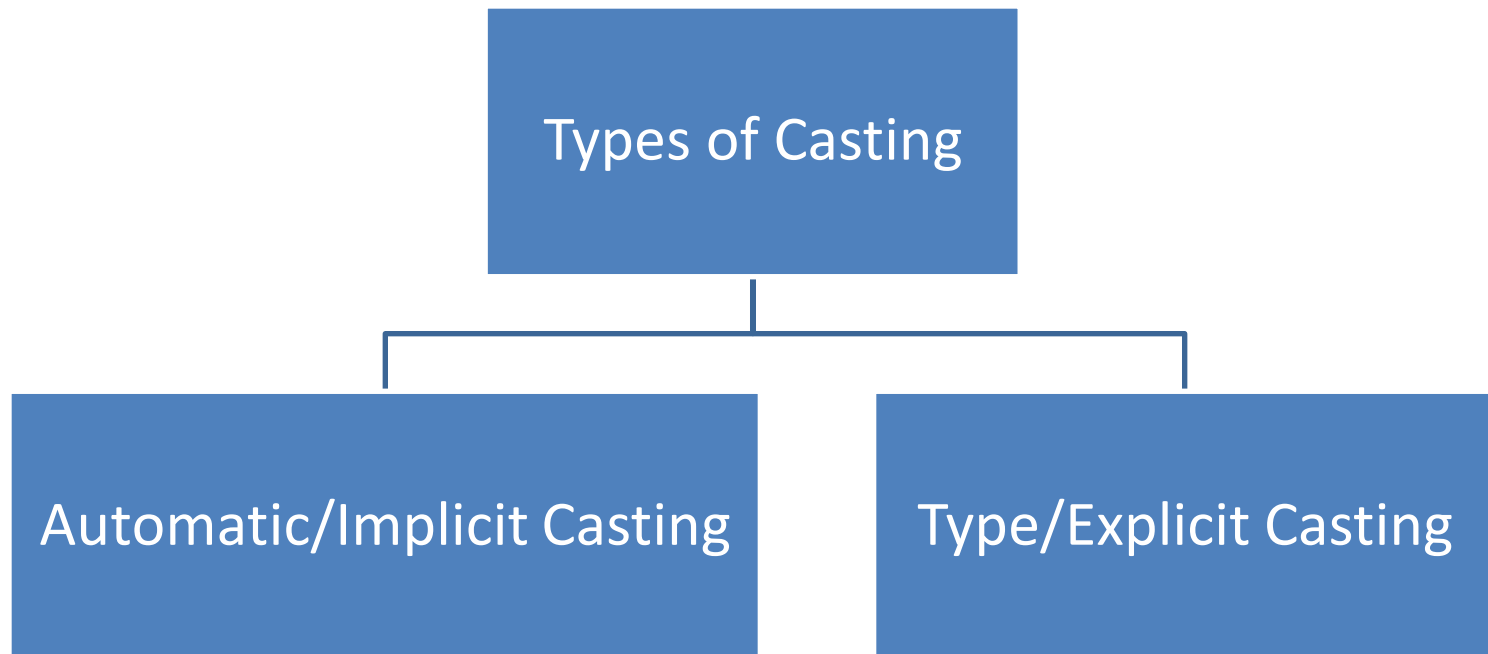
Casting is the process by which a liquid is poured into a mold and is allowed to solidify. In this process, it attains the shape of the mold

Similarly in Java, a value of one primitive type is assigned to a variable of another primitive type. This is called primitive type casting.

Example: A variable int is casted into float. Here float is the mold and int variable is casted as a float.

- **Casting is done with any of the primitive data types**
- **Boolean type variable cannot be casted**

Types of Primitive Casting



Implicit Casting

Java will performs automatic conversion when:

1. The two data types are compatible.
2. The destination data type is larger than the source data type.

Example: An int value (source) to a long variable (destination).

```
int mySal=500;  
long sal=mySal;
```

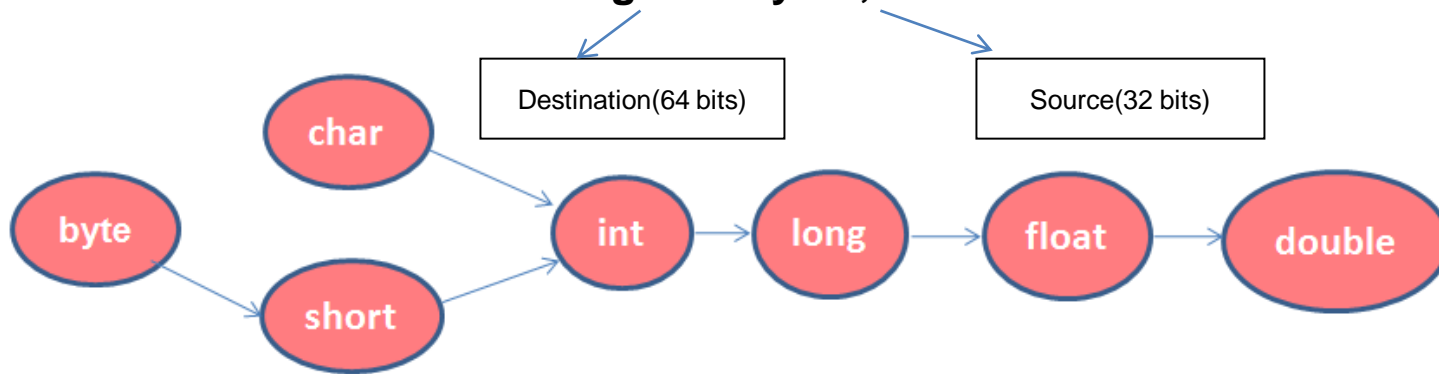


Figure depicts the allowed conversions for primitive data types (arrows indicate paths of allowed conversion).

Example: int can be type casted to long. Long can be type casted to float. etc.

Explicit Casting

Explicit Casting will be done by the programmers when the following condition is met.

The destination type is smaller than the source type.

Example:

```
double pf=12.67;  
float myPf=pf;           // compile-time error.  
float myPf=(float) pf;   // Explicit Casting.
```

A different type of conversion named “*truncation*” will occur when a floating-point value is assigned to an integer type.

Example: if the value 1.43 is assigned to an integer, the resulting value will be 1. The 0.43 will have been truncated.

Operators

What is an operator?

Operators are special symbols that perform specific operation on one, two or more operands and return a result.

Example:

$$a + b = c$$

Here,

- a and b are variables
- the data in a and b are operands
- '+' is the operator
- c contains the result

Navigate to the next slide for more explanation

Operators in Java

- Operators operate on the data represented by a *variable*.
- The data that is being operated on is called an operand.
- Operators operate on their operands in a variety of ways
- Operators change the values of their operands.
- Operators can produce a new value without changing the values of the operands.
- Operators can compare the values of two operands.

Types of Operators in Java

Types of data operators based on the number of operands,

- Unary operators: Require only *one* operand.

Example:

++ increments the value of its operand by 1.

-- decrement the value of its operand by 1.

- Binary operators: Require *two* operands.

Example:

+ adds the values of its two operands.

- Ternary operators: Operate on three operands(?:).

Unary Operators

Unary Arithmetic Operator:

The unary operators require only one operand. They perform various operations such as incrementing/decrementing a value by one, negating an expression, or inverting the value of a boolean.

Operator	Use	Description
-	<code>int a =-b;</code>	Negates the operand <code>b</code> and stores the value in <code>a</code>
++	<code>int b=a ++</code>	Increments the value of <code>a</code> by 1
--	<code>Int b=a --</code>	Decrements the value of <code>a</code> by 1

Unary Operators

Example of Increment & Decrement Operators:

Initial Value of x	Code Statement	Final Value of y	Final value of x
7	y=++x;	8	8
7	y=x++;	7	8
7	y=--x;	6	6
7	y=x--;	7	6

Ternary Operator

The `?` operator consists of three operands and is used to evaluate boolean expressions. The goal of the operator is to decide which value should be assigned to the variable.

Syntax:

variable x = (expression) ? value if true : value if false

Example:

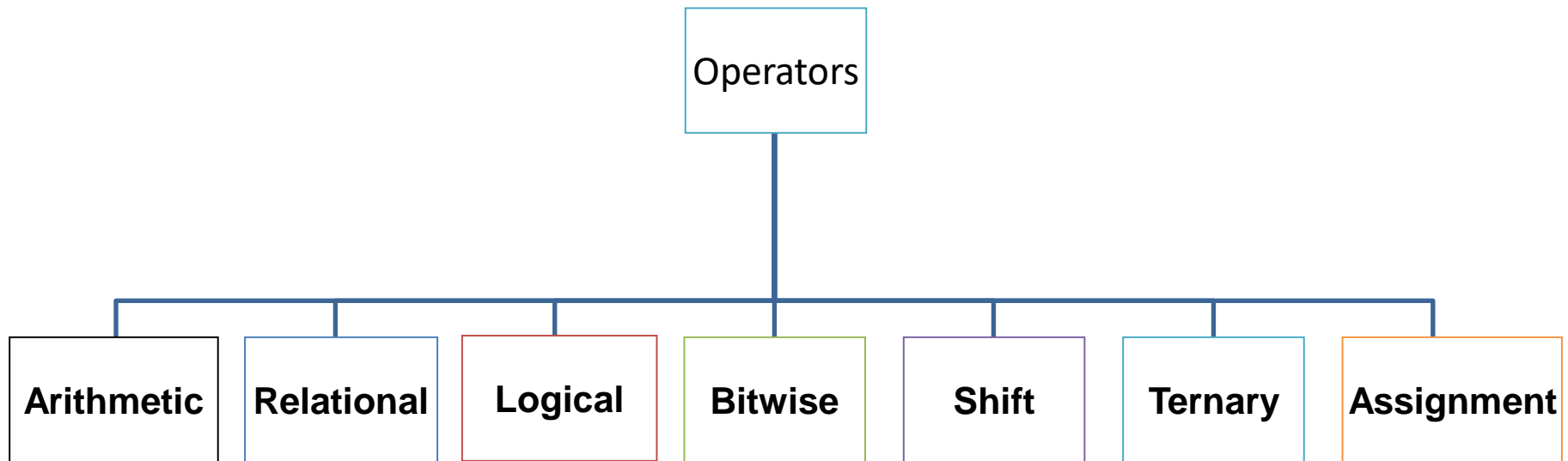
```
public class TernaryDemo {  
    public static void main(String[] args) {  
        int num1 = 35;  
        int num2 = (num1>15)?75:90;  
        System.out.println("The Value of num2 is "+num2);  
    }  
}
```

In the above example since the num1 is greater than 15 the value 75 will be assigned to num2.

If num1 is < 15 the value 90 will be assigned to num2.

Types of Operators

Types of Operators – Based on the Operation



You will learn more about operators in the next coming slides.

Arithmetic Operators

Arithmetic Operator:

Arithmetic operators are used for mathematical calculations. The following table lists the arithmetic operators

Operator	Value of a	Value of b	Usage	Value of c
+	6	5	$c=a+b$	11
-	6	5	$c=a-b$	1
*	6	5	$c=a* b$	30
/	6	5	$c=a/b$	1
%	6	5	$c=a\%b$	1

Example - Operators

1. Create a java class "**Arithmetic**" with three instance variable int num1, num2, result.
2. Create three methods "**addition**", "**subtraction**" and "**printSmaller**" which will add, subtract and print the smallest of two numbers respectively. The methods should display the results as follows,

Addition: "The Sum of the Two numbers <num1> and <num2> is <result>"

Subtraction: "The Subtracted Result of the two numbers <num1> and <num2> is <result>"

Print Smaller: "The Smallest of the Two numbers <num1> and <num2> is <result>"

3. Create a java class "**OperatorDemo**" add a main method which will
 1. Create a object instance of the Arithmetic .
 2. Set the value of num1 and num2 as 24 and 16.
 3. Invoke the methods "**addition**", "**subtraction**" and "**printSmaller**".
4. The message needs to be displayed in the console.

Operators - Solution

```
class Arithmetic{
    int num1;
    int num2;
    int result;
    public void addition() {
        result = num1+num2;
        System.out.println("The Sum of Two numbers "+num1+" and "+num2+" is "+result);
    }
    public void subtraction() {
        result = num1-num2;
        System.out.println("The Subtracted result of two numbers "+num1+" and "+num2+" is "+result);
    }
    public void printSmaller() {
        result = (num1<num2)?num1:num2;
        System.out.println("The Smallest of the Two numbers "+num1+" and "+num2+" is "+result);
    }
}

public class OperatorDemo {
    public static void main(String[] args) {
        Arithmetic arith = new Arithmetic();
        arith.num1 = 24;
        arith.num2 = 16;
        arith.addition();
        arith.subtraction();
        arith.printSmaller();
    }
}
```

Relational Operators

Relational Operator:

A Relational operator, also called a comparison operator, compares the values of two operands and returns a boolean value, true or false.

Operator	Use	Description
>	op1>op2	True if op1 is greater than op2, otherwise false
>=	op1>=op2	True if op1 is greater than or equal to op2, else false
<	op1 < op2	True if op1 is less than op2, otherwise false
<=	op1 <=op2	True if op1 is less than or equal to op2, otherwise false
==	op1 == op2	True if op1 and op2 are equal, otherwise false
!=	op != op2	True if op1 and op2 are not equal, otherwise false

Logical Operators

Logical Operator:

Logical operators are used to evaluate one or more conditions and return a true or false value based on conditions result.

Operator	Operator Type	Use	Description
&&	Binary	op1 &&op2	True if both the operands are true, otherwise false.
	Binary	op1 op2	True if any one of the operand is true , otherwise false.
!	Unary	! op1	True if operand is false and vice versa

Bitwise Operators

Bitwise Operator:

Bitwise operators that are used to manipulate bits of an integer (byte, short, char, int, long) value.

Operator	Type	Use	Description
&	Binary	op1 & op2	The AND operator & returns a value 1 bit if both operands are 1, a zero is produced in all other cases. It is equivalent to multiplying both the bits.
	Binary	op1 op2	The OR operator , returns a value 1 if one of the bits in the operands is a 1. Returns zero if both the operands has a value zero. It is equivalent to adding both the bits.
~	Unary	~ op1	The unary NOT operator, ~, inverts all of the bits of its operand., converts the 1 to zero and vice versa.
^	Binary	op1 ^ op2	The XOR operator, ^, It represents the inequality function, i.e., the output is HIGH (1) if the inputs are not alike otherwise the output is LOW (0).

Bitwise Operators

Example of Bitwise Operators:

Initial Value of x	Initial Value of y	Code Statement	Final Value of z
15(0000 1111)	5(0000 0101)	Z=x & y;	5 (0000 0101)
15(0000 1111)	5(0000 0101)	y=x y;	15 (0000 1111)
7(0000 1111)	-	z=~x;	-8 (~1111 0000)
15(0000 1111)	5(0000 0101)	z=x ^ y;	10 (0000 1010)

Example - Operators

1. Create a java class “**Bitwise**” add three integer instance variables num1, num2 and result.
2. Create four methods which will perform bitwise **AND**, **OR**, **XOR** and **unaryNOT** for the two numbers and print in following the format
 - “The Bitwise AND of the Two numbers <num1> and <num2> is <result>”
 - “The Bitwise OR Result of the two numbers <num1> and <num2> is <result>”
 - “The Bitwise XOR of the Two numbers <num1> and <num2> is <result>”
 - “The Unary NOT Result of the number <num1> is <result>”
3. Create a java class “**BitwiseOpDemo**” add a main method which will
 - Create a object instance of the **Bitwise** .
 1. Set the value of num1 and num2 as 15 and 5 .
 2. Invoke the four methods.
4. The message needs to be displayed in the console.

```

class Bitwise{
    int num1, num2, result;
    public void bitwiseAND() {
        result = num1&num2;
        System.out.println("The Bitwise AND of the Two numbers "+num1+" and "+num2+" is "+result);
    }
    public void bitwiseOR() {
        result = num1|num2;
        System.out.println("The Bitwise OR result of the two numbers "+num1+" and "+num2+" is "+result);
    }
    public void bitwiseXOR() {
        result = num1^num2;
        System.out.println("The Bitwise XOR of the Two numbers "+num1+" and "+num2+" is "+result);
    }
    public void unaryNOT() {
        result = ~num1;
        System.out.println("The Unary NOT result of the numbers "+num1+" is "+result);
    }
}

public class BitwiseOpDemo {
    public static void main(String[] args) {
        Bitwise obj = new Bitwise();
        obj.num1 = 15;
        obj.num2 = 5;
        obj.bitwiseAND();
        obj.bitwiseOR();
        obj.bitwiseXOR();
        obj.unaryNOT();
    }
}

```

Shift Operators

Shift Operator:

Shift operators operate on one or more bit patterns or binary numerals at the level of their individual bits.

Operator	Use	Description
<<	op1 <<op2	The left shift operator, <<, shifts all of the bits in a value to the left a specified number of times.
>>	op1>>op2	The right shift operator, >>, shifts all of the bits in a value to the right a specified number of times.
>>>	op1 >>> op2	Shift right zero fill operator. The left operands value is moved right by the number of bits specified by the right operand and shifted values are filled up with zeros.

Assignment Operators

Assignment Operators:

An assignment operator is used to set (or reset) the value of a variable

Operator	Use	Description
<code>+=</code>	<code>op1 +=op2</code>	Equivalent to <code>op1=op1+op2</code>
<code>-=</code>	<code>op1-=op2</code>	Equivalent to <code>op1=op1-op2</code>
<code>*=</code>	<code>op1*=op2</code>	Equivalent to <code>op1=op1 * op2</code>
<code>/=</code>	<code>op1 /=op2</code>	Equivalent to <code>op1=op1/op2</code>
<code>%=</code>	<code>op1%=op2</code>	Equivalent to <code>op1=op1%op2</code>

Assignment Operators

Operator	Use	Description
<code>&=</code>	<code>op1 &=op2</code>	Equivalent to <code>op1=op1& op2</code>
<code> =</code>	<code>Op1 =op2</code>	Equivalent to <code>op1=op1 op2</code>
<code>^=</code>	<code>op1 ^=op2</code>	Equivalent to <code>op1=op1 ^ op2</code>
<code><<=</code>	<code>op1 <<=op2</code>	Equivalent to <code>op1=op1<< op2</code>
<code>>>=</code>	<code>op1>>=op2</code>	Equivalent to <code>op1=op1>> op2</code>
<code>>>>=</code>	<code>op1 >>>=op2</code>	Equivalent to <code>op1=op1 >>> op2</code>

Operator Precedence

If in a expression there are more than one operators the order in which they are evaluated would be based on the operator precedence.

Precedence	Operator
1	(), []
2	++, --
3	*, /, %, +, -
4	<, <=, >, >=
5	==, !=
6	&&,

Operator Precedence Example

Let us analyze some examples to understand the precedence of operators.

Example 1:

```
result = (num1*num2) + num3 / num4;
```

Here is the order on how the above expression is processed

1. num1 * num2
2. num3 / num4
3. Final Result = result of step 1 + result of Step 2

Example 2:

```
result = num1>num2 && num3<=num4 || num5!=num6
```

Here is the order on how the above expression is processed

1. num3<=num4
2. num1>num2
3. num5!=num6
4. Result of Step1 && Result of Step2
5. Result of Step4 || Result of Step3

Thank you

You have successfully completed
**Language
Fundamentals &
Operators**