Chapter 4

Operators

Declaration

- These slides are made for UIT, BU students only. I am not holding any copy write of it as I had collected these study materials from different books and websites etc. I have not mentioned those to avoid complexity.

Topics

- Groups of operators
  - Arithmetic operators
  - Bitwize Operators
  - Relational Operators
  - Logical Operators
- Operator Precedence
- Special Operator

Operators

- An operations is an action performed on one or more values either to modify the value held by one or both of the values, or to produce a new value by combining existing values.
- Operators are special symbols that perform specific operations on one, two, or more operands.
- Operands can be constant or a variable.
- An expression is a combination of operands and operator that perform a specific operation.
- Therefore, an operation is performed using at least one symbol and at least one value.
  - The symbol used in an operation is called an operator.
  - A value involved in an operation is called an operand.

Forms Of Operators

- There are basically three forms of operators depending on the operands each takes. They are:
  - Unary Operators (One operand)
  - Binary Operators (Two operands)
  - Ternary Operators (Three operands).

Unary Operators

Operators	Description
+	Unary plus operator; indicates positive value (numbers are positive without this, however).
-	Unary minus operator; negates an expression.
++	Increment operator; increments a value by 1.
--	Decrement operator; decrements a value by 1.
!	Logical complement operator; inverts the value of a boolean.

```
int x = +5; // Unary plus, indicating x is positive 5
```

- **Explanation:**
- The unary plus operator doesn't change the value of the operand; it simply indicates that the operand is positive.
- In practice, the unary plus is rarely used because numbers are positive by default unless explicitly marked as negative with the unary minus (-) operator.

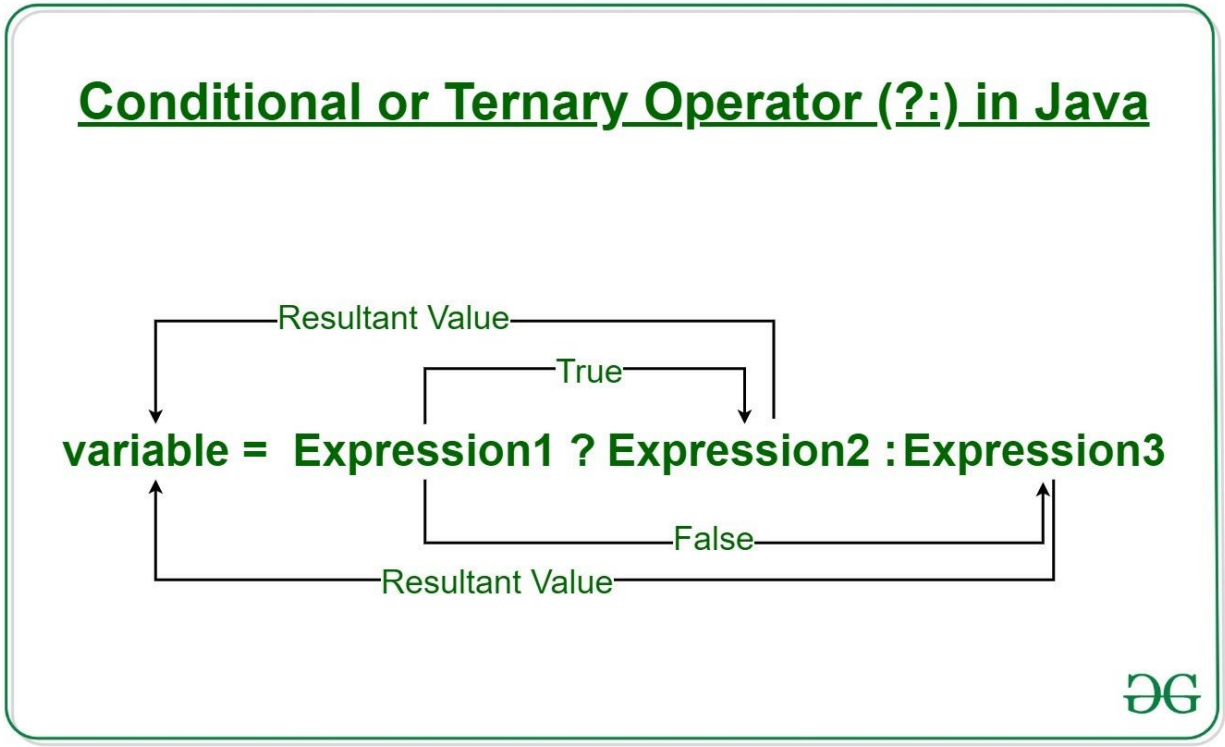
Binary operators

Operator	Meaning	Type	Example
+	Addition	Binary	`total = cost + tax;`
-	Subtraction	Binary	`cost = total - tax;`
*	Multiplication	Binary	`tax = cost * rate;`
/	Division	Binary	`salePrice = original / 2;`
%	Modulus	Binary	`remainder = value % 5;`

2-59

Ternary Operators

- The ternary operator "?:" is an operator which uses three operands. It is a conditional operator that provides a shorter syntax for the if..then..else statement. The first operand is a boolean expression; if the expression is true then the value of the second operand is returned otherwise the value of the third operand is returned:



Ternary Operators

```
public class TernaryOperatorExample
{
        public static void main(String[] args)
        {
                int number = 10;
                /* Using ternary operator to check if the number is
                   even or odd*/
                String result = (number % 2 == 0) ? "Even" : "Odd";
                // Print the result
                System.out.println("The number " + number + " is "
                + result);
        }
}
```

Ternary Operators

```
class Ternary {
  public static void main(String args[]) {
    int i, k;

    i = 10;
    k = i < 0 ? -i : i; // get absolute value of i
    System.out.print("Absolute value of ");
    System.out.println(i + " is " + k);

    i = -10;
    k = i < 0 ? -i : i; // get absolute value of i
    System.out.print("Absolute value of ");
    System.out.println(i + " is " + k);
  }
}
```

Types of Operators

- Java provides a rich set of operators to manipulate variables. We can divide java operators into the following groups :
  - Arithmetic Operators
  - Relational Operators
  - Logical Operators
  - Increment and Decrement Operators
  - Shorthand Operators
  - Bitwize operators

Arithmetic Operators

Operators	Result
+	Addition of two numbers
-	Subtraction of two numbers
*	Multiplication of two numbers
/	Division of two numbers
%	(Modulus Operator)Divides two numbers and returns the remainder
++	Increment Operator
--	Decrement Operator

Arithmetic operators

```
// Demonstrate the % operator.
class Modulus {
  public static void main(String args[]) {
    int x = 42;
    double y = 42.25;

    System.out.println("x mod 10 = " + x % 10);
    System.out.println("y mod 10 = " + y % 10);
  }
}
```

Arithmetic operators

Output:

x mod 10 = 2

y mod 10 = 2.25

Arithmetic operators

```
// Demonstrate ++ and --.
class IncDec {
  public static void main(String args[]) {
    int a = 1;
    int b = 2;
    int c;
    int d;
    c = ++b;
    d = a++;
    c++;
    System.out.println("a = " + a);
    System.out.println("b = " + b);
    System.out.println("c = " + c);
    System.out.println("d = " + d);
  }
}
```

Arithmetic operators

Output:

a = 2

b = 3

c = 4

d = 1

Relational Operators

Operator	Name	Example expression	Meaning
==	Equal to	x == y	true if x equals y, otherwise false
!=	Not equal to	x != y	true if x is not equal to y, otherwise false
>	Greater than	x > y	true if x is greater than y, otherwise false
<	Less than	x < y	true if x is less than y, otherwise false
>=	Greater than or equal to	x >= y	true if x is greater than or equal to y, otherwise false
<=	Less than or equal to	x <= y	true if x is less than or equal to y, otherwise false

Logical Operators

1. Logical AND Operator (& and &&)

Operand1	Operand2	Returned Value
False	False	False
False	True	False
True	False	False
True	True	True

2. Logical OR Operator (| and ||)

Operand1	Operand2	Returned Value
False	False	False
False	True	True
True	False	True
True	True	True

3. Logical NOT Operator (!)

Operand	Returned Value
False	True
True	False

Increment and Decrement Operators

Operator	Name	Example expression	Meaning
++	Postfix increment	x++	add 1 to x and return the old value
++	Prefix increment	++x	add 1 to x and return the new value
--	Postfix decrement	x--	take 1 from x and return the old value
--	Prefix decrement	--x	take 1 from x and return the new value

Shorthand Operators

Symbol	Operator Usage	Meaning
+=	x += y;	x = (x + y);
-=	x -= y;	x = (x - y);
*=	x *= y;	x = (x * y);
/=	x /= y;	x = (x / y);
%=	x %= y;	x = (x % y);
&=	x &= y;	x = (x & y);
|=	x != y;	x = (x ! y);
^=	x ^= y;	x = (x ^ y);
<<=	x <<= y;	x = (x << y);
>>=	x >>= y;	x = (x >> y);
>>>=	x >>>= y;	x = (x >>> y);

Bitwize Operators

- These operators which can be applied to the integer types, long, int, short, char, and byte.

Operator	Result
~	Bitwise unary NOT
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
>>	Shift right
>>>	Shift right zero fill
<<	Shift left
&=	Bitwise AND assignment
I =	Bitwise OR assignment
^=	Bitwise exclusive OR assignment
>>=	Shift right assignment
>>>=	Shift right zero fill assignment
<<=	Shift left assignment

Bitwise Logical Operators

- First, **logical operators work on *boolean* expressions** and return *boolean* values (either *true* or *false),* whereas **bitwise operators work on binary digits** of integer values (*long, int, short, char,* and *byte*) and return an integer.
- The bitwise logical operators are AND(&), OR(|), XOR(^), and NOT(~).
- For Example

```
public void bit_logi_op() {
        int value1 = 6;   int value2 = 5;
        int result = value1 | value2;
        System.out.println(result);}
```

0110

0101

0111

Binary Representation

- The byte value for 42 in binary is 00101010.
- All the integer types (except char) are signed integers. they can represent negative values as well as positive ones.
- Java uses an encoding known as two's compliment.
- Negative numbers are represented by inverting 1's to 0's and vice versa) all of the bits in a value, then adding 1 to the result.
- -42 is represented by inverting all of the bits in 42, or 00101010, which 11010101
- Then adding 1, which results in 11010110, or -42.
- To decode a negative number first invert all of the bits, then add 1. -42, or 11010110 inverted yields or 41, so when you add 1 you get 42.

Binary Representation

- The reason Java (and most other computer languages) uses two's complement is easy to see when you consider the issue of zero crossing.
- Assuming a byte value, zero is represented by 00000000. In one's complement, simply inverting all of the bits creates 11111111 which creates negative zero.
- This problem is solved by using two's complement to represent negative then using two's complement, 1 is added to the complement, producing 100000000.
- This produces a 1 bit too far to the left to fit back into the byte value, in the desired behavior, where -0 is the same as 0, and 11111111 is the for -1.

Bitwize Operators

A	B	A|B	A&B	A^B	~A
0	0	0	0	0	1
1	0	1	0	1	0
0	1	1	0	1	1
1	1	1	1	0	0

The Left Shift

- The left shift operator, <<, shifts all of the bits in a value to the left a specified number of times. It has this general form:
- value << num
- Here, num specifies the number of positions to left-shift the value in value.

The Left Shift

```
class ByteShift {
  public static void main(String args[]) {
    byte a = 64, b;
    int i;
    i = a << 2;
    b = (byte) (a << 2);
    System.out.println("Original value of a: " + a);
    System.out.println("i and b: " + i + " " + b);
  }
}
```

Arithmetic operators

Output:

Original value of a: 64

i and b: 256 0

The Right Shift

- The right shift operator, >>, shifts all of the bits in a value to the right a specified number of times.
- 11111000 -8

  >>1

  11111100 -4

Hierarchy of Operations

Operator	Description	Level	Associativity

`[]`
`.`
`()`
`++`
`--`
	access array element
	access object member
	invoke a method
	post-increment
	post-decrement
		1	left to right

`++`
`--`
`+`
`-`
`!`
`~`
	pre-increment
	pre-decrement
	unary plus
	unary minus
	logical NOT
	bitwise NOT
		2	right to left

`()` `new`	cast object creation	3	right to left

`*`
`/`
`%`
	multiplicative	4	left to right

`+ -` `+`	additive string concatenation	5	left to right
`<< >>` `>>>`	shift	6	left to right

Hierarchy of Operations

`< <= > >= instanceof`	relational type comparison	7	left to right
`== !=`	equality	8	left to right
`&`	bitwise AND	9	left to right
`^`	bitwise XOR	10	left to right
`|`	bitwise OR	11	left to right
`&&`	conditional AND	12	left to right
`||`	conditional OR	13	left to right
`?:`	conditional	14	right to left
`= += -= *= /= %= &= ^= |= <<= >>= >>>=`	assignment	15	right to left

Expression

- An expression in Java is any valid combination of operators, constants, variables and method invocations, which are constructed according to the syntax of the language that evaluates to a single value.
- Mathematical Expression: when an expression uses mathematical operator it is known as mathematical expression. It can be of different types
  - Pure Expression: When similar types of operands are present in an expression, it is known as Pure Expression.

    e.g int a =10, b = 20
  - Mixed Expression: When different types of operands are present in an expression it is known as Mixed Expression.

    e.g. . float total_sal = sal + da;

Expression

- Complex Expression: When more than one operator appears in an expression, evaluation is performed according to the rules of the precedence of operators.

  eg. short result = 100 + 4* 9 /3;

- Logical Expression : The expression that results into true or false ate called logical or Boolean expressions. .

  eg. if (x>y)

  System.out.println(" x is greater than y); ,

New Operator

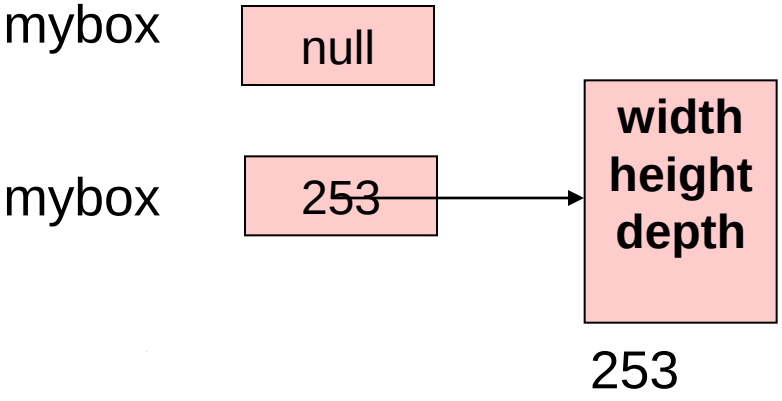
Box mybox = new Box();

It can be rewritten by

Box mybox;//declare reference to object

mybox = new Box(); //allocate a Box object

- Effect



Special Operator

- **instanceof** operator

person instanceof student

Is true if the object person belongs to the class student; otherwise it is false.

- **Dot** operator

The dot(.) operator is used to access the instance variables and methods to class objects.

person.age;

person.salary();

End of Chapter 4

Questions?