**JAVA COMMENTS**

The java comments are the statements in a program that are not executed by the compiler and interpreter.

**Why do we use comments in code?**

* Comments are used to make the program more readable by adding the details of the code.
* It makes easy to maintain the code and to find the errors easily.
* The comments can be used to provide information and explanation about the variable, method, class, or any statement.
* It can also be used to prevent the execution of program code while testing the alternative code.

**Types of Java Comments**

There are **3 types** of comments in java.

* Single Line comment
* Multi Line comment
* Documentation comment

**Single Line Comment:**

* The single line comment is used to comment only one line of the code.
* It is widely used and easiest way of commenting the statement.
* Single line comment starts with 2 forward slashes (//).
* Any text in front of // is not executed by java.

**Syntax**

//This is single line comment

**Example:**

public class Demo{

public static void main(String[] args){

int a = 25; //a is a variable with value 25

System.out.println(a) //printing the variable a

}

}

**Multi Line Comment:**

* The multi-line comment is used to comment multiple lines of code.
* It can be used to explain a complex code or to comment multiple lines at a time.
* Multi-Line comments are placed between /\* and \*/.

**Syntax**

/\*

This

Is

Multi-line

Comment

\*/

**Example:**

public class Demo{

public static void main(String[] args){

/\* let’s declare and

print the variable \*/

int a = 25;

System.out.println(a)

/\* float b = 65.2f;

float c = 45.6f;

System.out.println(b+c); \*/

}

}

**Java Documentation Comment**

* Documentation comments are usually used to write large programs for a project or software application as it helps to create documentation API.
* The documentation comments are placed between /\*\* and \*/.

**Syntax**

/\*\*

\*This contains data related to java

\*It is used to show information

\*/

**Example:**

public class Demo{

public static void main(String[] args){

int a = 25;

System.out.println(a)

/\*\*

\*The values are declared in float.

\*The final stage by calculation been based on addition

\*/

float b = 65.2f;

float c = 45.6f;

System.out.println(b+c);

}

}

**OPERATORS IN JAVA**

* Unary Operator
* Arithmetic Operator
* Shift Operator
* Relational Operator
* Bitwise Operator
* Logical Operator
* Ternary Operator
* Assignment Operator

**OPERATOR PRECEDENCE:**

|  |  |  |
| --- | --- | --- |
| **Operator Type** | **Category** | **Precedence** |
| Unary | Postfix | expr++ expr-- |
| Prefix | ++expr --expr +expr -expr ~[-(n+1)] ! |
| Arithmetic | multiplicative | \* / % |
| additive | + - |
| Shift | shift | << >> >>> |
| Relational | comparison | < > <= >= instanceof |
| equality | == != |
| Bitwise | bitwise AND | & |
| bitwise exclusive OR (XOR) | ^ |
| bitwise inclusive OR | | |
| bitwise NOT | ~ |
| Logical | logical AND | && |
| logical OR | || |
| Ternary | ternary | ? : |
| Assignment | assignment | = += -= \*= /= %= &= ^= |= <<= >>= >>>= |

**UNARY OPERATOR**

* Incrementing / decrementing value by one
* Negating the expression
* Inverting the value of boolean

**Example: ++, --**

public class OperatorExample1{

public static void main(String[] args){

int a = 5;

System.out.println(a++); //5(6)

System.out.println(++a); //6

System.out.println(a--); //6(5)

System.out.println(--a); //4

}

}

**Example: ~, !, +expr, -expr**

public class OperatorExample2{

public static void main(String[] args){

int a = 5;

int b = -5;

boolean c = true;

boolean d = false;

int e = +4; //unary plus operator indicates positive value

/\*unary minus operator indicates negative value\*/

int f = -4;

System.out.println(~a); //-6

System.out.println(~b); //4

System.out.println(!c); //false

System.out.println(!d); //true

System.out.println(e);

System.out.println(f);

}

}

**ARITHMETIC OPERATOR**

Arithmetic operators are used to perform addition, subtraction, multiplication, division, modulus.

**Example: +, -, \*, /, %**

public class ArithmeticOperator{

public static void main(String[] args){

int a = 20;

int b = 10;

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);

}

}

**LOGICAL (&&) and BITWISE (&) AND:**

* The logical && operator doesn’t check the second condition if the first condition is false.
* It checks the second condition only if the first one is true.
* Bitwise & operator always checks both conditions whether the first condition is true or false.

**Example:**

public class OperatorExample3{

public static void main(String[] args){

int a = 15;

int b = 8;

int c = 25;

System.out.println(a<b&&a<c); //false && true = false

System.out.println(a<b&a<c); //false && true = false

}

}

|  |  |  |
| --- | --- | --- |
| **Operand 1** | **Operand 2** | **Operand1 & Operand 2** |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

**The Bitwise AND Operation Results**

**Sample Example: &🡪multiplication**

8 & 5

8 4 2 1

8=> 1 0 0 0

5=> 0 1 0 1

---------

0=> 0 0 0 0

---------

**LOGICAL (||) and BITWISE (|) OR:**

* The logical || operator doesn’t check the second condition if the first condition is true.
* It checks the second condition only if the first one is false.
* Bitwise | operator always checks both conditions whether the first condition is true or false.

**Example:**

public class OperatorExample4{

public static void main(String[] args){

int a = 15;

int b = 8;

int c = 25;

System.out.println(a<b||a<c); //false || true = true

System.out.println(a<b|a<c); //false | true = true

}

}

|  |  |  |
| --- | --- | --- |
| **Operand 1** | **Operand 2** | **Operand1 | Operand 2** |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

**The Bitwise OR -inclusive Operation Results**

**Sample Example: |🡪addition**

8 | 5

8 4 2 1

8=> 1 0 0 0

5=> 0 1 0 1

---------

13=> 1 1 0 1

---------

**BITWISE (~) NOT:**

It negates the value

|  |  |
| --- | --- |
| **Operand 1** | **~Operand 2** |
| 0 | 1 |
| 1 | 0 |

**BITWISE (^) XOR:**

* The bitwise XOR(^) operator performs the XOR operation on two operands.
* The XOR operator applied on two bits results in 1, if exactly one bit is 1, else 0 in all other cases.

|  |  |  |
| --- | --- | --- |
| **Operand 1** | **Operand 2** | **Operand1 ^ Operand 2** |
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

**The Bitwise XOR -Exclusive Operation Results**

**TERNARY OPERATOR**

* Ternary operator is used as one line replacement for if-then-else statement and used a lot in java programming.
* It is the only conditional operator which takes 3 operands.

**Syntax:**

boolean\_expression ? expression 1 : expression 2

**Example: ? :**

public class OperatorExample5{

public static void main(String[] args){

int a = 15;

int b = 8;

int min = (a>b)?a:b;

System.out.println(min);

}

}

**ASSIGNMENT OPERATOR**

Assignment operator is one of the most common operators.

It is used to assign value on its right to the operand on its left.

**Example: =, +=, -=, \*=, /=, %=**

public class ArithmeticExample{

public static void main(String[] args){

int a = 5;

int b = 5;

int c = 8;

int d = 6;

int e = 8;

a += 2;

b -= 3;

c \*= 4;

d /= 2;

e % = 2;

System.out.println(a);

System.out.println(b);

System.out.println(c);

System.out.println(d);

System.out.println(e);

}

}

**SHIFT OPERATOR**

A shift operator is used to shift the bits of its operand either to the left or to the right.

The various types of shift operators are:

* Right shift operator (>>)
* Left shift operator (<<)
* Unsigned right shift operator (>>>)

**Right shift operator (>>)**

The right shift operator shifts all the bits of a binary number in the right direction.

**Syntax:**

operand >> num

**Example:**

5>>2

5🡺0101

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |

1st time

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

2nd time

**Left shift operator (<<)**

The left shift operator shifts all the bits of a binary number in the left direction.

**Syntax:**

operand << num

5<<2

5🡺0101

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |

1st time

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |

2nd time