Index

Operators

#. Operator in Java is a symbol that is used to perform operations. For example: +, -, *, / etc.

#. There are many types of operators in java which are given below:

- Unary Operator : expr++, expr --, ++expr, --expr etc
- Arithmetic Operator : +, -, *, /, % etc
- Relational Operator : ==, !=, <, >, <=, >= etc
- Bitwise Operator : &, ^, | etc
- Logical Operator : &&, || etc
- Assignment Operator : =, +=, -=, *=, /=, %= etc

Post and Pre Increment

Post Increment:-

```
int i = 1;

        // Print the value of i and then increment it
        System.out.println(i++); // 1, 2
        System.out.println(i); // 2

int x = 2;
int y = x++;
      // Print the value of x and then increment it
        System.out.println("y is " + y); // 2
        System.out.println("x is " + x); // 3
```

Pre Increment:-

```
int z = 9;

      // Increment the value of z and then print it
        System.out.println(++z); // 10

int u = 8;
int p = ++u;
      // Increment the value of u and then print it
        System.out.println("p is " + p); // 9
        System.out.println("u is " + u); // 9
```

Unary Operator

The Java unary operators require only one operand. Unary operators are used to perform various operations i.e.:

- Incrementing a value by one
- Decrementing a value by one

Example :-

```
public class UnaryOperator {
        public static void main(String[] args) {
    int a = 10;
    int b = 10;

            System.out.println(a++); // 10,11
            System.out.println(a++ + ++a); // 11,12,13
            System.out.println(b++ + ++b); // 10,11,12
            System.out.println(b++ + b++); // 12,13
            System.out.println(b);
     }
}
```

Output :- 10
24
22
25
14

Arithmetic Operator

Java arithmetic operators are used to perform addition, subtraction, multiplication, and division.

Example:-

```
public class ArithmeticOperator {
    public static void main(String[] args) {
        int a = 10;
        int b = 5;
        System.out.println(a + b); // Addition // 15
        System.out.println(a - b); // Subtraction // 5
        System.out.println(a * b); // Multiplication // 50
        System.out.println(a / b); // Division // 2
    }

}
```

Logical (&&) and Bitwise (&) Operator

#. The logical && operator doesn't check second condition if first condition is false. It checks second condition only if first one is true.

#. The bitwise & operator always checks both conditions whether first condition is true or false.

Example:-

```
public class LogicalOperator {

	public static void main(String[] args) {
		int a = 10;
		int b = 5;
		int c = 20;

		System.out.println(a < b && a++ < c); // false && true = false
		System.out.println(a); // 10 because second condition is not checked
		System.out.println(a < b & a++ < c); // false && true = false
		System.out.println(a); // 11 because second condition is checked
	}

}
```

Assignment Operator

Java assignment operator is one of the most common operator. It is used to assign the value on its right to the operand on its left.

Example:-

```
public class AssignmentOperator2 {

	public static void main(String[] args) {
		int a = 10;
		a += 3; // Do not give space between + and =
		System.out.println(a); // 13
		a -= 4; // Do not give space between - and =
		System.out.println(a); // 9
		a *= 2; // Do not give space between * and =
		System.out.println(a); // 18
		a /= 2; // Do not give space between / and =
		System.out.println(a); // 9

	}

}
```

Relational Operator

Java has six relational operators that compare two numbers and return a boolean value. The relational operators are <, >, <=, >=, ==, and !=.

x < y	*Less than*	True if x is less than y, otherwise false.
x > y	*Greater than*	True if x is greater than y, otherwise false.
x <= y	*Less than or equal to*	True if x is less than or equal to y, otherwise false.
x >= y	*Greater than or equal to*	True if x is greater than or equal to y, otherwise false.
x == y	*Equal*	True if x equals y, otherwise false.
x != y	*Not Equal*	True if x is not equal to y, otherwise false.

Example:-

```
public class RelationalOperator {

	public static void main(String[] args) {
		int a = 200;
		int b = 300;

		if (a != b) {
			System.out.println("a is not equal to b");
		}

		String p = "Selenium Class";
		String q = "Selenium Class";

		// Never Compare String Like This
		if (p == q) {
			System.out.println("This is not the correct way of comparing string");
		}

		// Always use equals() or equalsIgnoreCase()
		if (p.equals(q)) {
			System.out.println("p and q are equal");
		} else {
			System.out.println("p and q are unequal");
		}
	}

}
```

Question: Difference between ==, .equals() and equalsIgnoreCase()
Answer: Will See at the time of interview preparation.

Switch Statement

The Java switch statement executes one statement from multiple conditions. It is like if-else-if ladder statement.

- There can be one or N number of case values for a switch expression.
- The case value must be of switch expression type only.
- The case values must be unique. In case of duplicate value, it gives compile time error.
- The switch statement works with byte, short, int, long, String and some wrapper types like Byte, Short, Int, and Long.
- Each case statement can have a break statement which is optional. If a break statement is not found, it executes the next case.
- The case value can have a default label which is optional.

Syntax:-

```
switch(expression){
case value1:
 //code to be executed;
 break;  //optional
case value2:
 //code to be executed;
 break;  //optional
default:
Code to be executed if all cases are not matched;
}
```

Example:-

```
public class SwitchCase {
        public static void main(String[] args) {
                Integer age = 18;
                switch (age) { // Switch Expression
                case (16): // Case Statement
                        System.out.println("You are under 18.");
                        break;
                case (18):
                        System.out.println("You are eligible for vote.");
                        break;
                case (65):
                        System.out.println("You are senior citizen.");
                        break;
                default: // Default Statement
                        System.out.println("Please give the valid age.");
                        break;
                } } } // Output: You are eligible for vote.
```

Java Switch Statement is fall-through

The Java switch statement is fall-through. It means it executes all statements after the first match if a break statement is not present.

Example:-

```
public class SwitchCase {

        public static void main(String[] args) {
                int number = 20;
                // switch expression with int value
                switch (number) {
                // switch cases without break statements
                case 10:
                        System.out.println("10 - Not Matched");
                case 20:
                        System.out.println("20 - Matched");
                case 30:
                        System.out.println("30 - Not Matched");
                default:
                        System.out.println("No Any Case Get Matched");
                } } }
```

Output:-

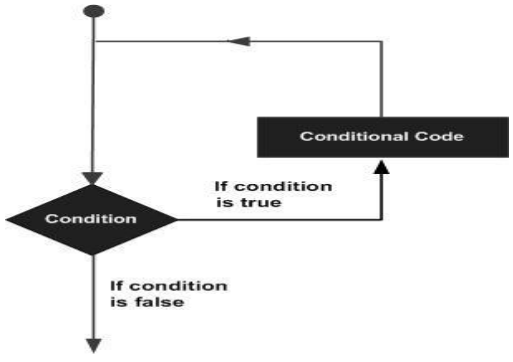
20 - Matched
30 - Not Matched
No Any Case Get Matched

Loops

When we need to execute a block of code several number of times, we use loop. Couple of popular loops are as below:

- for loop
- for-each loop
- while loop
- do-while loop

Architecture of loop:



For Loop - The Java for loop is used to iterate the part of program several times. If the number of iteration is fixed, it is recommended to use for loop.

Syntax:
for(initialization; condition ; incr / decr) {
//statement or code to be executed
}

Example:-

```
public class ForLoop {

	// For Loop
	public static void main(String[] args) {
		for (int i = 0; i <= 5; i++) { // initialization; condition ; increment
			System.out.println("Value of i is  : " + i);
		}
	}

}
```

Output:

```
Value of i is  : 0
Value of i is  : 1
Value of i is  : 2
Value of i is  : 3
Value of i is  : 4
Value of i is  : 5
```

For-each Loop - The for-each loop is used to traverse array or collection in java. It is easier to use than simple for loop because we don't need to increment value. It works on elements basis not index. It returns element one by one in the defined variable.

Syntax:
for(Type var:array){
//code to be executed
}

Example:-

```
public class ForEachLoop {

	public static void main(String[] args) {
		// Declaring An array
		int arr[] = { 1, 2, 3, 4, 5 };
```

```
		// Printing array using for-each loop
		for (int i : arr) {
			System.out.println(i);
		}
	}

}
```

Output:
1
2
3
4
5

While Loop - The Java while loop is used to iterate a part of the program several times. If the number of iteration is not fixed, it is recommended to use while loop.

Syntax:
while(condition){
//Code to be executed
}

Example:-

```
public class WhileLoop {

	public static void main(String[] args) {
		int i = 1;

		while (i <= 5) {
			System.out.println(i);
			i++; //if user forget to give this condition, program will run in infinite loop
		}
	}

}
```

Output:
1
2
3
4
5

Do While Loop - The Java do-while loop is used to iterate a part of the program several times. If the number of iteration is not fixed and you must have to execute the loop at least once, it is recommended to use do-while loop.

Syntax:
```
do{
//code to be executed
}while(condition);
```

Example:
```
public class DoWhileLoop {

	public static void main(String[] args) {
		int i = 1;

		do {
			System.out.println(i);
			i++;
		} while (i <= 5);
	}

}
```

Output:
```
1
2
3
4
5
```

Break Statement

The Java break statement is used to break loop or switch statement. It breaks the current flow of the program at specified condition. In case of inner loop, it breaks only inner loop. We can use Java break statement in all types of loops such as for loop, while loop and do-while loop.

Syntax:
```
break; // Jump Statement
```

Example:
```
public class BreakStatement1 {

	public static void main(String[] args) {
		for (int i = 1; i <= 10; i++) {
			if (i == 5) {
```

```
                    // breaking the loop
                    break;
                }
                System.out.println(i);
            }
        }

}
```

Output:

```
1
2
3
4
```

Break Statement with Inner Loop - It breaks inner loop only if you use break statement inside the inner loop.

Example:

```
public class BreakStatement2 {

    public static void main(String[] args) {
        // Outer Loop
        for (int i = 1; i <= 3; i++) {

            // Inner Loop
            for (int j = 1; j <= 3; j++) {
                if (i == 2 && j == 2) {
                    break;
                }
                System.out.println(i + "  " + j);
            }
        }
    }
}
```

Output:

```
1  1
1  2
1  3
2  1
3  1
3  2
3  3
```

Break Statement in while loop

Example:

```
public class BreakStatement3 {

	public static void main(String[] args) {
		int i = 1;
		// while loop
		while (i <= 10) {
			if (i == 5) {
				break; // using break statement, It will break the loop
			}
			System.out.println(i);
			i++;
		} } }
```

Output:

1
2
3
4

Break Statement in do-while loop

Example:

```
public class BreakStatement4 {

	public static void main(String[] args) {
		// declaring variable
		int i = 1;
		// do while loop
		do {
			if (i == 5) {
				break; // using break statement, it will break the loop
			}
			System.out.println(i);
			i++;
		} while (i <= 10);
	}
}
```

Output:

1
2
3
4

Java Continue Statement

The Java continue statement is used to continue the loop. It continues the current flow of the program and skips the remaining code at the specified condition. In case of an inner loop, it continues the inner loop only. We can use Java continue statement in all types of loops such as for loop, while loop and do-while loop.

Syntax:
Continue; // Jump Statement

Example:-

```
public class ContinueStatement1 {

	public static void main(String[] args) {
		// for loop
		for (int i = 1; i <= 7; i++) {
			if (i == 5) {
				continue; // it will skip the conditional statement '5'
			}
			System.out.println(i);
		}
	}

}
```

Output:

1
2
3
4
6 // You can see, 5 is not printed. It is because the loop is continued when it reaches to 5.
7

Java Continue Statement with Inner Loop - It continues inner loop only if we use the continue statement inside the inner loop.

Example:

```
public class ContinueStatement2 {

	public static void main(String[] args) {
		// Outer Loop
		for (int i = 1; i <= 3; i++) {
			// Inner Loop
			for (int j = 1; j <= 3; j++) {
				if (i == 2 && j == 2) {
```

```
                    continue; // using continue statement inside inner loop
                        }
                        System.out.println(i + "  " + j);
                    }

                }
            }
}
```

Output:

```
1  1
1  2
1  3
2  1
2  3 // You can see, 2 2 is not printed. It is because the loop is continued when it reaches to 2 2.
3  1
3  2
3  3
```

Java Comments - The comments can be used to provide information or explanation about the variable, method, class or any statement. It can also be used to hide program code for specific time.

Types of Java Comments:

- Single Line Comment
- Multi Line Comment

Single Line Comment - The single line comment is used to comment only one line.

Syntax:

```
// This is single line comment
```

Multi Line Comment - The multi line comment is used to comment multiple lines of code.

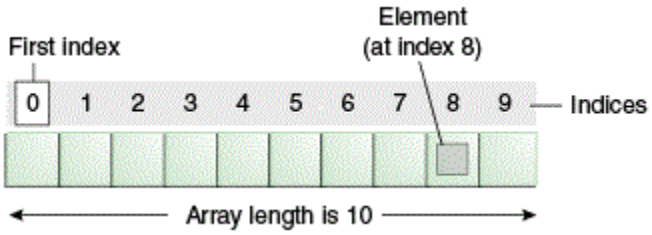
Syntax:

```
/*
This is
multiple line
comment
*/
```

Shortcut for Multi Line Comment:- Select the multiple lines of code and press " **Ctrl + Shift + ? "**

Java Array

1. Normally, an array is a collection of similar type of elements that have a contiguous memory location.
2. It can have element of similar data types (e.g., int array can not store character).
3. We can store only fixed set of elements in an array. It can not grow dynamically.
4. Array in java is index based, first element of the array is stored at 0 index.



Advantage of Java Array

- **Code Optimization:** It makes the code optimized, we can retrieve or short the data easily.
- **Random access:** We can get any data located at any index position

Disadvantage of Java Array

- **Size Limit:** We can store only fixed size of elements in the array. It does not grow its size at runtime. To solve this problem, collection framework (ArrayList) is used in java which grows automatically.

Types of Array in Java

- Single Dimensional Array
- Multi Dimensional Array

Single Dimensional Array

Syntax to Declare an Array in Java:

- dataType[] arr; (or)
- dataType []arr; (or)
- dataType arr[];

Instantiation of an Array in Java:

- arrayRefVar = **new** dataType[size];

Example:

```
public class SingleDimensionalArray {

    public static void main(String[] args) {
        int[] a = new int[5]; // Declaration & Instantiation
        a[0] = 5; // Initialization
```

```
                a[1] = 6;
                a[2] = 7;
                a[3] = 8;
                a[4] = 9;

                /*
                 We can also write in below fashion:-
                 int a[]={5,6,7,8,9};//declaration, instantiation and initialization
                 */

                // Printing length of array
                System.out.println("Length of array is " + a.length);

                for (int i = 0; i < a.length; i++) { // length is the property of array
                        System.out.println(a[i]);
                }
        }
}
```

Output:

```
Length of array is 5
5
6
7
8
9
```

Passing Array to Method in Java: We can pass the java array to method so that we can reuse the same logic on an array.

Example:

```
public class PassingArrayToMethod {
        public static void main(String[] args) {
                int a[] = { 33, 3, 4, 5 };// Declaration, Instantiation & Initialization of array
                min(a); // Passing array to method
        }
        static void min(int a[]) { // Creating a method which receives an array as parameter
                int b = a[0];
                for (int i = 1; i < a.length; i++) {
                        if (b > a[i])
                                System.out.println(b);
                }
           }
        } // Output: 33,33,33
```

Multi Dimensional Array in Java

In such case, data is stored in row and column based index (also known as matrix form).

Syntax to Declare Multidimensional Array in Java:

- dataType[][] arrayRefVar; (or)
- dataType [][]arrayRefVar; (or)
- dataType arrRefVar[][]; (or)
- dataType []arrayRefVar[];

Example to Instantiate Multi Dimensional Array in Java:

- **int**[][] arr = **new int**[3][3]; // 3 row and 3 column

Example:

```
public class TwoDimensionalArray {
        public static void main(String[] args) {
         /*int[][] arr = new int [3][3]; // Declaration, Instantiation//3 Row & 3 Column
           arr[0][0]=1; // Initialization
           arr[0][1]=2;
           arr[0][2]=3;
           arr[1][0]=4;
           arr[1][1]=5;       //Multi Line Comment
           arr[1][2]=6;
           arr[2][0]=7;
           arr[2][1]=8;
           arr[2][2]=9; */
                //we can write the same in below fashion
                int arr[][]={{1,2,3,10},{4,5,6,11},{7,8,9,12}}; // Preferable
                int row = arr.length;
                System.out.println("Total Number of Rows  " +row); // Printing Row Size
                int column = arr[0].length;
                System.out.println("Total Number of Column  " +column); //Printing Column Size
                //Printing 2D array
                for(int i=0;i<row;i++){
                        for(int j=0;j<column;j++){
                                System.out.print(arr[i][j]+ "  ");
                        }
                        System.out.println();
                } } }
```

Output:

```
Total Number of Rows  3
Total Number of Column  4
1  2  3  10
4  5  6  11
7  8  9  12
```

Important Note:

1. To print data of single dimensional array, use 1 for loop.
2. To print data of two dimensional array, use 2 for loops.
3. arrayName.length :- will give total number of rows.
4. arrayName[0].length :- will give total number of columns.

Method / Functions in Java

A java method is a collection of statements that are grouped together to perform an operation. When you call System.out.println() method, for example, the system actually executes several statements in order to display a message on the console.

Rules associated with methods:

1. All the functions has to be written inside body of a class.
2. One cannot write function inside function.
3. Execution of program always starts from main method / function.
4. One can call same function as many time as wanted.
5. One can pass any number of values inside functions.
6. Function can return only one value.
7. It is not mandatory to catch the value returned by function.
8. If user press "**Ctrl + Space bar**" inside a class, user can see all the functions / methods available inside class.

Example:

```
public class MethodExample {

	public static void main(String[] args) {
		int c = minFunction(15, 30); // function / method call
		System.out.println("Minimum Value = " + c);
	}

	// Function Definition
	public static int minFunction(int n1, int n2) { // int n1 & int n2 are arguments
		if (n1 > n2) {
			return n2; // return statement
		} else {
			return n1; // return statement
		}
	}
}
```

Output: Minimum Value = 15