Previous Year Questions

1. Describe the Anonymous array with examples.
2. Explain the jump statements with example.
3. Discuss the operators of Java in detail
4. Explain the implementation of interface.
5. What is Package? Explain with example.
6. Differentiate final class and super class.
7. Write a JAVA program to sort the element of one dimensional Array in descending order.
8. Describe the control statement using in JAVA.
9. Write a JAVA program to check that the given string is palindrome or not.
10. What are inheritance and explain variables in inheritance
11. What is Package ? Explain the procedure and importing package
12. What is abstract and final classes in Java ? explain
13. What is Inheritance? What are the basic types of inheritance?
14. Explain Method Overriding with Examples

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,
- Arithmetic Operator,
- Shift Operator,
- Relational Operator,
- Bitwise Operator,
- Logical Operator,
- Ternary Operator and
- Assignment Operator.

Java Operator Precedence

Operator Type	Category	Precedence
Unary	postfix	*expr*++ *expr*--
	prefix	++*expr* --*expr* +*expr* -*expr* ~ !
Arithmetic	multiplicative	* / %
	additive	+ -
Shift	shift	<< >> >>>
Relational	comparison	< > <= >= instanceof
	equality	== !=
Bitwise	bitwise AND	&
	bitwise exclusive OR	^

	bitwise inclusive OR	|
Logical	logical AND	&&
	logical OR	||
Ternary	ternary	? :
Assignment	assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=

Java Arithmetic Operators

These operators involve the mathematical operators that can be used to perform various simple or advanced arithmetic operations on the primitive data types referred to as the operands. These operators consist of various unary and binary operators that can be applied on a single or two operands. Let's look at the various operators that Java has to provide under the 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

Now let's look at each one of the arithmetic operators in Java:

1. Addition(+):

This operator is a binary operator and is used to add two operands.

Syntax:

num1 + num2

Example:

num1 = 10, num2 = 20

sum = num1 + num2 = 30

```
import java.io.*;

class Addition {
    public static void main(String[] args)
    {
        // initializing variables
        int num1 = 10, num2 = 20, sum = 0;

        // Displaying num1 and num2
        System.out.println("num1 = " + num1);
        System.out.println("num2 = " + num2);

        // adding num1 and num2
        sum = num1 + num2;
        System.out.println("The sum = " + sum);
    }
}
```

Output
num1 = 10
num2 = 20
The sum = 30

2. Subtraction(-): This operator is a binary operator and is used to subtract two operands.

Syntax:
num1 - num2

Example:
num1 = 20, num2 = 10
sub = num1 - num2 = 10

```
// Java code to illustrate Subtraction operator

import java.io.*;

class Subtraction {
    public static void main(String[] args)
    {
        // initializing variables
        int num1 = 20, num2 = 10, sub = 0;

        // Displaying num1 and num2
        System.out.println("num1 = " + num1);
        System.out.println("num2 = " + num2);
```

```
        // subtracting num1 and num2
        sub = num1 - num2;
        System.out.println("Subtraction = " + sub);
    }
}
```

Output
num1 = 20
num2 = 10
Subtraction = 10

3. Multiplication(*): This operator is a binary operator and is used to multiply two operands.

Syntax:
num1 * num2

Example:
num1 = 20, num2 = 10
mult = num1 * num2 = 200

```
// Java code to illustrate Multiplication operator

import java.io.*;

class Multiplication {
    public static void main(String[] args)
    {
        // initializing variables
        int num1 = 20, num2 = 10, mult = 0;

        // Displaying num1 and num2
        System.out.println("num1 = " + num1);
        System.out.println("num2 = " + num2);

        // Multiplying num1 and num2
        mult = num1 * num2;
        System.out.println("Multiplication = " + mult);
    }
}
```

Output
num1 = 20
num2 = 10
Multiplication = 200

4. Division(/): This is a binary operator that is used to divide the first operand(dividend) by the second operand(divisor) and give the quotient as a result.

Syntax:

num1 / num2

Example:

num1 = 20, num2 = 10

div = num1 / num2 = 2

```
// Java code to illustrate Division operator

import java.io.*;

class Division {
    public static void main(String[] args)
    {
        // initializing variables
        int num1 = 20, num2 = 10, div = 0;

        // Displaying num1 and num2
        System.out.println("num1 = " + num1);
        System.out.println("num2 = " + num2);

        // Dividing num1 and num2
        div = num1 / num2;
        System.out.println("Division = " + div);
    }
}
```

Output

num1 = 20

num2 = 10

Division = 2

5. Modulus(%): This is a binary operator that is used to return the remainder when the first operand(dividend) is divided by the second operand(divisor).

Syntax:

num1 % num2

Example:

num1 = 5, num2 = 2

mod = num1 % num2 = 1

```
import java.io.*;

class Modulus {
    public static void main(String[] args)
    {
        // initializing variables
        int num1 = 5, num2 = 2, mod = 0;

        // Displaying num1 and num2
        System.out.println("num1 = " + num1);
        System.out.println("num2 = " + num2);

        // Remaindering num1 and num2
        mod = num1 % num2;
        System.out.println("Remainder = " + mod);
    }
}
```

Output

num1 = 5

num2 = 2

Remainder = 1

Java Relational Operators

There are six types of relational operators in Java, these are:

Operator	Meaning
==	Is equal to
!=	Is not equal to
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to

These operators are mainly used when applying control statements in the program.

The output of the relational operator is (true/false) boolean value, and in Java, true or false is a non-numeric value that is not related to zero or one.

Program to Show Relational Operators Works

Example:

public class relation {

```
 public static void main(String[] args) {
  //Variables Definition and Initialization
  int num1 = 12, num2 = 4;

  //is equal to
  System.out.println("num1 == num2 = " + (num1 == num2) );

  //is not equal to
  System.out.println("num1 != num2 = " + (num1 != num2) );

  //Greater than
  System.out.println("num1 >  num2 = " + (num1 >  num2) );

  //Less than
  System.out.println("num1 <  num2 = " + (num1 <  num2) );

  //Greater than or equal to
  System.out.println("num1 >= num2 = " + (num1 >= num2) );

  //Less than or equal to
  System.out.println("num1 <= num2 = " + (num1 <= num2) );

 }
}
```

Output:
num1 == num2 = false
num1 != num2 = true
num1 > num2 = true
num1 < num2 = false
num1 >= num2 = true
num1 <= num2 = false

Java Logical Operators

You can also test for true or false values with logical operators.

Logical operators are used to determine the logic between variables or values:

Operator	Name	Description	Example
&&	Logical and	Returns true if both statements are true	x < 5 && x < 10
||	Logical or	Returns true if one of the statements is true	x < 5 || x < 4
!	Logical not	Reverse the result, returns false if the result is true	!(x < 5 && x < 10)

program –

Java

```
public class LogicalOperators {
    public static void main(String[] args) {
        boolean a = true;
        boolean b = false;

        System.out.println("a: " + a);
        System.out.println("b: " + b);
        System.out.println("a && b: " + (a && b));
        System.out.println("a || b: " + (a || b));
        System.out.println("!a: " + !a);
        System.out.println("!b: " + !b);
    }
}
```

Output

a: true

b: false

a && b: false

a || b: true

!a: false

!b: true

Bitwise Operators

Bitwise operators are used to performing the manipulation of individual bits of a number. They can be used with any integral type (char, short, int, etc.). They are used when performing update and query operations of the Binary indexed trees.

Now let's look at each one of the bitwise operators in Java:

1. Bitwise OR (|)

This operator is a binary operator, denoted by '|'. It returns bit by bit OR of input values, i.e., if either of the bits is 1, it gives 1, else it shows 0.

Example:

a = 5 = 0101 (In Binary)

b = 7 = 0111 (In Binary)

Bitwise OR Operation of 5 and 7

```
  0101
| 0111
  ____
  0111  = 7 (In decimal)
```

2. Bitwise AND (&)

This operator is a binary operator, denoted by '&.' It returns bit by bit AND of input values, i.e., if both bits are 1, it gives 1, else it shows 0.

Example:

a = 5 = 0101 (In Binary)

b = 7 = 0111 (In Binary)

Bitwise AND Operation of 5 and 7

```
  0101
& 0111
 ________
  0101  = 5 (In decimal)
```

3. Bitwise XOR (^)

This operator is a binary operator, denoted by '^.' It returns bit by bit XOR of input values, i.e., if corresponding bits are different, it gives 1, else it shows 0.

Example:

a = 5 = 0101 (In Binary)

b = 7 = 0111 (In Binary)

Bitwise XOR Operation of 5 and 7

```
  0101
^ 0111
 ________
  0010  = 2 (In decimal)
```

4. Bitwise Complement (~)

This operator is a unary operator, denoted by '~.' It returns the one's complement representation of the input value, i.e., with all bits inverted, which means it makes every 0 to 1, and every 1 to 0.

Example:

a = 5 = 0101 (In Binary)

Bitwise Complement Operation of 5

```
~ 0101
 ________
 1010  = 10 (In decimal)
```

```
public class operators {
   public static void main(String[] args)
   {
      // Initial values
      int a = 5;
      int b = 7;

      // bitwise and
      // 0101 & 0111=0101 = 5
      System.out.println("a&b = " + (a & b));

      // bitwise or
```

```
        // 0101 | 0111=0111 = 7
        System.out.println("a|b = " + (a | b));

        // bitwise xor
        // 0101 ^ 0111=0010 = 2
        System.out.println("a^b = " + (a ^ b));

    }
}
```

Output

a&b = 5

a|b = 7

a^b = 2

JAVA Conditional Operator

The Java Conditional Operator selects one of two expressions for evaluation, which is based on the value of the first operands. It is also called ternary operator because it takes three arguments.

The conditional operator is used to handling simple situations in a line.

Syntax:

expression1 ? expression2:expression3;

The above syntax means that if the value given in Expression1 is true, then Expression2 will be evaluated; otherwise, expression3 will be evaluated.

Example:

val == 0 ? you are right:you are not right;

P

Program to Show Conditional Operator Works

Example:

```
public class condiop
{
 public static void main(String[] args)
{
  String out;
  int  a = 6, b = 12;
  out = a==b ? "Yes":"No";
  System.out.println("Ans: "+out);
 }
}
```

Output:

Ans: No

In the above example, the condition given in expression1 is false because the value of a is not equal to the value of b.

The new operator in Java

The new operator is used in Java to create new objects. It can also be used to create an array object.

Let us first see the steps when creating an object from a class −

- **Declaration** − A variable declaration with a variable name with an object type.
- **Instantiation** − The 'new' keyword is used to create the object.
- **Initialization** − The 'new' keyword is followed by a call to a constructor. This call initializes the new object.

Now, let us see an example −

Example

```
public class Puppy
{
  public Puppy(String name)
 {
    // This constructor has one parameter, name.
    System.out.println("Passed Name is : " + name );
  }
  public static void main(String []args)
  {
    Puppy myPuppy = new Puppy( "jackie" );
  }
}
```

Output

Passed Name is : Jackie

Java instanceof Operator

The **java instanceof operator** is used to test whether the object is an instance of the specified type (class or subclass or interface).

The instanceof in java is also known as type *comparison operator* because it compares the instance with type. It returns either true or false. If we apply the instanceof operator with any variable that has null value, it returns false.

Simple example of java instanceof

Let's see the simple example of instance operator where it tests the current class.

```
1. class Simple1
2. {
3.  public static void main(String args[])
4. {
5.  Simple1 s=new Simple1();
6.  System.out.println(s instanceof Simple1);//true
```

7. }
8. }

Test it Now

Output:true

An object of subclass type is also a type of parent class. For example, if Dog extends Animal then object of Dog can be referred by either Dog or Animal class.

Java Control Statements | Control Flow in Java

Java compiler executes the code from top to bottom. The statements in the code are executed according to the order in which they appear. However, Java provides statements that can be used to control the flow of Java code. Such statements are called control flow statements. It is one of the fundamental features of Java, which provides a smooth flow of program.

Java provides three types of control flow statements.

1. Decision Making statements (Selection Statement)
    - if statements
    - switch statement
2. Loop statements (Iteration Statement)
    - do while loop
    - while loop
    - for loop
    - for-each loop
3. Jump statements
    - break statement
    - continue statement

Decision-Making statements: (Selection Statement)

As the name suggests, decision-making statements decide which statement to execute and when. Decision-making statements evaluate the Boolean expression and control the program flow depending upon the result of the condition provided. There are two types of decision-making statements in Java, i.e., If statement and switch statement.

1) If Statement:

In Java, the "if" statement is used to evaluate a condition. The control of the program is diverted depending upon the specific condition. The condition of the If statement gives a Boolean value, either true or false. In Java, there are four types of if-statements given below.

1. Simple if statement
2. if-else statement
3. if-else-if ladder
4. Nested if-statement

Let's understand the if-statements one by one.

1) Simple if statement:

It is the most basic statement among all control flow statements in Java. It evaluates a Boolean expression and enables the program to enter a block of code if the expression evaluates to true.

Syntax of if statement is given below.

```
1. if(condition)
2. {
3. statement 1; //executes when condition is true
4. }
```

Consider the following example in which we have used the **if** statement in the java code.

Student.java

Student.java

```
1. public class Student
2. {
3. public static void main(String[] args)
4. {
5. int x = 10;
6. int y = 12;
7. if(x+y > 20)
8. {
9. System.out.println("x + y is greater than 20");
10. }
11. }
12. }
```

Output:

x + y is greater than 20

2) if-else statement

The if-else statement is an extension to the if-statement, which uses another block of code, i.e., else block. The else block is executed if the condition of the if-block is evaluated as false.

Syntax:

```
1. if(condition)
2. {
3. statement 1; //executes when condition is true
4. }
5. Else
6. {
7. statement 2; //executes when condition is false
8. }
```

Consider the following example.

Student.java

```
1. public class Student
2. {
3. public static void main(String[] args)
4. {
5. int x = 10;
```

```
6. int y = 12;
7. if(x+y < 10)
8. {
9. System.out.println("x + y is less than      10");
10. }
11. else
12. {
13. System.out.println("x + y is greater than 20");
14. }
15. }
16. }
```

Output:

x + y is greater than 20

3) if-else-if ladder:

The if-else-if statement contains the if-statement followed by multiple else-if statements. In other words, we can say that it is the chain of if-else statements that create a decision tree where the program may enter in the block of code where the condition is true. We can also define an else statement at the end of the chain.

Syntax of if-else-if statement is given below.

```
1. if(condition 1)
2. {
3. statement 1; //executes when condition 1 is true
4. }
5. else if(condition 2)
6. {
7. statement 2; //executes when condition 2 is true
8. }
9. else
10. {
11. statement 2; //executes when all the conditions are false
12. }
```

Consider the following example.

Student.java

```
1. public class Student
2. {
3. public static void main(String[] args)
4. {
5. String city = "Delhi";
6. if(city == "Meerut")
7. {
8. System.out.println("city is meerut");
9. }
10. else if (city == "Noida")
11. {
12. System.out.println("city is noida");
```

```
13. }
14. else if(city == "Agra")
15. {
16. System.out.println("city is agra");
17. }
18. else
19. {
20. System.out.println(city);
21. }
22. }
23. }
```

Output:

Delhi

4. Nested if-statement

In nested if-statements, the if statement can contain a **if** or **if-else** statement inside another if or else-if statement.

Syntax of Nested if-statement is given below.

```
1. if(condition 1)
2. {
3. statement 1; //executes when condition 1 is true
4. if(condition 2)
5. {
6. statement 2; //executes when condition 2 is true
7. }
8. Else
9. {
10. statement 2; //executes when condition 2 is false
11. }
12. }
```

Consider the following example.

Student.java

```
1. public class Student
2. {
3. public static void main(String[] args)
4. {
5. String address = "Delhi, India";
6. if(address.endsWith("India"))
7. {
8. if(address.contains("Meerut"))
9. {
10. System.out.println("Your city is Meerut");
11. }
12. else if(address.contains("Noida"))
13. {
14. System.out.println("Your city is Noida");
```

```
15. }
16. else
17. {
18. System.out.println(address.split(",")[0]);
19. }
20. }
21. else
22. {
23. System.out.println("You are not living in India");
24. }
25. }
26. }
```

Output:

Delhi

Switch Statement:

In Java, Switch statements are similar to if-else-if statements. The switch statement contains multiple blocks of code called cases and a single case is executed based on the variable which is being switched. The switch statement is easier to use instead of if-else-if statements. It also enhances the readability of the program.

Points to be noted about switch statement:

- The case variables can be int, short, byte, char, or enumeration. String type is also supported since version 7 of Java
- Cases cannot be duplicate
- Default statement is executed when any of the case doesn't match the value of expression. It is optional.
- Break statement terminates the switch block when the condition is satisfied.
  It is optional, if not used, next case is executed.
- While using switch statements, we must notice that the case expression will be of the same type as the variable. However, it will also be a constant value.

The syntax to use the switch statement is given below.

```
1. switch (expression)
2. {
3.     case value1:
4.      statement1;
5.      break;
6.     .
7.     .
8.     .
9.     case valueN:
10.     statementN;
11.     break;
12.    default:
13.     default statement;
14. }
```

Consider the following example to understand the flow of the switch statement.

Student.java

```
1. public class Student implements Cloneable
```

```
2. {
3. public static void main(String[] args)
4. {
5. int num = 2;
6. switch (num)
7. {
8. case 0:
9. System.out.println("number is 0");
10. break;
11. case 1:
12. System.out.println("number is 1");
13. break;
14. default:
15. System.out.println(num);
16. }
17. }
18. }
```

Output:

2

While using switch statements, we must notice that the case expression will be of the same type as the variable. However, it will also be a constant value. The switch permits only int, string, and Enum type variables to be used.

Loop Statements (Iteration Statements)

In programming, sometimes we need to execute the block of code repeatedly while some condition evaluates to true. However, loop statements are used to execute the set of instructions in a repeated order. The execution of the set of instructions depends upon a particular condition.

In Java, we have three types of loops that execute similarly. However, there are differences in their syntax and condition checking time.

1. for loop
2. while loop
3. do-while loop

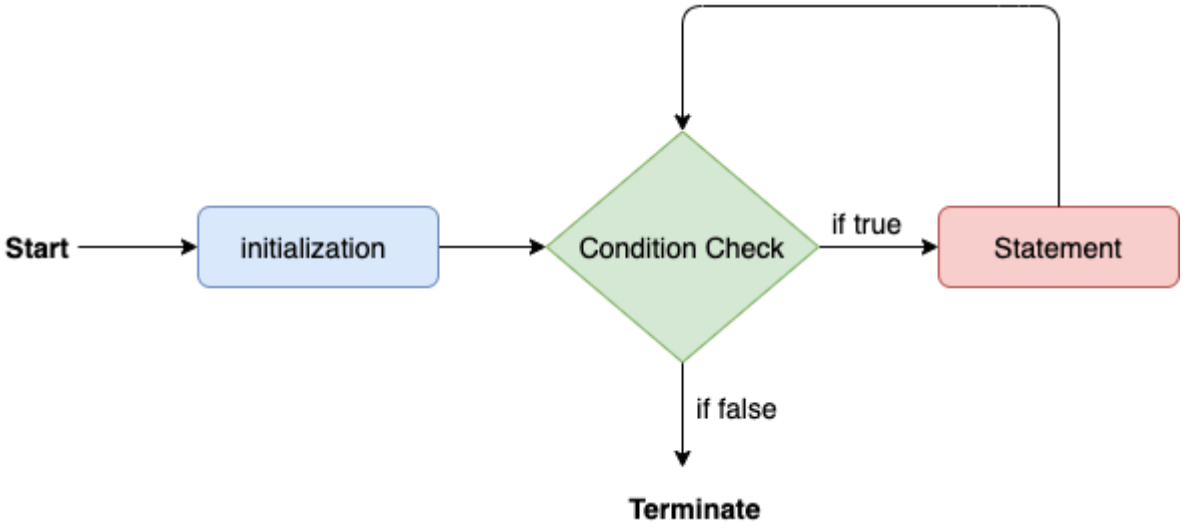
Let's understand the loop statements one by one.

Java for loop

In Java, for loop is similar to C and C++. It enables us to initialize the loop variable, check the condition, and increment/decrement in a single line of code. We use the for loop only when we exactly know the number of times, we want to execute the block of code.

```
1. for(initialization, condition, increment/decrement)
2. {
3. //block of statements
4. }
```

The flow chart for the for-loop is given below.



Consider the following example to understand the proper functioning of the for loop in java.

Calculation.java

```
1. public class Calculattion
2. {
3. public static void main(String[] args)
4. {
5. // TODO Auto-generated method stub
6. int sum = 0;
7. for(int j = 1; j<=10; j++)
8. {
9. sum = sum + j;
10. }
11. System.out.println("The sum of first 10 natural numbers is " + sum);
12. }
13. }
```

Output:

The sum of first 10 natural numbers is 55

Java for-each loop

Java provides an enhanced for loop to traverse the data structures like array or collection. In the for-each loop, we don't need to update the loop variable. The syntax to use the for-each loop in java is given below.

```
1. for(data_type var : array_name/collection_name)
2. {
3. //statements
4. }
```

Consider the following example to understand the functioning of the for-each loop in Java.

Calculation.java

```
1. public class Calculation
2. {
3. public static void main(String[] args)
4. {
5. // TODO Auto-generated method stub
```

```
6. String[] names = {"Java","C","C++","Python","JavaScript"};
7. System.out.println("Printing the content of the array names:\n");
8. for(String name:names)
9. {
10. System.out.println(name);
11. }
12. }
13. }
```

Output:

Printing the content of the array names:

Java

C

C++

Python

JavaScript

Java while loop

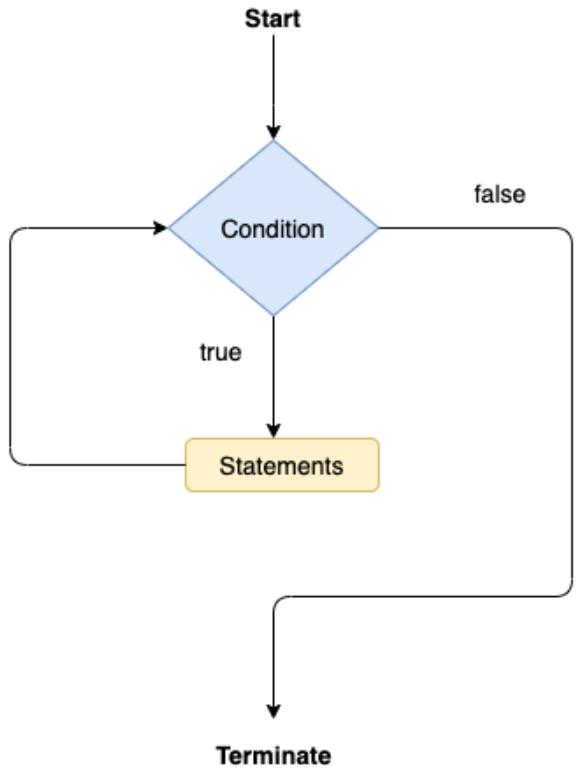
The while loop is also used to iterate over the number of statements multiple times. However, if we don't know the number of iterations in advance, it is recommended to use a while loop. Unlike for loop, the initialization and increment/decrement doesn't take place inside the loop statement in while loop.

It is also known as the entry-controlled loop since the condition is checked at the start of the loop. If the condition is true, then the loop body will be executed; otherwise, the statements after the loop will be executed.

The syntax of the while loop is given below.

```
1. while(condition)
2. {
3. //looping statements
4. }
```

The flow chart for the while loop is given in the following image.



Consider the following example.

Calculation .java

```
1. public class Calculation
2. {
3. public static void main(String[] args)
4. {
5. // TODO Auto-generated method stub
6. int i = 0;
7. System.out.println("Printing the list of first 10 even numbers \n");
8. while(i<=10)
9. {
10. System.out.println(i);
11. i = i + 2;
12. }
13. }
14. }
```

Output:

```
Printing the list of first 10 even numbers
0
2
4
6
8
10
```

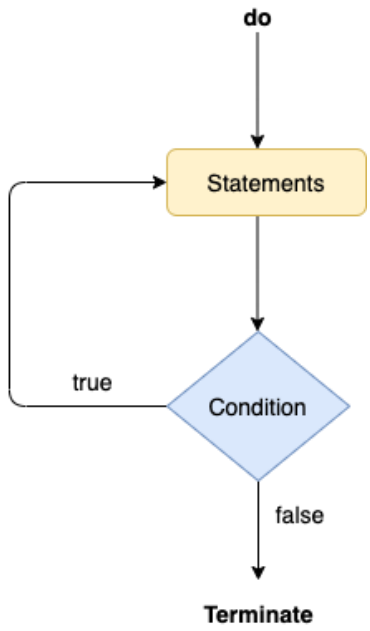
Java do-while loop

The do-while loop checks the condition at the end of the loop after executing the loop statements. When the number of iteration is not known and we have to execute the loop at least once, we can use do-while loop.

It is also known as the exit-controlled loop since the condition is not checked in advance. The syntax of the do-while loop is given below.

```
1. do
2. {
3. //statements
4. } while (condition);
```

The flow chart of the do-while loop is given in the following image.



Consider the following example to understand the functioning of the do-while loop in Java.

Calculation.java

```
1.  public class Calculation
2.  {
3.  public static void main(String[] args)
4.  {
5.  // TODO Auto-generated method stub
6.  int i = 0;
7.  System.out.println("Printing the list of first 10 even numbers \n");
8.  do
9.  {
10. System.out.println(i);
11. i = i + 2;
12. }
13. while(i<=10);
14. }
15. }
```

Output:

Printing the list of first 10 even numbers

0

2

4

6

8

10

Jump Statements

Jump statements are used to transfer the control of the program to the specific statements. In other words, jump statements transfer the execution control to the other part of the program. There are two types of jump statements in Java, i.e., break and continue.

Java break statement

As the name suggests, the break statement is used to break the current flow of the program and transfer the control to the next statement outside a loop or switch statement. However, it breaks only the inner loop in the case of the nested loop.

The break statement cannot be used independently in the Java program, i.e., it can only be written inside the loop or switch statement.

The break statement example with for loop

Consider the following example in which we have used the break statement with the for loop.

BreakExample.java

```
1. public class BreakExample
2. {
3. public static void main(String[] args)
4. {
5. // TODO Auto-generated method stub
6. for(int i = 0; i<= 10; i++)
7. {
8. System.out.println(i);
9. if(i==6)
10. {
11. break;
12. }
13. }
14. }
15. }
```

Output:

```
0
1
2
3
4
5
6
```

break statement example with labeled for loop

Calculation.java

```
1. public class Calculation
2. {
3. public static void main(String[] args)
4. {
5. // TODO Auto-generated method stub
6. a:
7. for(int i = 0; i<= 10; i++)
8. {
9. b:
```

```
10. for(int j = 0; j<=15;j++)
11. {
12. c:
13. for (int k = 0; k<=20; k++)
14. {
15. System.out.println(k);
16. if(k==5)
17. {
18. break a;
19. }
20. }
21. }
22. }
23. }
24. }
```

Output:

```
0
1
2
3
4
5
```

Java continue statement

Unlike break statement, the continue statement doesn't break the loop, whereas, it skips the specific part of the loop and jumps to the next iteration of the loop immediately.

Consider the following example to understand the functioning of the continue statement in Java.

```
1. public class ContinueExample
2. {
3. public static void main(String[] args)
4. {
5. // TODO Auto-generated method stub
6. for(int i = 0; i<= 2; i++)
7. {
8. for (int j = i; j<=5; j++)
9. {
10. if(j == 4)
11. {
12. continue;
13. }
14. System.out.println(j);
15. }
16. }
17. }
18. }
```

Output:

0
1
2
3
5
1
2
3
5
2
3
5

Java Arrays

Normally, an array is a collection of similar type of elements which has contiguous memory location.

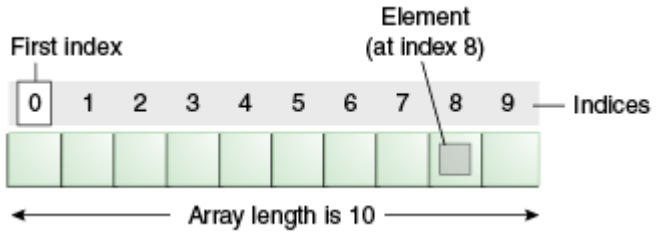
Java array is an object which contains elements of a similar data type. Additionally, The elements of an array are stored in a contiguous memory location. It is a data structure where we store similar elements. We can store only a fixed set of elements in a Java array.

Array in Java is index-based, the first element of the array is stored at the 0th index, 2nd element is stored on 1st index and so on.

Unlike C/C++, we can get the length of the array using the length member. In C/C++, we need to use the sizeof operator.

In Java, array is an object of a dynamically generated class. Java array inherits the Object class, and implements the Serializable as well as Cloneable interfaces. We can store primitive values or objects in an array in Java. Like C/C++, we can also create single dimentional or multidimentional arrays in Java.

Moreover, Java provides the feature of anonymous arrays which is not available in C/C++.



Advantages

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

Disadvantages

- **Size Limit:** We can store only the fixed size of elements in the array. It doesn't grow its size at runtime. To solve this problem, collection framework is used in Java which grows automatically.

Types of Array in java

There are two types of array.

- Single Dimensional Array
- Multidimensional Array

Single Dimensional Array in Java

Syntax to Declare an Array in Java

1. dataType[] arr; (or)
2. dataType []arr; (or)
3. dataType arr[];

Instantiation of an Array in Java

1. arrayRefVar=**new** datatype[size];

Example of Java Array

Let's see the simple example of java array, where we are going to declare, instantiate, initialize and traverse an array.

```
1. //Java Program to illustrate how to declare, instantiate, initialize
2. //and traverse the Java array.
3. class Testarray{
4. public static void main(String args[]){
5. int a[]=new int[5];//declaration and instantiation
6. a[0]=10;//initialization
7. a[1]=20;
8. a[2]=70;
9. a[3]=40;
10. a[4]=50;
11. //traversing array
12. for(int i=0;i<a.length;i++)//length is the property of array
13. System.out.println(a[i]);
14. }}
```

Output:

```
10
20
70
40
50
```

Declaration, Instantiation and Initialization of Java Array

We can declare, instantiate and initialize the java array together by:

```
1. int a[]={33,3,4,5};//declaration, instantiation and initialization
```

Let's see the simple example to print this array.

```
1. //Java Program to illustrate the use of declaration, instantiation
2. //and initialization of Java array in a single line
```

```
3. class Testarray1
4. {
5. public static void main(String args[])
6. {
7. int a[]={33,3,4,5};//declaration, instantiation and initialization
8. //printing array
9. for(int i=0;i<a.length;i++)//length is the property of array
10. System.out.println(a[i]);
11. }
12. }
```

Output:

33

3

4

5

Anonymous Array in Java

Java supports the feature of an anonymous array, so you don't need to declare the array while passing an array to the method.

```
1. //Java Program to demonstrate the way of passing an anonymous array
2. //to method.
3. public class TestAnonymousArray
4. {
5. //creating a method which receives an array as a parameter
6. static void printArray(int arr[])
7. {
8. for(int i=0;i<arr.length;i++)
9. System.out.println(arr[i]);
10. }
11. public static void main(String args[])
12. {
13. printArray(new int[]{10,22,44,66});//passing anonymous array to method
14. }
15. }
```

Output:

10

22

44

66

Multidimensional 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

```
1. dataType[][] arrayRefVar; (or)
```

2. dataType [][]arrayRefVar; (or)
3. dataType arrayRefVar[][]; (or)
4. dataType []arrayRefVar[];

Example to instantiate Multidimensional Array in Java

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

Example to initialize Multidimensional Array in Java

1. arr[0][0]=1;
2. arr[0][1]=2;
3. arr[0][2]=3;
4. arr[1][0]=4;
5. arr[1][1]=5;
6. arr[1][2]=6;
7. arr[2][0]=7;
8. arr[2][1]=8;
9. arr[2][2]=9;

Example of Multidimensional Java Array

Let's see the simple example to declare, instantiate, initialize and print the 2Dimensional array.

```
1. //Java Program to illustrate the use of multidimensional array
2. class Testarray3
3. {
4. public static void main(String args[])
5. {
6. //declaring and initializing 2D array
7. int arr[][]={{1,2,3},{2,4,5},{4,4,5}};
8. //printing 2D array
9. for(int i=0;i<3;i++)
10. {
11.  for(int j=0;j<3;j++)
12. {
13.    System.out.print(arr[i][j]+" ");
14.  }
15.  System.out.println();
16. }
17. }
18. }
```

Output:

```
1 2 3
2 4 5
4 4 5
```