

Java Tutorial

JAVA TUTORIAL	1
JAVA	2
APPLICATION.....	5
TYPES OF JAVA APPLICATIONS.....	5
JAVA PLATFORMS / EDITIONS	6
FEATURES OF JAVA	6
• A LIST OF THE MOST IMPORTANT FEATURES OF THE JAVA LANGUAGE IS GIVEN BELOW	6
C++ VS JAVA.....	7
C++ PROGRAM EXAMPLE	9
JAVA PROGRAM EXAMPLE.....	10
JAVA VARIABLES	10
VARIABLE.....	11
<i>Types of Variables</i>	<i>11</i>
<i>Example to understand the types of variables in java</i>	<i>12</i>
<i>Java Variable Example: Add Two Numbers.....</i>	<i>13</i>
<i>Java Variable Example: Widening.....</i>	<i>13</i>
<i>Java Variable Example: Narrowing (Typecasting).....</i>	<i>13</i>
<i>Java Variable Example: Overflow.....</i>	<i>14</i>
<i>Java Variable Example: Adding Lower Type.....</i>	<i>14</i>
DATA TYPES IN JAVA.....	15
JAVA PRIMITIVE DATA TYPES	15
BOOLEAN DATA TYPE.....	16
BYTE DATA TYPE	16
SHORT DATA TYPE.....	17
INT DATA TYPE	17
LONG DATA TYPE	17
FLOAT DATA TYPE	17
DOUBLE DATA TYPE.....	18
CHAR DATA TYPE	18
<i>Why char uses 2 byte in java and what is \u0000 ?.....</i>	<i>18</i>
OPERATORS IN JAVA.....	19
JAVA OPERATOR PRECEDENCE	19
<i>Java Unary Operator</i>	<i>20</i>
<i>Java Unary Operator Example: ++ and --</i>	<i>20</i>
<i>Java Unary Operator Example 2: ++ and --</i>	<i>21</i>
<i>Java Unary Operator Example: ~ and !</i>	<i>21</i>
<i>Java Arithmetic Operators</i>	<i>21</i>
<i>Java Arithmetic Operator Example</i>	<i>22</i>
<i>Java Arithmetic Operator Example: Expression</i>	<i>22</i>
<i>Java Left Shift Operator</i>	<i>22</i>
<i>Java Left Shift Operator Example.....</i>	<i>22</i>
<i>Java Right Shift Operator</i>	<i>23</i>
<i>Java Right Shift Operator Example</i>	<i>23</i>
<i>Java Shift Operator Example: >> vs >>></i>	<i>23</i>
<i>Java AND Operator Example: Logical && and Bitwise &</i>	<i>24</i>

Java AND Operator Example: Logical && vs Bitwise &	24
Java OR Operator Example: Logical and Bitwise 	25
Java Ternary Operator	25
Java Ternary Operator Example.....	26
Java Assignment Operator.....	26
Java Assignment Operator Example	26
Java Assignment Operator Example	27
Java Assignment Operator Example: Adding short.....	27
JAVA KEYWORDS	28
LIST OF JAVA KEYWORDS	28
JAVA CONTROL STATEMENTS CONTROL FLOW IN JAVA	31
2) if-else statement	33
Switch Statement:	36
Loop Statements	37
for loop.....	37
while loop.....	40
do-while loop	42
break statement.....	43
continue statement.....	44
JAVA ARRAYS.....	44
Advantages	44
Disadvantages	45
Types of Array in java	45
SINGLE DIMENSIONAL ARRAY	45
INITIALIZATION OF JAVA ARRAY.....	47
MULTIDIMENSIONAL ARRAY	47
JAGGED ARRAY	49

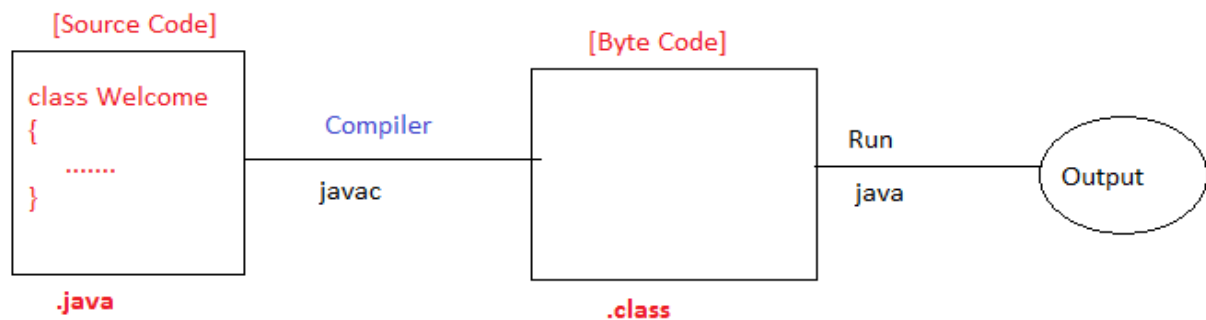
JAVA

- Java is a **programming language** and a **platform**.
- Java is a high level, robust, object-oriented and secure programming language.
- Java is a general purpose, object oriented programming language.
- General purpose – We can able to develop all types of application using java.
- Object Oriented – In java main component/ basic build block is class.

HISTORY OF JAVA

- Java was developed by *Sun Microsystems* (which is now the subsidiary of Oracle) in the year 1995.
- **James Gosling** is known as the father of Java. Before Java, its name was *Oak*. Since Oak was already a registered company, so James Gosling and his team changed the name from Oak to Java.

Java Code Execution Flow



Source Code

present in human understandable format
not present in machine understandable format

Byte Code

not present in human understandable format
not present in machine understandable format

? - JVM - Java Virtual Machine

JVM - It converts byte code to machine code.

Machine Code

not present in human understandable format
present in machine understandable format

OS --> Machine Code --> Output

Setup development environment

Install JDK (Java Development Kit)

Step 1: Verify/Check that it is already installed or not

Open command prompt and execute below commands:



javac

```
Microsoft Windows [Version 6.3.9600]
(c) 2013 Microsoft Corporation. All rights reserved.

C:\Users\hp>javac
'javac' is not recognized as an internal or external command,
operable program or batch file.
```

Step 2: Download JDK

- Click on the link: <https://www.oracle.com/java/technologies/downloads/>
- Search for Java 8, Choose type of operating system.

Linux macOS Solaris Windows		
Product/file description	File size	Download
x86 Installer	157.99 MB	 jdk-8u321-windows-i586.exe
x64 Installer	171.09 MB	 jdk-8u321-windows-x64.exe

- Create an account, Sign-in and download

Step 2: Install JDK

Double click on ***jdk-8u321-windows-x64.exe*** file



Step 3: Set the Path

Right click on "this PC". It can be named as "My Computer" in some systems. Choose "properties" from the options.

Writing your first Java program

Step1: Open any editor (Notepad, Edit plus, Sublime text etc.)

Step2: Write below java program

```
class Welcome
{
    public static void main(String args[])
    {
        System.out.println("Welcome to Java World !!");
    }
}
```

Step3: Save this program in a file (e.g. ***Welcome.java***).

Note: It is always recommended to give file name same as main class name.

Step4: Open command prompt and go to source code location.

Step5: Compile Java Program.

Syntax: *javac filename*

javac Welcome.java

Step6: Run Java Program.

Syntax: *java Main-Class-Name*

Java Welcome

Application

According to Sun, 3 billion devices run Java. There are many devices where Java is currently used. Some of them are as follows:

1. Desktop Applications such as acrobat reader, media player, antivirus, etc.
2. Web Applications such as irctc.co.in, javatpoint.com, etc.
3. Enterprise Applications such as banking applications.
4. Mobile
5. Embedded System
6. Smart Card
7. Robotics
8. Games, etc.

Types of Java Applications

1) Standalone Application

- Standalone applications are also known as desktop applications or window-based applications.

2) Web Application

- An application that runs on the server side and creates a dynamic page is called a web application.

3) Enterprise Application

- An application that is distributed in nature, such as banking applications, etc. is called an enterprise application.

4) Mobile Application

- An application which is created for mobile devices is called a mobile application.
- Currently, Android and Java ME are used for creating mobile applications.

Java Platforms / Editions

There are 4 platforms or editions of Java:

1) Java SE (Java Standard Edition)

- It is a Java programming platform. It includes Java programming APIs such as java.lang, java.io, java.net, java.util, java.sql, java.math etc.
- It includes core topics like OOPs, [String](#), Regex, Exception, Inner classes, Multithreading, I/O Stream, Networking, AWT, Swing, Reflection, Collection, etc.

2) Java EE (Java Enterprise Edition)

- It is an enterprise platform that is mainly used to develop web and enterprise applications.
- It is built on top of the Java SE platform. It includes topics like Servlet, JSP, Web Services, EJB, [JPA](#), etc.

3) Java ME (Java Micro Edition)

- It is a micro platform that is dedicated to mobile applications.

4) JavaFX

- It is used to develop rich internet applications.
- It uses a lightweight user interface API.

Features of Java

- A list of the most important features of the Java language is given below

1. [Simple](#)
2. [Object-Oriented](#)
3. [Portable](#)
4. [Platform independent](#)
5. [Secured](#)

6. Robust
7. Architecture neutral
8. Interpreted
9. High Performance
10. Multithreaded
11. Distributed
12. Dynamic

C++ vs Java

Comparison Index	C++	Java
Platform-independent	C++ is platform-dependent.	Java is platform-independent.
Mainly used for	C++ is mainly used for system programming.	Java is mainly used for application programming. It is widely used in Windows-based, web-based, enterprise, and mobile applications.
Design Goal	C++ was designed for systems and applications programming. It was an extension of the C programming language .	Java was designed and created as an interpreter for printing systems but later extended as a support network computing. It was designed to be easy to use and accessible to a broader audience.
Goto	C++ supports the goto statement.	Java doesn't support the goto statement.
Multiple inheritance	C++ supports multiple inheritance.	Java doesn't support multiple inheritance through class. It can be achieved by using interfaces in java .
Operator Overloading	C++ supports operator overloading .	Java doesn't support operator overloading.
Pointers	C++ supports pointers . You can write a pointer program in C++.	Java supports pointer internally. However, you can't write the pointer program in java. It means java has restricted pointer support in java.

Compiler and Interpreter	C++ uses compiler only. C++ is compiled and run using the compiler which converts source code into machine code so, C++ is platform dependent.	Java uses both compiler and interpreter. Java source code is converted into bytecode at compilation time. The interpreter executes this bytecode at runtime and produces output. Java is interpreted that is why it is platform-independent.
Call by Value and Call by reference	C++ supports both call by value and call by reference.	Java supports call by value only. There is no call by reference in java.
Structure and Union	C++ supports structures and unions.	Java doesn't support structures and unions.
Thread Support	C++ doesn't have built-in support for threads. It relies on third-party libraries for thread support.	Java has built-in thread support.
Documentation comment	C++ doesn't support documentation comments.	Java supports documentation comment (<code>/** ... */</code>) to create documentation for java source code.
Virtual Keyword	C++ supports virtual keyword so that we can decide whether or not to override a function.	Java has no virtual keyword. We can override all non-static methods by default. In other words, non-static methods are virtual by default.
unsigned right shift >>>	C++ doesn't support >>> operator.	Java supports unsigned right shift >>> operator that fills zero at the top for the negative numbers. For positive numbers, it works same like >> operator.

C++ Program Example

File: main.cpp

```

1. #include <iostream>
2. using namespace std;
3. int main() {
4.     cout << "Hello C++ Programming";
5.     return 0;
6. }
```

Output:

Hello C++ Programming

Java Program Example

File: Simple.java

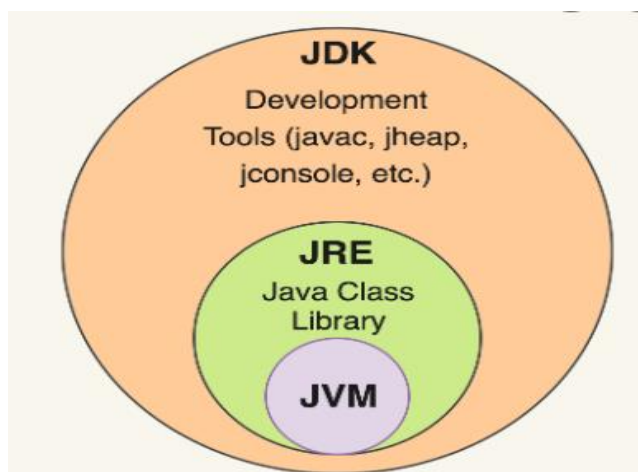
```
1. class Simple{
2.     public static void main(String args[]){
3.         System.out.println("Hello Java");
4.     }
5. }
```

Output:

```
Hello Java
```

JDK, JRE and JVM?

JDK	JRE	JVM
JDK stands for Java Development Kit	JRE stands for Java Runtime Environment	JVM Stands for Java Virtual Machine
It contains JRE + development tools (javac, javap etc.).	It contains JVM + Java class library.	It provides environment to run byte code.



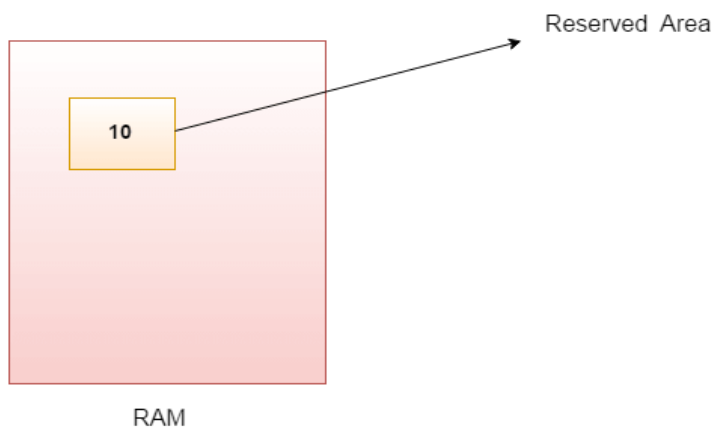
Java Variables

- A variable is a container which holds the value while the [Java program](#) is executed. A variable is assigned with a data type.
- Variable is a name of memory location.
- There are three types of variables in java: local, instance and static.

There are two types of [data types in Java](#): primitive and non-primitive.

Variable

- A variable is the name of a reserved area allocated in memory.
- In other words, it is a name of the memory location. It is a combination of "vary + able" which means its value can be changed.



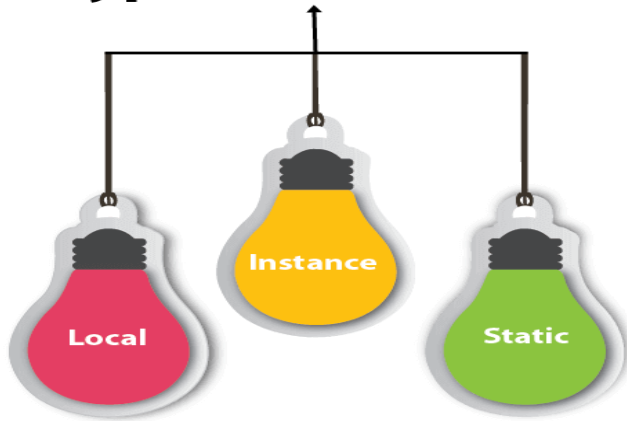
1. `int data=50;`//Here data is variable

Types of Variables

There are three types of variables in [Java](#):

- local variable
- instance variable
- static variable

Types of Variables



1) Local Variable

- A variable declared inside the body of the method is called local variable.
- You can use this variable only within that method and the other methods in the class aren't even aware that the variable exists.

A local variable cannot be defined with "static" keyword.

2) Instance Variable

- A variable declared inside the class but outside the body of the method, is called an instance variable. It is not declared as static.
- It is called an instance variable because its value is instance-specific and is not shared among instances.

3) Static variable

- A variable that is declared as static is called a static variable.
- It cannot be local. You can create a single copy of the static variable and share it among all the instances of the class.
- Memory allocation for static variables happens only once when the class is loaded in the memory.

Example to understand the types of variables in java

```
1. public class A
2. {
3.     static int m=100;//static variable
4.     void method()
5.     {
6.         int n=90;//local variable
7.     }
```

```
8.  public static void main(String args[])
9.  {
10.  int data=50;//instance variable
11.  }
12. }//end of class
```

Java Variable Example: Add Two Numbers

```
1.  public class Simple{
2.  public static void main(String[] args){
3.  int a=10;
4.  int b=10;
5.  int c=a+b;
6.  System.out.println(c);
7.  }
8.  }
```

Output:

```
20
```

Java Variable Example: Widening

```
1.  public class Simple{
2.  public static void main(String[] args){
3.  int a=10;
4.  float f=a;
5.  System.out.println(a);
6.  System.out.println(f);
7.  }}
```

Output:

```
10
10.0
```

Java Variable Example: Narrowing (Typecasting)

```
1.  public class Simple{
2.  public static void main(String[] args){
3.  float f=10.5f;
```

4. `//int a=f;//Compile time error`
5. `int a=(int)f;`
6. `System.out.println(f);`
7. `System.out.println(a);`
8. `}}`

Output:

```
10.5
10
```

Java Variable Example: Overflow

1. `class Simple{`
2. `public static void main(String[] args){`
3. `//Overflow`
4. `int a=130;`
5. `byte b=(byte)a;`
6. `System.out.println(a);`
7. `System.out.println(b);`
8. `}}`

Output:

```
130
-126
```

Java Variable Example: Adding Lower Type

1. `class Simple{`
2. `public static void main(String[] args){`
3. `byte a=10;`
4. `byte b=10;`
5. `//byte c=a+b;//Compile Time Error: because a+b=20 will be int`
6. `byte c=(byte)(a+b);`
7. `System.out.println(c);`
8. `}}`

Output:

```
20
```

Data Types in Java

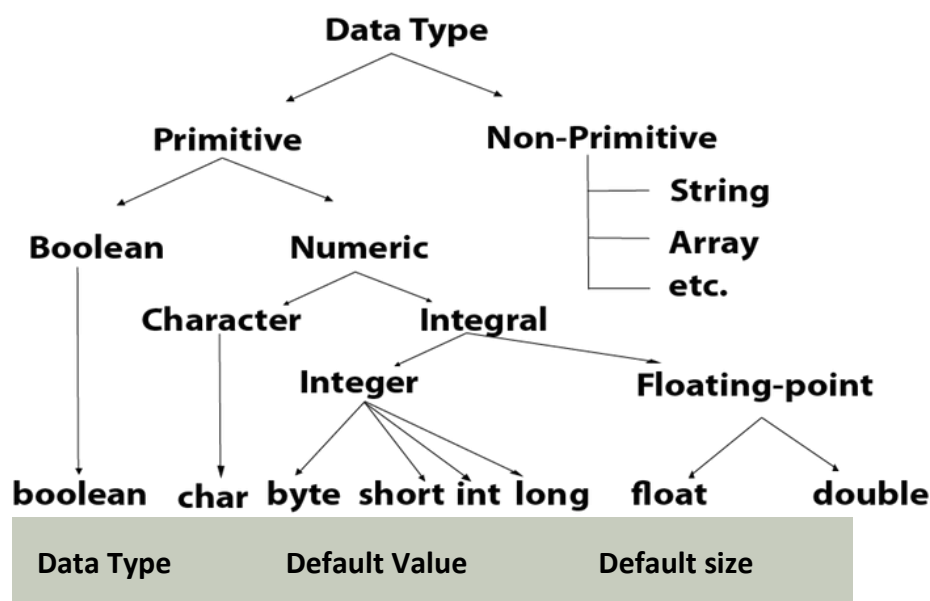
- Data types specify the different sizes and values that can be stored in the variable.
- There are two types of data types in Java:
 1. **Primitive data types:** The primitive data types include boolean, char, byte, short, int, long, float and double.
 2. **Non-primitive data types:** The non-primitive data types include [Classes](#), [Interfaces](#), and [Arrays](#).

Java Primitive Data Types

- In Java language, primitive data types are the building blocks of data manipulation.
- These are the most basic data types available in [Java language](#).

There are 8 types of primitive data types:

- boolean data type
- byte data type
- char data type
- short data type
- int data type
- long data type
- float data type
- double data type



boolean	false	1 bit
char	'\u0000'	2 byte
byte	0	1 byte
short	0	2 byte
int	0	4 byte
long	0L	8 byte
float	0.0f	4 byte
double	0.0d	8 byte

Boolean Data Type

- The Boolean data type is used to store only two possible values: true and false.
- This data type is used for simple flags that track true/false conditions.
- The Boolean data type specifies one bit of information, but its "size" can't be defined precisely.

Example:

1. Boolean one = **false**

Byte Data Type

- The byte data type is an example of primitive data type.
- It is an 8-bit signed two's complement integer.
- Its value-range lies between -128 to 127 (inclusive). Its minimum value is -128 and maximum value is 127. Its default value is 0.
- The byte data type is used to save memory in large arrays where the memory savings is most required.
- It saves space because a byte is 4 times smaller than an integer. It can also be used in place of "int" data type.

Example:

1. **byte** a = **10**, **byte** b = **-20**

Short Data Type

- The short data type is a 16-bit signed two's complement integer.
- Its value-range lies between -32,768 to 32,767 (inclusive). Its minimum value is -32,768 and maximum value is 32,767. Its default value is 0.
- The short data type can also be used to save memory just like byte data type.
- A short data type is 2 times smaller than an integer.

Example:

1. `short s = 10000, short r = -5000`

Int Data Type

- The int data type is a 32-bit signed two's complement integer.
- Its value-range lies between - 2,147,483,648 (-2^{31}) to 2,147,483,647 ($2^{31} - 1$) (inclusive).
- Its minimum value is - 2,147,483,648 and maximum value is 2,147,483,647. Its default value is 0.
- The int data type is generally used as a default data type for integral values unless if there is no problem about memory.

Example:

1. `int a = 100000, int b = -200000`

Long Data Type

- The long data type is a 64-bit two's complement integer. Its value-range lies between -9,223,372,036,854,775,808 (-2^{63}) to 9,223,372,036,854,775,807 ($2^{63} - 1$) (inclusive). Its minimum value is - 9,223,372,036,854,775,808 and maximum value is 9,223,372,036,854,775,807.
- Its default value is 0. The long data type is used when you need a range of values more than those provided by int.

Example:

1. `long a = 100000L, long b = -200000L`

Float Data Type

- The float data type is a single-precision 32-bit IEEE 754 floating point. Its value range is unlimited.

- It is recommended to use a float (instead of double) if you need to save memory in large arrays of floating point numbers.
- The float data type should never be used for precise values, such as currency. Its default value is 0.0F.

Example:

1. `float f1 = 234.5f`

Double Data Type

- The double data type is a double-precision 64-bit IEEE 754 floating point.
- Its value range is unlimited.
- The double data type is generally used for decimal values just like float.
- The double data type also should never be used for precise values, such as currency. Its default value is 0.0d.

Example:

1. `double d1 = 12.3`

Char Data Type

- The char data type is a single 16-bit Unicode character.
- Its value-range lies between '\u0000' (or 0) to '\uffff' (or 65,535 inclusive).
- The char data type is used to store characters.

Example:

1. `char letterA = 'A'`

Why char uses 2 byte in java and what is \u0000 ?

- It is because java uses Unicode system not ASCII code system.
- The \u0000 is the lowest range of Unicode system.
- To get detail explanation about Unicode visit next page.

Converting String to Integer

Integer

- parseInt
- parseInt is a static method.
- It is used to convert string to integer.

```
int Integer.parseInt(string)
```

Operators in Java

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	<i>expr++ expr--</i>
	prefix	<i>++expr --expr +expr -expr ~ !</i>
Arithmetic	multiplicative	<i>* / %</i>
	additive	<i>+ -</i>
Shift	shift	<i><< >> >>></i>

Relational	comparison	< > <= >= instanceof
	equality	== !=
Bitwise	bitwise AND	&
	bitwise exclusive OR	^
	bitwise inclusive OR	
Logical	logical AND	&&
	logical OR	
Ternary	ternary	? :
Assignment	assignment	= += -= *= /= %= &= ^= = <<= >>= >>>=

Java Unary Operator

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

- incrementing/decrementing a value by one
- negating an expression
- inverting the value of a boolean

Java Unary Operator Example: ++ and --

1. **public class** OperatorExample{
2. **public static void** main(String args[]){
3. **int** x=10;
4. System.out.println(x++);**//10 (11)**
5. System.out.println(++x);**//12**
6. System.out.println(x--);**//12 (11)**
7. System.out.println(--x);**//10**
8. **}}**

Output:

```
12
12
10
```

Java Unary Operator Example 2: ++ and --

1. **public class** OperatorExample{
2. **public static void** main(String args[]){
3. **int** a=10;
4. **int** b=10;
5. System.out.println(a++ + ++a);//10+12=22
6. System.out.println(b++ + b++);//10+11=21
- 7.
8. }}

Output:

```
22
21
```

Java Unary Operator Example: ~ and !

1. **public class** OperatorExample{
2. **public static void** main(String args[]){
3. **int** a=10;
4. **int** b=-10;
5. **boolean** c=true;
6. **boolean** d=false;
7. System.out.println(~a);//-11 (minus of total positive value which starts from 0)
8. System.out.println(~b);//9 (positive of total minus, positive starts from 0)
9. System.out.println(!c);//false (opposite of boolean value)
10. System.out.println(!d);//true
11. }}

Output:

```
-11
9
false
true
```

Java Arithmetic Operators

Java arithmetic operators are used to perform addition, subtraction, multiplication, and division. They act as basic mathematical operations.

Java Arithmetic Operator Example

```
1. public class OperatorExample{
2.     public static void main(String args[]){
3.         int a=10;
4.         int b=5;
5.         System.out.println(a+b);//15
6.         System.out.println(a-b);//5
7.         System.out.println(a*b);//50
8.         System.out.println(a/b);//2
9.         System.out.println(a%b);//0
10.    }
```

Output:

```
15
5
50
2
0
```

Java Arithmetic Operator Example: Expression

```
1. public class OperatorExample{
2.     public static void main(String args[]){
3.         System.out.println(10*10/5+3-1*4/2);
4.     }
```

Output:

```
21
```

Java Left Shift Operator

The Java left shift operator << is used to shift all of the bits in a value to the left side of a specified number of times.

Java Left Shift Operator Example

```
1. public class OperatorExample{
```

2. **public static void** main(String args[]){
3. System.out.println(10<<2);*//10*2^2=10*4=40*
4. System.out.println(10<<3);*//10*2^3=10*8=80*
5. System.out.println(20<<2);*//20*2^2=20*4=80*
6. System.out.println(15<<4);*//15*2^4=15*16=240*
7. }}

Output:

```
40
80
80
240
```

Java Right Shift Operator

The Java right shift operator >> is used to move the value of the left operand to right by the number of bits specified by the right operand.

Java Right Shift Operator Example

1. **public** OperatorExample{
2. **public static void** main(String args[]){
3. System.out.println(10>>2);*//10/2^2=10/4=2*
4. System.out.println(20>>2);*//20/2^2=20/4=5*
5. System.out.println(20>>3);*//20/2^3=20/8=2*
6. }}

Output:

```
2
5
2
```

Java Shift Operator Example: >> vs >>>

1. **public class** OperatorExample{
2. **public static void** main(String args[]){
3. *//For positive number, >> and >>> works same*
4. System.out.println(20>>2);
5. System.out.println(20>>>2);
6. *//For negative number, >>> changes parity bit (MSB) to 0*

```
7. System.out.println(-20>>2);
8. System.out.println(-20>>>2);
9. }}
```

Output:

```
5
5
-5
1073741819
```

Java AND Operator Example: Logical && and Bitwise &

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.

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

```
1. public class OperatorExample{
2.     public static void main(String args[]){
3.         int a=10;
4.         int b=5;
5.         int c=20;
6.         System.out.println(a<b&&a<c);//false && true = false
7.         System.out.println(a<b&a<c);//false & true = false
8.     }}
```

Output:

```
false
false
```

Java AND Operator Example: Logical && vs Bitwise &

```
1. public class OperatorExample{
2.     public static void main(String args[]){
3.         int a=10;
4.         int b=5;
5.         int c=20;
6.         System.out.println(a<b&&a++<c);//false && true = false
7.         System.out.println(a);//10 because second condition is not checked
8.         System.out.println(a<b&a++<c);//false && true = false
```


9. `System.out.println(a);`//11 because second condition is checked
10. `}}`

Output:

```
false
10
false
11
```

Java OR Operator Example: Logical `||` and Bitwise `|`

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.

The bitwise `|` operator always checks both conditions whether first condition is true or false.

1. `public class` OperatorExample{
2. `public static void` main(String args[]){
3. `int` a=10;
4. `int` b=5;
5. `int` c=20;
6. `System.out.println(a>b | a<c);`//true || true = true
7. `System.out.println(a>b | a<c);`//true | true = true
8. `||` vs `|`
9. `System.out.println(a>b | a++<c);`//true || true = true
10. `System.out.println(a);`//10 because second condition is not checked
11. `System.out.println(a>b | a++<c);`//true | true = true
12. `System.out.println(a);`//11 because second condition is checked
13. `}}`

Output:

```
true
true
true
10
true
11
```

Java Ternary Operator

Java 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 three operands.

Java Ternary Operator Example

```
1. public class OperatorExample{
2. public static void main(String args[]){
3. int a=2;
4. int b=5;
5. int min=(a<b)?a:b;
6. System.out.println(min);
7. }}
```

Output:

```
2
```

Another Example:

```
1. public class OperatorExample{
2. public static void main(String args[]){
3. int a=10;
4. int b=5;
5. int min=(a<b)?a:b;
6. System.out.println(min);
7. }}
```

Output:

```
5
```

Java Assignment Operator

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

Java Assignment Operator Example

```
1. public class OperatorExample{
2. public static void main(String args[]){
3. int a=10;
4. int b=20;
```

5. `a+=4; //a=a+4 (a=10+4)`
6. `b-=4; //b=b-4 (b=20-4)`
7. `System.out.println(a);`
8. `System.out.println(b);`
9. `}}`

Output:

```
14
16
```

Java Assignment Operator Example

1. `public class` OperatorExample{
2. `public static void` main(String[] args){
3. `int` a=10;
4. `a+=3; //10+3`
5. `System.out.println(a);`
6. `a-=4; //13-4`
7. `System.out.println(a);`
8. `a*=2; //9*2`
9. `System.out.println(a);`
10. `a/=2; //18/2`
11. `System.out.println(a);`
12. `}}`

Output:

```
13
9
18
9
```

Java Assignment Operator Example: Adding short

1. `public class` OperatorExample{
2. `public static void` main(String args[]){
3. `short` a=10;
4. `short` b=10;
5. `//a+=b; //a=a+b internally so fine`
6. `a=a+b; //Compile time error because 10+10=20 now int`

7. `System.out.println(a);`
8. `}}`

Output:

Compile time error

After type cast:

1. `public class` OperatorExample{
2. `public static void` main(String args[]){
3. `short` a=10;
4. `short` b=10;
5. `a=(short)(a+b);`//20 which is int now converted to short
6. `System.out.println(a);`
7. `}}`

Output:

20

Java Keywords

Java keywords are also known as reserved words. Keywords are particular words that act as a key to a code. These are predefined words by Java so they cannot be used as a variable or object name or class name.

List of Java Keywords

A list of Java keywords or reserved words are given below:

1. **abstract**: Java abstract keyword is used to declare an abstract class. An abstract class can provide the implementation of the interface. It can have abstract and non-abstract methods.
2. **boolean**: Java boolean keyword is used to declare a variable as a boolean type. It can hold True and False values only.
3. **break**: Java break keyword is used to break the loop or switch statement. It breaks the current flow of the program at specified conditions.
4. **byte**: Java byte keyword is used to declare a variable that can hold 8-bit data values.

5. **case**: Java case keyword is used with the switch statements to mark blocks of text.
6. **catch**: Java catch keyword is used to catch the exceptions generated by try statements. It must be used after the try block only.
7. **char**: Java char keyword is used to declare a variable that can hold unsigned 16-bit Unicode characters
8. **class**: Java class keyword is used to declare a class.
9. **continue**: Java continue keyword is used to continue the loop. It continues the current flow of the program and skips the remaining code at the specified condition.
10. **default**: Java default keyword is used to specify the default block of code in a switch statement.
11. **do**: Java do keyword is used in the control statement to declare a loop. It can iterate a part of the program several times.
12. **double**: Java double keyword is used to declare a variable that can hold 64-bit floating-point number.
13. **else**: Java else keyword is used to indicate the alternative branches in an if statement.
14. **enum**: Java enum keyword is used to define a fixed set of constants. Enum constructors are always private or default.
15. **extends**: Java extends keyword is used to indicate that a class is derived from another class or interface.
16. **final**: Java final keyword is used to indicate that a variable holds a constant value. It is used with a variable. It is used to restrict the user from updating the value of the variable.
17. **finally**: Java finally keyword indicates a block of code in a try-catch structure. This block is always executed whether an exception is handled or not.
18. **float**: Java float keyword is used to declare a variable that can hold a 32-bit floating-point number.
19. **for**: Java for keyword is used to start a for loop. It is used to execute a set of instructions/functions repeatedly when some condition becomes true. If the number of iteration is fixed, it is recommended to use for loop.
20. **if**: Java if keyword tests the condition. It executes the if block if the condition is true.
21. **implements**: Java implements keyword is used to implement an interface.
22. **import**: Java import keyword makes classes and interfaces available and accessible to the current source code.

23. **instanceof**: Java instanceof keyword is used to test whether the object is an instance of the specified class or implements an interface.
24. **int**: Java int keyword is used to declare a variable that can hold a 32-bit signed integer.
25. **interface**: Java interface keyword is used to declare an interface. It can have only abstract methods.
26. **long**: Java long keyword is used to declare a variable that can hold a 64-bit integer.
27. **native**: Java native keyword is used to specify that a method is implemented in native code using JNI (Java Native Interface).
28. **new**: Java new keyword is used to create new objects.
29. **null**: Java null keyword is used to indicate that a reference does not refer to anything. It removes the garbage value.
30. **package**: Java package keyword is used to declare a Java package that includes the classes.
31. **private**: Java private keyword is an access modifier. It is used to indicate that a method or variable may be accessed only in the class in which it is declared.
32. **protected**: Java protected keyword is an access modifier. It can be accessible within the package and outside the package but through inheritance only. It can't be applied with the class.
33. **public**: Java public keyword is an access modifier. It is used to indicate that an item is accessible anywhere. It has the widest scope among all other modifiers.
34. **return**: Java return keyword is used to return from a method when its execution is complete.
35. **short**: Java short keyword is used to declare a variable that can hold a 16-bit integer.
36. **static**: Java static keyword is used to indicate that a variable or method is a class method. The static keyword in Java is mainly used for memory management.
37. **strictfp**: Java strictfp is used to restrict the floating-point calculations to ensure portability.
38. **super**: Java super keyword is a reference variable that is used to refer to parent class objects. It can be used to invoke the immediate parent class method.
39. **switch**: The Java switch keyword contains a switch statement that executes code based on test value. The switch statement tests the equality of a variable against multiple values.
40. **synchronized**: Java synchronized keyword is used to specify the critical sections or methods in multithreaded code.

41. **this**: Java this keyword can be used to refer the current object in a method or constructor.
42. **throw**: The Java throw keyword is used to explicitly throw an exception. The throw keyword is mainly used to throw custom exceptions. It is followed by an instance.
43. **throws**: The Java throws keyword is used to declare an exception. Checked exceptions can be propagated with throws.
44. **transient**: Java transient keyword is used in serialization. If you define any data member as transient, it will not be serialized.
45. **try**: Java try keyword is used to start a block of code that will be tested for exceptions. The try block must be followed by either catch or finally block.
46. **void**: Java void keyword is used to specify that a method does not have a return value.
47. **volatile**: Java volatile keyword is used to indicate that a variable may change asynchronously.
48. **while**: Java while keyword is used to start a while loop. This loop iterates a part of the program several times. If the number of iteration is not fixed, it is recommended to use the while loop.

Java Control Statements | Control Flow in Java

Java provides three types of control flow statements.

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

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.
- if statement consists of an expression followed by one or more statements

Syntax

```
if(expression)
{
    // Statements will execute if the expression is true
}
```

Example#1

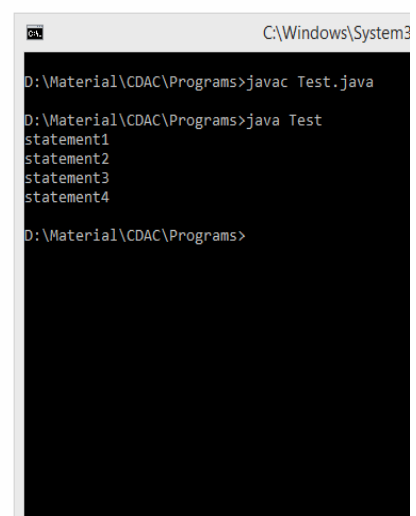
```
public class Test
{
    public static void main(String args[])
    {
        int x = 1;

        if( x < 2)
        {
            System.out.print("This is if statement");
        }
    }
}
```

Example-2

```
class Test
{
    public static void main(String args[])
    {
        int x = 1;

        if(x<2)
        {
            System.out.println("statement1");
            System.out.println("statement2");
            System.out.println("statement3");
        }
        System.out.println("statement4");
    }
}
```



```
C:\Windows\System32
D:\Material\CDAC\Programs>javac Test.java
D:\Material\CDAC\Programs>java Test
statement1
statement2
statement3
statement4
D:\Material\CDAC\Programs>
```

Example#3


```
class Test
{
    public static void main(String args[])
    {
        int x = 1;

        if(x>2)
            System.out.println("statement1");
            System.out.println("statement2");
            System.out.println("statement3");
            System.out.println("statement4");
    }
}
```

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

```
if(expression)
{
    // Executes when the expression is true
}
else
{
    // Executes when the expression is false
}
```

Example-1

```
class AgeChecker
{
    public static void main(String args[])
    {
        int age = 20;
        if(age >= 18)
        {
```

```
        System.out.println("You are eligible for voting");
    }
    else
    {
        System.out.println("You are not eligible for voting");
    }
}
}
```

Example-2

Student.java

```
public class Student {
    public static void main(String[] args) {
        int x = 10;
        int y = 12;
        if(x+y < 10) {
            System.out.println("x + y is less than 10");
        } else {
            System.out.println("x + y is greater than 20");
        }
    }
}
```

Output:

```
x + y is greater than 20
```

If-else-if :-

- 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

```
if(expression 1)
{
    Statement(s) to be executed if expression 1 is true
}
else if (expression2)
{
    Statement(s) to be executed if expression 2 is true
}
else if (expression3)
{
    Statement(s) to be executed if expression 3 is true
}
else
{
    Statement(s) to be executed if no expression is true
}
```

Example

```
class NumberChecker
{
    public static void main(String args[])
    {
        int number = 0;

        if (number > 0)
        {
            System.out.println("The number is positive.");
        }
        else if (number < 0)
        {
            System.out.println("The number is negative.");
        }
        else
        {
            System.out.println("The number is 0.");
        }
    }
}
```

Example-2

Student.java

1. **public class** Student {
2. **public static void** main(String[] args) {

```
3. String city = "Delhi";
4. if(city == "Meerut") {
5. System.out.println("city is meerut");
6. }else if (city == "Noida") {
7. System.out.println("city is noida");
8. }else if(city == "Agra") {
9. System.out.println("city is agra");
10. }else {
11. System.out.println(city);
12. }
13. }
14. }
```

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.

Syntax—

```
switch(expression)
{
    case x:
        // code block
        break;
    case y:
        // code block
        break;
    default:
        // code block
}
```

Example--1

```
class NumberChecker
{
    public static void main(String args[])
```

```
{
    int a = 3;
    switch(a)
    {
        case 1:
            System.out.println("a is 1");
            break;
        case 2:
            System.out.println("a is 2");
            break;
        case 3:
            System.out.println("a is 3");
            break;
        default:
            System.out.println("default");
    }
}
```

Loop 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.

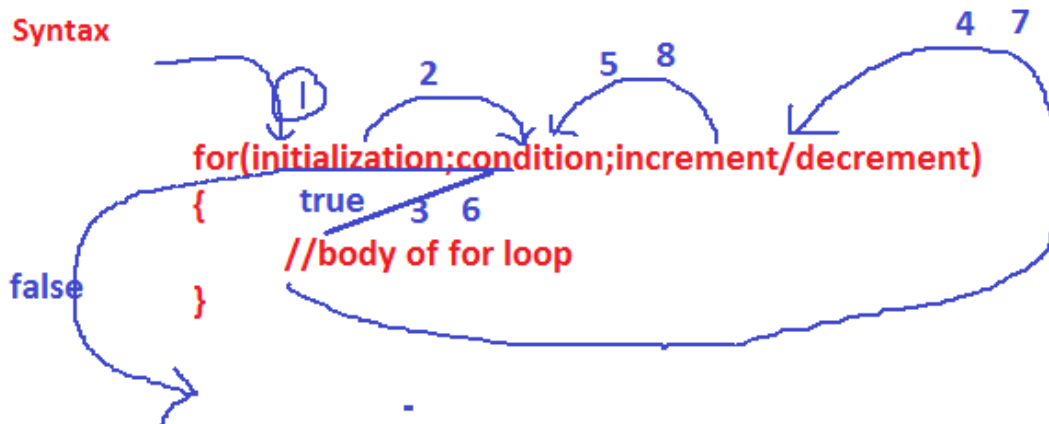
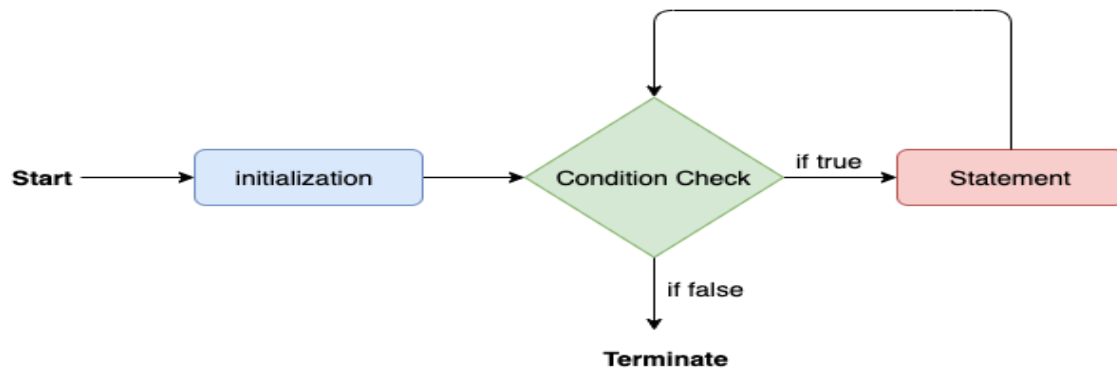
1. for loop
2. while loop
3. do-while loop

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. [//block of statements](#)
3. }

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



Example#1

```
class Test{
    public static void main(String args[]){
        for(int i=1;i<=5;i++){
            System.out.println(i);
        }
    }
}
```

Example#2

```
class Test{
    public static void main(String args[]){
        for(int i=5; i>=1; i--){
            System.out.println(i);
        }
    }
}
```

Example#3

Write a program to print all even numbers from 1 to 10.

```
class Test{
    public static void main(String args[]){
        for(int i=1; i<=10; i++){
            if(i%2 == 0){
                System.out.println(i);
            }
        }
    }
}
```

Example#4

Write a program to print all even numbers from 1 to n.

```
class Test{
    public static void main(String args[]){
        int n = Integer.parseInt(args[0]);

        for(int i=1; i<= n; i++){
            if(i%2 == 0){
                System.out.println(i);
            }
        }
    }
}
```

Example#5

Write a program to find sum of all numbers from 1 to 5.

```
class Test{
    public static void main(String args[]){
        int sum = 0;
        for(int i=1; i<=5; i++){
            sum = sum +i;
        }
        System.out.println(sum);
    }
}
```

Example#6

Write a program to find sum of all even numbers from 1 to 5.

```
class Test{
    public static void main(String args[]){
        int sum = 0;
        for(int i=1; i<=5; i++){
            if(i%2 == 0){
                sum = sum +i;
            }
        }
        System.out.println(sum);
    }
}
```

while loop

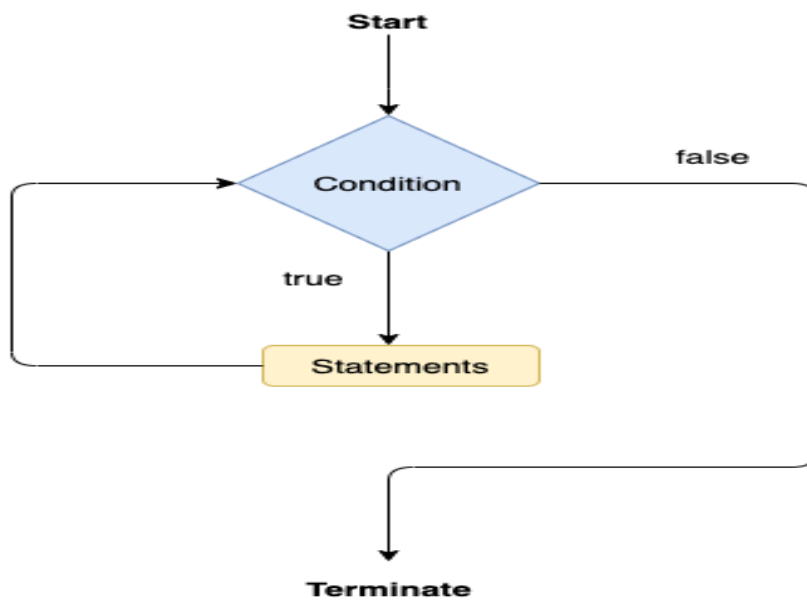
- The while loop is also used to iterate over the number of statements multiple times.
- However, if we don't know the number of iterations in advance, it is recommended to use a while loop.
- Unlike for loop, the initialization and increment/decrement doesn't take place inside the loop statement in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

Syntax

```
initialization;

while(condition)
{
    //body

    increment/decremnt
}
```

Example#1

```
class Test{
    public static void main(String args[]){
        int i = 1;
        while(i<=5){
            System.out.println(i);
            i++;
        }
    }
}
```

Example#2

Write a program to display 5 to 1 using while loop.

```
class Test{
    public static void main(String args[]){
        int i = 5;
        while(i>=1){
            System.out.println(i);
            i--;
        }
    }
}
```

}

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.

Syntax

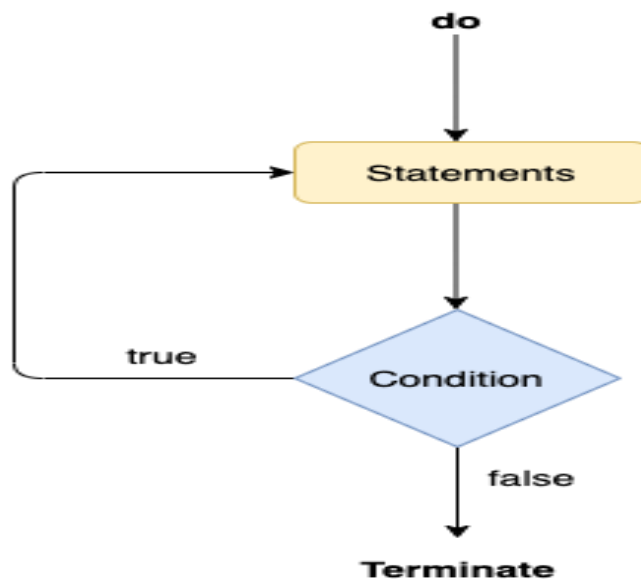
initialization; ↓

do
{

 //body
 increment / decrement
}while(condition);

for and while loop checks conditions before executing body part.

do while loop executes body part once without checking condition.



<pre>for(int i=10;i<=1;i++) { System.out.println(i); }</pre>	<pre>int i = 10; while(i<=1) { System.out.println(i); }</pre>	<pre>int i = 10; do { System.out.println(i); i++; }while(i<=1);</pre>
No Output	No Output	10

Example#1

Write a program to display 1 to 5 using do while loop.

```
class Test{
    public static void main(String args[]){
        int i = 1;
        do{
            System.out.println(i);
            i++;
        }while(i<=5);
    }
}
```

Example#2

Write a program to display 5 to 1 using do while loop.

```
class Test{
    public static void main(String args[]){
        int i = 5;
        do{
            System.out.println(i);
            i--;
        }while(i>=1);
    }
}
```

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.

Example-1

```
class Test{
    public static void main(String args[]){
        for(int i=1;i<=10;i++){
            if(i==5){
                break;
            }
            System.out.println(i);
        }
    }
}
```

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.

Example-1

```
class Test{
    public static void main(String args[]){
        for(int i=1;i<=10;i++){
            if(i==5){
                continue;
            }
            System.out.println(i);
        }
    }
}
```

Java Arrays

- Arrays are used to stores group of homogeneous and fixed size elements.
- The size of Array is fixed it means once we created Array it is not possible to increase and decrease the size.

Syntax

```
datatype arrayname[] = new datatype[size];
```

Example:

```
int arr[] = new int[5];
```

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

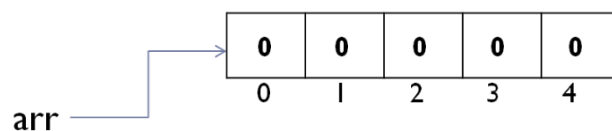
There are two types of array.

- Single Dimensional Array
- Multidimensional Array
- Jagged Array

Single Dimensional Array

- One dimensional array is a list of variables of same type that are accessed by a common name.
- An individual variable in the array is called an array element.

`int arr[] = new int[5];`



Example

```
class ArrayDemo{
    public static void main(String args[]){
        int arr[] = new int[5];

        System.out.println("Display Array Elements: ");
        for(int i=0;i<=4;i++){
            System.out.println(arr[i]);
        }
    }
}
```

Reading data from the keyboard using Scanner class

Step1:

- import Scanner class from java.util.package.
`import java.util.Scanner;`

Step2:

- Create object of Scanner class.


`Scanner sc = new Scanner(System.in);`

Step3:

- call method of scanner class to read data from the keyboard

```
integr value ----> sc.nextInt()
float value  ----> sc.nextFloat();
double      ----> sc.nextDouble();
.....
.....
string value --> sc.next() / sc.nextLine()
```

int value character

Enter your age : 20 

Enter ur name:


nextInt will read only int part and assign to age variable

age

20

nextLine will not wait for user input, it will take enter key and sign to name variable

name



```
import java.util.Scanner;
```

```
class Employee{
    public static void main(String args[]){
        Scanner sc = new Scanner(System.in);

        int age;
        String name;

        System.out.print("Enter your age : ");
        age = sc.nextInt();
        sc.nextLine();
        System.out.print("Enter your name : ");
        name = sc.nextLine();

        System.out.println("*** Employee Details ***");
        System.out.println("Your age is : "+age);
        System.out.println("Your name is : "+name);
    }
}
```

```
    }  
}
```

Read and Display Array Elements

```
import java.util.Scanner;  
  
class ReadArray{  
    public static void main(String args[]){  
        Scanner sc = new Scanner(System.in);  
        int arr[] = new int[5];  
  
        System.out.println("Read Array Elements : ");  
        for(int i=0;i<=4;i++){  
            arr[i] = sc.nextInt();  
        }  
        System.out.println("Display Array Elements : ");  
        for(int i=0;i<=4;i++){  
            System.out.println(arr[i]);  
        }  
    }  
}
```

Initialization of Java Array

- Giving values into the array at the time of declaration is known as initialization.
- The values are enclosed by braces and separated by commas.

```
int arr[] = {10,20,30,40,50}
```

Example

```
class ReadArray{  
    public static void main(String args[]){  
        int arr[] = {10,20,30,40,50};  
  
        System.out.println("Display Array Elements : ");  
        for(int i=0;i<=4;i++){  
            System.out.println(arr[i]);  
        }  
    }  
}
```

(2-D Array)

Multidimensional Array

In java multidimensional arrays are not implemented in matrix form, they implemented by using array of array concept. This approach will improve memory utilization.

2-D Array

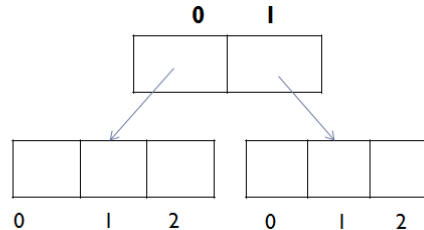
- In java multidimensional arrays are not implemented in matrix form, they implemented by using array of array concept.

```
int arr[][] = new int[2][3];
```

2-D Array : Matrix Representation

	0	1	2
0	00	01	02
1	10	11	12

2-D Array : Array of Array Representation

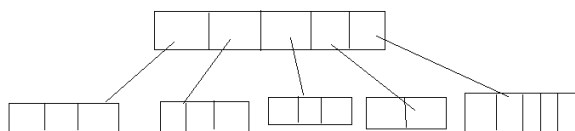


- This approach will improve memory utilization.

5 students

	P	C	M	B	IT
S1	40	40	50	.	.
S2	50	50	.	60	.
S3	60	60	.	.	50
S4	70	70	.	.	.
S5	40	40	40	50	50

Array of Array



```
import java.util.Scanner;
```

```
class Test{
```

```
    public static void main(String args[]){
```

```
        Scanner s = new Scanner(System.in);
```

```
        int arr[][] = new int[2][3];
```

```
        System.out.println("Read Array : ");
```

```
        for(int i=0;i<=1;i++){
```

```
            for(int j=0;j<=2;j++){
```

```
                arr[i][j] = s.nextInt();
```

```
            }
```

```
        }
```

```
        System.out.println("Display Array : ");
```

```
        for(int i=0;i<=1;i++){
```

```
            for(int j=0;j<=2;j++){
```

```
                System.out.print(arr[i][j]+"t");
```

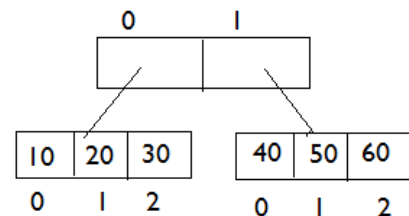
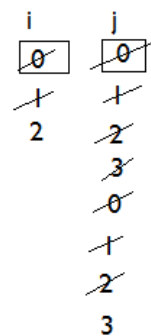
```
            }
```

```
            System.out.println();
```

```
        }
```

```
    }
```

```
}
```



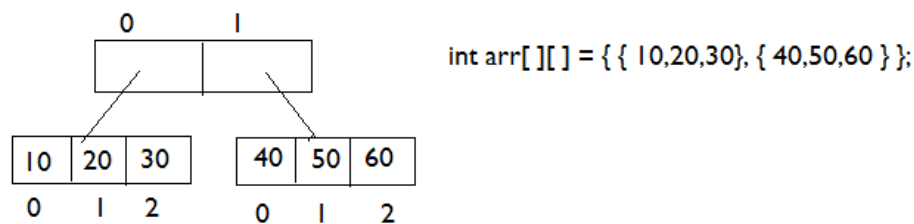
o/p

10 20 30

40 50 60

2-D Array Initialization

- Initializing a two dimensional array requires enclosing each row's initialization list in its own set of braces.



```
import java.util.Scanner;
class Test
{
    public static void main(String args[])
    {
        Scanner s = new Scanner(System.in);
        int arr[][] = { {1,2,3}, {4,5,6}};
        System.out.println("Display Array : ");
        for(int i=0;i<=1;i++)
        {
            for(int j=0;j<=2;j++)
            {
                System.out.print(arr[i][j]+"\\t");
            }
            System.out.println();
        }
    }
}
```

Jagged Array

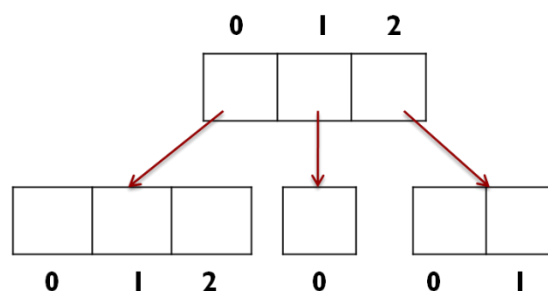
- 2-D arrays with variable number of columns in each row are known as Jagged arrays.

```
int arr[][] = new int[3][];
```

```
arr[0] = new int[3];
```

```
arr[1] = new int[1]
```

```
arr[2] = new int[2]
```



```
import java.util.Scanner;
class Test
{
```

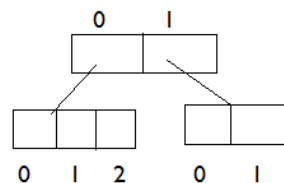
```

public static void main(String args[])
{
    Scanner s = new Scanner(System.in);
    int arr[][] = new int[2][];
    arr[0] = new int[3];
    arr[1] = new int[2];
    System.out.println("Read Array : ");
    for(int i=0;i<=1;i++)
    {
        for(int j=0;j<=arr[i].length-1;j++)
        {
            arr[i][j] = s.nextInt();
        }
    }
    System.out.println("Display Array : ");
    for(int i=0;i<=1;i++)
    {
        for(int j=0;j<=arr[i].length-1;j++)
        {
            System.out.print(arr[i][j]+"\\t");
        }
        System.out.println();
    }
}
}

```

Jagged Array Initialization

```
int arr[][] = { {10,20,30}, {40,50} };
```



```

class Test
{
    public static void main(String args[])
    {
        int arr[][] = { {10,20,30}, {40,50} };
        System.out.println("Display Array : ");
        for(int i=0;i<=1;i++)
        {
            for(int j=0;j<=arr[i].length-1;j++)
            {
                System.out.print(arr[i][j]+"\\t");
            }
            System.out.println();
        }
    }
}

```

```

    }
}

```

Assignment#1

1. Write a program to find largest element in the array?

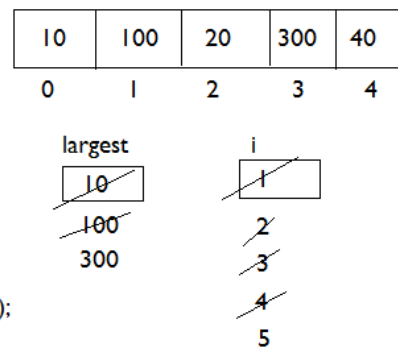
```
int arr[] = {10,100,20,300,40};
```

```

class FindLargestElement{
    public static void main(String args[]){
        int arr[] = {10,100,20,300,40};
        int largest = arr[0];

        for(int i=1;i<=4;i++){
            if(arr[i]>largest){
                largest = arr[i];
            }
        }
        System.out.println("Largest Element is : "+largest);
    }
}

```



2. Write a program to find smallest element in the array?

```
int arr[] = {10,100,20,300,40};
```

```

class FindSmallestElement{
    public static void main(String args[]){
        int arr[] = {10,100,20,300,40};
        int smallest = arr[0];

        for(int i=1;i<=4;i++){
            if(arr[i]<smallest){
                smallest = arr[i];
            }
        }
        System.out.println("Smallest Element is : "+smallest);
    }
}

```

3. Write a program to find sum of largest and smallest element of the array?

```

class FindSumSmallestLargest{
    public static void main(String args[]){
        int arr[] = {10,100,20,300,40};
        int largest = arr[0];

```

```
        int smallest = arr[0];

        for(int i=1;i<=4;i++){
            if(arr[i]>largest){
                largest = arr[i];
            }

            if(arr[i]<smallest){
                smallest = arr[i];
            }
        }
        System.out.println("Sum : "+(largest+smallest));
    }
}
```