



1.1 About Java

Java is a high level, robust, secured and object-oriented programming language.

Java is a powerful and robust programming language for developing software running on mobile devices, desktop computers, and servers.

Java was first introduced to the public in 1995 as core component of Sun Microsystems' Java platform. Initially known as OAK but was renamed as Java in 1995. It promised Write Once, Run Anywhere(WORA), providing no-cost run-times on popular platforms.

Java programming language was originally developed by Sun Microsystems which was initiated by James Gosling.



At that time Java was widely used to create Internet applications and other software programs. Sun Microsystems was purchased by Oracle in 2010. Today, Java is maintained and owned by Oracle.

Java can be used to create complete applications that may run on a single computer or be distributed among servers and clients in a network.

Since Java has its own runtime environment (JRE) and API, Java is also known as a platform.

1.2 Types of Program that can be developed using JAVA

There are mainly 2 type of applications that can be created using java programming: Application and Applet



1 Application

2 Applet

1. Application

- Java programs that run directly on your machine.
- Application can be executed on stand alone computer system.
- Applications must have a main().
- Java applications are compiled using the javac command and run using the java command.

2. Applets

- Applet is a Java program executed by a browser. Applet can run over the Internet.
- The server stores the Java Applet, which is sent to the client machine running the browser, where the Applet is then run.
- Applets do not require a main().
- Java Applets are also compiled using the javac command, but output either with a browser or with the appletviewer command.

1.3 Version of Java

Various version of Java Language along with its toolkit JDK. The Java language has undergone several changes since JDK 1.0 as well as numerous additions of classes and packages to the standard library.

Release	Year
JDK Beta	1995
JDK 1.0	1996
JDK 1.1	1997
J2SE 1.2	1998
J2SE 1.3	2000
J2SE 1.4	2002
J2SE 5.0	2004
Java SE 6	2006
Java SE 7	2011
Java SE 8	2014



1.4 Features of Java

Java syntax is defined in the Java language specification, and the Java library is defined in the Java API. The JDK is the software for developing and running Java programs. An IDE is an integrated development environment for rapidly developing programs.

Java is simple, object oriented, distributed, interpreted, robust, secure, architecture neutral, portable, high performance, multithreaded, and dynamic.

1) Java is simple language to learn.

- Java is easy to learn and its syntax is quite simple. Java is Easy to write and more readable compare to C++.
- Java has a concise, cohesive set of features that makes it easy to learn and use. It's easy to C++ programmers to learn Java because most of the concepts are drew from C++.
- It is free from pointer due to this execution time of application is improve.

2) Java can be compiled and interpreted

- Usually programming languages can be either compiled or interpret but Java is a language which can be compiled as well as interpreted.
- While running Java Program, First Java Compiler translates the Java Source program into byte code, then Java interpreter interprets this byte code to machine code.

3) Java is known as an Object Oriented programming language.

- In Java everything is Object which has some data and behavior. Java programming is object-oriented programming language.
- Java is purely follows the concepts of OOP. While C++ partially. For Example, if we want to create simple Hello world program, we need to create Class.
- Java has all OOP features such as class, object, abstraction, encapsulation, inheritance and polymorphism.

4) Java is Robust

- The Java programming language is designed for creating highly reliable software.
- Robust simply means strong. Java uses strong memory management.
- Java provides extensive compile-time checking, followed by a second level of run-time checking.





- Java focuses on Memory Management and mishandled Exceptions by introducing automatic Garbage Collector and Exception Handling.
- Mainly Java is Robust due to 1) Lack of pointers 2) Automatic Garbage Collection 3) Strong mechanism of Exception handling

5) Java is a Platform Independent

- Here Platform refers to the operating system. Java is guaranteed to be write-once, run-anywhere language.(WORA)
- A program written in Java can be executed on any operating system with any hardware.
- For Example, Word or PowerPoint, can be used on multiple operating systems like Windows 7, OS X and Linux.
- But just notice that these programs have different versions for different operating systems, which means that they are actually not platform independent, and programs written in Java, we can directly use them on either Windows or Mac OS or even Linux without any modification.
- It means, Imagine we can speak a language that everyone on the planet can understand, without them having to learn it!
- Java is Platform Independent due to Java Virtual Machine (JVM) , JVM provides the bridge between your code and the underlying platform. Usually all the operating system supports JVM.

6) Java is Secure

- Java is Secure Language because of its many features it enables to develop virus-free, tamper-free systems.
- Java is secured because:
 - No use of pointers for memory management.
 - Java programs run inside a virtual machine (the JVM).
 - Use of try-catch block to save a program from exiting due to exceptions.
 - Java code is verified before execution.
 - Defined order execution

7) Java supports concept of Multi Threading

- Modern programs typically need to do several things at the same time.
- Java multithreading feature makes it possible to write program that can do many tasks simultaneously.



- For Example, In Microsoft Word, we can download an image, printing an image and scrolling the page.
- Thread is nothing but a flow of control. When any Language execute multiple thread at a time that language is known as multithreaded Language.
- Java technology's multithreading capability provides the means to build applications with many concurrent threads of activity.
- Advantage of multithreading is that it utilizes same memory and other resources to execute multiple threads at the same time.

8) Java is Portable programming language

- If any language supports platform independent and architectural neutral feature known as portable.
- In java, all primitive types (integers, longs, floats, doubles, and so on) are of defined sizes, regardless of the machine or operating system on which the program is run.
- Java is Portable due to (1) Java programs can execute in any environment (2) Java programs can be run on any platform (Linux, Window, Mac)

1.5 Java Terms JDK

The Java Development Kit is nothing but a collection of tools that are used for development and runtime programs.



JDK is a program development environment for writing Java applets and applications.



It consist of various parts.

javac

The compiler for the Java Programming language.
The Javac compiler translates the source code to the bytecode.
Syntax# javac FileName.java

java

The launcher for Java applications
As we know java is both compile and interpret.
The java interprets the bytecode stored in a class file and execute the program to generate output.
Syntax# java ClassName

javadoc

API documentation generator
Generating Java code documentation in HTML format from Java source code which has required documentation in a predefined format.
Syntax# \$ javadoc FileName.java

appletviewer

Run and debugge applets without a web browser.
Syntax# appletviewer FileName.java

jar

Manage Java Archive(JAR)files.
.jar is a general-purpose archiving and compression tool, based on ZIP.
Syntax# jar FileName

jdb

The java Debugger.
Syntax# jdb FullClassName

javah

Generates C header and source files from a Java class.
The generated header and source files are used by C programs.
Syntax# javah FullClassName

javap

The javap command displays information about the fields,constructors and methods present in a class file.
Syntax# javap FullClassName

1.6 Java VS other Languages



**1.6.1 | Java Vs C**

	Java	C
1	Java follows the concepts of Object Oriented	C follows the concepts of Procedure Oriented
2	The program written in Java needs to be compiled and interpreted	The program written in C needs to only compile.
3	Java is Platform Independent	C is Platform dependent
4	Java is a high-level language.	C is a low-level language
5	Java uses the bottom-up approach for any program	C uses the top-down approach for any program
6	Java supports Inheritance	C does not support Inheritance
7	Security is built-in to language	Limited security
8	The way we access library file is <code>import java.io.File;</code>	The way we access library file is <code>#include <stdio.h></code>
9	There must be at least one class present	Does not support concept of class
10	Java does not support concept of pointers	C uses pointers
11	We cannot declare global variables in Java.	We can declare global variables in C.
12	Java doesn't support a goto statement.	We can write goto statment.

1.6.2 | Java Vs C++

	Java	C++
1	The program written in Java needs to be compiled and interpreted	The program written in C++ needs to only compile.
2	Java is Platform Independent	C++ is Platform dependent
3	Security is built-in to language	Limited security
4	The way we access library file is <code>import java.io.File;</code>	The way we access library file is <code>#include <iostream.h></code>





5	There must be at least one class present	It is not compulsory to declare class. We can write a program without class.
6	Java does not support concept of pointers	C++ uses pointers
7	We cannot declare global variables in Java.	We can declare global variables in C++
8	Java doesn't support a goto statement.	We can write goto statment.
9	Java doesn't support multiple and hybrid inheritance.	C++ supports multiple and hybrid inheritance.
10	Java is purely object oriented programming language.	C++ is partially object oriented.
11	Java supports multi-threading programming.	C++ does not support multi-threading programming.
12	In Java, Operators are not overridable.	In C++ , We can overload Operators.

1.7 Basic Structure of Java

Documentation Section

Package Statement

Import Statements

Class definitions

Main method class

Documentation Section

Documentation section contains comments for the programmer. For example, Programmer Name, Date of Program etc.





There are two types of comment.

- Single Line comments: It start with two forward slashes (//) and continue to the end of the current line.
- Multi Line comments: It start with a forward slash and an asterisk (/*) and end with an asterisk and a forward slash (*/).

Package Statement

Here we can declare package statement. Package is nothing but a group of classes and other functionality. Package is used to organize the classes based on functionality.

Import Statement

Like header files in C or C++ .A package is a collection of classes, interfaces and sub-packages. We can include both user define and inbuilt package using `import packagename;`. For Example, `import java.io.*;` There are two types of package user define or inbuilt.

Class Definitions

Java is purely object oriented so we need to declare class in each and every program. Class is keyword used for developing user defined data type and every java program must start with a concept of class. It is common in Java that every program have more than one class and one class contains `main()` method.

main() method

Like C and C++ , every program needs `main()` method from where execution starts. Here the syntax of `main()` method is `public static void main(String args[])`

1.8 Creating , Compiling and Executing a Java Program

Any Java program can be written using simple text editors like notepad, notepad++ or popular IDE like NetBeans , Eclipse , etc. The extension to the file name should be `.java`. We have saved our first program by the `j1.java`

Step 1: Write a code in Notepad++ , Save it with `j1.java` (Here we store in `e:\DemoJava` folder)



```
File Edit View Language Settings Macro Run Plugins Window ?
j1.java
1 class j1
2 {
3     public static void main(String s[])
4     {
5         System.out.println("Welcome to Java World!");
6     }
7 }
```

Line 1 defines a class.

It is compulsory in Java to define at least one class.

Each class has a name. In this example, the class name is j1. Usually we make name of the class and name of our Java Program same. But it is not compulsory. It is possible that our file name is jDemo.java and class name is j1.

Line 2 curly braces

The class definition should be within the curly brackets.

Line 3 defines the main method.

This line is for a main() function. A class may contain several methods. Same as C and C++, the main method is the entry point where the program begins execution. The main() method starts with public static void. Here public means that main() can be called from anywhere related with visibility of main() method, static means that main() doesn't belong to a specific object and void means that main() returns no value. The argument which is passed to main is string args[]. String[] is the type of the parameter, indicating an array of Strings.

Here string is a class name and args is the name of the parameter. Array args[] receives the command line arguments when the program runs.

Line 4 curly braces

The method definition should be within the curly brackets.

Line 5 prints "Welcome to Java world!"

System.out.println(...) is used to display a message on the console. This statement displays the string Welcome to Java! on the console. In C++ we are using cout<< object to print



something here we are using `System.out.println(...)` or `System.out.print(...)`. `System` is a final class from `java.lang` package. `out` is the reference of `PrintStream` class and a static member of `System` class. `println` is a method of `PrintStream` class.

Difference between `System.out.println(...)` and `System.out.print(...)` is after `println` method, the newline is invoked. That means next output if any is on the new line (effect of `"\n"`).

Line 6 curly braces

End of the `main()` method.

Line 7 curly braces

End of the class definition.

Step 2: To run Java Program written in Notepad or Notepad++, Open Command Prompt and Compile program, there are two steps to run any java program.

- 1) **javac fileName.java**
'javac' Compiler compiles source code to bytecode and generate class file
- 2) **java className**
'java' Interprets the bytecode stored in class file and executes the program

```
E:\DemoJava>javac j1.java
E:\DemoJava>java j1
Welcome to Java World!
E:\DemoJava>
```

1.9 ByteCode and JVM

Byte code

Byte codes are the machine language of the Java virtual machine. Java byte code is an intermediate language between machine code and Java.

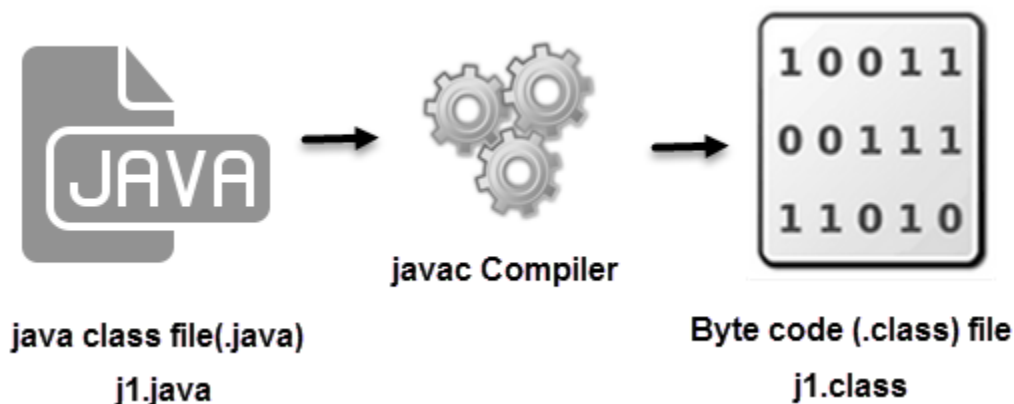
In previous topic, A Java compiler translates a Java source file "j1.java" into a Java bytecode file.



is PC > Local Disk (E:) > DemoJava			
Name	Date modified	Type	Size
j1	6/25/2016 3:23 PM	Java source file	1 KB

Before compile

The following command compiles j1.java: **javac j1.java.**



If there aren't any syntax errors, the compiler generates a bytecode file with a .class extension. Thus, the preceding command generates a file named **j1.class**.

The java interprets the bytecode stored in a class file and execute the program to generate output.

PC > Local Disk (E:) > DemoJava			
Name	Date modified	Type	Size
j1.class	6/25/2016 3:27 PM	CLASS File	1 KB
j1	6/25/2016 3:23 PM	Java source file	1 KB

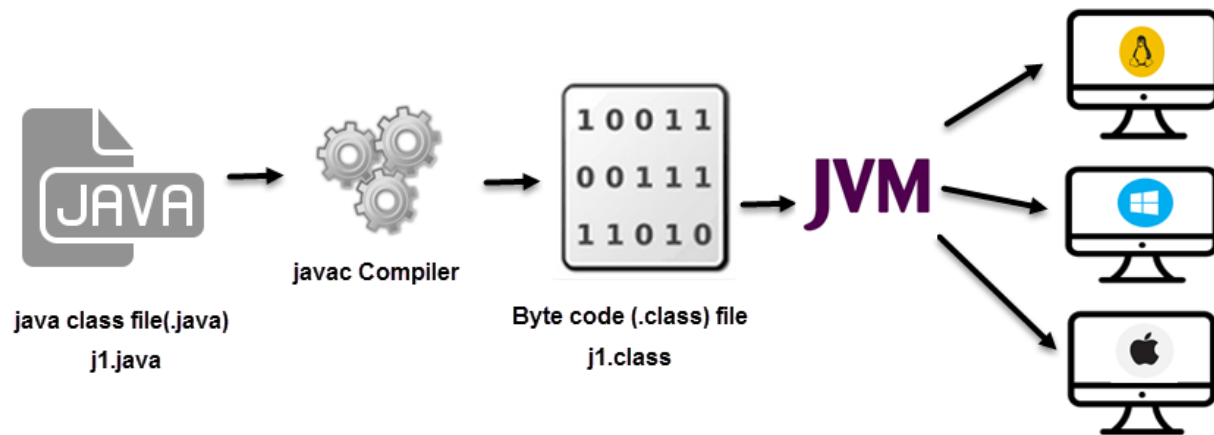
After compile

Bytecode stored in .class file to run on a Java virtual machine (JVM) instead of a central processing unit (CPU).

The bytecode is similar to machine instructions but is architecture neutral and can run on any platform that has a Java Virtual Machine (JVM). When a JVM loads a class file, it gets one stream of byte codes for each method in the class.

Byte code, also known as portable code (p-code), is a form of instruction set designed for efficient execution by a software interpreter

JVM



JVM converts byte code file (.class file) in output file.

A Java virtual machine (JVM) is a software that creates an environment between the computer and end user in which end user can operate programs.

A Java virtual machine enables a computer to run a Java program.

JVM is a platform-independent execution environment that converts Java bytecode into machine language and executes it.

The bytecode is generated by java compiler in a JVM understandable format. The Java virtual machine can generate an output corresponding to the underlying any operating system.

JVM does following task

- Garbage collection
- Guarantees platform independence by clearly defining the primitive data type:
- Network byte order

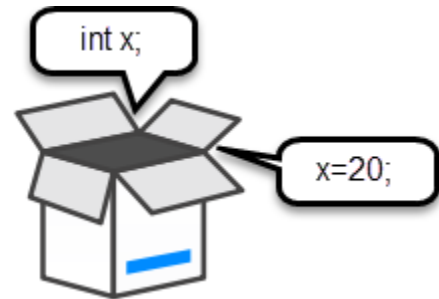
1.10 Some Facts of Java Program

- Every line of Java program should be end with ;
- Extension of Java program is ".java".
- The types of comments are same as C or C++ language.
 - Singline using //
 - Multiline using /* */
- A program can have one or more class definition and each class can have one or more data members and data functions.
- Java is case sensitive
- If we select netbeans or eclipse as an editor then there is no need to take two steps to runt java program that is javac compilation and java to run.
- If we select notepad or notepad++ that time we need command prompt and then we need to write java for compile the program and java to run.



1.11 Variable

- A variable is something that is used in a program to store data in memory.
- A variable is like a box and it contains some value.
- A variable has a name (the word you use to refer to the value the variable contains) and a data type (which determines the kind of data the variable can store).
- Data type gives meaning and capacity to variable.



Syntax#

Datatypesname variablename;

DataTypesname: The type of data that the variable can hold

VariableName: The variable we declare for hold the values.

Example#

```
int a;  
float b,c,d;
```

Variable Naming Convention

- Every variable name should start with either alphabets or underscore (_) or dollar (\$) symbol.
- The variable name cannot have a space
- Variable name Unique in its scope.
- Variable names are case-sensitive.
- Variable name must not be a keyword or reserved word.
- Except underscore (_) no special symbol are allowed in the middle of variable declaration
- Maximum length of variable is 64 characters.
- No keywords should access variable name.

Example# Variable Demo

Output View

```
E:\DemoJava>javac jVariableDemo.java  
E:\DemoJava>java jVariableDemo  
No = 10 m1 = 55  
Eng Marks = 33 A = 40
```

Coding View

```
class jVariableDemo  
{  
    public static void main(String s[])  
    {  
        int _no, m1, eng_mark,$a;  
        _no=10;  
        m1=55;  
        eng_mark=33;  
        $a=40;  
        System.out.println("No = " + _no + " m1 = " + m1 );  
        System.out.println("Eng Marks = " + eng_mark + " A = " + $a);  
    }  
}
```

Types of Variable

There are three types of variable in Java.



Local Variable

A variable which is between the opening and closing braces of a method means declared inside the method is called local variable.

Local variables are created when the method, constructor or block is entered and the variable will be destroyed once it exits the method, constructor or block.

Example#

```
void printAll()  
{  
    int x=5;  
    for(int i=1;i<=x;i++)
```



```
{  
    //Here i,x are local variables  
}  
}
```

Instance Variable

Datamembers of a class, and not declared as static.

Instance variable access outside a method, constructor or any block.

Instance variables are created when an object is created with the new key word and destroyed when the object is destroyed.

The value of instance variables are depends on the object. Each object have its unique values of instance variables.

Instance variables can be accessed by calling with the object name .
ObjectName.VariableName.

Example

```
class Stu  
{  
    int rno;  
    String name; //Here rno and name both are instance variable  
}  
  
Stu s1;  
s1.rno=5;
```

Class/Static variable

A variable that is declared as static is called static variable. It cannot be local.

It does not contain inside any object.

There would only be one copy of each class variable per class, regardless of how many objects are created from it. For example, we have created 10 objects then all of the objects can share same variable, value remains same for all the object.

Static variables are created when the program starts and destroyed when the program stops.

Static variables can be accessed by calling with the class name . ClassName.VariableName.

Example#

```
class Stu  
{  
    static int cnt; //Here cnt is static variable  
}
```





```
Stu.cnt=5;
```

1.12 Keywords

Any programming language have own set of reserved keywords and Java keywords are:

Abstract	continue	For	new	switch
assert***	default	goto*	package	synchronized
Boolean	Do	If	private	this
Break	double	implements	protected	throw
Byte	Else	import	public	throws
Case	Enum	instanceof	return	transient
Catch	extends	int	short	try
Char	Final	interface	static	void
Class	Finally	long	strictfp**	volatile
const*	Float	native	super	while

*not used **added in 1.2 ***added in 1.4 **** added in 5.0

Some Facts about keyword:

- We cannot create the variable name from above listed words.
- const and goto are reserved words but not used.
- true, false and null are literals, not keywords.
- All keywords are in lower-case.

Keywords group by Category

Category	Keywords
Access modifiers	private, protected, public
Class, method, variable modifiers	abstract, class, extends, final, implements, interface, native, new, static, strictfp, synchronized, transient, volatile
Flow control	break, case, continue, default, do, else, for, if, instanceof, return, switch, while
Package control	import, package
Primitive types	boolean, byte, char, double, float, int, long, short
Error handling	assert, catch, finally, throw, throws, try
Enumeration	enum
Others	super, this, void
Unused	const, goto





1.13 Data type

- 📌 The data type of a programming element refers to what kind of data it can hold and how that data is stored.
- 📌 Data type also can be identified as **Value type and Reference type**.
- 📌 **The difference between Value Types and Reference Types is the way of storing the values in memory.**
- 📌 The value of Value Type stores in Stack.
- 📌 The value of Reference Types store references to the actual data and stores in Heap.
- 📌 Example of Reference type is Class.

Value type's data type list are given below:

Data Type	Default Value	Default size	Range
Boolean	False	1 bit	True or false
Char	'\u0000'	2 byte	0 to 65,536
Byte	0	1 byte	-128 to 127
Short	0	2 byte	-32,768 to 32,767
Int	0	4 byte	2,147,483,648 to 2,147,483,647
Long	0L	8 byte	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
Float	0.0f	4 byte	1.4e - 045 to 3.4e + 038
Double	0.0d	8 byte	4.9 - 324 to 1.8e + 308

1.14 Escape Sequence

Java language supports few special escape sequences for String and char literals as well.

Notation	Meaning
\n	Newline
\r	Carriage return



\f	Formfeed
\b	Backspace
\s	Space
\t	Tab
\"	Double quote
\'	Single quote
\\	Backslash
\ddd	Octal character
\uxxxx	Hexadecimal UNICODE character

1.15 Operators

An operator is a symbol or character.

It allows us to manipulate data. An operator performs a function on one or more operands.

Example#

```
int ans;
```

```
ans=2+5
```

Here 2 and 5 are operands and + is an operator.

Operators that take one operand are called unary operators. For example, a=-5

Operators that take two operands are called binary operators. For example, a=5+9


Category of Operators

Operator	Operators
Arithmetic	+, -, *, \, %, ++, --
Assignment	=, +=, -=, *= , \= , %=



Comparison/Relational	<code>==, !=, <, >, <=, >=</code>
Logical	<code>&&, , !</code>
Bitwise operations	<code>&, , ^, ~, <<, >>, >>></code>
Miscellaneous operations	Ternary, instance of

1.15.1 Arithmetic operators

 Arithmetic operators are used in mathematical expression in the same way that are used in algebra.

Operator	Meaning	Example
<code>+</code>	Addition	<code>5+2 =7</code>
<code>-</code>	Subtraction	<code>5-2 =3</code>
<code>X</code>	Multiplication	<code>5*3 =15</code>
<code>/</code>	Regular Division	<code>20/2=10</code>
<code>%</code>	Modulus (Reminder)	<code>20%2=0</code>
<code>++</code>	Increment operator increases integer value by one	<code>a=5</code> <code>a++</code>
<code>--</code>	Decrement operator decreases integer value by one	<code>a=5</code> <code>a--</code>

Example# All Arithmetic operators

Output View

```
E:\DemoJava>javac jOperator.java

E:\DemoJava>java jOperator
A = 22 B = 2
Sum = 24
Sub = 20
Mul = 44
Div = 11
A = 23
B = 1
```

Coding View

```
class jOperator
{
    public static void main(String s[])
```



```
{
    int a=22;
    int b=2;
    System.out.println("A = " + a + " B = " + b);
    System.out.println("Sum = " + (a+b));
    System.out.println("Sub = " + (a-b));
    System.out.println("Mul = " + (a*b));
    System.out.println("Div = " + (a/b));
    a++;
    System.out.println("A = " + a);
    b--;
    System.out.println("B = " + b);
}
```

1.15.2 Assignment operators

Assignment operators are used to assign some value to the operand.

Operator	Meaning	Example
=	Assigns from one value to other one	a=5
+=	All are shorthand	a+=5
-=		a-=5
=		a=5
/=		a/=5
%=		a%=5

Example#

Output View

```
E:\DemoJava>javac jOperator2.java

E:\DemoJava>java jOperator2
A = 20
a = 30
a = 20
a = 200
a = 100
a = 0
```

Coding View

```
class jOperator2
```

```
{
public static void main(String s[])
{
    int a=20;
    System.out.println("A = " + a);
    a+=10;
    System.out.println("a = " + a);
    a-=10;
    System.out.println("a = " + a);
    a*=10;
    System.out.println("a = " + a);
    a/=2;
    System.out.println("a = " + a);
    a%=2;
    System.out.println("a = " + a);
}
}
```

1.15.3 Relational/Comparison operators

All of the relational operators result in a **Boolean** value.

Operator	Meaning	Example
==	Equality	if(a==b)
!=	Not equal	if(a<>b)
<	Less than	if(a<b)
>	Greater than	if(a>b)
<=	Less than equal to	if(a<=b)
>=	Greater than equal to	If(a>=b)

Example# All Relational operators

Output View

```
E:\DemoJava>javac jOperator3.java

E:\DemoJava>java jOperator3
a==b false
A <100 and a>b
A is not equal to 10
```

Coding View


```
class jOperator3
{
    public static void main(String s[])
    {
        int a=20,b=5;

        System.out.println("a==b " + (a==b));

        if(a>b && a<100)
        {
            System.out.println("A <100 and a>b");
        }

        if(a!=10)
        {
            System.out.println("A is not equal to 10");
        }
    }
}
```

1.15.4 Logical Operator

 Logical operators are used to combine more than one conditions. Usually Logical operators are used with if() condition and with relational operators.

Operator	Meaning	Example
&&	Logical and	(a>b && a>c)
	Logical or	(a>b a>c)
!	Logical not	!(a>b)

Example# All Logical operators

Output View

```
E:\DemoJava>javac jOperator4.java

E:\DemoJava>java jOperator4
A is max
a is > b
a>b || a>c
```

Coding View

```
class jOperator4
{
    public static void main(String s[])
```



```

{
    int a=20,b=5,c=12;

    if(a>b && a>c)
    {
        System.out.println("A is max");
    }

    if(!(b>a))
    {
        System.out.println("a is > b");
    }

    if(a>b || a>c)
    {
        System.out.println("a>b || a>c");
    }
}

```

1.15.5 BitWise Operator

Bitwise operators can be applied to the integer types long, int, short, char and byte

Operator	Meaning	Example
&	Bitwise AND	a & b
	Bitwise OR	a b
^	Bitwise exclusive OR	a ^ b
<<	left shift	a << b
>>	right shift	a >> b

Truth table for bitwise **&**, **|** and **^**

A	b	a & b	a b	a ^ b
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0



The bitwise shift operators shifts the bit value. The left operand specifies the value to be shifted and the right operand specifies the number of positions that the bits in the value are to be shifted.

Example# All Bitwise operators

Output View

```
E:\DemoJava>javac jOperator5.java

E:\DemoJava>java jOperator5
a&b = 0
a!b = 6
a^b = 6
no = 8
value of no after left shift: 16
value of no after right shift with sign: -4
```

Coding View

```
class jOperator5
{
    public static void main(String s[])
    {
        int a = 2; //0010
        int b = 4; //0100
        int c=a&b;
        System.out.println("a&b = " + c); // and condition = 0000
        c=a|b;
        System.out.println("a!b = " + c); // or condition = 0110
        c=a^b;
        System.out.println("a^b = " + c); // ^(a and b)

        int no = 8; //0000 1000
        System.out.println("no = " + no);

        //left shifting bytes with 1 position
        no = no<<1; //should be 16 i.e. 0001 0000

        //equivalent of multiplication of 2
        System.out.println("value of no after left shift: " + no);

        no = -8;
        //right shifting bytes with sign 1 position
        no = no>>1; //should be 16 i.e. 0001 0000

        //equivalent of division of 2
        System.out.println("value of no after right shift with sign: " + no);
    }
}
```

1.15.6 Miscellaneous Operator

1.15.6.1 instanceof Operator

- In Java, instanceof operator is used to check the type of an object at runtime.
- instanceof operator checks whether the object is of a particular type(class type or interface type).
- instanceof operator is also known as type comparison operator because it compares the instance with type.
- It returns either true or false.

Syntax#

(Object reference variable) instanceof (class/interface type)

Example# instanceof Operator

Output View

```
E:\DemoJava>javac jOperator6.java

E:\DemoJava>java jOperator6
s1 instance of Stu = true
s2 instance of Stu (which is null) = false
College instance of String = true
```

Coding View

```
class stu
{
int sno;
String sname;
};

class jOperator6
{
public static void main(String s[])
{
stu s1=new stu();
stu s2=null;


String college="LJ";

System.out.println("s1 instance of Stu = " + (s1 instanceof stu));
System.out.println("s2 instance of Stu (which is null) = " + (s2 instanceof stu));
System.out.println("College instance of String = " + (college instanceof String));
```




```
}  
}
```

1.15.6.2 Ternary Operator

 The ternary operator ":" is an operator to take three operands. It is a conditional operator that provides a shorter syntax for the if..then..else statement.

Syntax # value=(expression? value1:value2);

 The first operand is a boolean expression; if the expression is true then the value of the second operand is returned otherwise the value of the third operand is returned.

Example# Ternary Operator

Output View

```
E:\DemoJava>javac jOperator7.java  
  
E:\DemoJava>java jOperator7  
12 is max
```

Coding View

```
class jOperator7  
{  
    public static void main(String s[])  
    {  
        int a=12;  
        int b=3;  
  
        int ans=a>b?a:b;  
  
        System.out.println(ans + " is max");  
    }  
}
```

1.15.7 Precedence of Operator

The precedence of operator defines the step of execution of expression or which operation must be done first during the expression evaluation.

Example# $(1+2) * 3$ is not equal to $1+2*3$, value of the first expression is 9 and second one is 7.





Operator	Meaning
[]	access array element
.	access object member
()	invoke a method
++	post-increment
--	post-decrement
++	pre-increment
--	pre-decrement
+	unary plus
-	unary minus
!	logical NOT
~	bitwise NOT
()	cast
new	object creation
*	Multiplicative
/	
%	
+ -	additive
+	string concatenation
<< >>	Shift
>>>	
< <=	relational
> >=	type comparison
instanceof	
==	Equality
!=	
&	bitwise AND
^	bitwise XOR
	bitwise OR
&&	conditional AND
	conditional OR
?:	Conditional
= += -=	Assignment
*= /= %=	
&= ^= =	
<<= >>= >>>=	

1.16 Control statement

We can control the order of execution of the program, based on logic and values.

In the sequential flow of control statements are executed line by line, but in the case of conditional program the order of execution are based on data values and conditional logic.

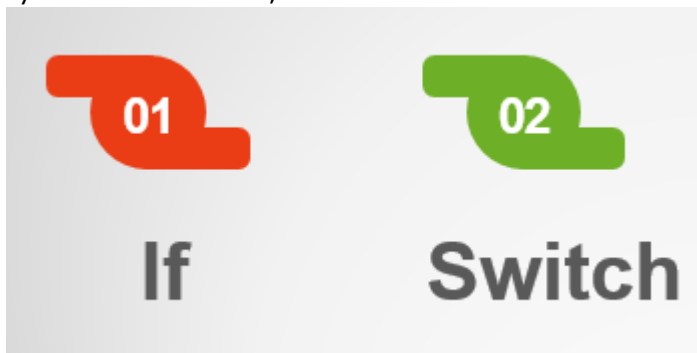


The control statements can be classified into two groups: Selection statements , Iteration statements and Branching statements.

1.16.1 Selection Statement

Selection statement is also known as conditional or decision making statement.

There are two ways to take decision,



1.16.1.1 Type of if statements

- There are cases when we would like to execute some logic when a condition is TRUE, and some other logic when the condition is FALSE.
- The condition usually results from a comparison of two values.
- The condition is an expression that is made up by using relational operators (>,<,>=,<=,==,etc) and logical operators (&& , || ,etc) and returns true or false.
- There are 4 ways to write if statements as below given image.





1.16.1.1.2 If

- There are cases when we would like to execute some logic when a condition is TRUE.
- The condition usually results from a comparison of two values.
- If the condition is TRUE then the control goes to between **if**.

Syntax#

if(condition)

{

Logic..

}

Example# Check whether number is 5 .

Output View

```
E:\DemoJava>javac if1.java
E:\DemoJava>java if1
Yes the value of A = 5
```

Coding View

```
class if1
{
    public static void main(String s[])
    {
        int a;
        a=5;
        if(a==5)
        {
            System.out.println("Yes the value of A = " + a);
        }
    }
}
```

1.16.1.1.3 If...else

- There are cases when we would like to execute some logic when a condition is TRUE, and some other logic when the condition is FALSE.
- If the condition is TRUE then the control goes to between if and else block, that is the program will execute the code between if and else statements, else program will execute the code after else.



**Syntax#**

```
if(condition)
{
    Logic..
}
else
{
    Logic..
}
```

Example# Check whether number is Positive or Negative


Output View

```
E:\DemoJava>javac if2.java
E:\DemoJava>java if2
No is positive
```

Coding View

```
class if1
{
    public static void main(String s[])
    {
        int a;
        a=5;
        if(a==5)
        {
            System.out.println("Yes the value of A = " + a);
        }
    }
}
```

1.16.1.1.4 Nested If...else

 There are cases when we would like to execute some more condition when a condition is TRUE, and some other condition logic when the condition is FALSE.

Syntax#

```
if(condition)
{
    if(condition)
    {
    }
}
else
{
    if(condition)
    {
    }
}
```



```
Else
{
```

```
}
```

```
}
```

Example# Max between 3

Output View

```
E:\DemoJava>javac if3.java
```

```
E:\DemoJava>java if3
B is max
```

Coding View

```
class if3
{
    public static void main(String s[])
    {
        int a=11,b=33,c=22;
        a=5;
        if(a>b)
        {
            if(a>c)
                System.out.println("A is max");
            else
                System.out.println("C is max");
        }
        else
        {
            if(c>a)
                System.out.println("B is max");
            else
                System.out.println("C is max");
        }
    }
}
```

1.16.1.1.5 If..else else if

- There are cases when we would like to execute some more condition when a condition is TRUE, and some other condition logic when the condition is FALSE.

Syntax#

```
if(condition)
{
}
else if(condition)
{
}
```



```
}  
else  
{  
  
}
```

Example# Odd Even

Output View

```
E:\DemoJava>javac if4.java  
  
E:\DemoJava>java if4  
No is even
```

Coding View

```
class if4  
{  
    public static void main(String s[])  
    {  
        int a=20;  
  
        if(a%2==0)  
        {  
            System.out.println("No is even");  
        }  
        else if(a%2!=0)  
        {  
            System.out.println("No is odd");  
        }  
        else  
        {  
            System.out.println("No is 0");  
        }  
    }  
}
```

Some good Examples of if-else:

String compare	String str="Shyam"; if(str.equals("Shyam")) { Logic.. }
----------------	---



Character is in uppercase	<pre>char ans='A'; if(Character.isUpperCase(ans)) { Logic.. }</pre>
Total range	<pre>int a=22,b=33,c=55; if((a+b+c)>0 && (a+b+c)<50) { Logic.. }</pre>

1.16.1.2 Switch-case

- When you are comparing the same expression to several different values, we can use the switch-case statements as an alternative to the if-else statements.
- It executes one of several groups of statements depending on the value of an expression.
- If-else block executes different condition or expression in each statement, the switch statement evaluates a single expression and compares it for every comparison.
- If any case **statement** is satisfied logic will run that only and then control will come out of **switch** block. This one will not happen with if-else block.
- Remember that switch block used with a break statement at the end of each case. C# simply requires that we leave the block before it ends.
- If-else block are time consuming then switch case block.

Syntax#

Switch(testexpression)

```
{
case expression1:
    Logic..
    break;
case expression2:
    Logic..
    break;
case expression N:
    Logic..
    break;
default:
    break;
}
```



Example# Demo of Switch Case

Output View

```
E:\DemoJava>javac Switch1.java
E:\DemoJava>java Switch1
Stop the car
```

Coding View

```
class Switch1
{
public static void main(String s[])
{
    String color="red";

    switch(color)
    {
        case "red":
            System.out.println("Stop the car");
            break;
        case "green":
            System.out.println("Drive the car");
            break;
        case "orange":
            System.out.println("Wait");
            break;
        default:
            System.out.println("wrong opt");
    }
}
}
```

Some good Examples of switch-case:

<pre>int no=5; switch(no) { case 1: logic.... break; case 2: logic.... break; case 10: logic.... break;</pre>	<pre>char a='e'; switch (a) { case 'a': case 'e': case 'i': case 'o': case 'u': logic.. break; default:</pre>	<pre>switch (op) { case '+': logic.... break; case '-': logic.... break; case '*': logic.... break; case '/':</pre>
---	--	---

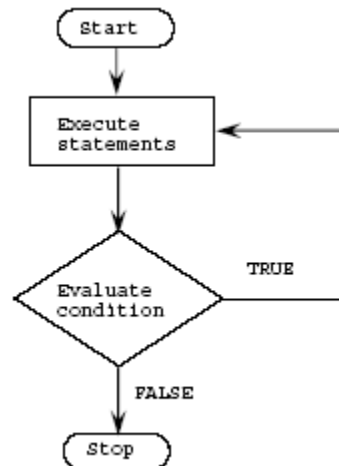


default: logic.... break; }	logic... break; }	logic.... break; default: logic.... break; }
---	-------------------------	--

1.16.1.3 Iteration Statements

Iteration statements is also known as a Looping statements. Looping structures are used when a group of statements is to be executed repeatedly, until a condition is TRUE or until a condition is FALSE.

The following illustration shows a loop structure that runs a set of statements until a condition becomes true.



Java supports the following loop structures.





1.16.1.3.1 For loop

- The for loop executes a block of statements a specified number of times.
- It's most commonly and most popular loop used in any programming language.

Syntax#

```
for(initialization; condition; step)
{
    Logic....
}
```

Example# Print 0 to 9

Output View



```
E:\DemoJava>javac loop1.java
```

```
E:\DemoJava>java loop1
```

```
1
2
3
4
5
6
7
8
9
10
```

Coding View

```
class loop1
{
    public static void main(String s[])
    {
        int i;

        for(i=1;i<=10;i++)
        {
            System.out.println(i);
        }
    }
}
```

Example# Table

Output View



```
E:\DemoJava>javac loop2.java
```

```
E:\DemoJava>java loop2
```

```
7 X 1 = 7
7 X 2 = 14
7 X 3 = 21
7 X 4 = 28
7 X 5 = 35
7 X 6 = 42
7 X 7 = 49
7 X 8 = 56
7 X 9 = 63
7 X 10 = 70
```

Coding View

```
class loop2
{
    public static void main(String s[])
    {
        int i,no=7;

        for(i=1;i<=10;i++)
        {
            System.out.println(no + " X " + i + " = " + (no*i));
        }
    }
}
```

Some good examples:

<pre>for (int i = 1; i < 20; i = i + 5) { Logic..... }</pre>	<pre>for (int i = 20; i >=0 ; i --) { Logic..... }</pre>
<pre>for (int i = 1; i <=5 ; i ++) { for (int j = 1; j <= 5; j++) { Logic..... } }</pre>	<pre>for (int i = 1; i <=5 ; i ++) { for (int j = 1; j <= 5; j++) { for (int k = 1; k <= 5; k++) { Logic..... } } }</pre>

1.16.1.3.2 While loop

- While loop keeps executing until the condition against which it tests remain true.
- The While statement always checks the condition before it begins the loop.
- In While loop programmer have to maintain or keep track of increment or decrement value.
- An entry controls loop checks the condition first and then enters the loop body. So for...and while..both are known as a entry controls loop.

Syntax#

```
while(upto condition)
{
    logic....
}
```

Example# while

Output View

```
E:\DemoJava>javac loop3.java
E:\DemoJava>java loop3
1 == 1
2 == 4
3 == 9
4 == 16
5 == 25
```

Coding View

```
class loop3
{
    public static void main(String s[])
    {
        int no=1;
        while (no <= 5)
        {
            System.out.println(no + " == " + (no*no));
            no++;
        }
    }
}
```


}

1.16.1.3.3 Do while loop

- Repeats a block of statements while a Boolean condition is FALSE or until the condition becomes TRUE.
- Do while loop makes sure that the code block is always executed at least once.
- Usually we used switch case inside do while loop.
- In Do.. While loop programmer have to maintain or keep track of increment or decrement value.
- Do..while loops is known as an exit controlloed loop. An exit controlled loop checks the condition at the end of the loop.Hence even if the condition is false the loop will be executed once.

Syntax#

```
do
{
    logic...;
}while(upto condition);
```

Example# do...while

Output View

```
E:\DemoJava>javac loop4.java



E:\DemoJava>java loop4
Square of 1 is 1
Square of 2 is 4
Square of 3 is 9
Square of 4 is 16
```

Coding View

```
class loop4
{
    public static void main(String s[])
    {
        int no=1;
        do{
            System.out.println("Square of "+ no +" is " + (no*no));
            no++;
        }while (no <=4);
    }
}
```



1.16.1.3.4 For each loop

-  The foreach loop is similar to the for loop, but it executes the statement block for each element in a collection or array.
-  Repeats a group of statements for each element in a collection.

Syntax#

```
foreach (datatype variablename : groupname)
{
    logic.....
}
```

Example# for...each

Output View

```
E:\DemoJava>javac loop5.java
E:\DemoJava>java loop5

Now collection
Mayur
Aalok
Pooja

Now array
2
23
55
77
88
```

Coding View

```
import java.util.*;

class loop5
{
    public static void main(String s[])
    {
        ArrayList<String> friends=new ArrayList<String>();
        friends.add("Mayur");
        friends.add("Aalok");
        friends.add("Pooja");
    }
}
```



```
System.out.println("\nNow collection");

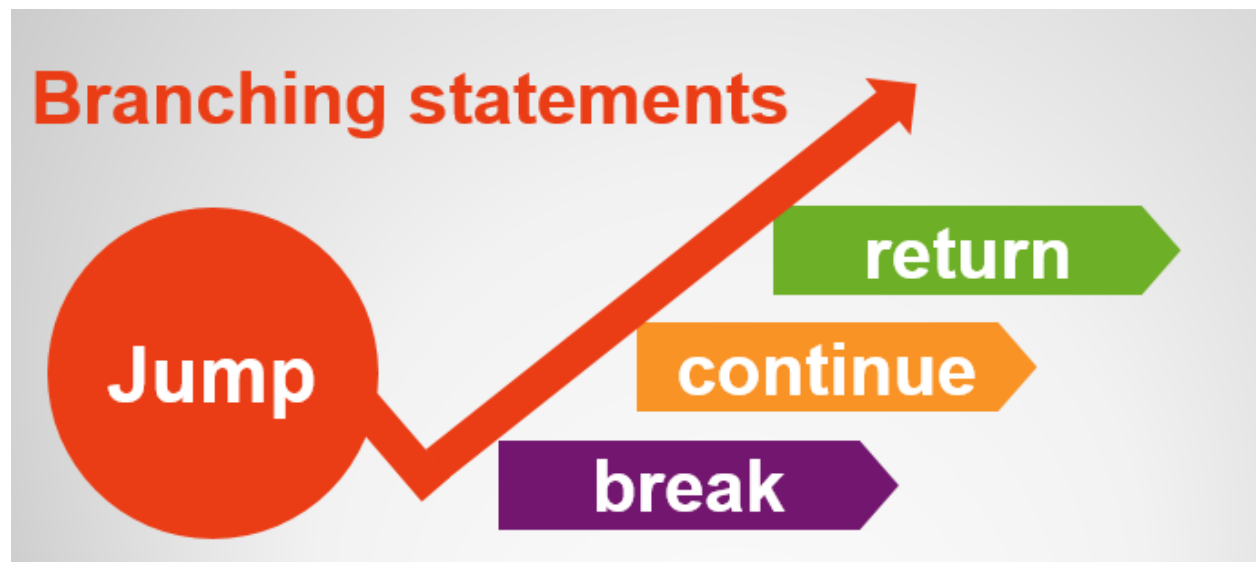
for(String str:friends){
    System.out.println(str);
}

System.out.println("\nNow array");
int arr[]={2,23,55,77,88};

for(int no:arr){
    System.out.println(no);
}
}
```

1.16.1.4 Branching Statement


Java's branching statements also known as jumping statements. Jumping statements jumps from one condition to other conditionally or uncordially.




1.16.1.4.1break

 The break statement is used to stop current the execution and comes out of loop.





 In many situations we need to get out of the loop before the loop execution is complete.

 There are two types of break

1. unlabeled
2. labeled

Unlabeled break


 We see the unlabeled form in the topic of the switch statement.

 Unlabeled break statement is used to get out of a single loop and for the case of inside the nested loops it will exit only from the loop in which break is used.

Syntax# unlabeled

Break;

Labeled break

 Labeled breaks statement is used when we need to break the nested loops.

Syntax# labeled

Placename:

Break placename;

Example# unlabeled break

Output View

```
E:\DemoJava>javac break1.java
E:\DemoJava>java break1
No is prime
E:\DemoJava>
```

Coding View

```
class break1
{
    public static void main(String s[])
    {
        int no=22,divide=0;
        for(int i=2; i<no ; i++)
        {
            if(no%i==0)
```





```
        {
            divide=1;
            break;
        }
    }
    if(divide==0)
    {
        System.out.println("No is prime");
    }
    else
    {
        System.out.println("No is not prime");
    }
}
```

Example# labeled break

Output View

```
E:\DemoJava>javac break2.java

E:\DemoJava>java break1
1 1
1 2
1 3
2 1
```

Coding View

```
class break1
{
    public static void main(String s[])
    {
        int no=3;
        outer:
        for(int i=1;i<=no;i++)
        {
            for(int j=1; j<=no ; j++)
            {
                System.out.println(i + " " + j);
                if(i==2)
                {
                    break outer;
                }
            }
        }
    }
}
```



1.16.1.4.2 Continue

- Continue is similar to break but when break statement executes it comes out of loop where as continue comes out of loop and jumps to the conditional statement of loop.
- It executes the loop to immediately jump to the next iteration of the loop means it skips the Loop and Re-Executes Loop with new condition.
- We can use Continue statment with For Loop. | While Loop. | do-While Loop.
- There are two types of continue
 - unlabeled
 - labeled

Unlabeled continue

- Unlabeled continue statement is used to get out of a single loop and for the case of inside the nested loops it will exit only from the loop in which continue is used.

Syntax# unlabeled
Break;

Labeled continue

- Labeled continue statement is used when we need to continue the nested loops.

Syntax# labeled

Placename:
Break placename;

Example# unlabeled continue

Output View

```
E:\DemoJava>javac continue1.java
E:\DemoJava>java continue1
1 1
1 2
1 3
2 1
2 2
2 3
3 1
3 2
3 3
```



Coding View

```
class continue1
{
    public static void main(String s[])
    {
        int no=3;
        for(int i=1;i<=no;i++)
        {
            for(int j=1; j<=no ; j++)
            {
                System.out.println(i + " " + j);
                if(i==2)
                {
                    continue;
                }
            }
        }
    }
}
```

Example# labeled continue

Output View

```
E:\DemoJava>javac continue2.java

E:\DemoJava>java continue2
1 1
1 2
1 3
2 1
3 1
3 2
3 3
```

Coding View

```
class continue2
{
    public static void main(String s[])
    {
        int no=3;
        outer:
        for(int i=1;i<=no;i++)
        {
            for(int j=1; j<=no ; j++)
            {
                System.out.println(i + " " + j);
                if(i==2)
                {
                    continue outer;
                }
            }
        }
    }
}
```



```
        continue outer;
    }
}
}
```

1.16.1.4.3 return

- 📄 We specify return in the method declaration.
- 📄 Default return type of any method is void, means the method with define void, it does not return any value.
- 📄 Return statement decide control flow of the method. The return statement is used to explicitly return from a method.
- 📄 Usually we write return statement at the end of the any method.
- 📄 When we want to return something from the method to the caller that time we can use.
- 📄 Return statement stops execution of the method and transfer the control back to the caller of the method.
- 📄 The return statement can be used in two ways, return statement that returns a value and another that does not return a value.

Syntax#

```
public static <return type> methodName(arguments) {
    // body
}
```

Example# Return

Output View

```
E:\DemoJava>javac return1.java
E:\DemoJava>java return1
Add = 29
```

Coding View

```
class return1
{
    static int add(int a,int b)
    {
        return a+b;
    }
    public static void main(String s[])
    {
    }
```



```
{
    int a,b,ans;
    a=22;
    b=7;
    ans=add(a,b);
    System.out.println("Add = "+ (ans));
}
```

How to Scan value from the user?

Java uses System.out to refer to the standard output device and System.in to the standard input device.

Instead of assigning static values to the variable we can use Scanner class for console input.

How to use Scanner class?

- 1) Import - import java.util.*;
- 2) Create an object of Scanner class - Scanner s1=new Scanner(System.in);
- 3) Use method of scanner class to scan something



Method	Meaning
nextByte()	reads an integer of the byte type.
nextShort()	reads an integer of the short type.
nextInt()	reads an integer of the int type.
nextLong()	reads an integer of the long type.
nextFloat()	reads a number of the float type.
nextDouble()	reads a number of the double type.
next()	reads a string that ends before a whitespace character.
nextLine()	reads a line of text

Example# Scan the values

Output View

```
E:\DemoJava>javac jValue.java

E:\DemoJava>java jValue
Enter a =>22
Enter b =>3
Add = 25
```

Coding View

```
import java.util.*;
```

```
class jValue
{
```



```
public static void main(String s[])
{
    int a,b;

    Scanner s1=new Scanner(System.in);

    System.out.print("Enter a =>");
    a=s1.nextInt();

    System.out.print("Enter b =>");
    b=s1.nextInt();

    System.out.println("Add = "+ (a+b));
}
}
```

