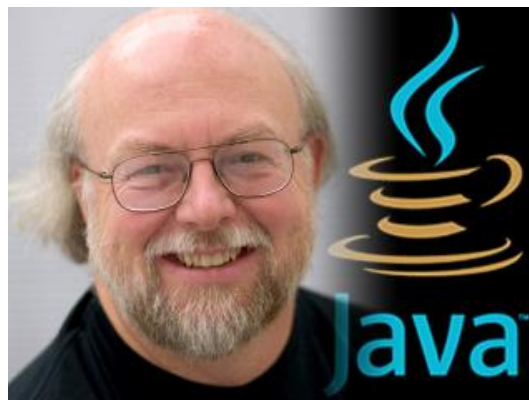


Java History

Java is a purely object-oriented programming language, originally developed by James Gosling, Patrick Naughton, Chris Warth, Ed Frank and Mike Sheridan at Sun Microsystems of USA in 1991. It was originally called “Oak” but was renamed “Java” in 1994. It was designed for the development of software for consumer electronic devices like TV, VCR, Toasters, Cable set top boxes, PDA and other electronic machines. It was originally meant to be a programming language for embedded systems. The Language was created with 5 main goals:

1. It should be object oriented.
2. A single representation of a program could be executed on multiple operating systems.
3. It should fully support network programming.
4. It should execute code from remote sources securely.
5. It should be easy to use.



James Gosling

Java acronym for the names of the team members: **J**ames Gosling, **A**rthur Van Hoff, and **A**ndy Bechtolsheim.

Java Milestones

Year	Development
1990	Sun Microsystems decided to develop special software that could be used for electronic devices. A project called Green Project created and head by James Gosling.
1991	Explored possibility of using C++, with some updates announced a new language named “Oak”
1992	The team demonstrated the application of their new language to control a list of home appliances using a hand held device with a tiny touch sensitive screen.
1993	The World Wide Web appeared on the Internet and transformed the text-based interface to a graphical rich environment. The team developed Web applets (time programs) that could run on all types of computers connected to the Internet.
1994	The team developed a new Web browser called “Hot Java” to locate and run Applet programs on internet. Hot-Java gained instance success.
1995	Oak was renamed to Java, as it did not survive “legal” registration. Many companies such as Netscape and Microsoft announced their support for Java
1996	Java established itself as both 1 1. The language for Internet programming 2. A general purpose Object Oriented Programming language.
1997	A class libraries, Community effort and standardization, Enterprise Java, Clustering, etc. Sun releases Java Development Kit 1.1 (JDK 1.1)
1998	Sun releases Java-2 of the software Development Kit (SDK 1.2)
1999	Sun releases Java-2 platform, Standard Edition (J2SE) and Enterprise Edition (J2EE)
After 2000	Sun releases J2SE with SDK 1.3, 1.4 J2SE with JDK 5.0

Q. What are the features of Java? Or State any four features of Java

- **Java is Compiled and Interpreted language:** Java is a two stage system: In a first stage java compiler translates source code into byte code instructions. Byte codes are not machine instructions. In a second stage, Java interpreter generates machine code that can be directly executed by the machine. The two steps of compilation and interpretation allow for extensive code checking and improved security.
- **Java is Platform-Independent and Portable language:** The feature "Write-once-run-anywhere" (known as the Platform independent) states that programs written on one platform can run on any platform provided the platform must have the JVM. Java programs can be easily moved from one computer system to another, anywhere and anytime. Changes and upgrades in operating systems, processors and system resources will not force any changes in Java programs. Due to this reason Java has become a popular language for programming on Internet. We can download a Java applet from remote computer onto our local system via Internet and execute it locally.
- **Java is an Object Oriented Language:** Everything in java is an object. All program code and data reside within objects and classes. Java comes with an extensive set of classes, arranged in packages that can use in our programs by inheritance. As the languages like Objective C, C++ fulfills the four characteristics (Inheritance, Encapsulation, Polymorphism and Dynamic Binding) yet they are not fully object oriented languages because they are structured as well as Object Oriented Languages. But in case of java, it is a fully Object Oriented Language because object is at the outer most level of data structure in java. No stand alone methods, constants, and variables are there in java. Everything in java is object even the primitive data types can also be converted into object by using the wrapper class.
- **Java is Robust and Secure language:** Java provides many safeguards to ensure reliable code. Java has the strong memory allocation and automatic garbage collection mechanism. It provides the powerful exception handling and type checking mechanism as compared to other programming languages. Compiler checks the program whether there are any errors and interpreter checks any run time errors and makes the system secure from crash. All of the above features make the java language robust. With Java secure feature it enables to develop virus-free, tamper-free systems used in internet. The absence of pointers in Java ensures that programs cannot gain access to memory locations without proper authorization. Authentication techniques are based on public-key encryption.
- **Java is Distributed language:** Java is designed for the distributed environment of the internet. The widely used protocols like HTTP and FTP are developed in java. Internet programmers can call functions on these protocols and can get access the files from any remote machine on the internet rather than writing codes on their local system.
- **Java is Familiar, Simple and Small language:** Java is designed to be easy to learn. If you understand the basic concept of OOP java would be easy to master. There are various features that make the java as a simple language. Programs are easy to write and debug because java does not use the pointers explicitly. It is much harder to write the java programs that can crash the system but we cannot say about the other programming languages. Java provides the bug free system due to the strong memory management. It also has the automatic memory allocation and deallocation system. It also eliminates operator overloading and multiple inheritance. Java uses many Constructs of C and C++ and therefore Java code "looks like a C++" code. Thus Java is Simplified version of C++.
- **Java is Multithreaded and Interactive language:** Multithreading means a single program having different threads executing independently at the same time. Java supports multithreaded programs. Multiple threads execute instructions according to the program code in a process or a program. Multithreading works the similar way as multiple processes

run on one computer. This design feature allows developers to construct smoothly running interactive applications.

- **Java is High Performance language:** Interpretation of byte codes slowed performance in early versions, but advanced virtual machines with adaptive and just-in-time compilation and other techniques now typically provide performance up to 50% to 100% the speed of C++ programs.
- **Java is Dynamic:** By connecting to the Internet, a user immediately has access to thousands of programs and other computers. During the execution of a program, Java can dynamically load classes that it requires either from the local hard drive, from another computer on the local area network or from a computer somewhere on the Internet.
- **Architecture Neutral:** The Java compiler generates an architecture-neutral object file format to enable a Java application to execute anywhere on the network and then the compiled code is executed on many processors, given the presence of the Java runtime system. Hence Java was designed to support applications on network. This feature of Java has thrived the programming language.
- **Java is popular for Internet:** The main reason why Java is popular on the Internet is that Java offers applets. Applets are tiny programs which run inside the browser. Before the introduction of applets, the web pages were mostly static. After the use of this application, they have become dynamic, making browsing richer. Applets are very safe. They cannot write to the local hard disk. Hence, there is no threat of viruses while surfing the Internet. This has made applets (and thereby Java) highly popular on the Internet. They also support graphical user interface (GUI). Hence, surfing the Internet can be quite enjoying when web pages use applets.

Q. Differentiate between Java and C++

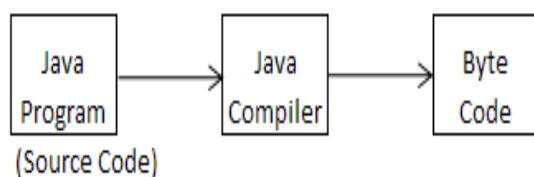
Java	C++
Java is a true object oriented language.	C++ is basically C with object oriented extension.
Java does not support operator overloading.	C++ support operator overloading.
Java does not have template classes.	C++ has template classes.
Java does not support multiple inheritances of classes. This is accomplished using a new feature called "interface".	C++ supports multiple inheritances of classes.
Java does not support global variables. Every variable and method is declared within a class and forms part of that class.	C++ support global variables.
Java does not use pointers.	C++ uses pointers.
Java has replaced the destructor function with a finalize() function.	C++ has the destructor function.
There are no header files in java.	C++ includes header file in program.
Java supports special methods to convert values between class objects and primitive types.	C++ does not support this.
Java does not support typedef keyword.	C++ support typedef keyword.
Java adds a new right shift operator >>> which inserts zeros at the top end.	C++ supports only >> and <<.
Operator overloading is not possible in java. + Operator can be used to concatenate strings.	Operator overloading is possible in C++.
Constants can be created using the final modifier when declaring class and instance variables.	Constants can be created using const modifier.
There are no scope resolution operator :: in java.	C++ has scope resolution operator ::
There is no virtual keyword in java.	C++ has virtual keyword.

Java	C++
Strings in java are objects. They are not terminated by a null.	Strings in C and C++ are arrays of characters, terminated by a null character.
Java does not support multidimensional array. It is possible to create arrays of arrays to represent multidimensional arrays.	C/C++ support multidimensional arrays.
There is multi-thread support.	There is support for multithreading.
Java support automatic garbage collection and makes a lot of programming problems simply vanish.	There is no garbage collection.
In java the destructor function is replaced with a finalize function.	C++ uses destructor function.
Write Once Run Anywhere/Everywhere (WORA/WORE).	Write Once Compile Anywhere (WOCA).
Call through the Java Native Interface and recently Java Native Access.	Allows direct calls to native system libraries.
Runs in a protected virtual machine.	Exposes low-level system facilities.
Primitive and reference data types always passed by value.	Pointers, References, and pass by value are supported.
Javadoc standard documentation.	No standard inline documentation mechanism. 3rd party software (e.g. <u>Doxygen</u>) exists.
Supports labels with loops and statement blocks.	Supports the goto statement.

Java Components

- The byte code
- Java Development Kit (JDK)
- Java Virtual Machine (JVM)

The Byte Code: When java program is compiled, then java compiler produces an intermediate code for a machine that does not exist instead of machine code. This intermediate form of code is known as byte code. By Java interpreter we can interpret this Byte code on any machine and run on any machine. The file which contains this code has **.class** extension; e.g. if we compile a file Prog1.java then after successful compilation Prog1.class containing byte code will be created. It is not machine specific. Byte code is computer object code that is processed by a program, usually referred to as a virtual machine, rather than by the "real" computer machine, the hardware processor. The virtual machine converts each generalized machine instruction into a specific machine instruction or instructions that this computer's processor will understand. Byte code is the result of compiling source code written in a language that supports this approach.



- The intermediate form of java program is known as byte code.
- The file which contains byte code has **.class** extension.

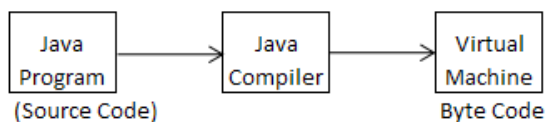
Java Development kit (JDK): The Java Development Kit comes with a collection of tools that are used for developing and running Java programs. The development tools are part of the system known as **Java Development Kit (JDK)** and the classes and methods are part of the **Java Standard Library (JSL)**, also known as the **Application Programming Interface (API)**.

Java Development kit (JDK) includes following tools:

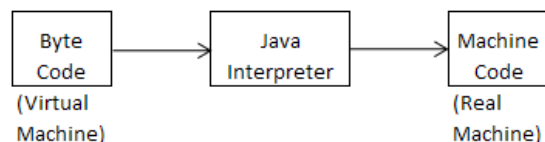
Tool	Description
Javac (Java compiler)	It is java compiler and used to compile java program. It translates java source code to byte code files that the interpreter can understand.
Java (Java interpreter)	It is java interpreter and used to run an applet and application program by reading and interpreting byte code files.
Appletviewer (view java applet)	it enables us to run java applets (without actually using a java-compatible browser).
Javah (C header files)	It generates C header files and source files that are needed to implement native methods.
Javap (Java disassembler)	It is java disassembler and used to convert byte code files into a program description.
Jdb (java debugger)	It is used to find errors in our programs
Javadoc (java document for creating HTML documents)	It is used to create HTML-format documentation from java source code files

Java Virtual Machine (JVM):

All language compilers translate source code into *machine code* for a specific computer. But, Java compiler produces an intermediate code known as **byte code** for a machine that does not exist. This machine is called the **Java Virtual Machine (JVM)** and it exists only inside the computer memory. It is a simulated computer within the computer and does all major functions of a real computer. The virtual machine code is not machine specific. The machine specific code (known as machine code) is generated by the Java interpreter by acting as an intermediary between the virtual machine and the real machine. The interpreter is different for different machines.



1. Process of compilation

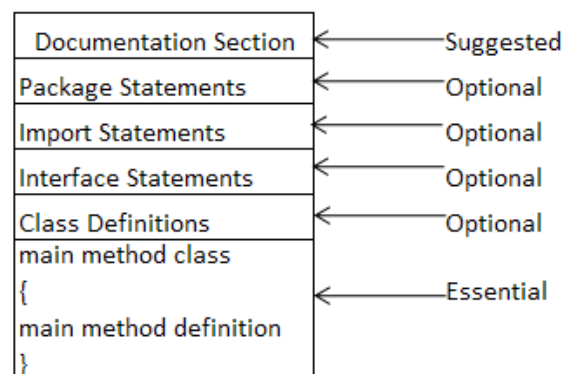


2. Process of converting byte code into machine code

Java Program Structure

A java program may contain many classes of which only one class defines a main method. A class contains data members and methods that operate on the data members of the class. Methods may contain data type declarations and executable statements. A java program may contain one or more sections

- **Documentation Section:** Documentation section comprises of a set of lines giving the name of the program, author name and other details. Comments like `//` or `/* ...*/` and `/** ...*/` known as documentation comment.



Structure of Java Program

- **Package Statement:** Package statement declares package name and informs the compiler that the classes defined here belong to this package. For example *package college;*
- **Import Statement:** This statement is similar to the #include statement in C. This statement instructs the interpreter to load the classes contained in the package. For example *import java.lang.*;*
- **Interface Statement:** An interface like a class but includes a group of method declarations. This is an optional section and is used only when we wish to implement the multiple inheritance features in the program.
- **Class Definition:** Classes are the primary and essential elements of a java program. The number of classes used depends on the complexity of the program.
- **Main method class:** Every java program requires a main method. The main method creates objects of various classes and establishes communications between them. On reaching the end of main, the program terminates and the control passes back to the operating system.

Simple Java Program

```
Class Test
{
Public static void main(String args[])
{
System.out.println("Java is better than C++");
}
}
```

There are three permissible styles for inserting comments.

- *// comment on one line*
- */* comment on one or more lines */*
- */** documentation comment */*

- **Class Declaration:** Java is true object-oriented language so everything must be placed inside a class. Class is a keyword to define a new class.
- **Opening Brace:** The class definition must begin with opening curly brace ({}) and ends with closing curly brace ({}). The rest of the things defined inside these braces are called member of the class
- **The main line:** The *public static void main(String args[])* defines a method named main. The program will start execute by calling this main method. All the java program will start execute by calling the main method. This line contains a number of keywords, public, static and void.
- **Public:-**The keyword public is an access specifier that declares the main method as unprotected and therefore making it accessible to all other classes. The main() must be declared as public because it must be called by code outside of its class when the program is started.
- **Static:-**It declares this method as one that belongs to the entire class and not a part of any objects of the class. The main must always be declared as static since the interpreter uses this method before any objects are created. It allows main() to be called without having to instantiate a particular instance of the class. This is necessary since main() is called by the JVM before any objects are made.
- **Void:-** A keyword simply tells the compiler that the main() does not return values. In a main() method there is only one parameter ,String args[] . args[] is a name of the parameter that is an array of the objects of data type String. String store sequences of characters and args will receive the command line arguments when the program is executed. Any information that we need to pass to method is received by variables specified within the set of parentheses that follow the name of the method. These variables are called parameters. If no parameters are passed then the parentheses are empty.
- **The Output line:-**The only executable statement in the program is *System.out.println("Java is better than C++");* It is similar to printf() statement of C or cout<< construct of C++. The

println method is a member of the out object which is a static data member of system class. This line prints the string- *"Java is better than C++"* to the screen. The println method always appends a new line character to the end of the string. I.e. any subsequent output will start on a new line. Every statement must end with a semicolon. **(System is a class, out is an object and println is a method)**

Advantages of Java

- It is an open source, so users do not have to struggle with heavy license fees each year.
- The tools needed to build and test Java programs are available without charge. Sun makes the Java Development Kit (JDK) available over the Internet (at www.javasoft.com), where faculty and students alike can download it. The JDK includes the Java compiler and interpreter, among the other tools.
- Java API's can easily be accessed by developers.
- Multi-platform support language and support for web-services.
- Using JAVA we can develop dynamic web applications.
- Java embraced the concept of exception specifications.
- It allows you to create modular programs and reusable codes.
- By using Java, one program can be run on many different platforms. This means that you do not need to put your efforts on developing a different version of software for each platform. You can use the same code on windows, Linux, and Macintosh and so on. This is necessary when programs are downloaded over the internet to run on a variety of platforms.
- The syntax of java is similar of C and C++, making it easy to learn.
- Java is fully Object Oriented Programming to make bug free code than C++.
- Java eliminates manual memory allocation and de-allocation. Memory in java is automatically garbage collected. So programmer does not worry about memory corruption.
- Java eliminates pointer arithmetic.
- Java eliminates multiple inheritances, replacing it with a new notion of **interface** that they derived from objective C.

Why java is not 100% object oriented language

According to this assumption, there are several reasons why Java is not a 100% Object Oriented Programming Language:

- 1) Primitive data types (int, char, floats, etc.) are not an object and couldn't be defined as an object, neither a class. It is clear enough, because primitive data don't have any ability/characteristic such as inheritance or polymorphism. So, we can say that Java can be defined as not fully OOP supported language, since it's allowed a non object-oriented thing to exist. int, float, etc. primitive data types are not objects. But java implemented Wrapper classes for these primitive types to make them in to object types.
- 2) **Multiple-inheritance through classes is not possible:** As same as the operator overloading, multiple inheritance is also an (ad-hoc) abilities. It does support multiple inheritances directly, instead of that we can use interfaces, and with the help of interface implicitly we can use multiple inheritances.
- 3) **Availability of static methods and variables:** Since this ability exists in Java, which allows developers to invoke a method without instantiating any object like breaking rules of encapsulation.
- 4) **Operator overloading is not possible in java:** Except "+" operator which can be used in two different operations (addition operation and concatenation of string), there is no possibilities in Java to do an operator-overloading. Since, this is a kind of polymorphism ability (specific for operators) we also could say that Java isn't 100% OOP Language.

Java Tokens

Java Character set

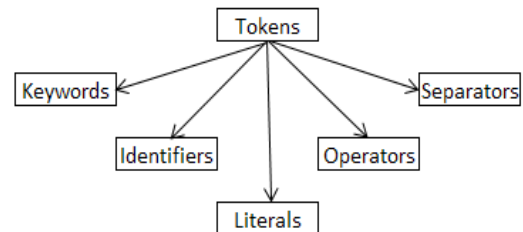
The character set of java consists of following set of characters, numbers and special symbols.

- Alphabets : lower case a to z, upper case A to Z
- Numbers : 0 to 9
- Special symbols : +, -, *, /, %, {, }, [,], ?, |, &, !, ; etc.

Java Tokens:

A Java program is basically a collection of classes. A class is defined by a set of declaration statements and methods containing executable statements. Most statements contain expressions, which describe the actions carried out on data. The word formed from the character set is building block of java and are known as **token**. These tokens represent the individual entity of language.

- Smallest individual units in a program are known as tokens.



The following different types of tokens used in java are:

- **Reserved Keywords:** names already in the programming language
- **Identifiers:** names the programmer chooses
- **Literals** (specified by their type)
 - Numeric: **int** and **double**
 - Logical: **boolean**
 - Textual: **char** and **String**
 - Reference: **null**
- **Operators:** symbols that operate on arguments and produce results
- **Separators** (also known as punctuators): punctuation characters and paired delimiters

Java Keywords

Keywords are reserved words of language. These keywords combined with operators and separators according to syntax, form definition of the java language. They have specific meaning in the language and the meaning is never change during the execution of program. Keywords cannot be used by the programmer as variable, classes, methods, constant names and so on. All keywords are to be written in lower case letters.

abstract	assert	boolean	break	byte	case
catch	char	class	const	continue	default
do	double	else	enum	extends	final
finally	float	for	goto	if	implements
import	instanceof	int	interface	long	native
new	package	private	protected	public	return
short	static	strictfp	super	switch	synchronized
this	throw	throws	transient	try	void
volatile	While				

Identifiers

An identifier is a word used by a programmer to name a variable, method (function), class or label, packages, objects and interfaces in a program. Keywords which are reserved words may not be used as identifiers. Identifier must be meaningful, short enough to be quickly and easily typed and long enough to be descriptive and easily read.

Java developers have followed some naming conventions.

- Names of all public methods and instance variables start with a leading lowercase letter. Example: average, sum
- When more than one word is used in name, the second and subsequent words are marked with a leading uppercase letters. Example: firstDayOfMonth, totalMarks
- All private and local variables use only lowercase letters combined with underscores. Example: length, total_marks
- All classes and interfaces start with a leading uppercase letter (and each subsequent word with a leading uppercase letter). Examples: Student, Employee, MotorCycle
- Variables that represent constant values use all uppercase letters and underscores between words. Example: TOTAL_MARKS, EMPLOYEE_SALARY
- Identifiers are like symbolic constants in C. All these are conventions and not rules. User may follow our own conventions as long as we do not break the basic rules of naming identifiers.

Literals

Literals in java are a sequence of characters (digits, letters and other characters) that represent constant values to be stored in variables. Literals can be any number, text, or other information that represents a value. This means what you type is what you get. A constant value in a program is denoted by a *literal*. Integer literals are whole numbers that can be assigned to variables of the following data types: int, long, float, double, char etc.

int num1=123;	Integer literals are written as plain integer numbers.
long int num1=123456789L long int num2=123456789l	Long literals are written as integer numbers with L suffix.
float num1=3.14f float num2=11.22F	Float literals are written as floating-point numbers with F suffix.
double num1=3.14; double num2=11.22d	Double literals are written as plain floating-point numbers. Optionally suffix D (case doesn't matter) can be provided.
String str="hello";	String literals are enclosed within double quotes.
char ch='a'; char ch1=65;	Char literals are enclosed within single quotes.
boolean a=true; boolean b=false;	Boolean literals are perhaps the simplest of all the literals. The value can either be true or false.

An integer literal can be represented using one of the following four numbering systems: hexadecimal (base 16), decimal (base 10), octal (base 8), and binary (base 2). Prefix 0 is used to indicate octal, and prefix 0x indicates hexadecimal when using these number systems for literals.

Int a=10;	Decimal literals are whole numbers of the base 10 numbering system.
int num = 0xFF;	Hexadecimal literals are hexadecimal numbers with 0x or 0X prefix.
Int num=0b1101	Binary literals are binary numbers with 0b or 0B prefix.
Int num=017	Octal literals are octal numbers with 0 prefix.

An escape sequence uses a special syntax to represent a character. The syntax begins with a single backslash character.

Escape	Meaning	Escape	Meaning
\n	New line	\'	Single quotation mark
\t	Tab	\"	Double quotation mark
\b	Backspace	\d	Octal
\r	Carriage return	\xd	Hexadecimal
\f	Form feed	\ud	Unicode character

\\	Backslash		
----	-----------	--	--

Null Literals: The final literal that we can use in Java programming is a Null literal. We specify the Null literal in the source code as 'null'. To reduce the number of references to an object, use null literal. The type of the null literal is always null. We typically assign null literals to object reference variables.

For instance s = null;

In this example, an object is referenced by s. We reduce the number of references to an object by assigning null to s. Now, as in this example the object is no longer referenced so it will be available for the garbage collection, i.e. the compiler will destroy it and the free memory will be allocated to the other object.

Separators

- Separators help define the structure of a program. The separators are parenthesis, (), braces, { }, the period, ., and the semicolon, ;.
- Separators are symbols used to indicate where groups of code are divided and arranged. They basically define the shape and function of our code.

Separator	Purpose
()	Encloses arguments in method definitions and calling; adjusts precedence in arithmetic expressions; surrounds cast types and delimits test expressions in flow control statements.
{ }	Defines blocks of code and automatically initializes arrays.
[]	Declares array types and dereferences array values.
;	Terminates statements.
,	Separates successive identifiers in variable declarations; chains statements in the test, expression of a for loop.
.	Selects a field or method from an object; separates package names from sub-package and class names.
:	Used after loop labels.

Constants

Constant is a fixed value that does not change during the execution of a program. Constant in java can be declared by using the keyword **final**. The keyword **final** indicates that once you assign some value to a variable, the value does not change throughout the program. In java by convention, constants are variables generally declared in uppercase only. Example: final int MAX=100;

Integer constant: It is a sequence of digits. There are three types of integer constants decimal, octal and hexadecimal.

1. Decimal integer constant: It consists of digits 0 through 9 preceded by an optional - or + sign.

E.g. valid decimal integers constant are:-123,-321, 0, 654321, +78

E.g. invalid decimal integer constant is:-Embedded spaces, commas and non-digit characters are not permitted between digits:-

15 750 Here no space is allowed.

20,000 Here no comma is allowed

\$1000 Here non digit character \$ is not permitted.

2. Octal integer constant:-It consists of digits from the set 0 through 7 with a leading 0, e.g., 037,0,0435,0551.

3. Hexadecimal integer constants:-A sequence of digits preceded by 0x or 0X is known as hexadecimal integer constants. They may also include alphabets A through F .The letters A through F represents the numbers 10 through 15.

e.g. 0x2, 0x9F, 0Xbcd, 0x.

Real constants:-The numbers with floating point are known as real integer constants, e.g., 0.0083,- 0.75, 435.36, +247.0

We can omit digits before or after the decimal point, e.g., 215. , .95, -.71, +.5

A real number may also be expressed in exponential (scientific) notation.

E.g. A value 215.65 may be written as 2.1565e2 in exponential notation. e2 means multiply by 10².

General form:-mantissa e exponent

Here mantissa is either real or an integer and exponent is an integer number with an optional plus or minus sign. E.g. 0.65e4, 12e-2, 1.5e+5, 3.18E3, -1.2E-1.

Single character constant:-It contains a single character enclosed within a pair of single quote marks.

E.g. '5', 'X', ';', ''

Character constants have integer values known as ASCII values

e.g. 'a' represents ASCII value 97.

String constants:-It is sequence of characters enclosed in double quotes. It may be letters, numbers, special characters and blank space. e.g. "Hello!", "1987", "WELL DONE", "?...!", "5+3", "X".

Backslash Character Constants: Some special backslash character constants that are used in output functions.

Constant	Meaning	Constant	Meaning
\a'	Audible alert (bell)	\v'	Vertical tab
\b'	Back space	\''	Single quote
\f'	Form feed	\'''	Double quote
\n'	New line	\?'	Question mark
\r'	Carriage return	\\'	Backslash
\t'	Horizontal tab	\0'	Null

Variables

A variable represents a memory location that can store a value.

Rules for naming variables

- The name of a variable needs to be meaningful, short and without any embedded space or symbol like ? ! @ # % ^ & * () [] { } . , ; : " ' / and \. However underscore can be used wherever a space is required for example basic_salary.
- Variable name must be unique. For example to store four different numbers, four unique variable names need to be used.
- A variable name must begin with a letter, a dollar symbol ('\$') or an underscore ('_'), which may be followed by a sequence of letters or digits (0-9), '\$' or '_'.
- Keywords cannot be used for variable names. For example, you cannot declare a variable called *switch*.
- Variable names must be meaningful. The names must reflect the data that the variables contain. Example to store the age of an employee, the variable name could be *employeeage*.
- Variable names are nouns and begin with a lowercase letter.
- If a variable name contains two or more words, join the words and begin each word with an uppercase letter. The first word, however, starts with a lowercase letter.
- Valid variable names: address1, studentname, total_salary
- Invalid variable names: \$salary, 1stname

Assigning values to variables:

- We can assign the value to variable at the time of declaration of a variable or we can read the value and assign it to the variable.
- Two ways to assign values to the variables:
 - By using assignment statement
 - By using read statement

- **By using assignment statement:** the value is assign to variable by using assignment operator (= sign). It is also referred to as variable initialization. If we do not initialize the variable then it is automatically set to zero.
- **Syntax:** [type] variable_name=value;
- For example:
 - int salary=5000;
 - a=b=0;
 - float marks=80;
- **By using Read statement:** From the keyboard at runtime we can also assign the value to a variable. This is done with the help of a keyboard readline.

Scope of variables:

- The area of the program where the variable is accessible is called its scope.
- Java variables are classified into three kinds
 - local variable
 - instance variable
 - class variable or static field
- Variables are declared and used inside methods which are called local variables. They are so called because they are not available for use outside the method definition. These are visible to the program only from the beginning of its program block to the end of the program block. Local variables can also be declared inside program blocks that are defined between an opening brace { and a closing brace }. These variables are visible to the program only from the beginning of its program block to the end of the program block. When the program control leaves a block, all the variables in the block will cease to exist. The area of the program where the variable is accessible is called its scope. There is no designated access modifier for local variables. Also, there is no default value for local variables, therefore, local variables must be initialized or assigned some value before they are used first time.
- Instance and class variables are declared inside a class. Instance variables are created when the objects are instantiated and they are associated with the objects. They take different values for each object. Instance variables are declared private or public or protected or default (no keyword). Instance variables are initialized to some default values by the compiler, in case they are not initialized by the creator of class. Instance variables are destroyed along with the object they have been created for.
- Class variables are global to a class and belong to the entire set of object that class creates. Only one memory is created for each variable. Class variables in Java are fields declared with static keyword. Modifier static informs the compiler that there will be only one copy of such variables created regardless of how many objects of this class are created. Static variables are created when a class is loaded into the memory by the class loader and destroyed when a class is destroyed or unloaded. Visibility of static fields will depend upon access modifiers. Default vales given to the static members will follow the same rule as it is done in instance variables.

Dynamic initialization:

Initialization is the process of providing value to a variable at declaration time. A variable is initialized once in its life time. Any attempt of setting a variable's value after its declaration is called assignment. To use a local variable you have to either initialize or assign it before the variable is first used. But for class members, the compulsion is not so strict. If you don't initialize them then compiler takes care of the initialization process and set class members to default values. Java allows its programmers to initialize a variable at run time also. Initializing a variable at run time is called

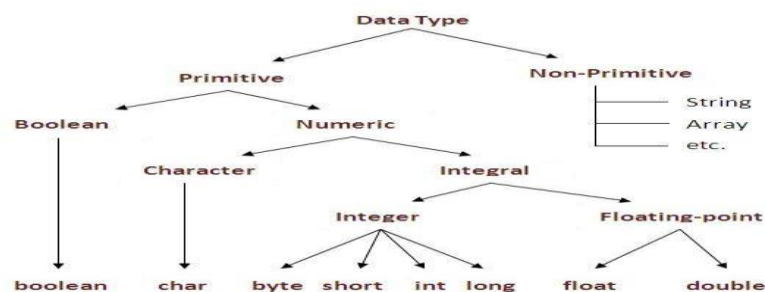
dynamic initialization. Java allows variables to be initialized dynamically, using any expression valid at the time the variable is declared.

Example:

```
public class MainClass
{
    public static void main(String args[])
    {
        double a = 3.0, b = 4.0;
        // c is dynamically initialized
        double c = Math.sqrt(a * a + b * b);
        System.out.println("Hypotenuse is " + c);
    }
}
```

Data Types in Java

Data types specify the size and type of values that can be stored. Primitive data types, also known as standard data types. When we are declaring a variable, we must specify the type of the variable, which means the type of data that the variable contains. i.e. numbers, characters or some value such as true or false.

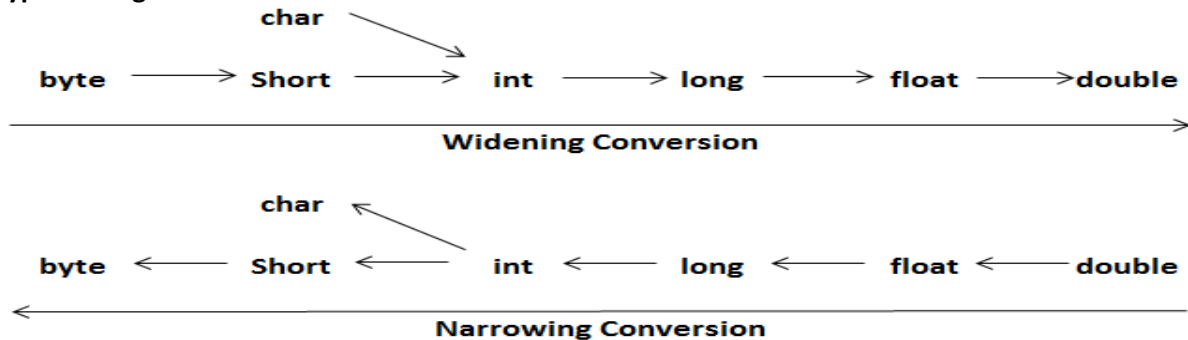


Data Types in Java

Data type	Size /format	Description	Range	Standard Default Value
(Numbers)				
byte	8 bit (1 byte)	Byte length integer	-128 to +127 if signed (-2^7 to 2^7-1) 0 to 255 if unsigned	0
short	16 bit (2 byte)	Short integer	-32,768 to +32,767 (-2^{15} to $2^{15}-1$)	0
int	32 bit (4 byte)	Integer	-2,147,483,648 to 2,147,483,647 (-2^{31} to $2^{31}-1$)	0
long	64 bit (8 byte)	Long integer	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 (-2^{63} to $2^{63}-1$)	0L
float	32 bit (4 byte)	Single precision floating point	+/- about 10^{39}	0.0f
double	64 bit (8 byte)	Double precision floating point	+/- about 10^{317}	0.0d
(Other types)				
char	16 bit (2 byte)	A single character		-
boolean	1 bit	A Boolean value (true or false)		False

Type Conversion (Casting)

In Java, type conversion are performed automatically when the type of the expression on the right hand side of an assignment operation can be safely promoted to the type of the variable on the left hand side of the assignment. Assigning a value of one type to a variable of another type is known as **Type Casting**.



Every expression has a type that is determined by the components of the expression.

Example:

```
double x;
int y=2;
float z=2.2f;
x=y+z;
```

The expression to the right of the “=” operator is solved first and the result is stored in x. The int value is automatically promoted to the higher data type float (float has a larger range than int) and then, the expression is evaluated. The resulting expression is of float data type. This value is then assigned to x, which is a double (larger range than float) and therefore, the result is a double.

Java automatically promotes values to a higher data type, to prevent any loss of information.

Example: `int x=5.5/2`

In above expression the right evaluates to a decimal value and the expression on left is an integer and cannot hold a fraction. Compiler will give you following error.

“Incompatible type for declaration, Explicit cast needed to convert double to int.”

This is because data can be lost when it is converted from a higher data type to a lower data type.

The compiler requires that you typecast the assignment.

Solution: `int x=(int)5.5/2;`

Casting into smaller type may result in loss of data. For example casting of float and double into integer will result in loss of fractional part. Following table shows casting that do not result in any loss of data.

Data type from	Data type to
Byte	Short, char, int, long, float, double
Short	int, long, float, double
Char	int, long, float, double
Int	long, float, double
Long	float, double
float	double

Operators

Java provides a rich set of operators. An operator is a symbol that tells the computer to perform certain mathematical or logical manipulations. Operators are used to compute and compare values, and test multiple conditions. Java operators can be classified into a number of categories:

- Arithmetic operators

- Assignment Operators
- Increment and Decrement Operators
- Relational Operators (comparison operators)
- Bitwise Operators
- Bitwise Shift Operators
- Logical Operators
- Conditional Operators (Ternary Operators)
- Special Operators

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

Operator	Description	Example	Explanation
+	Addition	$x = y + z$	Adds the value of y and z and stores the result in x.
-	Subtraction	$x = y - z$	Subtracts z from y and store the result in x.
*	Multiplication	$x = y * z$	Multiplies the values y and z and stores result in x.
/	Division	$x = y / z$	Divides y by z and stores the result in x.
%	Modulo	$x = y \% z$	Divides y by z and stores the remainder in x.

Assignment Operators: Assignment operators are used to assign the value of an expression to a variable.

Syntax: var op=exp;

Where var is a variable, op is a java binary operator and exp is an expression.

Example: $x=x+y$; will be written as $x+=y$; where the value of x is added into the value of y and answer is stored into x;

Operator	Description	Example	Explanation
=	Assigns the value of the right operand to the left.	$x = y$	Assigns the value of y to x.
+=	Adds the operands and assigns the result to the left operand.	$x += y$	Adds the value of y to x i.e., ($x = x + y$)
-=	Subtracts the right operand from the left operand and stores the result in the left operand.	$x -= y$	Subtract the value of y to x i.e., ($x = x - y$)
*=	Multiplies the left operand by the right operand and stores the result in the left operand.	$x *= y$	Multiply the value of y to x i.e., ($x = x * y$)
/=	Divides the left operand by the right operand and stores the result in the left operand.	$x /= y$	Divide the value of y to x i.e., ($x = x / y$)
%=	Divides the left operand by the right operand and stores the remainder in the left operand.	$x \% = y$	Stores remainder after divide the value of y to x i.e., ($x = x \% y$)

Increment and Decrement Operators:

Unary Operators: The operator ++ adds one to the operand and -- subtract one from the operand.

Operator	Description	Example	Explanation
++	Increases the value of the operand by one.	$x++$	Equivalent to $x = x + 1$
--	Decreases the value of the operand by one.	$x--$	Equivalent to $x = x - 1$

Pre-increment and post-increment: The `x++` and `++x` means the same thing when they form statements independently; they behave differently when they are used in expressions on the right hand side of an assignment statement.

Initial value of x	Expression	Final value of y	Final value of x
3	<code>y=x++</code>	3	4
3	<code>y=++x</code>	4	4
3	<code>y=x--</code>	3	2
3	<code>y=--x</code>	2	2

Comparison operators/Relational Operators: It is used to compare two quantities depending on their relation and take certain decisions. The relational operators determine the relationship that one operand has to the other. The expression use relational operators to control the if statement and the various loop statements.

Operator	Description	Example	Explanation
<code>==</code>	Evaluate whether the operands are equal.	<code>x == y</code>	Returns true if the values are equal and false if otherwise.
<code>!=</code>	Evaluates whether the operands are not equal.	<code>x != y</code>	Returns true if the values are not equal and false if otherwise.
<code>></code>	Evaluates whether the left operand is greater than the right operand.	<code>x > y</code>	Returns true if x is greater than y and false if otherwise.
<code><</code>	Evaluates whether the left operand is less than the right operand.	<code>x < y</code>	Returns true if x is less than y and false if otherwise.
<code>>=</code>	Evaluates whether the left operand is greater than or equal to the right operand.	<code>x >= y</code>	Returns true if x is greater than or equal to y and false if otherwise.
<code><=</code>	Evaluates whether the left operand is less than or equal to the right operand.	<code>x <= y</code>	Returns true if x is less than or equal to y and false if otherwise.

Bitwise operators: Java has a distinction of supporting special operators known as bitwise operators for manipulation of data at values of bit level. These operators are used for testing the bits or shifting bits right or left in the operations. Bitwise operators may not be applied to float or double.

Operator	Description	Example	Explanation
<code>&</code> (AND)	Evaluates to a binary value after a bit wise AND on the operands.	<code>x & y</code>	AND results in 1 if both the bits are 1, any other combination results in a 0. Example: If <code>x=00111100</code> and <code>y=00001101</code> <code>x&y=00001100</code>
<code> </code> (OR)	Evaluates to a binary value after a bit wise OR on the two operands.	<code>x y</code>	OR results in 0 when both the bits are 0, any other combination results in a 1. Example: If <code>x=00111100</code> and <code>y=00001101</code> <code>x y=00111101</code>
<code>^</code> (XOR)	Evaluates to a binary value after a bit wise XOR on the two operands.	<code>x ^ y</code>	XOR results in 0 if both the bits are of the same value and 1 if the bits have different values. Example: If <code>x=00111100</code> and <code>y=00001101</code> <code>x^y=00110001</code>
<code>~</code> (inversion)	Converts all 1 bits to 0s and all 0s bits to 1s.	<code>~x</code>	Example: <code>x=00111100</code> <code>~x=11000011</code>

These operators' act on boolean operands according to this table

A	B	A B	A&B	A^B	!A
False	False	False	False	False	True
True	False	True	False	True	False
False	True	True	False	True	True
True	True	True	True	False	False

Shift Operators:

Operator	Description	Example	Explanation
>>	Shifts bits to the right, filling sign bits at the left and it is also called the signed right shift operator.	x = 10 >> 3	The result of this is 10 divided by 2 ³ . Ex. 00001010 = 00000001
<<	Shifts bits to the left filling zeros at the right.	x = 10 << 3	The result of this is 10 multiplied by 2 ³ . Ex. 00001010 = 01010000
>>>	Also called the unsigned shift operator, works like the >> operator, but fills in zeroes from the left.	x = -10 >>> 3	If n is positive, then the result is the same as that of n>>s; if n is negative, the result is equal to that of the expression (n>>s)+(2<<~s) if the type of the left-hand operand is int.

Logical operators: Java provides an easy way to handle multiple conditions using the logical operators.

Operator	Description	Example	Explanation
&&	Evaluates to true if both the conditions evaluate to true, false if otherwise.	x > 5 && y < 10	The result is true if condition1 (x>5) and condition2 (y<10) are both true. If one of them is false, the result is false.
	Evaluates to true if at least one of the conditions evaluates to true, and false if none of the conditions evaluates to true.	x > 5 y < 10	The result is true if condition1 (x>5) or condition2 (y<10) or both true. If both the condition is false, the result is false.

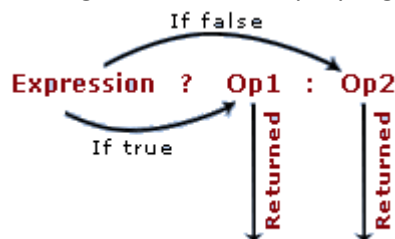
Conditional operators: The goal of the operator is to decide which value should be assigned to the variable. Ternary operator "? : " basically is used for an if-then-else as shorthand as:

Boolean expression ? operand1: operand2;

First expression is evaluated. If expression is true then returns operand1; otherwise returns operand2, if the expression is false.

For example: **x=(y>z)?y:z**

X is assigned the value of y if y is greater than z, else x is assigned the value of z.



Special Operators

The new operator: When you create an instance of a class, you need to allocate memory for it. When you declare an object, you merely state its data type. For example

Pen blackPen;

This tells the compiler that the variable blackPen is an object of the Pen class. It does not allocate memory for the object.

To allocate memory you need to use the new operator. Allocate memory for an object in two ways

Syntax:

1) Declare the object and then allocate memory using the new operator

```
<class_name><object_name>;
<object_name>=new<class_name>();
```

Example:

```
Pen blackPen;
blackPen=new Pen();
```

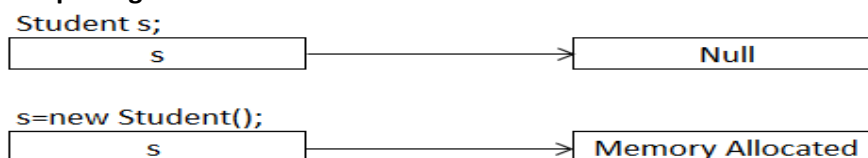
2) Declare the object and at the same time allocate memory using new operator.

```
<class_name><object_name>=new<class_name>();
```

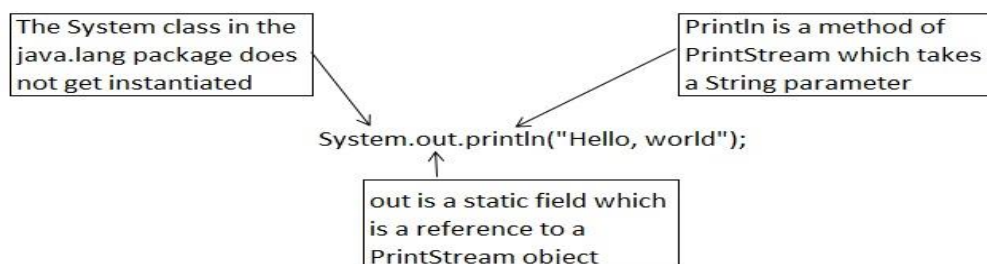
Example:

```
Pen blackPen=new Pen();
```

Concept Diagram:



The . Operator: The dot (.) operator access instance members of an object or class members of a class. For example, emp.salary()



The () operator: When declaring or calling a method, we list the methods arguments between (and). We can specify an empty argument list by using () with nothing between them.

[] operator: We can use square brackets to declare arrays, to create arrays and to access particular elements in an array. For example, int name[];

Type operator: Casts (or “converts”) a value to the specified type.

Instanceof operator: The instanceof operator tests whether its first operand is an instance of the second operand. Example x instanceof y; x must be the name of an object and y must be the name of a class. This operator is used only for object reference variables.

Example:

```
class Vehicle{}
```

```
public class Car extends Vehicle
{
public static void main(String args[])
{
Vehicle a = new Car();
boolean result = a instanceof Car;
System.out.println( result);
}
}
```

Output: true

Operator Precedence in Java

- When several operations occur in an expression, each part is evaluated and resolved in a predetermined order called operator precedence. Parenthesis can be used to override the order of precedence and force some parts of an expression to be evaluated before other parts. Operations within parenthesis are always performed before those outside. Within parenthesis, however, normal operator precedence is maintained.
- When expressions contain operators from more than one category, arithmetic operators are evaluated first, comparison operators are evaluated next and logical operators are evaluated last. Comparison operators all have equal precedence; that is they are evaluated in the left-to-right order in which they appear. Arithmetic and logical operators are evaluated in the following order of precedence.
- Table shows all java operators from highest to lowest precedence, along with their associativity.

Operator	Description	Level	Associativity
[] . () ++ --	Access array element Access object member Invoke a method Post-increment Post-decrement	1	Left to Right
++ -- + - ! ~	Pre-increment Pre-decrement Unary plus Unary minus Logical NOT Bitwise NOT	2	Right to Left
() New	Cast Object creation	3	Right to Left
* / %	Multiplication Division Modulus	4	Left to Right
+ +	Additive String concatenation	5	Left to Right
<< >> >>>	Left Shift Right shift Right shift with zero fill	6	Left to Right
< <= >	Relational, less than Less than or equal to Greater than	7	Left to Right

Operator	Description	Level	Associativity
>=	Greater than or equal to		
instanceof	Type comparison		
==	Equality	8	Left to Right
!=	Inequality		
&	Bitwise AND	9	Left to Right
^	Bitwise XOR	10	Left to Right
	Bitwise OR	11	Left to Right
&&	Conditional AND	12	Left to Right
	Conditional OR	13	Left to Right
?:	Conditional	14	Right to Left
= += -= *= /= %= &= ^= = <<== >>== >>>=	assignment	15	Right to Left

Precedence order: When two operators share an operand, the operator with the higher precedence goes first. For example $1+2*3$ is treated as $1+(2*3)$, whereas $1*2+3$ is treated as $(1*2)+3$ since multiplication has a higher precedence than addition.

Associativity: When two operators with the same precedence, the expression is evaluated according to its associativity. For example $x=y=z=17$ is treated as $x=(y=(z=17)))$, leaving all three variables with the value 17, since the = operator has right-to-left associativity. On the other hand, $70/2/3$ is treated as $(70/2)/3$ since the / operator has left-to-right associativity.

Mathematical Function:

Mathematical function such as sin, cos, sqrt used to analyze of real life problems. Java supports these basic math functions through Math class defined in the java.lang package. The mathematical functions are:

Abs()=abs method is used to find absolute value of variable.

Syntax: Math.abs(variable);

Here variable can be double, float, int and long datatype.

Example: Math.abs(-2.3);

Output: 2.3

Min()=min method is used to find minimum value between variable1 and variable2.

syntax: Math.min(variable1,variable2)

Here variable can be double,float,int and long datatype

Example: Math.min(2,8);

output = 2

Max()=max method is used to find maximum value between variable1 and variable2.

syntax: Math.max(variable1,variable2)

Here variable can be double,float,int and long datatype

Example: Math.max(2,8);

Output: 8

Pow()=Power of the number. Here variable1 is base value and variable2 is power value

Syntax: Math.pow(variable1,variable2)

Here variable is double datatype

Example: Math.pow(2,3);

Output: 8

Sqrt()=sqrt method is used for finding square root of a number.

Syntax: `Math.sqrt(variable)`

Here variable is double datatype

Example: `Math.sqrt(9);`

Output: 3

Exp()=Euler's number e raised to the power of a variable.

Syntax: `Math.exp(variable)`

Here variable is double datatype

Example: `Math.exp(2);`

Output: 7.38905609893065

Sin():To calculate the sine of a value.

Syntax: `Math.sin(double variable)`

Example: `Math.sin(82.55);`

Output: 0.763419622322519

Cos()=To calculate the cosine of an angle

Syntax: `Math.cos(variable)`

Example: `Math.cos(82);`

Output: 0.949677697882543

Tan()= To calculate the tangent of a number.

Syntax: `Math.tan(variable)`

Example: `Math.tan(80);`

Output: 9.00365494560708

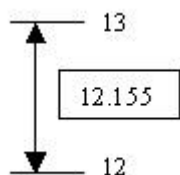
Ceil()=In arithmetic, the ceiling of a number is the closest integer that is greater or higher than the number considered. ceil method is used round up the variable.

Syntax: `Math.ceil(variable)`

Example: `Math.ceil(12.15)=13`

`Math.ceil(-24.06)=-24`

Consider a floating-point number such as 12.155. This number is between integer 12 and integer 13. In the same way, consider a number such as -24.06. As this number is negative, it is between -24 and -25, with -24 being greater.



Floor()=This is the floor() function which returns you the largest value but less than the argument. floor method is used round down the variable.

Syntax: `Math.floor(variable)`

Here variable is double datatype

Example: `Math.floor(8.4) = 8`

`Math.floor(-24.06)=-25`

copySign()= copySign method is used to copy sign of variable2 and assign that sign to variable1.

Syntax: `Math.copySign(variable1, variable2)`

Here variable is float and double datatype

Example: `Math.copySign(2,-8.3) = -2.0`
`Math.copySign(-2,8.3) = 2.0`

Round()=Rounds to the nearest integer.

Syntax: `Math.round(variable)`

Example: `Math.round(1.02)=1`
`Math.round(-2.1)=-2`

Random()=This is the `random()` function which returns you the random number. It is absolutely system generated. This method can be used to produce a random number between 0 and 100

Log()=Returns the natural logarithm of the argument.

Syntax: `Math.log(variable)`

Example: `Math.log(2)= 0.6931471805599453`

Decision Making and Branching

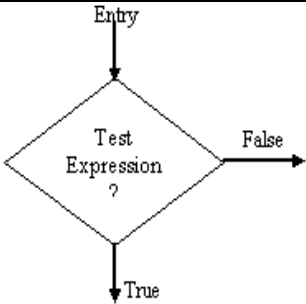
Generally instructions are executed sequentially. In some cases it is necessary to change the sequence of executions based on certain conditions. For this purpose decision control structure is required. When program breaks the sequential flow and jumps to another part of the code, it is called **branching**. When branching is based on a particular condition, it is known as **conditional branching**. If branching takes place without any decision, it is known as **unconditional branching**.

Decision Control Structure is a structure which executes a set of instructions depending on certain conditions.

An if statement tests a particular condition; if the condition evaluates to true, a course of action is followed, i.e., a statement or set of statements is executed. Otherwise (if the condition evaluates to false), the course-of-action is ignored. The if statement may be implemented in different forms depending on the complexity of conditions to be tested.

- Simple if statement
- if....else statement
- Nested if....else statement
- else if ladder

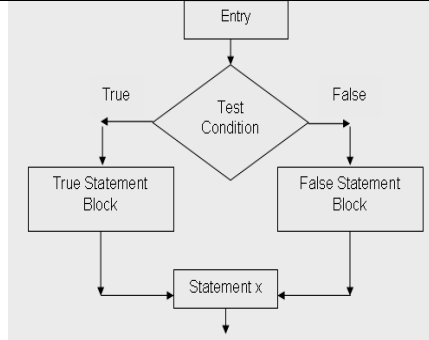
The if Statement: If the condition is true then the statements in the block are executed otherwise no statements in the block are executed.

<p>Syntax: if (test condition is true) { Execute Statements; }</p>	 <pre> graph TD Entry --> Test{Test Expression ?} Test -- True --> TrueOut[] Test -- False --> FalseOut[] style TrueOut fill:none,stroke:none style FalseOut fill:none,stroke:none </pre>	<pre> class ifelse { Public static void main(String args[]) { int a=10; int b=20; if (a<b) System.out.println("a is small"); } } </pre>
---	---	--

The if.....else statement: If condition is true then the True-block statement(s) is executed otherwise False-block statement(s) is executed.

Syntax:

```
if(test condition)
{
    True statement block;
}
else
{
    False statement block;
}
```



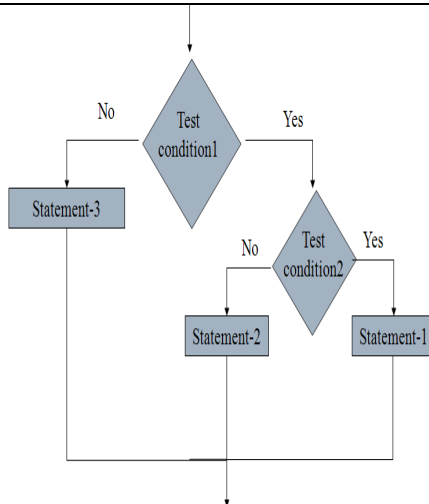
class ifelse

```
{
    Public static void main(String args[])
    {
        int a=10;
        int b=20;
        if (a<b)
            System.out.println("a is small");
        else
            System.out.println("b is small");
    }
}
```

Nested-If Statement: When a series of decisions are involved, we may have to use more than one if-else statement in nested form.

Syntax:

```
If (condition1)
{
    If (condition2)
    {
        statement-1;
    }
    else
    {
        statement-2;
    }
}
else
{
    statement-3;
}
statement-x;
```



class largest

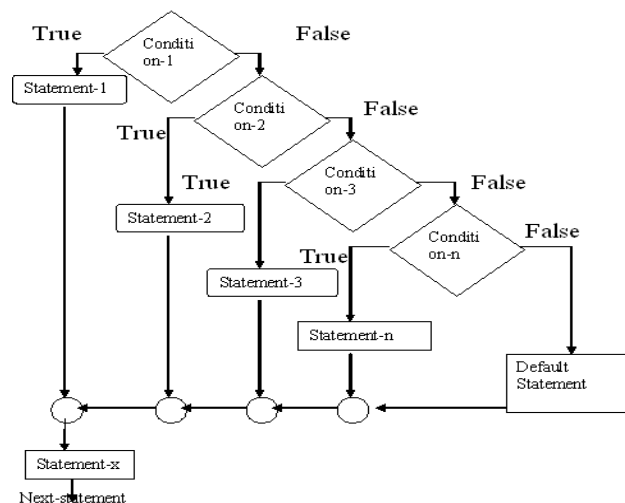
```
{
    public static void main(String args[])
    {
        int x=10, y=30, z=20;
        System.out.println("Largest number is=");
        if(x>y)
        {
            if(x>z)
            {
                System.out.println(x);
            }
            else
            {
                System.out.println(z);
            }
        }
        else
        {
            if(z>y)
            {
                System.out.println(z);
            }
            else
            {
                System.out.println(y);
            }
        }
    }
}
```

The Else-If Ladder: This construct is known as if else if ladder. The conditions are evaluated from the top to downwards. As soon as the true condition is found, the statement associated with it is executed and the control is transferred to the statement –x by skipping the rest of the ladder. When all the n conditions become false then the final else if containing the default- statement will be executed.

Syntax:

```
if(condition1)
    statement-1;
    else if(condition2)
        statement-2;
        else if(condition3)
            statement-3;
            .....
            .....
            else if(condition-n)
                statement-n;
                else
                    default- statement ;
                    statement-x;
```

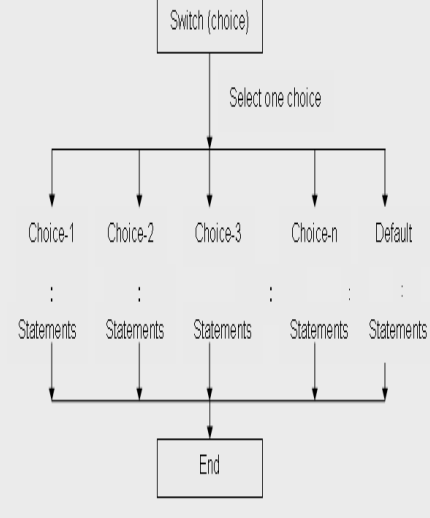
Concept Diagram:



Example of Else-if Ladder statement

```
if (marks > 79)
    grade="Honours";
else if(marks > 59)
    grade="First Class";
    else if(marks > 49)
        grade="Second Class";
        else if(marks >39)
            grade="Pass Class";
            else
                grade="Fail";
System.out.println("Grade="+grade);
```

Switch Statement: The switch statement causes a particular group of statement to be chosen from several available groups. The selection is based upon the value of an expression (int, char) which is included in the switch statement.

<p>Syntax:</p> <pre> switch (expression) { case expr-1: Statements; break; case expr-2: Statements; break; : : case expr-n: Statements; break; default: Statements; } </pre>		<pre> class switchcheck { Public static void main(String args[]) { char grade='A'; switch(grade) { case 'A' : System.out.println("Your Performance is Excellent"); break; case 'B' : System.out.println("Your Performance is Good"); break; case 'C' : System.out.println("Your Performance is Better"); break; case 'D' : System.out.println("Your Performance is not Satisfactory"); break; default : System.out.println("Invalid grade"); break; } } } </pre>
---	---	--

The default keyword: The statements associated with the default keyword are executed if the value of the switch variable does not match any of the case constants.

Nested Switch Statement: We can use a switch as part of the statement sequence of an outer switch. This is called nested switch. We know that each switch defines its own block of code but case constants of outer switch have no conflicts with the inner switch's case constants.

<p>Syntax:</p> <pre> switch (value) { case 1: switch (value) { case 1: statement(s) break; . . case n: statement (n) break; default: statement } } </pre>	<p>Example:</p> <pre> import java.util.*; class nswitch { public static void main (String arg[]) { Scanner in = new Scanner (System.in); int a,b; a=3; b=2; switch (a) { case 1: switch (b) { case 1: </pre>
--	---

<pre> } case 2: statement(s) break; . . case n: statement (n) break; default: statement } </pre>	<pre> System.out.println ("\n\t Inner One "); break; case 2: System.out.println ("\n\t Inner Two "); default: System.out.println ("\n\t Inner :: Invalid "); } case 2: System.out.println ("\n\t Outer Two "); break; case 3: System.out.println ("\n\t Outer Three "); break; default: System.out.println ("\n\t Outer :: Invalid "); } } } Output: Outer Three </pre>
--	---

Decision Making and Looping

A loop causes a section of a program to be repeated a certain number of times. The repetition continues while the condition set for it remains true. When the condition becomes false, the loop ends and the control is passed to the statement following the loop.

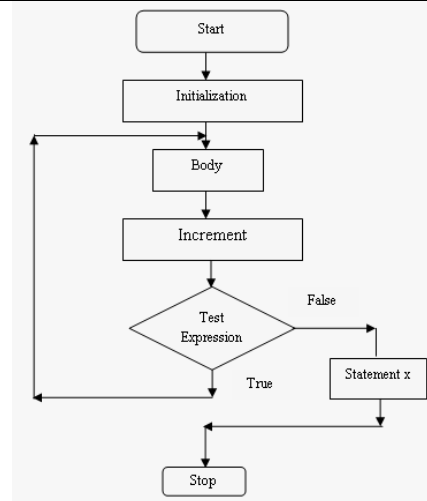
The While loop: The while is an entry controlled loop statement. Before executing any statements, the test condition is evaluated. If the condition is true, then the body of the loop is executed.

<p>Syntax:</p> <pre> while(test counter) { Body of the loop; Increment loop counter; } </pre>	<pre> graph TD A[Initialize counter] --> B{Test Condition} B -- True --> C[Statement Block] C --> D[Update counter] D --> B B -- False --> E[Statement x] E --> F[Stop] </pre>	<pre> class whileloop { public static void main(String args[]) { int i=0; while (i<10) { System.out.println("Number : "+ i); i++; } System.out.println("End of while loop"); } } </pre>
--	--	--

Do...while loop: In this the program proceeds to evaluate the body of the loop first. At the end of the loop the test condition in the while statement is evaluated.

Syntax:

```
do
{
  Body of the loop
} while(test expression);
```



```
class dowhileloop
{
  public static void main(String args[])
  {
    int i=0;
    do
    {
      System.out.println("Number : "+ i);
      i++;
    }
    while (i<10);
    System.out.println("End of dowhile
    loop");
  }
}
```

Entry-Controlled and Exit-Controlled statement: In Entry controlled loop, the test condition is checked first and if that condition is true then the block of statement in the loop body will be executed while in exit controlled loop the body of loop will be executed first and at the end the test condition is checked, if condition is satisfied then the body of loop will be executed again.

Example of Entry-Controlled loop is for loop, while loop

Example of Exit-Controlled loop is do...while loop.

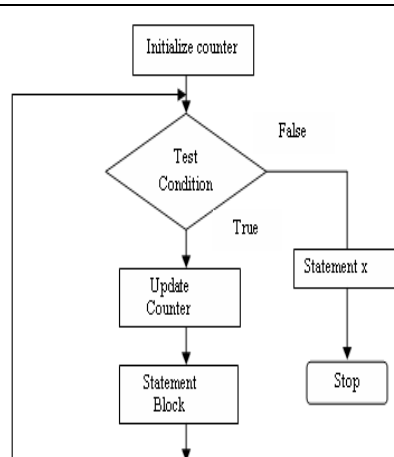
For loop

The for allows us to specify three things about a loop in a single line.

- Setting a loop counter to an initial value.
- Testing the loop counter to determine whether its value has reached the number of repetitions desired.
- Increasing the value of loop counter each time the program segment within the loop has been executed.

Syntax:

```
for ( counter initialization ;
test counter condition ;
increment counter )
{
  body of the loop;
}
```



```
class forloop
{
  public static void main(String args[])
  {
    int i;
    for(i=1;i<=10;i++)
    {
      System.out.println("Number : "+ i);
    }
  }
}
```

For-each version of the for loop

The for-each loop introduced in Java5. It is mainly used to traverse array or collection elements. The advantage of for-each loop is that it eliminates the possibility of bugs and makes the code more readable.

Syntax:

```
for(data_type variable : array | collection)
{ }
```

Example:

```
public class testforeach
{
    public static void main(String[] args)
    {
        int[] arr={ 11,22,33,44};

        System.out.println("Display an array using for loop");
        for (int i = 0; i < arr.length; i++)
        {
            System.out.print(arr[i] + " ");
        }
        System.out.println();

        System.out.println("Display an array using for each loop");
        for (int a : arr)
        {
            System.out.print(a+ " ");
        }
    }
}
```

Output:

```
Display an array using for loop
11 22 33 44
Display an array using for each loop
11 22 33 44
```

Break Statement and continue statement

Break Statement: The break statement causes the program flow to exit from the body of the switch construct. Control goes to the first statement following the end of the switch construct. If the break statement is not used, the control passes to the next case statement and the remaining statements in the switch construct are executed.

Example:

```
public class TestBreak
{
    public static void main(String args[])
    {
        int [] numbers = {10, 20, 30, 40, 50};
        for(int x : numbers )
        {
            if( x == 30 )
            {
                break;
            }
        }
    }
}
```

```
System.out.println( "value of x="+x );  
}  
}  
}
```

Output:

value of x=10
value of x=20

Continue statement: It is used to skip a part of the body of the loop under certain conditions. It causes the loop to be continued with the next iteration after skipping any statement in between. The continue statement tells the compiler "SKIP THE FOLLOWING STATEMENTS AND CONTINUE WITH THE NEXT ITERATION".

Example:

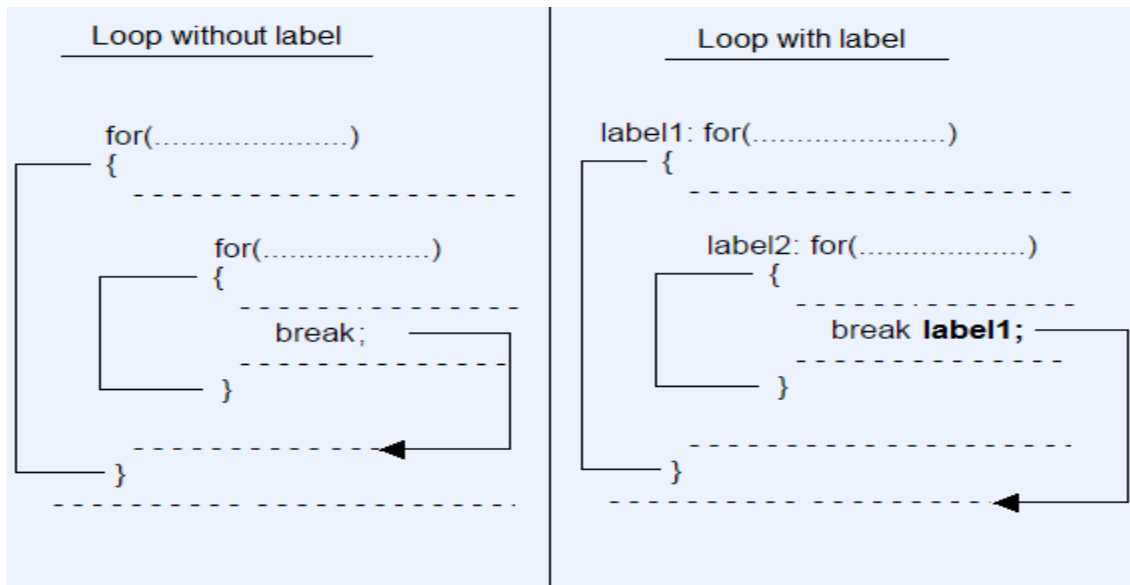
```
public class TestContinue  
{  
    public static void main(String args[])  
    {  
        int [] numbers = {10, 20, 30, 40, 50};  
        for(int x : numbers )  
        {  
            if( x == 30 )  
            {  
                continue;  
            }  
            System.out.print( x ); System.out.print("\n");  
        }  
    }  
}
```

Output:

10
20
40
50

Labeled Statement:

A *labeled statement* is simply a statement that has been given a name by prepending an identifier and a colon to it. Java doesn't provide goto statements. Java contains several ways to jump from one statement to another in certain circumstances. Labels are used by the break and continue statements. The continue statement (with or without a label reference) can only be used inside a loop. The break statement, without a label reference, can only be used inside a loop or a switch. With a label reference, it can be used to "jump out of" any Javascript code block:



Example:

```
class LabelledLoop
{
    public static void main(String args[])
    {
        int i,j;
```

```
        loop1: for(i=1;i<=10;i++)
        {
            System.out.println();
```

```
        loop2: for(j=1;j<=10;j++)
        {
            System.out.print(j + " ");
            if(j==5)
                break loop1; //Statement 1
        }
    }
}
```

Output:

1 2 3 4 5

Solved Programs**1. Program to print Natural Numbers.**

```
class Naturalnumbers
{
    public static void main(String args[])
    {
        int sum=0;
        for (int i=1;i<=10;i++)
        {
            System.out.println(" "+i);
            Sum=sum+i;
        }
        System.out.println("Sum="+sum);
    }
}
```

Output:

```
1 2 3 4 5 6 7 8 9 10
Sum=55
```

2. Program to check whether the character is vowel or not.

```
class Vowel
{
    public static void main(String args[])
    {
        char ch='a', ch1='z';
        if(ch=='a' || ch=='A' || ch=='e' || ch=='E' || ch=='i' || ch=='I' || ch=='o' || ch=='O' || ch=='u' ||
        ch=='U')
            System.out.println("ch=" +ch + " is Vowel");
        else
            System.out.println("ch=" +ch + " is not a Vowel");
        if(ch1=='a' || ch1=='A' || ch1=='e' || ch1=='E' || ch1=='i' || ch1=='I' || ch1=='o' || ch1=='O' ||
        ch1=='u' || ch1=='U')
            System.out.println("ch1=" +ch1 + " is Vowel");
        else
            System.out.println("ch1=" +ch1 + " is not a Vowel");
    }
}
```

Output:

```
ch=a is Vowel
ch1=z is not a Vowel
```

3. Program to check whether the entered character is a lower case, upper case, digit or special character.

```
class check
{
    public static void main(String args[])
    {
        char ch;
        try
        {
```

```
System.out.println("Enter any character=");
ch=(char)System.in.read();
if((ch>='a')&&(ch<='z'))
System.out.println("The character is lower case");
else if((ch>='A')&&(ch<='Z'))
System.out.println("The character is Upper case");
else if((ch>='0')&&(ch<='9'))
System.out.println("The character is Digit");
else
System.out.println("The character is Special Character");
}
catch(Exception e)
{
System.out.println("Error");
}
}
}
```

Output:

Enter any character=3
The character is Digit

4. Program to count even and odd numbers.

```
class Count
{
public static void main(String args[])
{
int num[]={50, 65, 56, 11, 81};
int even=0, odd=0;
for(int i=0; i<num.length;i++)
{
if((num[i]%2)==0)
{ even+=1;
}
else
{ odd+=1;
}
}
System.out.println("Even numbers="+even);
System.out.println("Odd numbers="+odd);
}
}
```

Logic

Even number=number%2=0
Odd number=number%2!=0

Output:

Even numbers=2
Odd numbers=3

5. Program to generate Fibonacci series i.e. 1 1 2 3 5 8 ...

```
class Fibonacci
{
    public static void main(String args[])
    {
        int f1=1,f2=1,f3,n=1;
        System.out.print(f1+" "+f2);
        while(n<10)
        {
            f3=f1+f2;
            System.out.print(" "+f3);
            f1=f2;
            f2=f3;
            n++;
        }
    }
}
```

Logic

1st no=1 2nd no=1
 3rd no=1st no + 2nd no
 =1 + 1 = 2
 Shift 2nd no to 1st no
 Shift 3rd no to 2nd no
 Again perform addition
 3rd no=1st no + 2nd no

Output:

1 1 2 3 5 8 13 21 34 55 89

6. Program to find sum of odd numbers up to 10.

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

Output:

1 3 5 7 9
 Sum=25

7. Define a class having three numbers as data members. Initialize and display the greatest of three numbers.

```
class Greatestnum
{
    public static void main(String args[])
    {
        int a,b,c;
        a=90;
        b=100;
        c=150;
        if((a>b)&&(a>c))
```

```
System.out.println("a is greater");
else if(b>c)
System.out.println("b is greater");
else
System.out.println("c is greater");
}
}
```

Output

c is greater

8. Define a class having one number, N as a data member, initializes and displays prime numbers from 1 to N.

```
class primenum
{
public static void main(String args[])
{
int i,num=0,n=20;
System.out.println("value of n="+n);
while(num<=n)
{
i=2;
while(i<=num)
{
if(num%i==0)
break;
i++;
}
if(i==num)
System.out.println(num+" is prime");
num++;
}
}
}
```

Output:

value of n=20
2 is prime
3 is prime
5 is prime
7 is prime
11 is prime
13 is prime
17 is prime
19 is prime

Logic

Prime numbers are those numbers which are divisible by 1 or itself.

Example:

7 can be divided only by 1 or 7, so 7 is a prime number

6 can be divided by 1, 2, 3 and 6, so it is a composite number, not a prime number

9. Define a class having one 3 digit number, num as data member, initialize and display reverse of that number.

```
class reversenum
{
    public static void main(String args[])
    {
        int num, temp;
        num=321;
        System.out.println("The number is="+num);
        System.out.println("Reverse number is=");
        do
        {
            temp=num%10;
            System.out.print(temp);
            num=num/10;
        }
        while(num!=0);
    }
}
```

Output

The number is=321
Reverse number is=123

Logic

1. Continue the steps until number is not equal to zero.
2. Divide the number by 10, and return the remainder. This is done using the % modulus operator.
3. Print the remainder
4. Divide the number by 10 and return the quotient.
5. Go to step 2 and continue.

10. Program to print factorial of a number.

```
import java.io.*;
import java.lang.*;
class Factorialn
{
    public static void main(String args[])
    {
        DataInputStream in=new DataInputStream(System.in);
        int n, fact=1;
        try
        {
            System.out.print("Enter Number:");
            n=Integer.parseInt(in.readLine());
            do
            {
                fact=fact*n;
                n--;
            }while(n>=1);
            System.out.println("Factorial of Number="+fact);
        }
        catch(Exception e)
        {
            System.out.println("Error");
        }
    }
}
```

Output:

Factorial of 4 is=24

Logic

$n! = n \times (n-1) \times (n-2) \dots$
using recursive function

11. Program to print even numbers up to 10 and sum of them.

```
class sumeven
{
    public static void main(String args[])
    {
        int sum=0,i=0;
        do
        {
            System.out.println(" "+i);
            sum=sum+i;
            i=i+2;
        }while(i<=10);
        System.out.println("Sum= "+sum);
    }
}
```

Output:

0 2 4 6 8 10
Sum= 30

12. Program to swap values of two variables.

```
class swap
{
    public static void main(String args[])
    {
        int a=10,b=20,c;
        System.out.println("Before Swapping");
        System.out.println("a="+a+" "+"b="+b);
        c=a;
        a=b;
        b=c;
        System.out.println("After Swapping");
        System.out.println("a="+a+" "+"b="+b);
    }
}
```

Logic
Before swapping
a=10 b=20
Swap using third variable
c=a value of c=10
a=b value of a=20
b=c value of b=10
After swapping
a=20 b=10

13. Program to find whether the given number is even or odd by using command line arguments.

```
class oddeven
{
    public static void main(String args[])
    {
        int a;
        a=Integer.parseInt(args[0]);
        if(a%2==0)
            System.out.println("The given number is even");
        else
            System.out.println("The given number is odd");
    }
}
```

Output:

C:\j2sdk1.4.0\bin>Javac oddeven.java
C:\j2sdk1.4.0\bin>Java oddeven 15
The given number is odd

14. Program to find sum of digits of number. (input number=321 sum=3+2+1=6)

```
import java.lang.*;
import java.io.*;
class sumdigit
{
    public static void main(String args[])
    {
        DataInputStream in=new DataInputStream(System.in);
        int number, sum=0;
        try
        {
            System.out.print("Enter Number:");
            number=Integer.parseInt(in.readLine());
            while(number>0)
            {
                sum=sum+number%10;
                number=number/10;
            }
            System.out.println("Sum of digit is="+sum);
        }
        catch(Exception e)
        {
            System.out.println("Error");
        }
    }
}
```

Logic

1st Execution

sum=sum+number%10
=0+321%10 =0+1=1
number=number/10
=321/10=32

2nd Execution

sum=sum+number%10
=1+32%10 =1+2=3
number=number/10
=32/10=3

3rd Execution

sum=sum+number%10
=2+3%10 =3+3=6
number=number/10
=3/10=0

Output:

Enter Number:321
Sum of digit is=6

15. Program to generate Harmonic Series .

```
class harmonic
{
    public static void main(String args[])
    {
        int num=Integer.parseInt(args[0]);
        double result=0.0;
        while(num>0)
        {
            result=result+(double) 1/num;
            num--;
        }
        System.out.println("Output of Harmonic series is"+result);
    }
}
```

Logic:

Input=5,
output=1+1/2+1/3+1/4+1/5=2.28
(Approximately)

Output:

```
C:\j2sdk1.4.0\bin>Javac harmonic.java
C:\j2sdk1.4.0\bin>Java harmonic 5
Output of Harmonic series is 2.28333333333333
```

16. Program to find whether given number is Armstrong or not.

```
class armstrong
{
public static void main(String args[])
{
int num=Integer.parseInt(args[0]);
int n=num, sum=0, remainder;
while(num>0)
{
remainder=num%10;
sum=sum+(int)Math.pow(remainder,3);
num=num/10;
}
if(sum==n)
System.out.println(n+" is an Armstrong Number");
else
System.out.println(n+" is not an Armstrong Number");
}
}
```

Logic

An Armstrong number is a number such that the sum of its digits raised to the third power is equal to the number itself. For example number=153, then $1^3+5^3+3^3=153$ so 153 is an Armstrong number.
Armstrong Numbers: 0, 1, 153, 370, 371, 407

Output:

```
C:\j2sdk1.4.0\bin>Javac armstrong.java
C:\j2sdk1.4.0\bin>Java armstrong 153
153 is an Armstrong Number
C:\j2sdk1.4.0\bin>Java armstrong 15
15 is not a Armstrong Number
```

17. Program to print triangle

```
class triangle
{
public static void main(String args[])
{
int num=5;
while(num>0)
{
for(int j=1;j<=num;j++)
{
System.out.print(" "+"*"+" ");
}
System.out.print("\n");
num--;
}
}
}
```

Output:

```
* * * * *
* * * *
* * *
* *
*
```

18. Program to print triangle

```
class triangle
{
public static void main(String args[])
{
int num=5;
while(num>0)
{
for(int j=1;j<=num;j++)
{
System.out.print(" "+num+" ");
}
System.out.print("\n");
num--;
}
}
}
```

Logic

	Columns				
Rows	c1	c2	c3	c4	c5
r1	5	5	5	5	5
r2	4	4	4	4	
r3	3	3	3		
r4	2	2			
r5	1				

Output:

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

19. Program to generate triangle

```
class triangle
{
public static void main(String args[])
{
for(int i=1;i<=5;i++)
{
for(int j=1;j<=i;j++)
{
System.out.print(" "+j+" ");
}
System.out.print("\n");
}
}
}
```

Logic

	Columns				
Rows	c1	c2	c3	c4	c5
r1	1				
r2	2	2			
r3	3	3	3		
r4	4	4	4		
r5	5	5	5	5	5

Output:

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

20. Program to print triangle

```
import java.lang.*;
import java.io.*;
class Temp
{
    public static void main(String args[])
    {
        int num=Integer.parseInt(args[0]);
        for(int i=1;i<=num;i++)
        {
            for(int j=1;j<=i;j++)
            {
                System.out.print((i*j)+ " ");
            }
            System.out.print("\n");
        }
    }
}
```

Output:

```
1
2 4
3 6 9
4 8 12 16
```

21. Program to display Floyd's triangle

```
class pascal
{
    public static void main(String args[])
    {
        int c=0;
        int n=Integer.parseInt(args[0]);
        loop1: for(int i=1;i<=n;i++)
        {
            loop2: for(int j=1;j<=i;j++)
            {
                if(c!=n)
                {
                    c++;
                    System.out.print(c+" ");
                }
                else
                    break loop1;
            }
            System.out.print("\n");
        }
    }
}
```

Logic

Rows	Columns			
	c1	c2	c3	c4
r1	1			
r2	2	3		
r3	4	5	6	
r4	7	8	9	10

Output:

```
C:\j2sdk1.4.0\bin>Javac pascal.java
C:\j2sdk1.4.0\bin>Java pascal 10
1
2 3
4 5 6
7 8 9 10
```

22. Program to print following pattern

```
public class Stars
{
    public static void main(String args[])
    {
        int c=1;
        for(int i=1;i<=5;i++)
        {
            for(int j=i;j<=5;j++)
            {
                System.out.print(" ");
            }
            for(int k=1;k<=c;k++)
            {
                if(k%2==0)
                    System.out.print(" ");
                else
                    System.out.print("*");
            }
            System.out.println();
            c+=2;
        }
    }
}
```

Output:

```
*
* *
* * *
* * * *
* * * * *
```

23. Program to check whether entered number is palindrome or not?

```
class palindrome
{
    public static void main(String args[])
    {
        int num=Integer.parseInt(args[0]);
        int n=num, reverse=0,remainder;
        while(num>0)
        {
            remainder=num%10;
            reverse=reverse*10+remainder;
```

Logic: The program expects the user to enter the number to check if palindrome. The entered number is reversed and stored in another variable. Finally the reversed number is checked for equality with the entered number. If they are the same, then it is a palindrome.

```
num=num/10;
}
if(reverse==n)
System.out.println(n+" is a Palindrome number");
else
System.out.println(n+" is not a Palindrome number");
}
}
```

Output:

```
C:\j2sdk1.4.0\bin>Javac palindrome.java
C:\j2sdk1.4.0\bin>Java palindrome 12321
12321 is a Palindrome number
C:\j2sdk1.4.0\bin>Java palindrome 12345
12345 is not a Palindrome number
```

24. Program to display Multiplication Table.

```
class multi
{
public static void main(String args[])
{
int num=Integer.parseInt(args[0]);
System.out.println("Multiplication Table");
for(int i=1; i<=num;i++)
{
for(int j=1;j<=num;j++)
{
System.out.print(" "+i*j+" ");
}
System.out.print("\n");
}
}
}
```

Output:

```
Multiplication Table
1 2 3 4 5 6 7 8 9 10
2 4 6 8 10 12 14 16 18 20
3 6 9 12 15 18 21 24 27 30
4 8 12 16 20 24 28 32 36 40
5 10 15 20 25 30 35 40 45 50
6 12 18 24 30 36 42 48 54 60
7 14 21 28 35 42 49 56 63 70
8 16 24 32 40 48 56 64 72 80
9 18 27 36 45 54 63 72 81 90
10 20 30 40 50 60 70 80 90 100
```

25. Program to check whether the entered number is prime or not.

```
class prime
{
    public static void main(String args[])
    {
        int num=Integer.parseInt(args[0]);
        int flag=0;
        for(int i=2;i<num;i++)
        {
            if(num%i==0)
            {
                System.out.println(num+"is not a Prime Number");
                flag=1;
                break;
            }
        }
        if(flag==0)
            System.out.println(num+"is a Prime Number");
    }
}
```

Output:

```
C:\j2sdk1.4.0\bin>Javac prime.java
C:\j2sdk1.4.0\bin>Java prime 5
5 is a Prime Number
C:\j2sdk1.4.0\bin>Java prime 6
6 is not a Prime Number
```

26. Program to calculate percentage of 4 subjects and print in suitable format.

```
import java.lang.*;
import java.io.*;
class percentage
{
    public static void main(String args[])
    {
        float sub1, sub2, sub3, sub4, per;
        DataInputStream in=new DataInputStream(System.in);

        try
        {
            System.out.print("Enter marks of subject1:");
            sub1=Float.parseFloat(in.readLine());
            System.out.print("Enter marks of subject2:");
            sub2=Float.parseFloat(in.readLine());
            System.out.print("Enter marks of subject3:");
            sub3=Float.parseFloat(in.readLine());
            System.out.print("Enter marks of subject4:");
            sub4=Float.parseFloat(in.readLine());
            per=(sub1+sub2+sub3+sub4)/4;
            System.out.print("Percentage="+per);
        }
    }
}
```

```
catch(Exception e)
{
    System.out.println("Error");
}
}
```

Output:

Enter marks of subject1: 60.00
Enter marks of subject2: 50.00
Enter marks of subject3: 60.00
Enter marks of subject4: 50.00
Percentage= 55.00

27. Write a program to check whether the given number is divisible by 5 or not.

```
import java.io.*;
class test
{
    public static void main(String args[])
    {
        BufferedReader br=new BufferedReader( new InputStreamReader(System.in));
        int n;
        try
        {
            System.out.println("Enter a number :");
            n=Integer.parseInt(br.readLine());
            if(n%5==0)
                System.out.println(n + " is divisible by 5 ");
            else
                System.out.println(n + " is not divisible by 5 ");
        }
        catch(IOException e)
        { System.out.println("I/O error"); }
    }
}
```

Output:

Enter a number: 5
5 is divisible by 5
Enter a number: 6
6 is not divisible by 5

28. Write a program to print prime numbers in the given range.

```
class primenum
{
    public static void main(String args[])
    {
        int i,num=0,n=20;
        System.out.println("value of n="+n);
        while(num<=n)
```

```
{  
i=2;  
while(i<=num)  
{  
if(num%i==0)  
break;  
i++;  
}  
if(i==num)  
System.out.print(num+" ");  
num++;  
}  
}  
}
```

Output:

2 3 5 7 11 13 17 19 23 29 31 37 41 43 47