**1.BASICS OF JAVA**



James Gosling initiated Java language project in June 1991 for use in one of his many set-top box projects. The language, initially called ‘Oak’ after an oak tree that stood outside Gosling's office, also went by the name ‘Green’ and ended up later being renamed as Java, from a list of random words.

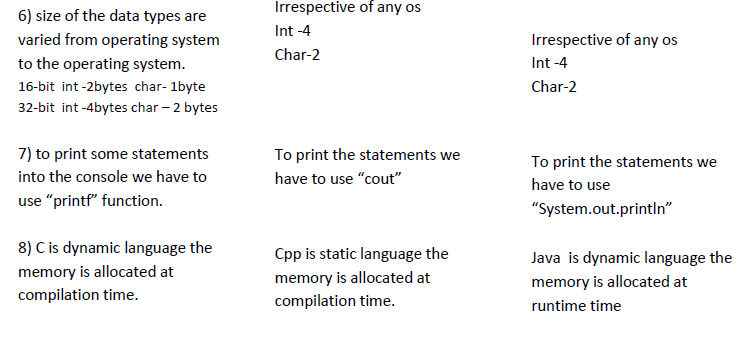
Sun released the first public implementation as Java 1.0 in 1995. It promised **Write Once, Run Anywhere** (WORA), providing no-cost run-times on popular platforms.

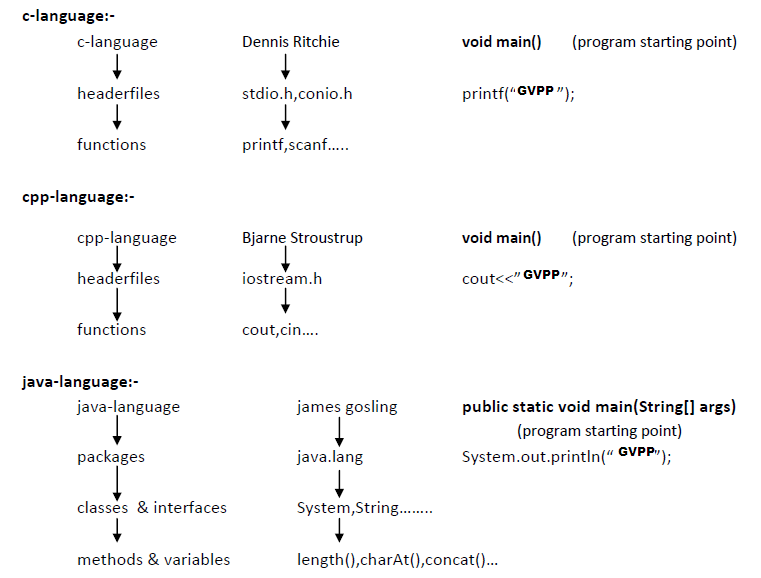
On 13 November, 2006, Sun released much of Java as free and open source software under the terms of the GNU General Public License (GPL).

Java is a general purpose, [high-level , object oriented programming language](http://www.webopedia.com/TERM/H/high_level_language.html).

**Difference between**







**JAVA FEATURES (BUZZWORDS)**

 **Object Oriented:** In Java, everything is an Object. Java can be easily extended since it is based on the Object model.

 **Platform Independent:** Unlike many other programming languages including C and C++, when Java is compiled, it is not compiled into platform specific machine, rather into platform independent byte code. This byte code is distributed over the web and interpreted by the Virtual Machine (JVM) on whichever platform it is being run on.

 **Simple:** Java is designed to be easy to learn. If you understand the basic concept of OOP Java, it would be easy to master.

 **Secure:** With Java's secure feature it enables to develop virus-free, tamper-free systems. Authentication techniques are based on public-key encryption.

 **Architecture-neutral:** Java compiler generates an architecture-neutral object file format, which makes the compiled code executable on many processors, with the presence of Java runtime system.

 **Portable:** Being architecture-neutral and having no implementation dependent aspects of the specification makes Java portable. Compiler in Java is written in ANSI C with a clean portability boundary, which is a POSIX subset.

 **Robust:** Java makes an effort to eliminate error prone situations by emphasizing mainly on compile time error checking and runtime checking.

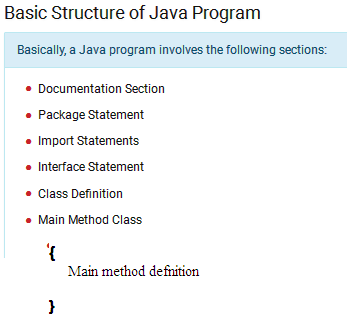
 **Multithreaded:** With Java's multithreaded feature it is possible to write programs that can perform many tasks simultaneously. This design feature allows the developers to construct interactive applications that can run smoothly.

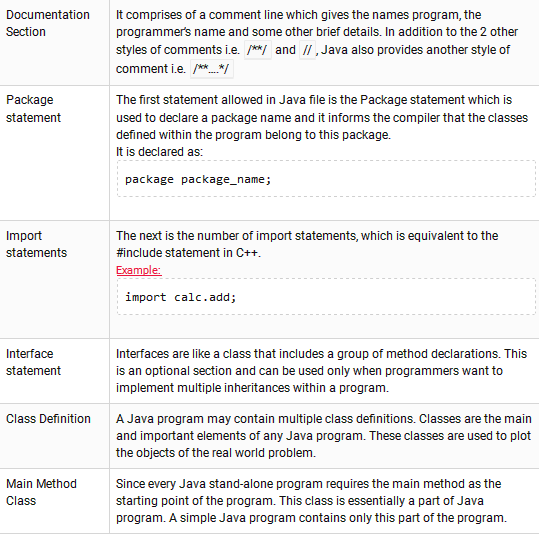
 **Interpreted:** Java byte code is translated on the fly to native machine instructions and is not stored anywhere. The development process is more rapid and analytical since the linking is an incremental and light-weight process.

 **High Performance:** With the use of Just-In-Time compilers, Java enables high performance.

 **Distributed:** Java is designed for the distributed environment of the internet.

 **Dynamic:** Java is considered to be more dynamic than C or C++ since it is designed to adapt to an evolving environment. Java programs can carry extensive amount of run-time information that can be used to verify and resolve accesses to objects on run-time.





**Java Tokens :**

A token is the smallest element of a program that is meaningful to the compiler.

Java tokens can be broken into sisx categories: identifiers, keywords, literals, operators, and separators,Variables.

**Identifiers:-**

any name in the java program like variable name,classname,methodname,interface name is called identifier.

class Test Test-------identifier

{

void add() add-------identifier

{

int a=10; a----------identifiers

int b=20; b----------identifiers

}

};

###### Rules of Identifiers Java

1. Should be single word. That is spaces are not allowed.  
   Example: mangoprice is valid but mango price is not valid.
2. Should start with a letter (alphabet) or underscore or $ symbol.  
   Example: price, \_price and $price are valid identifiers.
3. Should not be a keyword of Java as keyword carries special meaning to the compiler.  
   Example: class or void etc.
4. Should not start with a digit but digit can be in the middle or at the end.  
   Example: 5mangoescost is not valid and mango5cost and mangocost5 are valid.
5. Length of an identifier in Java can be of 65,535 characters and all are significant.
6. Identifiers are case-sensitive. That is both mango and Mango are treated differently.
7. Can contain all uppercase letters or lowercase letters or a mixture.

**Valid and invalid Java identifiers.**

|  |  |
| --- | --- |
| Valid | Invalid |
| HelloWorld | Hello World (uses a space) |
| Hi\_JAVA | Hi JAVA! (uses a space and punctuation mark) |
| value3 | 3value(begins with a number) |
| Tall | short (this is a Java keyword) |
| $age | #age (does not begin with any other symbol except \_ $ ) |

## ****Keywords**** :

“It is a special type of reserved word for a specific purpose which can not be use as a identifier means cannot be used as names for a variable, class, or method.”

Ex: **abstract,enum,instanceof,private,protected,public,static,int,float etc..**

## ****Separators :****

Separators are used to inform the Java compiler of how things are grouped in the code. For example, items in a list are separated by commas much like lists of items in a sentence.

|  |  |  |
| --- | --- | --- |
| **Symbol** | **Name** | **Purpose** |
| **;** | Semicolon | Terminates statements. |
| **,** | Comma | Separates consecutive identifiers in a variable  declaration.  Also used to chain statements together inside a **for**statement. |
| **{ }** | Braces | Used to contain the values of automatically initialized arrays.  Also used to define a block of code, for classes, methods, and local scopes. |
| **( )** | Parentheses | Used to contain lists of parameters in method definition and invocation.  Also used for defining precedence in expressions, containing expressions in control statements.  Also used for surrounding cast types. |
| **[ ]** | Brackets | Used to declare array types.  Also used when dereferencing array values. |
| **.** | Period | Used to separate package names from subpackages and classes Also used to separate a variable or method from a reference variable. |

**Variables:** is a Value that changes during the execution of the program.

Types

There are three types of variables in java:

* local variable
* instance variable
* static variable

#### 1) Local Variable

A variable which is declared inside the method is called local variable.

#### 2) Instance Variable

A variable which is declared inside the class but outside the method, is called instance variable . It is not declared as static.

#### 3) Static variable

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

### Example to understand the types of variables in java

class A{

int data=50; //instance variable

static int m=100; //static variable

void method(){

int n=90; //local variable

}

} //end of class

**Operator: Operator is a symbol that operates between two or more operands.**

Types:

* Arithmetic Operators
* Relational Operators
* Logical Operators
* Bitwise Operators
* Assignment Operators
* Ternary or Conditional Operators
* Increment and Decrement Operators.

**Arithmetic Operators**

Given table shows all the Arithmetic operator supported by C Language. Lets suppose variable **A** hold 8 and **B** hold 3.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Example (int A=8, B=3)** | **Result** |
| + | A+B | 11 |
| - | A-B | 5 |
| \* | A\*B | 24 |
| / | A/B | 2 |
| % | A%B | 2 |

**Relational Operators**

Which can be used to check the Condition, it always return true or false. Lets suppose variable **A** hold 8 and **B** hold 3.

|  |  |  |
| --- | --- | --- |
| **Operators** | **Example (int A=8, B=3)** | **Result** |
| < | A<B | False (0) |
| <= | A<=10 | True (1) |
| > | A>B | True (1) |
| >= | A<=B | False (0) |
| == | A== B | False (0) |
| != | A!=(-4) | True (1) |

**Logical Operator**

## Which can be used to combine more than one Condition?. Suppose you want to combined two conditions A<B and B>C, then you need to use Logical Operator like (A<B) && (B>C). Here && is Logical Operator. Truth table of Logical Operator

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **x** | **y** | **x && y** | **x ||y** | **!x** | **!y** |
| 1 | 1 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 | 0 |
| 0 | 0 | 0 | 0 | 1 | 1 |

**Assignment operators**

Which can be used to assign a value to a variable.Lets suppose variable **A** hold 8 and **B** hold 3.

|  |  |  |
| --- | --- | --- |
| **Operator** | **Example (int A=8, B=3)** | **Result** |
| += | A+=B or A=A+B | 11 |
| -= | A-=3 or A=A+3 | 5 |
| \*= | A\*=7 or A=A\*7 | 56 |
| /= | A/=B or A=A/B | 2 |
| %= | A%=5 or A=A%5 | 3 |
| =a=b | Value of b will be assigned to a |  |

**Bitwise Operators**

Bitwise operator works on bits and perform bit-by-bit operation. The truth tables for &, |, and ^ are as follows −

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **p** | **q** | **p & q** | **p | q** | **p ^ q** |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 | 1 |

[**Conditional operator (?)**](https://www.tutorialspoint.com/cplusplus/cpp_conditional_operator.htm)**:  
Condition ? X : Y**

[Conditional operator (?)](https://www.tutorialspoint.com/cplusplus/cpp_conditional_operator.htm). If Condition is true then it returns value of X otherwise returns value of Y.  
Ex: per>35? P: F

# Increment/decrement operators

Increment/decrement operators increment or decrement the value by 1

## Literals :

A constant value in Java is created by using a literal representation of it.

For example, Here are some java literals :

integer literal value : 100

floating-point literal value : 98.6

character  literal value : ‘X’

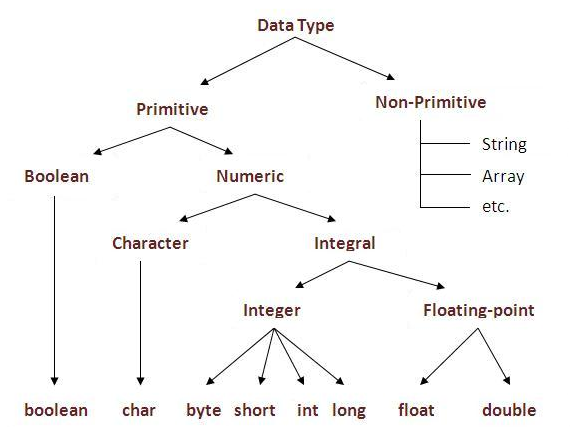
string literal value : “This is a test”

A literal can be used anywhere a value of its type is allowed

## Data Types in Java

Data types represent the different values to be stored in the variable. In java, there are two types of data types:

* Primitive data types
* Non-primitive data types



**Array :**It is a bunch or you can say group of variables with same data types.

|  |  |
| --- | --- |
| **Array** | A collection of several items of the same data type. For example names of student. |

**Arrays can be declared in three ways:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Way** | **Description** | **Syntax** | **Example** |
| **Only**  **Declaration** | Just declares the array. | data type identifier[] | char ch[ ]; declares a character array named ch. |
| **Declaration and creation** | Declares and allocates memory for the array elements using the reserved word ‘new’. | data type identifier[] =new data type[size]; | charch[] = new char[10]; declares an array ch to store 10 characters. |
| **Declaration, creation and initialization** | Declares the array, allocates memory for it and assigns initial values to its elements. | data type identifier[]  = {value1, value2, ….ValueN}; | charch[] ={‘A’,’B’,’C’,’D’}; declares an array ch to store 4 pre-as signed character values. |

**One dimensional array :**

Declaration syntax : type var-name[ ];

**EX.**

int month[ ];

**Multidimensional array :**

Declaration syntax : type var-name[ ][ ];

**EX.**

int mat[ ][ ] = new int[4][5];

**Java String**

In java, string is basically an object that represents sequence of char values. An array of characters works same as java string. For example:

1. char[] ch={'j','a','v','a','t','p','o','i','n','t'};
2. String s=new String(ch);

is same as:

1. String s="javatpoint";

**Conditional Statements**

The Java Conditional Statementsis used to test the condition. It checks boolean condition: *true* or *false*. There are various types of if statement in java.

* if statement
* if-else statement
* if-else-if ladder
* nested if statement
* **if statement**The Java if statement tests the condition. It executes the *if block* if condition is true.

**Syntax:**

if(condition)  
{

//code to be executed

}

**Example:**

public class IfExample {

public static void main(String[] args)  
 {

    int age=20;

    if(age>18){

        System.out.print("Age is greater than 18");   Output:

    }  }}   Age is greater than 18

## Java IF-else Statement

The Java if-else statement also tests the condition. It executes the if block if condition is true otherwise else block is executed.

**Syntax:**

if(condition){

//code if condition is true

}else{

//code if condition is false

}

## Java IF-else-if ladder Statement

The if-else-if ladder statement executes one condition from multiple statements.

**Syntax:**

if(condition1){

//code to be executed if condition1 is true

} else if(condition2){

//code to be executed if condition2 is true

}

else if(condition3){

//code to be executed if condition3 is true

}

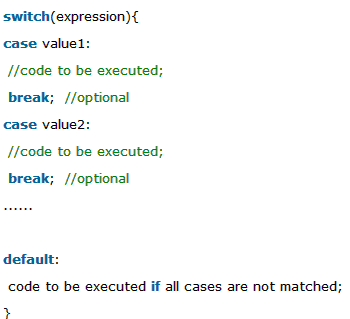
else{

//code to be executed if all the conditions are false

}

**Java Switch Statement**

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



# Looping statements:

# Java For Loop

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

There are three types of for loop in java.

* Simple For Loop
* For-each or Enhanced For Loop
* Labeled For Loop

## Java Simple For Loop

The simple for loop is same as C/C++. We can initialize variable, check condition and increment/decrement value.

**Syntax:**

for(initialization;condition;incr/decr){

//code to be executed

}

**Example:**public class ForExample {

public static void main(String[] args) {

    for(int i=1;i<=10;i++){

        System.out.println(i);

    }  } }

## Java For-each Loop

The for-each loop is used to traverse array or collection in java. It is easier to use than simple for loop because we don't need to increment value and use subscript notation.

It works on elements basis not index. It returns element one by one in the defined variable.

**Syntax:**

for(Type var:array){

//code to be executed

}

**Example:**

public class ForEachExample {

public static void main(String[] args) {

    int arr[]={12,23,44,56,78};

    for(int i:arr){

        System.out.println(i);

    }  }}

## Java Labeled For Loop

We can have name of each for loop. To do so, we use label before the for loop. It is useful if we have nested for loop so that we can break/continue specific for loop.

Normally, break and continue keywords breaks/continues the inner most for loop only.

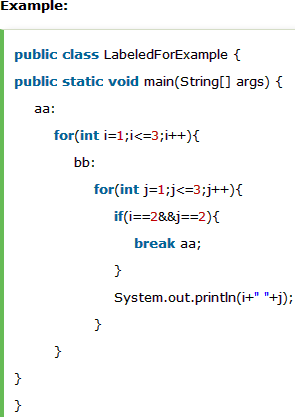
**Syntax:**

labelname:

for(initialization;condition;incr/decr){

//code to be executed

}



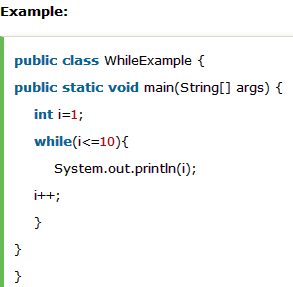
**Java While Loop**

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

**Syntax:**while(condition){

//code to be executed

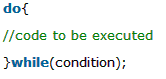
}

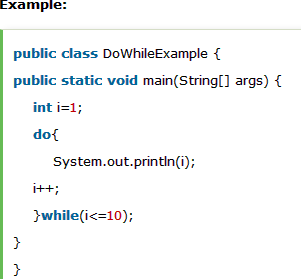


# Java do-while Loop

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

The Java do-while loop is executed at least once because condition is checked after loop body.

**Syntax:**



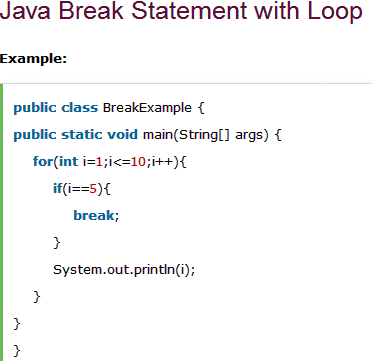
**Java Break Statement**

The Java *break* is used to break loop or switch statement. It breaks the current flow of the program at specified condition. In case of inner loop, it breaks only inner loop.

**Syntax:**

jump-statement;

break;   

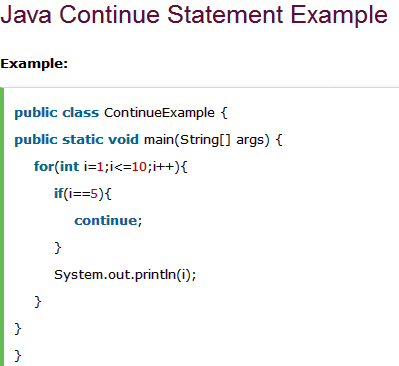


**Java Continue Statement**

The Java *continue statement* is used to continue loop. It continues the current flow of the program and skips the remaining code at specified condition. In case of inner loop, it continues only inner loop.

**Syntax:**jump-statement;

continue;



# Java Comments

The java comments are statements that are not executed by the compiler and interpreter. The comments can be used to provide information or explanation about the variable, method, class or any statement. It can also be used to hide program code for specific time.

## Types of Java Comments

There are 3 types of comments in java.

1. Single Line Comment
2. Multi Line Comment
3. Documentation Comment

## 1) Java Single Line Comment

The single line comment is used to comment only one line.

**Syntax:**

1. //This is single line comment

## 2) Java Multi Line Comment

The multi line comment is used to comment multiple lines of code.

**Syntax:**

/\* This  is  multi line  comment \*/

## 3) Java Documentation Comment

The documentation comment is used to create documentation API. To create documentation API, you need to use **javadoc tool**.

**Syntax:**

/\*\* This  is  documentation  comment \*/