**Unit-IV**

**Java as Object Oriented Programming Language-Overview**

# History of Java

**Java history** is interesting to know. The history of java starts from Green Team. Java team members (also known as **Green Team**), initiated a revolutionary task to develop a language for digital devices such as set-top boxes, televisions etc.

For the green team members, it was an advance concept at that time. But, it was suited for internet programming. Later, Java technology as incorporated by Netscape.

[**James Gosling**](http://en.wikipedia.org/wiki/James_Gosling)

Currently, Java is used in internet programming, mobile devices, games, e-business solutions etc. There are given the major points that describes the history of java.

1) **James Gosling**, **Mike Sheridan**, and **Patrick Naughton** initiated the Java language project in June 1991. The small team of sun engineers called **Green Team**.

2) Originally designed for small, embedded systems in electronic appliances like set-top boxes.

3) Firstly, it was called **"Greentalk"** by James Gosling and file extension was .gt.

4) After that, it was called **Oak** and was developed as a part of the Green project.

## Why sun choosed "Oak" name?

5) **Why Oak?** Oak is a symbol of strength and choosen as a national tree of many countries like U.S.A., France, Germany, Romania etc.

6) In 1995, Oak was renamed as **"Java"** because it was already a trademark by Oak Technologies.

# Features of Java

There is given many features of java. They are also known as java buzzwords. The Java Features given below are simple and easy to understand.

1. Simple
2. Object-Oriented
3. Platform independent
4. Secured
5. Robust
6. Architecture neutral
7. Portable
8. Dynamic
9. Interpreted
10. High Performance
11. Multithreaded
12. Distributed

### Simple

|  |
| --- |
| According to Sun, Java language is simple because: |
| syntax is based on C++ (so easier for programmers to learn it after C++). |
| removed many confusing and/or rarely-used features e.g., explicit pointers, operator overloading etc. |
| No need to remove unreferenced objects because there is Automatic Garbage Collection in java. |

### Object-oriented

|  |
| --- |
| Object-oriented means we organize our software as a combination of different types of objects that incorporates both data and behaviour. |
| Object-oriented programming(OOPs) is a methodology that simplify software development and maintenance by providing some rules. |
| Basic concepts of OOPs are: |
| 1. Object 2. Class 3. Inheritance 4. Polymorphism 5. Abstraction 6. Encapsulation |

### Platform Independent

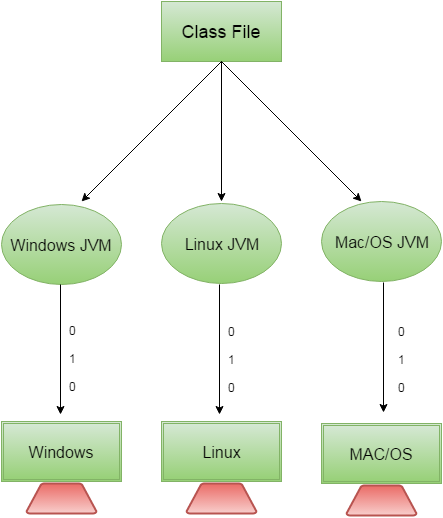
A platform is the hardwae or software environment in which a program runs.

There are two types of platforms software-based and hardware-based. Java provides software-based platform.

The Java platform differs from most other platforms in the sense that it is a software-based platform that runs on the top of other hardware-based platforms. It has two components:

1. Runtime Environment
2. API(Application Programming Interface)

Java code can be run on multiple platforms e.g. Windows, Linux, Sun Solaris, Mac/OS etc. Java code is compiled by the compiler and converted into bytecode. This bytecode is a platform-independent code because it can be run on multiple platforms i.e. Write Once and Run Anywhere(WORA).



### Secured

Java is secured because:

* No explicit pointer
* Java Programs run inside virtual machine sandbox



* Classloader: adds security by separating the package for the classes of the local file system from those that are imported from network sources.
* Bytecode Verifier: checks the code fragments for illegal code that can violate access right to objects.
* Security Manager: determines what resources a class can access such as reading and writing to the local disk.

These security are provided by java language. Some security can also be provided by application developer through SSL, JAAS, Cryptography etc.

### Robust

Robust simply means strong. Java uses strong memory management. There are lack of pointers that avoids security problem. There is automatic garbage collection in java. There is exception handling and type checking mechanism in java. All these points makes java robust.

### Architecture-neutral

There is no implementation dependent features e.g. size of primitive types is fixed.

In C programming, int data type occupies 2 bytes of memory for 32-bit architecture and 4 bytes of memory for 64-bit architecture. But in java, it occupies 4 bytes of memory for both 32 and 64 bit architectures.

### Portable

We may carry the java bytecode to any platform.

### High-performance

|  |
| --- |
| Java is faster than traditional interpretation since byte code is "close" to native code still somewhat slower than a compiled language (e.g., C++) |

### Distributed

|  |
| --- |
| We can create distributed applications in java. RMI and EJB are used for creating distributed applications. We may access files by calling the methods from any machine on the internet. |

### Multi-threaded

A thread is like a separate program, executing concurrently. We can write Java programs that deal with many tasks at once by defining multiple threads. The main advantage of multi-threading is that it doesn't occupy memory for each thread. It shares a common memory area. Threads are important for multi-media, Web applications etc.

# Difference between JDK, JRE and JVM

Understanding the difference between JDK, JRE and JVM is important in Java. We are having brief overview of JVM here.

If you want to get the detailed knowledge of Java Virtual Machine, move to the next page. Firstly, let's see the basic differences between the JDK, JRE and JVM.

### JVM

JVM (Java Virtual Machine) is an abstract machine. It is a specification that provides runtime environment in which java bytecode can be executed.

JVMs are available for many hardware and software platforms. JVM, JRE and JDK are platform dependent because configuration of each OS differs. But, Java is platform independent.

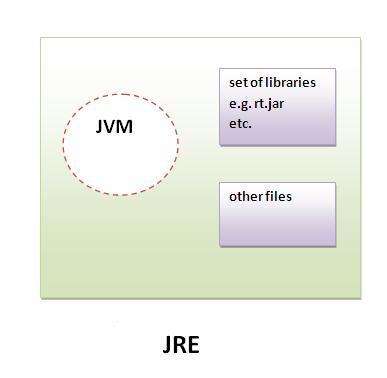
The JVM performs following main tasks:

* Loads code
* Verifies code
* Executes code
* Provides runtime environment

### JRE

JRE is an acronym for Java Runtime Environment.It is used to provide runtime environment.It is the implementation of JVM. It physically exists. It contains set of libraries + other files that JVM uses at runtime.

Implementation of JVMs are also actively released by other companies besides Sun Micro Systems.



### JDK:

JDK is an acronym for Java Development Kit.It physically exists.It contains JRE + development tools.



JVM (Java Virtual Machine)

JVM (Java Virtual Machine) is an abstract machine. It is a specification that provides runtime environment in which java bytecode can be executed.

JVMs are available for many hardware and software platforms (i.e. JVM is platform dependent).

### What is JVM

It is:

1. **A specification** where working of Java Virtual Machine is specified. But implementation provider is independent to choose the algorithm. Its implementation has been provided by Sun and other companies.
2. **An implementation** Its implementation is known as JRE (Java Runtime Environment).
3. **Runtime Instance** Whenever you write java command on the command prompt to run the java class, an instance of JVM is created.

### What it does

The JVM performs following operation:

* Loads code
* Verifies code
* Executes code
* Provides runtime environment

JVM provides definitions for the:

* Memory area
* Class file format
* Register set
* Garbage-collected heap
* Fatal error reporting etc.

### 

### Internal Architecture of JVM

|  |
| --- |
| Let's understand the internal architecture of JVM. It contains classloader, memory area, execution engine etc. |



### 1) Classloader

Classloader is a subsystem of JVM that is used to load class files.

### 2) Class(Method) Area

Class(Method) Area stores per-class structures such as the runtime constant pool, field and method data, the code for methods.

### 3) Heap

It is the runtime data area in which objects are allocated.

### 4) Stack

|  |
| --- |
| Java Stack stores frames.It holds local variables and partial results, and plays a part in method invocation and return. |
| Each thread has a private JVM stack, created at the same time as thread. |
| A new frame is created each time a method is invoked. A frame is destroyed when its method invocation completes. |

### 5) Program Counter Register

PC (program counter) register. It contains the address of the Java virtual machine instruction currently being executed.

### 6) Native Method Stack

It contains all the native methods used in the application.

### 7) Execution Engine

|  |
| --- |
| It contains: |
| 1) A virtual processor |
| 2) Interpreter: Read bytecode stream then execute the instructions. |
| 3) Just-In-Time(JIT) compiler: It is used to improve the performance.JIT compiles parts of the byte code that have similar functionality at the same time, and hence reduces the amount of time needed for compilation.Here the term ?compiler? refers to a translator from the instruction set of a Java virtual machine (JVM) to the instruction set of a specific CPU. |

# Variables and Data Types in Java

Variable is a name of memory location. There are three types of variables in java: local, instance and static.

There are two types of data types in java: primitive and non-primitive.

## Variable

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

int data=50;//Here data is variable

### Types of Variable

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.

We will have detailed learning of these variables in next chapters.

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

1. class A{
2. int data=50;//instance variable
3. static int m=100;//static variable
4. void method(){
5. int n=90;//local variable
6. }
7. }//end of class

## 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



|  |  |  |
| --- | --- | --- |
| Data Type | Default Value | Default size |
| boolean | false | 1 bit |
| char | '\u0000' | 2 byte |
| byte | 0 | 1 byte |
| short | 0 | 2 byte |
| int | 0 | 4 byte |
| long | 0L | 8 byte |
| float | 0.0f | 4 byte |
| double | 0.0d | 8 byte |

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

It is because java uses Unicode system than ASCII code system. The \u0000 is the lowest range of Unicode system. To get detail explanation about Unicode visit next page.

# Java Array

Normally, array is a collection of similar type of elements that have contiguous memory location.

**Java array** is an object the contains elements of similar data type. It is a data structure where we store similar elements. We can store only fixed set of elements in a java array.

Array in java is index based, first element of the array is stored at 0 index.



### Advantage of Java Array

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

### Disadvantage of Java Array

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

### Types of Array in java

There are two types of array.

* Single Dimensional Array
* Multidimensional Array

### Single Dimensional Array in java

### Syntax to Declare an Array in java

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

### Instantiation of an Array in java

1. arrayRefVar=new datatype[size];

### Example of single dimensional java array

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

1. class Testarray{
2. public static void main(String args[]){
4. int a[]=new int[5];//declaration and instantiation
5. a[0]=10;//initialization
6. a[1]=20;
7. a[2]=70;
8. a[3]=40;
9. a[4]=50;
11. //printing array
12. for(int i=0;i<a.length;i++)//length is the property of array
13. System.out.println(a[i]);
15. }}

[Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=Testarray)

Output: 10

20

70

40

50

## Declaration, Instantiation and Initialization of Java Array

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

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

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

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

[Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=Testarray1)

Output:33

3

4

5

### Passing Array to method in java

We can pass the java array to method so that we can reuse the same logic on any array.

Let's see the simple example to get minimum number of an array using method.

1. class Testarray2{
2. static void min(int arr[]){
3. int min=arr[0];
4. for(int i=1;i<arr.length;i++)
5. if(min>arr[i])
6. min=arr[i];
8. System.out.println(min);
9. }
11. public static void main(String args[]){
13. int a[]={33,3,4,5};
14. min(a);//passing array to method
16. }}

[Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=Testarray2)

Output:3

### Multidimensional array in java

In such case, data is stored in row and column based index (also known as matrix form).

### Syntax to Declare Multidimensional Array in java

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

### Example to instantiate Multidimensional Array in java

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

### Example to initialize Multidimensional Array in java

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

### Example of Multidimensional java array

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

1. class Testarray3{
2. public static void main(String args[]){
4. //declaring and initializing 2D array
5. int arr[][]={{1,2,3},{2,4,5},{4,4,5}};
7. //printing 2D array
8. for(int i=0;i<3;i++){
9. for(int j=0;j<3;j++){
10. System.out.print(arr[i][j]+" ");
11. }
12. System.out.println();
13. }
15. }}

[Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=Testarray3)

Output:1 2 3

2 4 5

4 4 5

### What is the class name of java array?

In java, array is an object. For array object, an proxy class is created whose name can be obtained by getClass().getName() method on the object.

1. class Testarray4{
2. public static void main(String args[]){
4. int arr[]={4,4,5};
6. Class c=arr.getClass();
7. String name=c.getName();
9. System.out.println(name);
11. }}

[Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=Testarray4)

Output:I

### Copying a java array

We can copy an array to another by the arraycopy method of System class.

### Syntax of arraycopy method

1. public static void arraycopy(
2. Object src, int srcPos,Object dest, int destPos, int length
3. )

### Example of arraycopy method

1. class TestArrayCopyDemo {
2. public static void main(String[] args) {
3. char[] copyFrom = { 'd', 'e', 'c', 'a', 'f', 'f', 'e',
4. 'i', 'n', 'a', 't', 'e', 'd' };
5. char[] copyTo = new char[7];
7. System.arraycopy(copyFrom, 2, copyTo, 0, 7);
8. System.out.println(new String(copyTo));
9. }
10. }

[Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=TestArrayCopyDemo)

Output:caffein

### Addition of 2 matrices in java

Let's see a simple example that adds two matrices.

1. class Testarray5{
2. public static void main(String args[]){
3. //creating two matrices
4. int a[][]={{1,3,4},{3,4,5}};
5. int b[][]={{1,3,4},{3,4,5}};
7. //creating another matrix to store the sum of two matrices
8. int c[][]=new int[2][3];
10. //adding and printing addition of 2 matrices
11. for(int i=0;i<2;i++){
12. for(int j=0;j<3;j++){
13. c[i][j]=a[i][j]+b[i][j];
14. System.out.print(c[i][j]+" ");
15. }
16. System.out.println();//new line
17. }
19. }}

[Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=Testarray5)

Output:2 6 8

6 8 10

# Java If-else Statement

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

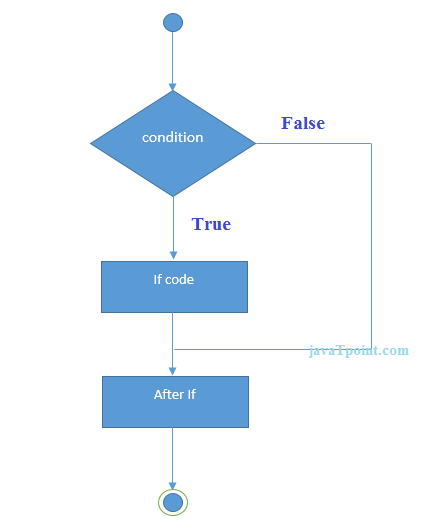
* if statement
* if-else statement
* nested if statement
* if-else-if ladder

## Java IF Statement

The Java if statement tests the condition. It executes the if block if condition is true.

Syntax:

1. if(condition){
2. //code to be executed
3. }



Example:

1. public class IfExample {
2. public static void main(String[] args) {
3. int age=20;
4. if(age>18){
5. System.out.print("Age is greater than 18");
6. }
7. }
8. }

Output:

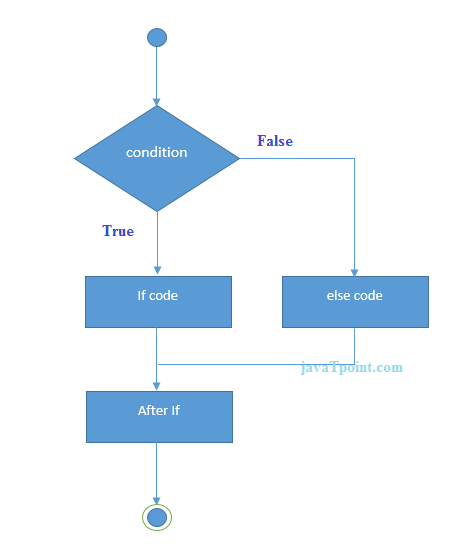
Age is greater than 18

## 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:

1. if(condition){
2. //code if condition is true
3. }else{
4. //code if condition is false
5. }



Example:

1. public class IfElseExample {
2. public static void main(String[] args) {
3. int number=13;
4. if(number%2==0){
5. System.out.println("even number");
6. }else{
7. System.out.println("odd number");
8. }
9. }
10. }

Output:

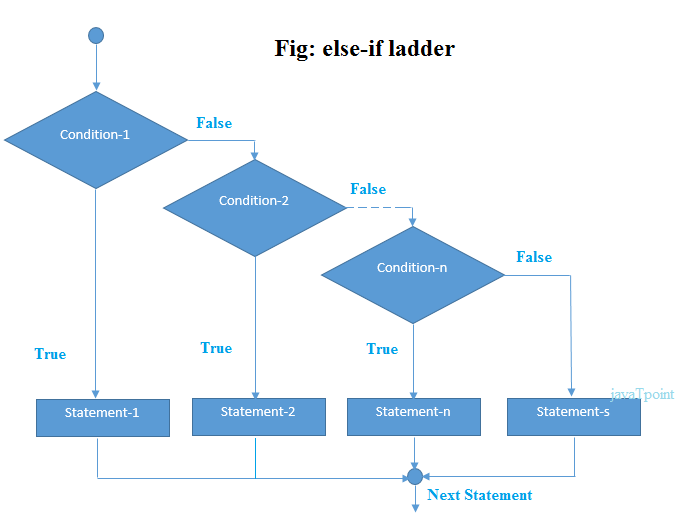
odd number

## Java IF-else-if ladder Statement

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

Syntax:

1. if(condition1){
2. //code to be executed if condition1 is true
3. }else if(condition2){
4. //code to be executed if condition2 is true
5. }
6. else if(condition3){
7. //code to be executed if condition3 is true
8. }
9. ...
10. else{
11. //code to be executed if all the conditions are false
12. }



Example:

1. public class IfElseIfExample {
2. public static void main(String[] args) {
3. int marks=65;
5. if(marks<50){
6. System.out.println("fail");
7. }
8. else if(marks>=50 && marks<60){
9. System.out.println("D grade");
10. }
11. else if(marks>=60 && marks<70){
12. System.out.println("C grade");
13. }
14. else if(marks>=70 && marks<80){
15. System.out.println("B grade");
16. }
17. else if(marks>=80 && marks<90){
18. System.out.println("A grade");
19. }else if(marks>=90 && marks<100){
20. System.out.println("A+ grade");
21. }else{
22. System.out.println("Invalid!");
23. }
24. }
25. }

Output:

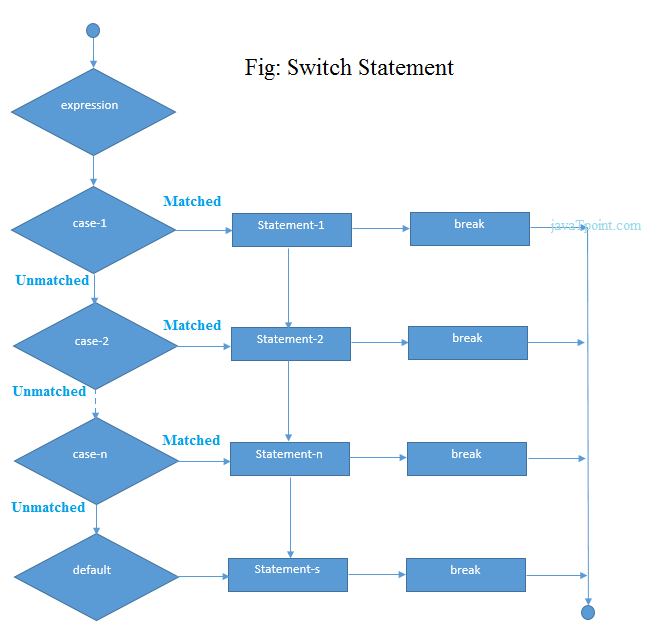
C grade

# Java Switch Statement

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

Syntax:

1. switch(expression){
2. case value1:
3. //code to be executed;
4. break;  //optional
5. case value2:
6. //code to be executed;
7. break;  //optional
8. ......
10. default:
11. code to be executed if all cases are not matched;
12. }



Example:

1. public class SwitchExample {
2. public static void main(String[] args) {
3. int number=20;
4. switch(number){
5. case 10: System.out.println("10");break;
6. case 20: System.out.println("20");break;
7. case 30: System.out.println("30");break;
8. default:System.out.println("Not in 10, 20 or 30");
9. }
10. }
11. }

Output:

20

# 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.

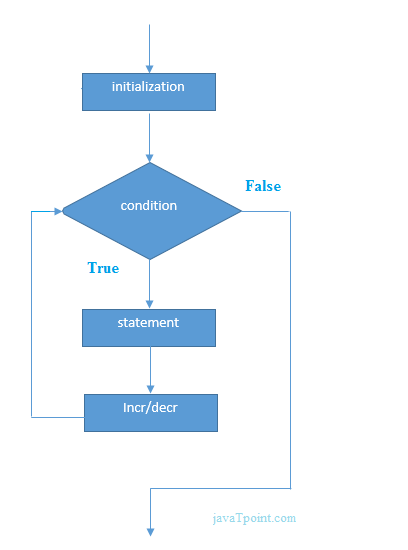
* Simple For Loop
* For-each or Enhanced For Loop
* Labeled For Loop

## Java Simple For Loop

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

Syntax:

1. for(initialization;condition;incr/decr){
2. //code to be executed
3. }



## 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:

1. for(Type var:array){
2. //code to be executed
3. }

## 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:

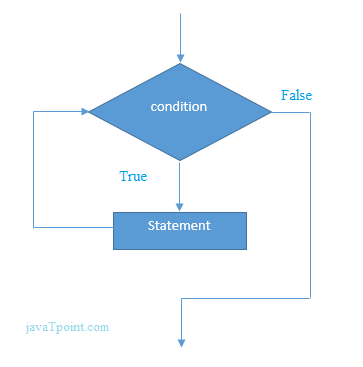
1. labelname:
2. for(initialization;condition;incr/decr){
3. //code to be executed
4. }

# 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:

1. while(condition){
2. //code to be executed
3. }



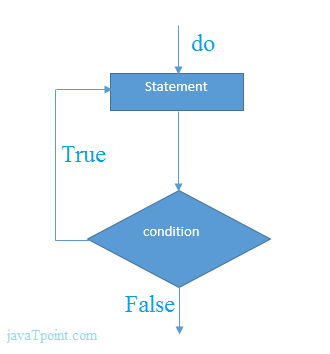
# Java do-while Loop

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:

1. do{
2. //code to be executed
3. }while(condition);

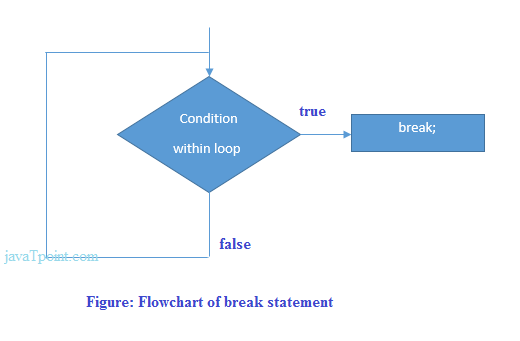


# Java Break Statement

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:

1. jump-statement;
2. 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:

1. jump-statement;
2. continue;

# 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:

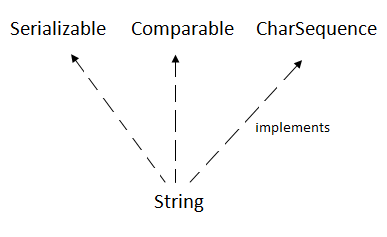
1. char[] ch={'j','a','v','a','t','p','o','i','n','t'};
2. String s=new String(ch);

is same as:

1. String s="javatpoint";

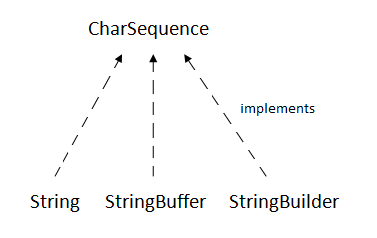
Java String class provides a lot of methods to perform operations on string such as compare(), concat(), equals(), split(), length(), replace(), compareTo(), intern(), substring() etc.

The java.lang.String class implements *Serializable*, *Comparable* and *CharSequence* interfaces.



## CharSequence Interface

The CharSequence interface is used to represent sequence of characters. It is implemented by String, StringBuffer and StringBuilder classes. It means, we can create string in java by using these 3 classes.



The java String is immutable i.e. it cannot be changed. Whenever we change any string, a new instance is created. For mutable string, you can use StringBuffer and StringBuilder classes.

We will discuss about immutable string later. Let's first understand what is string in java and how to create the string object.

### What is String in java

Generally, string is a sequence of characters. But in java, string is an object that represents a sequence of characters. The java.lang.String class is used to create string object.

### How to create String object?

|  |
| --- |
| There are two ways to create String object:   1. By string literal 2. By new keyword |

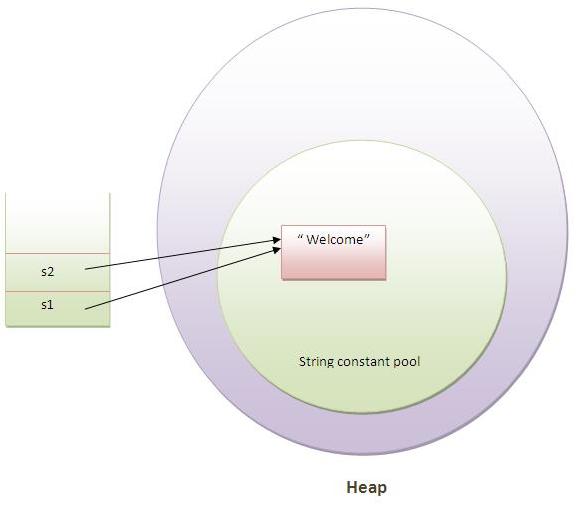
### 1) String Literal

Java String literal is created by using double quotes. For Example:

1. String s="welcome";

Each time you create a string literal, the JVM checks the string constant pool first. If the string already exists in the pool, a reference to the pooled instance is returned. If string doesn't exist in the pool, a new string instance is created and placed in the pool. For example:

1. String s1="Welcome";
2. String s2="Welcome";//will not create new instance



In the above example only one object will be created. Firstly JVM will not find any string object with the value "Welcome" in string constant pool, so it will create a new object. After that it will find the string with the value "Welcome" in the pool, it will not create new object but will return the reference to the same instance.

#### Note: String objects are stored in a special memory area known as string constant pool.

### Why java uses concept of string literal?

To make Java more memory efficient (because no new objects are created if it exists already in string constant pool).

### 2) By new keyword

1. String s=new String("Welcome");//creates two objects and one reference variable

In such case, JVM will create a new string object in normal(non pool) heap memory and the literal "Welcome" will be placed in the string constant pool. The variable s will refer to the object in heap(non pool).

### Java String Example

1. public class StringExample{
2. public static void main(String args[]){
3. String s1="java";//creating string by java string literal
4. char ch[]={'s','t','r','i','n','g','s'};
5. String s2=new String(ch);//converting char array to string
6. String s3=new String("example");//creating java string by new keyword
7. System.out.println(s1);
8. System.out.println(s2);
9. System.out.println(s3);
10. }}

[Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=StringExample)

java

strings

example

### Java String class methods

The java.lang.String class provides many useful methods to perform operations on sequence of char values.

|  |  |  |
| --- | --- | --- |
| No. | Method | Description |
| 1 | [char charAt(int index)](http://www.javatpoint.com/java-string-charat) | returns char value for the particular index |
| 2 | [int length()](http://www.javatpoint.com/java-string-length) | returns string length |
| 3 | [static String format(String format, Object... args)](http://www.javatpoint.com/java-string-format) | returns formatted string |
| 4 | [static String format(Locale l, String format, Object... args)](http://www.javatpoint.com/java-string-format) | returns formatted string with given locale |
| 5 | [String substring(int beginIndex)](http://www.javatpoint.com/java-string-substring) | returns substring for given begin index |
| 6 | [String substring(int beginIndex, int endIndex)](http://www.javatpoint.com/java-string-substring) | returns substring for given begin index and end index |
| 7 | [boolean contains(CharSequence s)](http://www.javatpoint.com/java-string-contains) | returns true or false after matching the sequence of char value |
| 8 | [static String join(CharSequence delimiter, CharSequence... elements)](http://www.javatpoint.com/java-string-join) | returns a joined string |
| 9 | [static String join(CharSequence delimiter, Iterable<? extends CharSequence> elements)](http://www.javatpoint.com/java-string-join) | returns a joined string |
| 10 | [boolean equals(Object another)](http://www.javatpoint.com/java-string-equals) | checks the equality of string with object |
| 11 | [boolean isEmpty()](http://www.javatpoint.com/java-string-isempty) | checks if string is empty |
| 12 | [String concat(String str)](http://www.javatpoint.com/java-string-concat) | concatinates specified string |
| 13 | [String replace(char old, char new)](http://www.javatpoint.com/java-string-replace) | replaces all occurrences of specified char value |
| 14 | [String replace(CharSequence old, CharSequence new)](http://www.javatpoint.com/java-string-replace) | replaces all occurrences of specified CharSequence |
| 15 | [static String equalsIgnoreCase(String another)](http://www.javatpoint.com/java-string-equalsignorecase) | compares another string. It doesn't check case. |
| 16 | [String[] split(String regex)](http://www.javatpoint.com/java-string-split) | returns splitted string matching regex |
| 17 | [String[] split(String regex, int limit)](http://www.javatpoint.com/java-string-split) | returns splitted string matching regex and limit |
| 18 | [String intern()](http://www.javatpoint.com/java-string-intern) | returns interned string |
| 19 | [int indexOf(int ch)](http://www.javatpoint.com/java-string-indexof) | returns specified char value index |
| 20 | [int indexOf(int ch, int fromIndex)](http://www.javatpoint.com/java-string-indexof) | returns specified char value index starting with given index |
| 21 | [int indexOf(String substring)](http://www.javatpoint.com/java-string-indexof) | returns specified substring index |
| 22 | [int indexOf(String substring, int fromIndex)](http://www.javatpoint.com/java-string-indexof) | returns specified substring index starting with given index |
| 23 | [String toLowerCase()](http://www.javatpoint.com/java-string-tolowercase) | returns string in lowercase. |
| 24 | [String toLowerCase(Locale l)](http://www.javatpoint.com/java-string-tolowercase) | returns string in lowercase using specified locale. |
| 25 | [String toUpperCase()](http://www.javatpoint.com/java-string-touppercase) | returns string in uppercase. |
| 26 | [String toUpperCase(Locale l)](http://www.javatpoint.com/java-string-touppercase) | returns string in uppercase using specified locale. |
| 27 | [String trim()](http://www.javatpoint.com/java-string-trim) | removes beginning and ending spaces of this string. |
| 28 | [static String valueOf(int value)](http://www.javatpoint.com/java-string-valueof) | converts given type into string. It is overloaded. |