**Java Tutorial**

**What is Java?**

Java is a **programming language** and a **platform**.

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

**Platform**: Any hardware or software environment in which a program runs, is known as a platform. Since Java has its own runtime environment (JRE) and API, it is called platform.

**Java Example**

Let's have a quick look at java programming example. A detailed description of hello java example is given in next page.

**class** Simple{

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

     System.out.println("Hello Java");

    }

}

## Where it is used?

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

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

## Types of Java Applications

There are mainly 4 type of applications that can be created using java programming:

#### 1) Standalone Application

It is also known as desktop application or window-based application. An application that we need to install on every machine such as media player, antivirus etc. AWT and Swing are used in java for creating standalone applications.

#### 2) Web Application

An application that runs on the server side and creates dynamic page, is called web application. Currently, servlet, jsp, struts, jsf etc. technologies are used for creating web applications in java.

#### 3) Enterprise Application

An application that is distributed in nature, such as banking applications etc. It has the advantage of high level security, load balancing and clustering. In java, EJB is used for creating enterprise applications.

#### 4) Mobile Application

An application that is created for mobile devices. Currently Android and Java ME are used for creating mobile applications.

**History of Java**

1. Brief history of Java
2. Java Version History

**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.

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

1. Features of Java
   1. Simple
   2. Object-Oriented
   3. Platform Independent
   4. secured
   5. Robust
   6. Architecture Neutral
   7. Portable
   8. High Performance
   9. Distributed
   10. Multi-threaded

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

### 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 hardware 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's a software-based platform that runs on top of other hardware-based platforms.It has two components:   1. Runtime Environment 2. API(Application Programming Interface) |

|  |
| --- |
| java is platform independentJava 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 * Programs run inside virtual machine sandbox. |

|  |  |
| --- | --- |
| how java is secured | how java is secured |

|  |
| --- |
| * **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 set. |

### Portable

|  |
| --- |
| We may carry the java bytecode to any platform. |

### 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 shares the same memory. Threads are important for multi-media, Web applications etc.

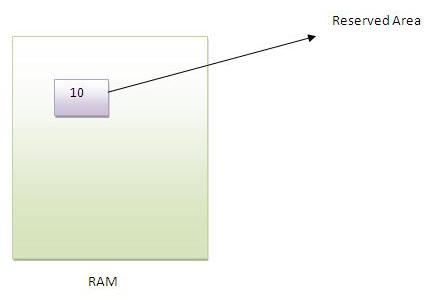
# Variable and Datatype in Java

1. Variable
2. Types of Variable
3. Data Types in Java

|  |
| --- |
| In this page, we will learn about the variable and java data types. Variable is a name of memory location. There are three types of variables: local, instance and static. There are two types of datatypes in java, primitive and non-primitive. |

### Variable

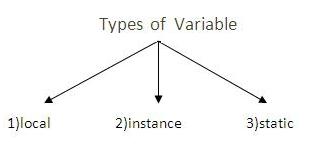
Variable is name of reserved area allocated in memory.



int data=50;//Here data is variable

### Types of Variable

|  |
| --- |
| There are three types of variables in java   * local variable * instance variable * static variable |



#### Local Variable

|  |
| --- |
| A variable that is declared inside the method is called local variable. |

#### Instance Variable

|  |
| --- |
| A variable that is declared inside the class but outside the method is called instance variable . It is not declared as static. |

#### 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

class A{

int data=50;//instance variable

static int m=100;//static variable

void method(){

int n=90;//local variable

}

}//end of class

## Data Types in Java

|  |
| --- |
| 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 ?

because java uses unicode system rather than ASCII code system. \u0000 is the lowest range of unicode system.To get detail about Unicode see below.

# Operators in java

**Operator** in java is a symbol that is used to perform operations. There are many types of operators in java such as unary operator, arithmetic operator, relational operator, shift operator, bitwise operator, ternary operator and assignment operator.

|  |  |
| --- | --- |
| **Operators** | **Precedence** |
| Postfix | expr++ expr-- |
| Unary | ++expr --expr +expr -expr ~ ! |
| Multiplicative | \* / % |
| Additive | + - |
| Shift | << >> >>> |
| Relational | < > <= >= instanceof |
| Equality | == != |
| bitwise AND | & |
| bitwise exclusive OR | ^ |
| bitwise inclusive OR | | |
| logical AND | && |
| logical OR | || |
| Ternary | ? : |
| Assignment | = += -= \*= /= %= &= ^= |= <<= >>= >>>= |

**Java If-else Statement**

The Java *if statement* is used to test the condition. It returns *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

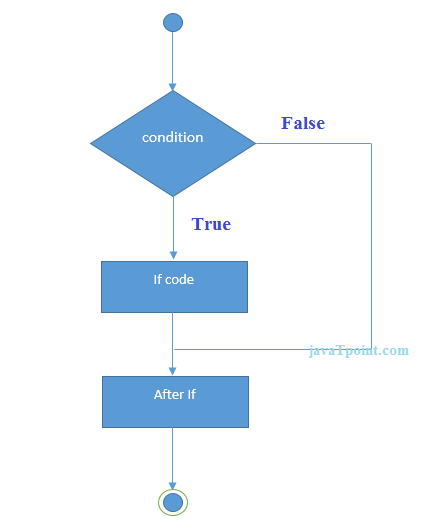
The if statement tests the condition. It executes the if statement if condition is true.

**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 if-else statement also tests the condition. It executes the *if block* if condition is true otherwise *else block*.

**Syntax:**

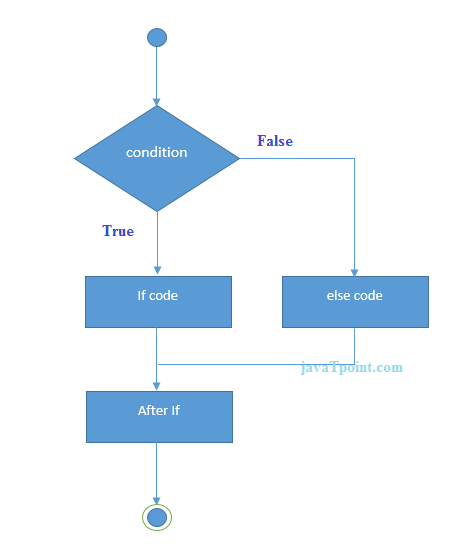
**if**(condition){

//code if condition is true

}**else**{

//code if condition is false

}



**Example:**

**public** **class** IfElseExample {

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

**int** number=13;

**if**(number%2==0){

        System.out.println("even number");

    }**else**{

        System.out.println("odd number");

    }

}

}

Output:

odd number

**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){

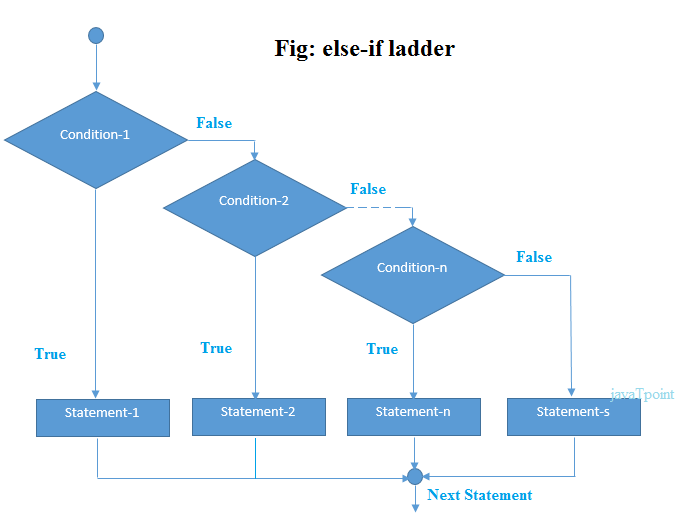
//code to be executed if condition3 is true

}

**else**{

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

}



**Example:**

**public** **class** IfElseIfExample {

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

**int** marks=65;

**if**(marks<50){

        System.out.println("fail");

    }

**else** **if**(marks>=50 && marks<60){

        System.out.println("D grade");

    }

**else** **if**(marks>=60 && marks<70){

        System.out.println("C grade");

    }

**else** **if**(marks>=70 && marks<80){

        System.out.println("B grade");

    }

**else** **if**(marks>=80 && marks<90){

        System.out.println("A grade");

    }**else** **if**(marks>=90 && marks<100){

        System.out.println("A+ grade");

    }**else**{

        System.out.println("Invalid!");

    }

}

}

Output:

C grade

**Java Switch Statement**

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

**Syntax:**

**switch**(expression){

**case** value1:

 //code to be executed;

**break**;  //optional

**case** value2:

 //code to be executed;

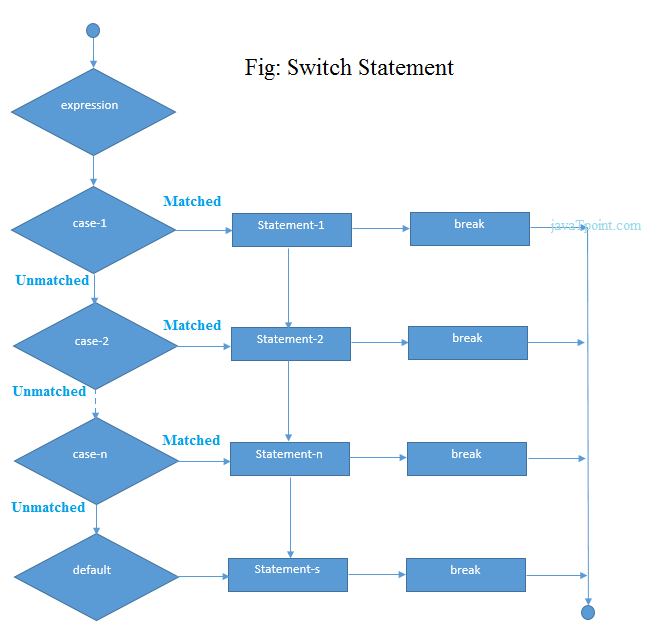
**break**;  //optional

......

**default**:

 code to be executed **if** all cases are not matched;

}



**Example:**

**public** **class** SwitchExample {

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

**int** number=20;

**switch**(number){

**case** 10: System.out.println("10");**break**;

**case** 20: System.out.println("20");**break**;

**case** 30: System.out.println("30");**break**;

**default**:System.out.println("Not in 10, 20 or 30");

    }

}

}

Output:

20

**Java Switch Statement is fall-through**

The java switch statement is fall-through. It means it executes all statement after first match if break statement is not used with switch cases.

**Example:**

**public** **class** SwitchExample2 {

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

**int** number=20;

**switch**(number){

**case** 10: System.out.println("10");

**case** 20: System.out.println("20");

**case** 30: System.out.println("30");

**default**:System.out.println("Not in 10, 20 or 30");

    }

}

}

Output:

20

30

Not in 10, 20 or 30

**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**

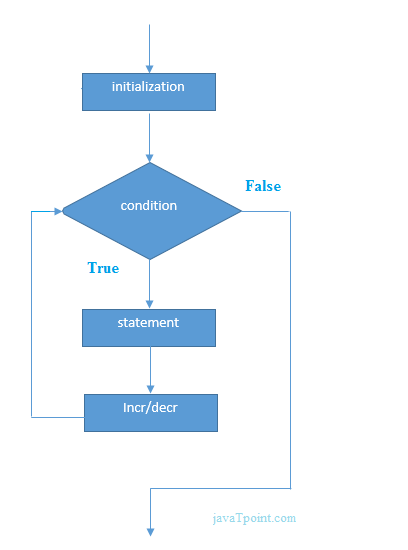
The simple for loop is same as C/C++. We can initialize variable, check condition and increment/decrement value.

**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);

    }

}

}

Output:

1

2

3

4

5

6

7

8

9

10

**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);

    }

}

}

Output:

12

23

44

56

78

**Java While Loop**

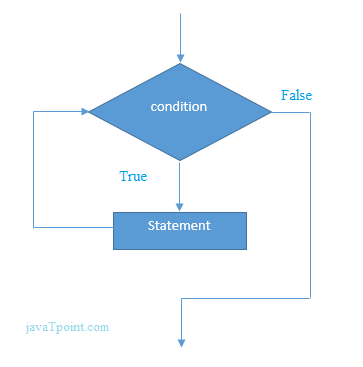
The Java *while loop* is used to iterate a part of the program several times. If the number of iteration is not fixed, it is recommended to use while loop.

**Syntax:**

**while**(condition){

//code to be executed

}



**Example:**

**public** **class** WhileExample {

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

**int** i=1;

**while**(i<=10){

        System.out.println(i);

    i++;

    }

}

}

Output:

1

2

3

4

5

6

7

8

9

10

**Java Infinitive While Loop**

If you pass **true** in the while loop, it will be infinitive while loop.

**Syntax:**

**while**(**true**){

//code to be executed

}

**Example:**

**public** **class** WhileExample2 {

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

**while**(**true**){

        System.out.println("infinitive while loop");

    }

}

}

Output:

infinitive while loop

infinitive while loop

infinitive while loop

infinitive while loop

infinitive while loop

ctrl+c

**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 while loop.

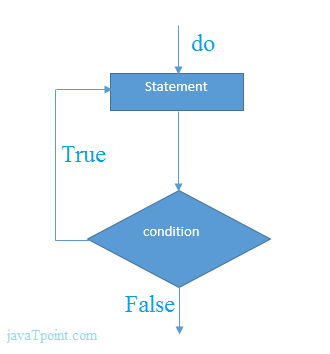
It is executed at least once because condition is checked after loop body.

**Syntax:**

**do**{

//code to be executed

}**while**(condition);



**Example:**

**public** **class** DoWhileExample {

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

**int** i=1;

**do**{

        System.out.println(i);

    i++;

    }**while**(i<=10);

}

}

Output:

1

2

3

4

5

6

7

8

9

10

**Java Infinitive do-while Loop**

If you pass **true** in the do-while loop, it will be infinitive do-while loop.

**Syntax:**

**while**(**true**){

//code to be executed

}

**Example:**

**public** **class** DoWhileExample2 {

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

**do**{

        System.out.println("infinitive do while loop");

    }**while**(**true**);

}

}

Output:

infinitive do while loop

infinitive do while loop

infinitive do while loop

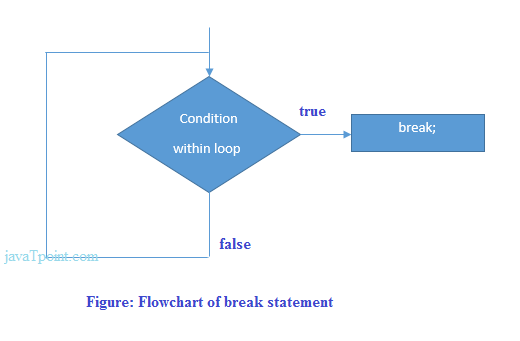
ctrl+c

**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 Break Statement with Loop**

**Example:**

**public** **class** BreakExample {

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

**for**(**int** i=1;i<=10;i++){

**if**(i==5){

**break**;

        }

        System.out.println(i);

    }

}

}

Output:

1

2

3

4

**Java Break Statement with Inner Loop**

It breaks inner loop only if you use break statement inside the inner loop.

**Example:**

**public** **class** BreakExample2 {

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

**for**(**int** i=1;i<=3;i++){

**for**(**int** j=1;j<=3;j++){

**if**(i==2&&j==2){

**break**;

                        }

                        System.out.println(i+" "+j);

                    }

            }

}

}

Output:

1 1

1 2

1 3

2 1

3 1

3 2

3 3

**Java Break Statement with Switch**

To understand the example of break with switch statement, please visit here: Java Switch Statement.

**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 Continue Statement Example**

**Example:**

**public** **class** ContinueExample {

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

**for**(**int** i=1;i<=10;i++){

**if**(i==5){

**continue**;

        }

        System.out.println(i);

    }

}

}

Output:

1

2

3

4

6

7

8

9

10

**Java Continue Statement with Inner Loop**

It continues inner loop only if you use continue statement inside the inner loop.

**Example:**

**public** **class** ContinueExample2 {

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

**for**(**int** i=1;i<=3;i++){

**for**(**int** j=1;j<=3;j++){

**if**(i==2&&j==2){

**continue**;

                        }

                        System.out.println(i+" "+j);

                    }

            }

}

}

Output:

1 1

1 2

1 3

2 1

2 3

3 1

3 2

3 3

# 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

copy to clipboard

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



### Instantiation of an Array in java

copy to clipboard

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.

copy to clipboard

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**

Output: 10

20

70

40

50

## Declaration, Instantiation and Initialization of Java Array

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

copy to clipboard

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



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

copy to clipboard

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**

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.

copy to clipboard

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**

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

copy to clipboard

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



### Example to instantiate Multidimensional Array in java

copy to clipboard

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



### Example to initialize Multidimensional Array in java

copy to clipboard

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.

copy to clipboard

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**

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.

copy to clipboard

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**

Output:I

### Copying a java array

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

### Syntax of arraycopy method

copy to clipboard

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



### Example of arraycopy method

copy to clipboard

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**

Output:caffein

### Addition of 2 matrices in java

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

copy to clipboard

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**

Output:2 6 8

6 8 10

# 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:

copy to clipboard

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



is same as:

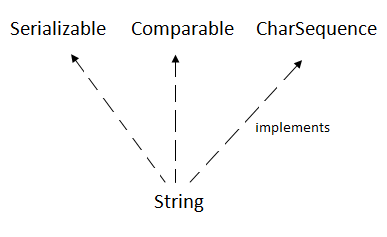
copy to clipboard

1. String s="javatpoint";



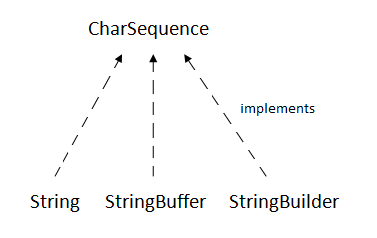
**Java String** class provides a lot of methods to perform operations on string such as compare(), concat(), equals(), split(), length(), replace(), compareTo(), intern(), substring() etc.

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:

copy to clipboard

1. String s="welcome";

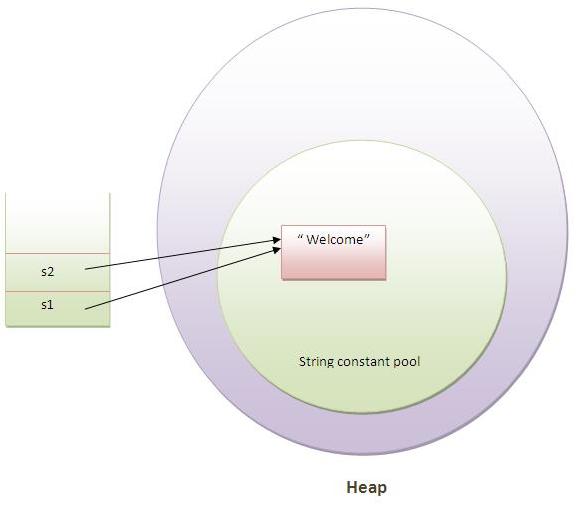


Each time you create a string literal, the JVM checks the string constant pool first. If the string already exists in the pool, a reference to the pooled instance is returned. If string doesn't exist in the pool, a new string instance is created and placed in the pool. For example:

copy to clipboard

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

copy to clipboard

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

copy to clipboard

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**

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) | returns char value for the particular index |
| 2 | int length() | returns string length |
| 3 | static String format(String format, Object... args) | returns formatted string |
| 4 | static String format(Locale l, String format, Object... args) | returns formatted string with given locale |
| 5 | String substring(int beginIndex) | returns substring for given begin index |
| 6 | String substring(int beginIndex, int endIndex) | returns substring for given begin index and end index |
| 7 | boolean contains(CharSequence s) | returns true or false after matching the sequence of char value |
| 8 | static String join(CharSequence delimiter, CharSequence... elements) | returns a joined string |
| 9 | static String join(CharSequence delimiter, Iterable<? extends CharSequence> elements) | returns a joined string |
| 10 | boolean equals(Object another) | checks the equality of string with object |
| 11 | boolean isEmpty() | checks if string is empty |
| 12 | String concat(String str) | concatinates specified string |
| 13 | String replace(char old, char new) | replaces all occurrences of specified char value |
| 14 | String replace(CharSequence old, CharSequence new) | replaces all occurrences of specified CharSequence |
| 15 | String trim() | returns trimmed string omitting leading and trailing spaces |
| 16 | String split(String regex) | returns splitted string matching regex |
| 17 | String split(String regex, int limit) | returns splitted string matching regex and limit |
| 18 | String intern() | returns interned string |
| 19 | int indexOf(int ch) | returns specified char value index |
| 20 | int indexOf(int ch, int fromIndex) | returns specified char value index starting with given index |
| 21 | int indexOf(String substring) | returns specified substring index |
| 22 | int indexOf(String substring, int fromIndex) | returns specified substring index starting with given index |
| 23 | String toLowerCase() | returns string in lowercase. |
| 24 | String toLowerCase(Locale l) | returns string in lowercase using specified locale. |
| 25 | String toUpperCase() | returns string in uppercase. |
| 26 | String toUpperCase(Locale l) | returns string in uppercase using specified locale. |
| 27 | String trim() | removes beginning and ending spaces of this string. |
| 28 | static String valueOf(int value) | converts given type into string. It is overloaded. |

Do You Know ?

* Why String objects are immutable?
* How to create an immutable class?
* What is string constant pool?
* What code is written by the compiler if you concat any string by + (string concatenation operator)?
* What is the difference between StringBuffer and StringBuilder class?

What we will learn in String Handling ?

* Concept of String
* Immutable String
* String Comparison
* String Concatenation
* Concept of Substring
* String class methods and its usage
* StringBuffer class
* StringBuilder class
* Creating Immutable class
* toString() method
* StringTokenizer class

# Exception Handling in Java

1. Exception Handling
2. Advantage of Exception Handling
3. Hierarchy of Exception classes
4. Types of Exception
5. Scenarios where exception may occur

The **exception handling in java** is one of the powerful *mechanism to handle the runtime errors* so that normal flow of the application can be maintained.

In this page, we will learn about java exception, its type and the difference between checked and unchecked exceptions.

### What is exception

**Dictionary Meaning:** Exception is an abnormal condition.

In java, exception is an event that disrupts the normal flow of the program. It is an object which is thrown at runtime.

### What is exception handling

Exception Handling is a mechanism to handle runtime errors such as ClassNotFound, IO, SQL, Remote etc.

### Advantage of Exception Handling

The core advantage of exception handling is **to maintain the normal flow of the application**. Exception normally disrupts the normal flow of the application that is why we use exception handling. Let's take a scenario:

copy to clipboard

1. statement 1;
2. statement 2;
3. statement 3;
4. statement 4;
5. statement 5;//exception occurs
6. statement 6;
7. statement 7;
8. statement 8;
9. statement 9;
10. statement 10;

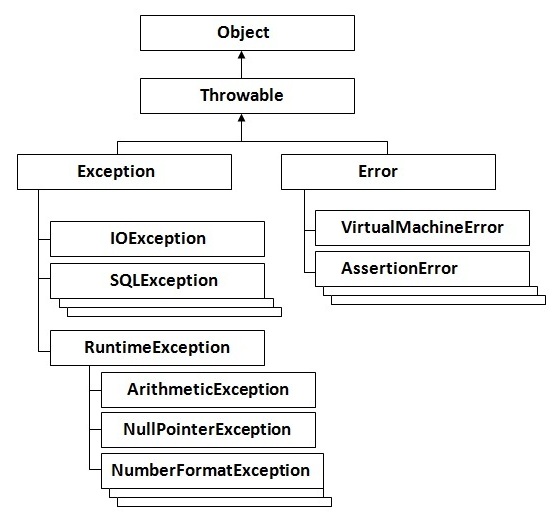


Suppose there is 10 statements in your program and there occurs an exception at statement 5, rest of the code will not be executed i.e. statement 6 to 10 will not run. If we perform exception handling, rest of the statement will be executed. That is why we use exception handling in java.

Do You Know ?

|  |
| --- |
| * What is the difference between checked and unchecked exceptions ? * What happens behind the code int data=50/0; ? * Why use multiple catch block ? * Is there any possibility when finally block is not executed ? * What is exception propagation ? * What is the difference between throw and throws keyword ? * What are the 4 rules for using exception handling with method overriding ? |

## Hierarchy of Java Exception classes



### Types of Exception

There are mainly two types of exceptions: checked and unchecked where error is considered as unchecked exception. The sun microsystem says there are three types of exceptions:

1. Checked Exception
2. Unchecked Exception
3. Error

## Difference between checked and unchecked exceptions

### 1) Checked Exception

The classes that extend Throwable class except RuntimeException and Error are known as checked exceptions e.g.IOException, SQLException etc. Checked exceptions are checked at compile-time.

### 2) Unchecked Exception

The classes that extend RuntimeException are known as unchecked exceptions e.g. ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc. Unchecked exceptions are not checked at compile-time rather they are checked at runtime.

### 3) Error

Error is irrecoverable e.g. OutOfMemoryError, VirtualMachineError, AssertionError etc.

### Common scenarios where exceptions may occur

There are given some scenarios where unchecked exceptions can occur. They are as follows:

### 1) Scenario where ArithmeticException occurs

If we divide any number by zero, there occurs an ArithmeticException.

copy to clipboard

1. int a=50/0;//ArithmeticException



### 2) Scenario where NullPointerException occurs

If we have null value in any variable, performing any operation by the variable occurs an NullPointerException.

copy to clipboard

1. String s=null;
2. System.out.println(s.length());//NullPointerException



### 3) Scenario where NumberFormatException occurs

The wrong formatting of any value, may occur NumberFormatException. Suppose I have a string variable that have characters, converting this variable into digit will occur NumberFormatException.

copy to clipboard

1. String s="abc";
2. int i=Integer.parseInt(s);//NumberFormatException



### 4) Scenario where ArrayIndexOutOfBoundsException occurs

If you are inserting any value in the wrong index, it would result ArrayIndexOutOfBoundsException as shown below:

copy to clipboard

1. int a[]=new int[5];
2. a[10]=50; //ArrayIndexOutOfBoundsException



## Java Exception Handling Keywords

There are 5 keywords used in java exception handling.

1. try
2. catch
3. finally
4. throw
5. throws

Next TopicTry catch block