Java

It is one of the programming language or technology used for developing web applications. Using this technology you can develop distributed application. A Java language developed at SUN Micro Systems in the year 1995 under the guidance of James Gosling and their team. In other word It is a programming language suitable for the development of web applications. It is also used for developing desktop and mobile application.

Overview of Java

Java is one of the programming language or technology used for developing web applications. Java language developed at SUN Micro Systems in the year 1995 under the guidance of James Gosling and there team. Originally SUN Micro Systems is one of the Academic university (Standford University Network)

Whatever the software developed in the year 1990, SUN Micro Systems has released on the name of oak, which is original name of java (scientifically oak is one of the tree name). The OAK has taken 18 months to develop.

The oak is unable to fulfill all requirements of the industry. So James Gosling again reviews this oak and released with the name of java in the year 1995. Scientifically java is one of the coffee seed name.

Java divided into three categories, they are

- J2SE (Java 2 Standard Edition)
- J2EE (Java 2 Enterprise Edition)
- J2ME (Java 2 Micro or Mobile Edition)

J2SE

J2SE is used for developing client side applications.

J2EE

J2EE is used for developing server side applications.

J2ME

J2ME is used for developing mobile or wireless application by making use of a predefined protocol called WAP(wireless Access / Application protocol).

Basic Points of Java

Java is a platform independent, more powerful, secure, high performance, multithreaded programming language. Here we discuss some points related to Java.

Define byte

Byte code is the set of optimized instructions generated during compilation phase and it is more powerful than ordinary pointer code.

Define JRE

The **Java Runtime Environment (JRE)** is part of the Java Development Kit (JDK). It contains a set of libraries and tools for developing Java application. The Java Runtime Environment provides the minimum requirements for executing a Java application.

Define JVM

JVM is set of programs developed by sun Micro System and supplied as a part of the JDK for reading line by line line of byte code and it converts into a native understanding form of operating system. The Java language is one of the compiled and interpreted programming language.

Garbage Collector

The **Garbage Collector** is the system Java program which runs in the background along with a regular Java program to collect un-Referenced (unused) memory space for improving the performance of our applications.

Note: Java programming does not support destructor concept in place of destructor, we have garbage collector program.

Define an API

An **API (Application Programming Interface)** is a collection of packages, a package is the collection of classes, interfaces and sub-packages. A sub-package is a collection of classes, Interfaces and sub sub packages etc.

Java programming contains user friendly syntax so that we can develop effective applications. in other words if any language is providing user friendly syntax, we can develop error free applications.

Definition of JIT

JIT is the set of programs developed by SUN Micro System and added as a part of JVM, to speed up the interpretation phase.

The major drawback in this architecture is if any problem occurred on server system that will be reflected on every client system.

Distributed applications

In this scenario multiple client system are depends on multiple server system so that even problem occurred in one server will never be reflected on any client system.

Note: In this architecture same application is distributed in multiple server system.

Java is a very powerful language can be used to developed both client server architecture and distributed architecture based application.

Features of Java

Features of a language are nothing but the set of services or facilities provided by the language vendors to the industry programmers. Some important **features of java** are;

Important Features of Java

- Simple
- Platform Independent
- Architectural Neutral
- Portable
- Multi Threading
- Distributed
- Networked
- Robust
- Dynamic
- Secured
- High Performance
- Interpreted
- Object Oriented

1. Simple

It is simple because of the following factors:

- It is **free from pointer** due to this execution time of application is improved. [Whenever we write a Java program without pointers then internally it is converted into the equivalent pointer program].
- It has **Rich set of API** (application protocol interface).
- It hs **Garbage Collector** which is always used to collect un-Referenced (unused) Memory location for improving performance of a Java program.

- It contains user friendly syntax for developing any applications.

2. Platform Independent

A program or technology is said to be platform independent if and only if which can run on all available operating systems with respect to its development and compilation. (Platform represents O.S).

3. Architectural Neutral

Architecture represents processor.

A Language or Technology is said to be Architectural neutral which can run on any available processors in the real world without considering their development and compilation.

The languages like C, CPP are treated as architectural dependent.

4. Portable

If any language supports platform independent and architectural neutral feature known as portable. The languages like C, CPP, Pascal are treated as non-portable language. It is a portable language. According to SUN microsystem.

5. Multithreaded

A flow of control is known as a threa. When any Language executes multiple thread at a time that language is known as multithreaded e. It is multithreaded.

6. Distributed

Using this language we can create distributed applications. RMI and EJB are used for creating distributed applications. In distributed application multiple client system depends on multiple server systems so that even problem occurred in one server will never be reflected on any client system.

Note: In this architecture same application is distributed in multiple server system.

7. Networked

It is mainly designed for web based applications, J2EE is used for developing network based applications.

8. Robust

Simply means of Robust are strong. It is robust or strong Programming Language because of its capability to handle Run-time Error, automatic garbage collection, the lack of pointer concept, Exception Handling. All these points make It robust Language.

9. Dynamic

It supports Dynamic memory allocation due to this memory wastage is reduce and improve performance of the application. The process of allocating the memory space to the input of the program at a run-time is known as dynamic memory allocation, To programming to allocate memory space by dynamically we use an operator called 'new' 'new' operator is known as dynamic memory allocation operator.

10. Secure

It is a more secure language compared to other language; In this language, all code is covered in byte code after compilation which is not readable by human.

11. High performance

It have high performance because of following reasons;

- This language **uses Bytecode** which is faster than ordinary pointer code so Performance of this language is high.
- **Garbage collector**, collect the unused memory space and improve the performance of the application.
- It has **no pointers** so that using this language we can develop an application very easily.
- It **support multithreading**, because of this time consuming process can be reduced to executing the program.

12. Interpreted

It is one of the highly interpreted programming languages.

13. Object Oriented

It supports OOP's concepts because of this it is most secure language, for this topic you can read our oop's concepts in detail.

Object and class in Java

Object is the physical as well as logical entity where as class is the only logical entity.

Class: Class is a blue print which is containing only list of variables and method and no memory is allocated for them. A class is a group of objects that has common properties.

A class in java contains:

- Data Member

- Method
- Constructor
- Block
- Class and Interface

Object: Object is a instance of class, object has state and behaviors.

An Object in java has three characteristics:

- State
- Behavior
- Identity

State: Represents data (value) of an object.

Behavior: Represents the behavior (functionality) of an object such as deposit, withdraw etc.

Identity: Object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. But,it is used internally by the JVM to identify each object uniquely.

Class is also can be used to achieve user defined data types.

Difference between Class and Object in Java

	Class	**Object**
1	Class is a container which collection of variables and methods.	object is a instance of class
2	No memory is allocated at the time of declaration	Sufficient memory space will be allocated for all the variables of class at the time of declaration.
3	One class definition should exist only once in the program.	For one class multiple objects can be created.

Data Type in Java

Datatype is a spacial keyword used to allocate sufficient memory space for the data, in other words Data type is used for representing the data in main memory (RAM) of the computer.

In general every programming language is containing three categories of data types. They are

- Fundamental or primitive data types
- Derived data types
- User defined data types.

Primitive data types

Primitive data types are those whose variables allows us to store only one value but they never allows us to store multiple values of same type. This is a data type whose variable can hold maximum one value at a time.

y one value at a time because it is primitive type variable.

Derived data types

Derived data types are those whose variables allow us to store multiple values of same type. But they never allows to store multiple values of different types. These are the data type whose variable can hold more than one value of similar type. In general derived data type can be achieve using array.

Here derived data type store only same type of data at a time not store integer, character and string at same time.

User defined data types

User defined data types are those which are developed by programmers by making use of appropriate features of the language.

User defined data types related variables allows us to store multiple values either of same type or different type or both. This is a data type whose variable can hold more than one value of dissimilar type, in java it is achieved using class concept.

Note: In java both derived and user defined data type combined name as reference data type.

Float category data types

Float category data type are used for representing float values. This category contains two data types, they are in the given table

Data Type	**Size**	**Range**	**Number of decimal places**
Float	4	+2147483647 to -2147483648	8
Double	8	+ 9.223*1018	16

Boolean category data types

Boolean category data type is used for representing or storing logical values is true or false. In java programming to represent Boolean values or logical values, we use a data type called Boolean.

Why Boolean data types take zero byte of memory ?

Boolean data type takes zero bytes of main memory space because Boolean data type of java implemented by Sun Micro System with a concept of flip - flop. A flip - flop is a general purpose register which stores one bit of information (one true and zero false).

Note: In C, C++ (Turbo) Boolean data type is not available for representing true false values but a true value can be treated as non-zero value and false values can be represented by zero

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

Variable Declaration Rules in Java

Variable is an identifier which holds data or another one variable is an identifier whose value can be changed at the execution time of program. Variable is an identifier which can be used to identify input data in a program.

Syntax

```
Variable_name = value;
```

Rules to declare a Variable

- Every variable name should start with either alphabets or underscore (_) or dollar ($) symbol.
- No space are allowed in the variable declarations.
- Except underscore (_) no special symbol are allowed in the middle of variable declaration
- Variable name always should exist in the left hand side of assignment operators.
- Maximum length of variable is 64 characters.
- No keywords should access variable name.

Note: Actually a variable also can start with ¥,¢, or any other currency sign.

Variable initialization

It is the process of storing user defined values at the time of allocation of memory space.

Variable assignment

Value is assigned to a variable if that is already declared or initialized.

Syntax

```
Variable_Name = value
int a = 100;
```

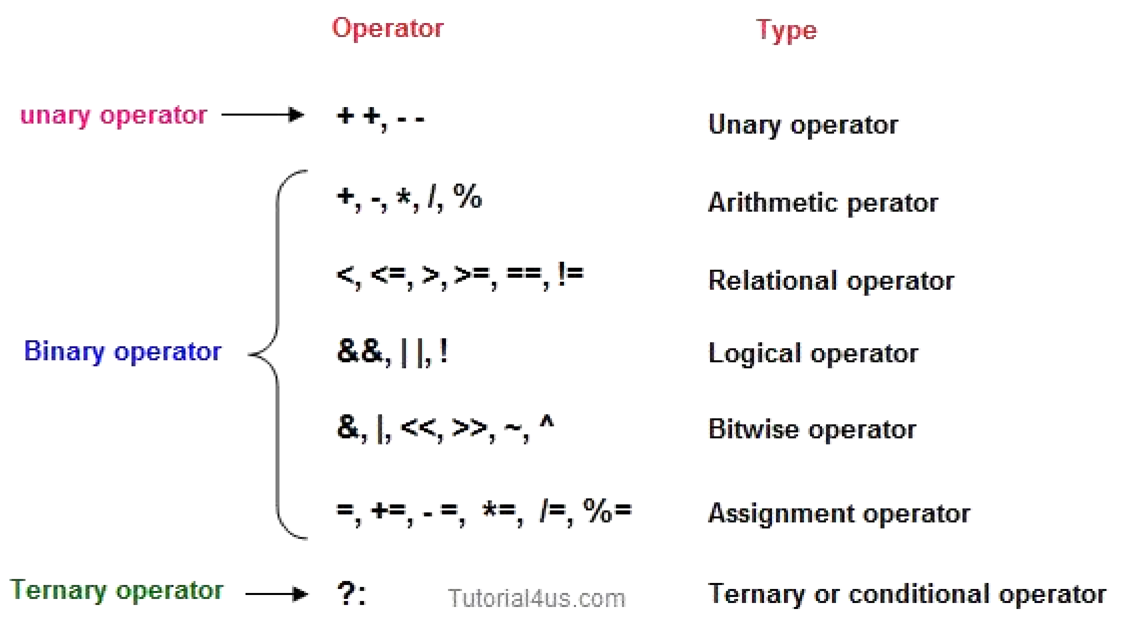
Syntax

```
int  a= 100;
int b;
b = 25;  //  ------>  direct assigned variable
b = a;    //  ------>  assigned value in term of variable
b = a+15;  //  ------>  assigned value as term of expression
```

Operators in Java

Operator is a special symbol that tells the compiler to perform specific mathematical or logical Operation. Java supports following lists of operators.

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Bitwise Operators
- Assignment Operators
- Ternary or Conditional Operators



Arithmetic Operators

Given table shows all the Arithmetic operator supported by Java 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%4	0

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
<=	A<=10	True
>	A>B	True
>=	A<=B	False
==	A== B	False
!=	A!=(-4)	True

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.

Operator	**Example (int A=8, B=3, C=-10)**	**Result**
&&	(A<B) && (B>C)	False
||	(B!=-C) || (A==B)	True
!	!(B<=-A)	True

Truth table of Logical Operator

C1	**C2**	**C1 && C2**	**C1 || C2**	**!C1**	**!C2**
T	T	T	T	F	F
T	F	F	T	F	T
F	T	F	T	T	F
F	F	F	F	T	T

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	

Ternary operator

If any operator is used on three operands or variable is known as ternary operator. It can be represented with " ?: "

Decision Making Statement in Java

Decision making statement statements is also called selection statement. That is depending on the condition block need to be executed or not which is decided by condition. If the condition is "true" statement block will be executed, if condition is "false" then statement block will not be executed. In java there are three types of decision making statement.

- if
- if-else
- switch

if-then Statement

if-then is most basic statement of Decision making statement. It tells to program to execute a certain part of code only if particular condition is true.

Syntax

```
if(condition)
 {
   Statement(s)
 }
```

if-else statement

In general it can be used to execute one block of statement among two blocks, in java language **if** and **else** are the keyword in java.

Syntax

```
if(condition)
 {
  Statement(s)
 }
 else
 {
  Statement(s)
 }
  ........
```

Switch Statement

The **switch** statement in java language is used to execute the code from multiple conditions or case. It is same like if else-if ladder statement.

A switch statement work with byte, short, char and int primitive data type, it also works with enumerated types and string.

Syntax

```
switch(expression/variable)

{

 case  value:

 //statements

 // any number of case statements

 break;  //optional

 default: //optional

 //statements

}
```

Rules for apply switch statement

With switch statement use only byte, short, int, char data type (float data type is not allowed). You can use any number of case statements within a switch. Value for a case must be same as the variable in switch.

Limitations of switch statement

Logical operators cannot be used with switch statement. For instance

Example

```
case k>=20: // not allowed
```

Looping Statement in Java

Looping statement are the statements execute one or more statement repeatedly several number of times. In java programming language there are three types of loops; while, for and do-while.

Why use loop ?

When you need to execute a block of code several number of times then you need to use looping concept in Java language.

Advantage with looping statement

- Reduce length of Code
- Take less memory space.

- Burden on the developer is reducing.
- Time consuming process to execute the program is reduced.

Difference between conditional and looping statement

Conditional statement executes only once in the program where as looping statements executes repeatedly several number of time.

While loop

In **while loop** first check the condition if condition is true then control goes inside the loop body otherwise goes outside of the body. while loop will be repeats in clock wise direction.

Syntax

```
while(condition)

{

 Statement(s)

 Increment / decrements (++ or --);

}
```

for loop

for loop is a statement which allows code to be repeatedly executed. For loop contains 3 parts Initialization, Condition and Increment or Decrements

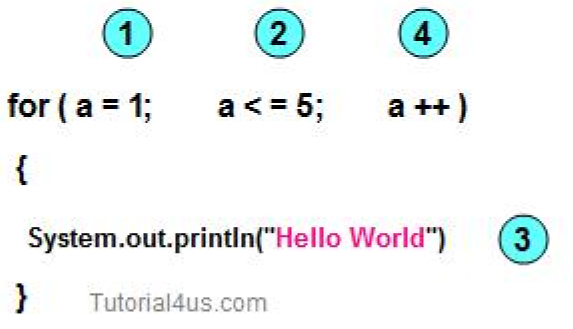
Syntax

```
for ( initialization; condition; increment )

{

 statement(s);

}
```

- **Initialization:** This step is execute first and this is execute only once when we are entering into the loop first time. This step is allow to declare and initialize any loop control variables.
- **Condition:** This is next step after initialization step, if it is true, the body of the loop is executed, if it is false then the body of the loop does not execute and flow of control goes outside of the for loop.
- **Increment or Decrements:** After completion of Initialization and Condition steps loop body code is executed and then Increment or Decrements steps is execute. This statement allows to update any loop control variables.

Flow Diagram

Control flow of for loop



```
for ( a = 1;     a < = 5;     a ++ )
{
 System.out.println("Hello World")
}
```

- First initialize the variable
- In second step check condition
- In third step control goes inside loop body and execute.
- At last increase the value of variable
- Same process is repeat until condition not false.

Improve your looping conceptFor Loop

Display any message exactly 5 times.

```
Hello Friends !
```

do-while

A **do-while** loop is similar to a while loop, except that a do-while loop is execute at least one time.

A do while loop is a control flow statement that executes a block of code at least once, and then repeatedly executes the block, or not, depending on a given condition at the end of the block (in while).

When use do..while loop

when we need to repeat the statement block **at least one time** then use do-while loop. In do-while loop post-checking process will be occur, that is after execution of the statement block condition part will be executed.

Syntax

```
do

{

Statement(s)
increment/decrement (++ or --)

}while();
```

In below example you can see in this program i=20 and we check condition i is less than 10, that means condition is false but do..while loop execute onec and print Hello world ! at one time.

Example do..while loop

```
class  dowhileDemo
{
 public static void main(String args[])
 {
 int i=20;
do
 {
System.out.println("Hello world !");
 i++;
}
while(i<10);
}
}
```

Output

```
Hello world !
```

Wrapper classes in java

For each and every fundamental data type there exist a pre-defined class, Such predefined class is known as wrapper class. The purpose of wrapper class is to convert numeric string data into numerical or fundamental data.

Why use wrapper classes ?

We know that in java whenever we get input form user, it is in the form of string value so here we need to convert these string values in different different datatype (numerical or fundamental data), for this conversion we use wrapper classes.

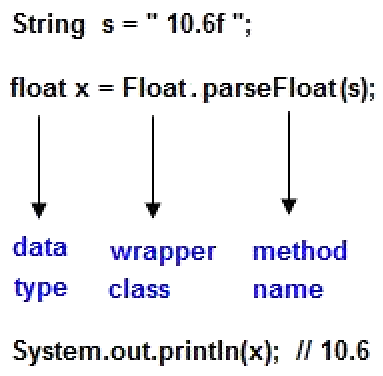
Example of wrapper class

```
class WraperDemo
{
public static void main(String[] args)
{
String  s[] = {"10", "20"};
System.out.println("Sum before:"+ s[0] + s[1]); // 1020
int x=Integer.parseInt(s[0]); // convert String to Integer
int y=Integer.parseInt(s[1]); // convert String to Integer
int z=x+y;
System.out.println("sum after: "+z); // 30
}
}
```

Output

```
Sum before: 1020

Sum after: 30
```

Explanation: In the above example 10 and 20 are store in String array and next we convert these values in Integer using "int x=Integer.parseInt(s[0]);" and "int y=Integer.parseInt(s[1]);" statement In "System.out.println("Sum before:"+ s[0] + s[1]);" Statement normally add two string and output is 1020 because these are String numeric type not number.



```
String  s = " 10.6f ";

float x = Float.parseFloat(s);

System.out.println(x);  // 10.6
```

Converting String data into fundamental or numerical

We know that every command line argument of java program is available in the main() method in the form of array of object of string class on String data, one can not perform numerical operation. To perform numerical operation it is highly desirable to convert numeric String into fundamental numeric value.

Example

```
"10"  --> numeric String --> only numeric string convert into numeric type or value.

"10x" --> alpha-numeric type --> this is not conversion.

"ABC" --> non-numeric String no conversion.

"A"   --> char String no conversion.
```

Only 'A' is convert into ASCII value that is 65 but 'A' is not convert into numeric value because it is a String value.

Fundamental data type and corresponding wrapper classes

The following table gives fundamental data type corresponding wrapper class name and conversion method from numerical String into numerical values or fundamental value.

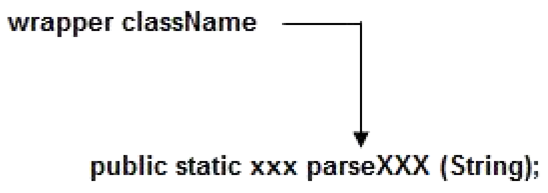
Fundamental DataType	**Wrapper CalssName**	**Conversion method from numeric string into fundamental or numeric value**
byte	Byte	public static byte parseByte(String)
short	Short	public static short parseShort(String)
int	Integer	public static integer parseInt(String)
long	Long	public static long parseLong(String)
float	Float	public static float parseFloat(String)
double	Double	public static double parseDouble(String)
char	Character	
boolean	Boolean	public static boolean parseBoolean(String)

How to use wrapper class methods

All the wrapper class methods are static in nature so we need to call these method using class.methodName().

- for Integer: int x=Integer.parseInt(String);
- for float: float x=Float.parseFloat(String);
- for double: double x=Double.parseDouble(String);

Each and every wrapper class contains the following generalized method for converting numeric String into fundamental values.



Here xxx represents any fundamental data type.

Naming Conversion of Java

Sun micro system was given following conversions by declaring class, variable, method etc. So that it is highly recommended to follow this conversion while writing real time code.

Why Using naming Conversion

Different Java programmers can have different styles and approaches to write program. By using standard Java naming conventions they make their code easier to read for themselves and for other programmers. Readability of Java code is important because it means less time is spent trying to figure out what the code does, and leaving more time to fix or modify it.

1. Every package name should exist a lower case latter.

Example

```
package student;  // creating package

import java.lang;  // import package
```

2. First letter of every word of class name or interface name should exists in upper case.

Example

```
class StudentDetails
 {
  .....
  .....
 }
interface FacultyDetail
 {
  .....
  .....
 }
```

3. Every constant value should exists in upper case latter. It is containing more than one word than it should be separated with underscore (-).

Example

```
class  Student
 {
 final   String  COLLEGE_NAME="abcd";
  ....
  ....
 }
```

Note: if any variable is preceded by final keyword is known as constant value.

Example

```
class  Student
 {
 Final String Student_name="abcd";
 }
```

While declaring variable name, method, object reference the first letter of first word should be exits in lower case but from the second words onward the first letter should exists in upper case.

Example

```
class  Student
{
String  StudentName="xyz";
void  instantStudentDetails();
{
  ....
  ....
}
Student  final
```

CamelCase in java naming conventions

Java follows camelcase syntax for naming the class, interface, method and variable. According to CamelCase if name is combined with two words, second word will start with uppercase letter always. General Example studentName, customerAccount. In term of java programming e.g. actionPerformed(), firstName, ActionEvent, ActionListener etc.

Access Modifiers in Java

Access modifiers are those which are applied before data members or methods of a class. These are used to where to access and where not to access the data members or methods. In Java programming these are classified into four types:

- Private
- Default (not a keyword)
- Protected
- Public

Note: Default is not a keyword (like public, private, protected are keyword)

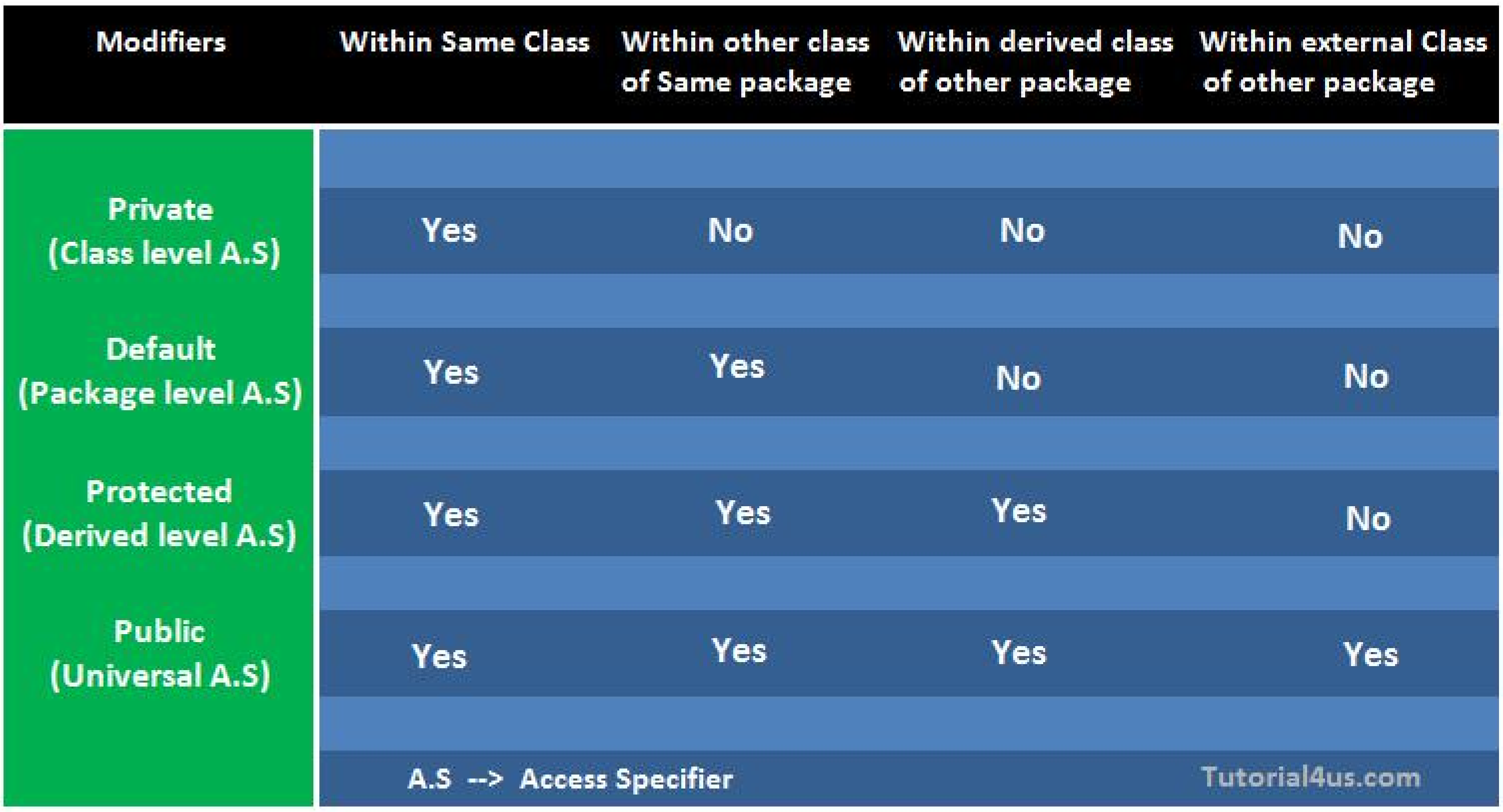
If we are not using private, protected and public keywords, then JVM is by default taking as default access modifiers.

Access modifiers are always used for, how to reuse the features within the package and access the package between class to class, interface to interface and interface to a class. Access modifiers provide features accessing and controlling mechanism among the classes and interfaces.

Note: Protected members of the class are accessible within the same class and another class of same package and also accessible in inherited class of another package.

Rules for access modifiers:

The following diagram gives rules for Access modifiers.



Modifiers	Within Same Class	Within other class of Same package	Within derived class of other package	Within external Class of other package
Private (Class level A.S)	Yes	No	No	No
Default (Package level A.S)	Yes	Yes	No	No
Protected (Derived level A.S)	Yes	Yes	Yes	No
Public (Universal A.S)	Yes	Yes	Yes	Yes

A.S --> Access Specifier

Tutorial4us.com

private: Private members of class in not accessible anywhere in program these are only accessible within the class. Private are also called class level access modifiers.

Example

```
class Hello
{
private int a=20;
private void show()
{
System.out.println("Hello java");
}
}

public class Demo
{
 public static void main(String args[])
 {
  Hello obj=new Hello();
  System.out.println(obj.a); //Compile Time Error, you can't access private data
  obj.show();   //Compile Time Error, you can't access private methods
 }
}
```

public: Public members of any class are accessible anywhere in the program in the same class and outside of class, within the same package and outside of the package. Public are also called universal access modifiers.

Example

```
class Hello
{
public int a=20;
public void show()
{
System.out.println("Hello java");
}
}

public class Demo
{
 public static void main(String args[])
 {
  Hello obj=new Hello();
  System.out.println(obj.a);
  obj.show();
 }
}
```

Output

```
20

Hello Java
```

protected: Protected members of the class are accessible within the same class and another class of the same package and also accessible in inherited class of another package. Protected are also called derived level access modifiers.

In below the example we have created two packages pack1 and pack2. In pack1, class A is public so we can access this class outside of pack1 but method show is declared as a protected so it is only accessible outside of package pack1 only through inheritance.

Example

```
// save A.java
package pack1;
public class A
{
protected void show()
{
System.out.println("Hello Java");
}
}
```

```
//save B.java
package pack2;
import pack1.*;

class B extends A
{
 public static void main(String args[]){
 B obj = new B();
 obj.show();
  }
}
```

Output

```
Hello Java
```

default: Default members of the class are accessible only within the same class and another class of the same package. The default are also called package level access modifiers.

Example

```
//save by A.java
package pack;
class A
{
 void show()
{
System.out.println("Hello Java");
}
}
//save by B.java
package pack2;
import pack1.*;
class B
{
 public static void main(String args[])
  {
  A obj = new A(); //Compile Time Error, can't access outside the package
  obj.show();   //Compile Time Error, can't access outside the package
 }
}
```

Output

```
Hello Java
```

Note: private access modifier is also known as native access modifier, default access modifier is also known as package access modifier, protected access modifier is also known as an inherited access modifier, public access modifier is also known as universal access modifier.

Array in java

Array is a collection of similar type of data. It is fixed in size means that you can't increase the size of array at run time. It is a collection of homogeneous data elements. It stores the value on the basis of the index value.

Advantage of Array

One variable can store multiple value: The main advantage of the array is we can represent multiple value under the same name.
Code Optimization: No, need to declare a lot of variable of same type data, We can retrieve and sort data easily.
Random access: We can retrieve any data from array with the help of the index value.

Disadvantage of Array

The main limitation of the array is **Size Limit** when once we declare array there is no chance to increase and decrease the size of an array according to our requirement, Hence memory point of view array concept is not recommended to use. To overcome this limitation in Java introduce the collection concept.

Types of Array

There are two types of array in Java.

- Single Dimensional Array
- Multidimensional Array

Array Declaration

Single dimension array declaration.

Syntax

```
1. int[]  a;
2. int  a[];
3. int  []a;
```

Note: At the time of array declaration we cannot specify the size of the array. For Example int[5] a; this is wrong.

2D Array declaration.

Syntax

```
1. int[][]  a;

2. int  a[][];

3. int  [][]a;

4. int[]  a[];

5. int[]  []a;

6. int   []a[];
```

Array creation

Every array in a Java is an object, Hence we can create array by using **new** keyword.

Syntax

```
int[] arr = new int[10];    // The size of array is 10.
or
int[] arr = {10,20,30,40,50};
```

Accessing array elements

Access the elements of array by using index value of an elements.

Syntax

```
arrayname[n-1];
```

Access Array Elements

```
int[] arr={10,20,30,40};
System.out.println("Element at 4th place"+arr[2]);
```

Example of Array

```
public class ArrayEx
{
public static void main(String []args)
{
int arr[] = {10,20,30};
for (int i=0; i < arr.length; i++)
{
 System.out.println(arr[i]);
 }
 }
 }
```

Output

```
10

20

39
```

Note:

1) At the time of array creation we must be specify the size of array otherwise get an compile time error. For Example int[] a=new int[]; Invalid.
int[] a=new int[5]; Valid

2) If we specify the array size as negative int value, then we will get run-time error, NegativeArraySizeException.

3) To specify the array size the allowed data types are byte, short, int, char If we use other data type then we will get an Compile time error.

4) The maximum allowed size of array in Java is 2147483647 (It is the maximum value of int data type)

Difference Between Length and Length() in Java

length: It is a final variable and only applicable for array. It represent size of array.

Example

```
int[] a=new int[10];
System.out.println(a.length);  // 10
System.out.println(a.length());  // Compile time error
```

length(): It is the final method applicable only for String objects. It represents the number of characters present in the String.

Example

```
String s="Java";
System.out.println(s.length());  // 4
System.out.println(s.length);  // Compile time error
```

Final keyword in java

It is used to make a variable as a constant, Restrict method overriding, Restrict inheritance. It is used at variable level, method level and class level. In java language final keyword can be used in following way.

- Final at variable level
- Final at method level
- Final at class level

Final at variable level

Final keyword is used to make a variable as a constant. This is similar to const in other language. A variable declared with the final keyword cannot be modified by the program after initialization. This is useful to universal constants, such as "PI".

Final Keyword in java Example

```
public class Circle
{
public  static final double PI=3.14159;

public static void main(String[] args)
{
System.out.println(PI);
}
}
```

Final at method level

It makes a method final, meaning that sub classes can not override this method. The compiler checks and gives an error if you try to override the method.

When we want to restrict overriding, then make a method as a final.

Example

```
public class A
{
public void fun1()
{
.......
}
public final void fun2()
{
.......
}


}
class B extends A
{
public void fun1()
{
.......
}
public void fun2()
{
 // it gives an error because we can not override final method
}
}
```

Example of final keyword at method level

Example

```
class Employee
{
final void disp()
{
System.out.println("Hello Good Morning");
}
}
class Developer extends Employee
{
void disp()
{
System.out.println("How are you ?");
}
}
class FinalDemo
{
public static void main(String args[])
{
Developer obj=new Developer();
obj.disp();
}
}
```

Output

```
It gives an error
```

Final at class level

It makes a class final, meaning that the class can not be inheriting by other classes. When we want to restrict inheritance then make class as a final.

Example

```
public final class A
{
......
......
}
public class B extends  A
```

```
{
// it gives an error, because we can not inherit final class
}
```

Example of final keyword at class level

Example

```
final class Employee
{
int salary=10000;
}
class Developer extends Employee
{
void show()
{
System.out.println("Hello Good Morning");
}
}
class FinalDemo
{
public static void main(String args[])
{
Developer obj=new Developer();
Developer obj=new Developer();
obj.show();
}
}
```

Output

```
Output:

It gives an error
```

This keyword in java

this is a reference variable that refers to the current object. It is a keyword in java language represents current class object

Usage of this keyword

- It can be used to refer current class instance variable.
- this() can be used to invoke current class constructor.
- It can be used to invoke current class method (implicitly)
- It can be passed as an argument in the method call.

- It can be passed as argument in the constructor call.
- It can also be used to return the current class instance.

Why use this keyword in java ?

The main purpose of **using this keyword** is to differentiate the formal parameter and data members of class, whenever the formal parameter and data members of the class are similar then jvm get ambiguity (no clarity between formal parameter and member of the class)

To differentiate between formal parameter and data member of the class, the data member of the class must be preceded by "this".

"this" keyword can be use in two ways.

- this . (this dot)
- this() (this off)

this . (this dot)

which can be used to differentiate variable of class and formal parameters of method or constructor.

"this" keyword are used for two purpose, they are

- It always points to current class object.
- Whenever the formal parameter and data member of the class are similar and JVM gets an ambiguity (no clarity between formal parameter and data members of the class).

To differentiate between formal parameter and data member of the class, the data members of the class must be preceded by **"this"**.

Syntax

```
this.data member of current class.
```

Note: If any variable is preceded by **"this"** JVM treated that variable as class variable.

Example without using this keyword

```
class Employee
{
 int id;
 String name;

 Employee(int id,String name)
 {
 id = id;
 name = name;
 }
void show()
 {
  System.out.println(id+" "+name);
 }
  public static void main(String args[])
  {
   Employee e1 = new Employee(111,"Harry");
   Employee e2 = new Employee(112,"Jacy");
   e1.show();
   e2.show();
  }
}
```

Output

```
Output:

0 null

0 null
```

In the above example, parameter (formal arguments) and instance variables are same that is why we are using **"this"** keyword to distinguish between local variable and instance variable.

Example of this keyword in java

```
class Employee
{
  int id;
  String name;

  Employee(int id,String name)
  {
  this.id = id;
  this.name = name;
  }
    void show()
  {
  System.out.println(id+" "+name);
  }
  public static void main(String args[])
  {
   Employee e1 = new Employee(111,"Harry");
   Employee e2 = new Employee(112,"Jacy");
   e1.show();
   e2.show();
  }
}
```

Output

```
111 Harry

112 Jacy
```

Note 1: The scope of **"this"** keyword is within the class.

Note 2: The main purpose of using **"this"** keyword in real life application is to differentiate variable of class or formal parameters of methods or constructor (it is highly recommended to use the same variable name either in a class or method and constructor while working with similar objects).

Difference between this and super keyword

Super keyword is always pointing to base class (scope outside the class) features and**"this"** keyword is always pointing to current class (scope is within the class) features.

Example when no need of this keyword

```
class Employee
{
  int id;
  String name;

  Employee(int i,String n)
 {
  id = i;
  name = n;
 }
 void show()
 {
  System.out.println(id+" "+name);
 }
  public static void main(String args[])
  {
   Employee e1 = new Employee(111,"Harry");
   Employee e2 = new Employee(112,"Jacy");
   e1.show();
   e2.show();
  }
}
```

Output

```
111 Harry

112 Jacy
```

In the above example, no need of use this keyword because parameter (formal arguments) and instance variables are different. This keyword is only use when parameter (formal arguments) and instance variables are same.

this ()

which can be used to call one constructor within the another constructor without creation of objects multiple time for the same class.

Syntax

```
this(); // call no parametrized or default constructor
this(value1,value2,.....) //call parametrize constructor
```

this keyword used to invoke current class method (implicitly)

By using this keyword you can invoke the method of the current class. If you do not use the this keyword, compiler automatically adds this keyword at time of invoking of the method.

Example of this keyword

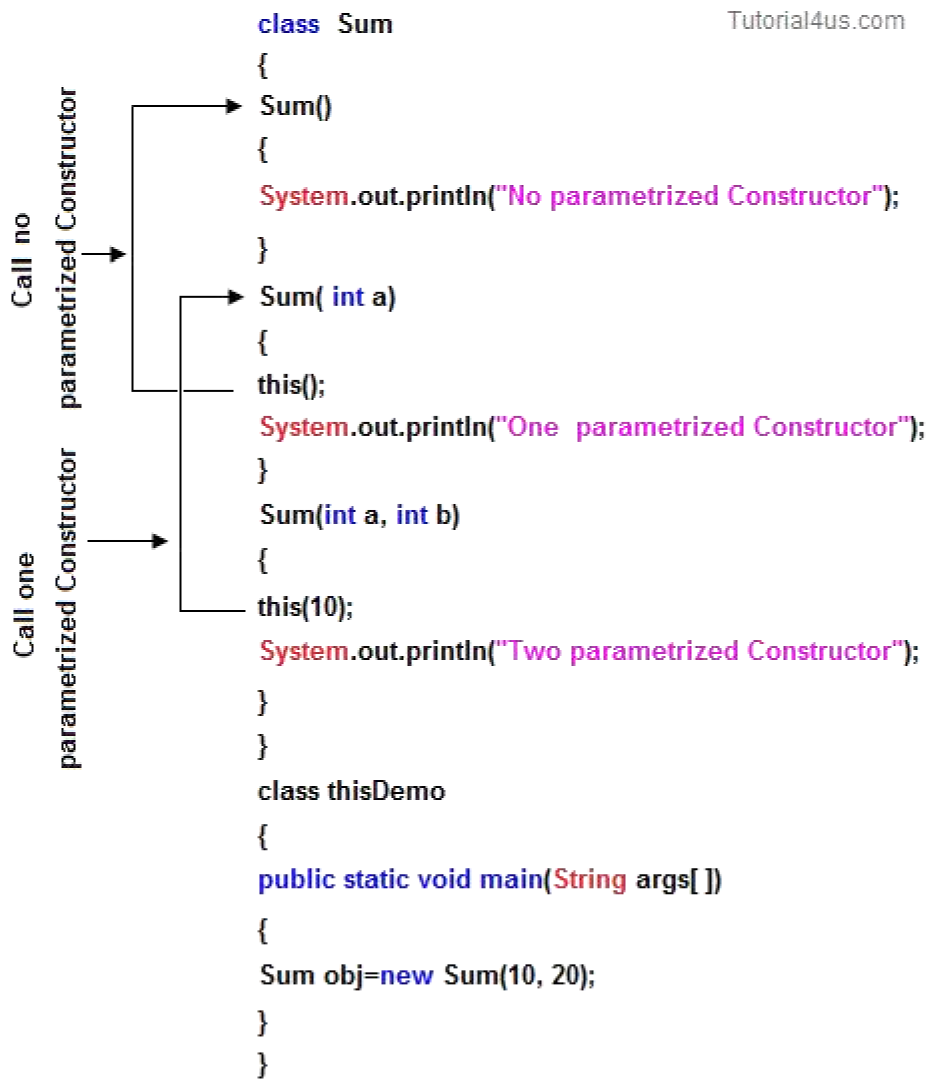
```
class Student
{
 void show()
  {
  System.out.println("You got A+");
  }
  void marks()
  {
  this.show(); //no need to use this here because compiler does it.
  }
  void display()
  {
  show(); //compiler act marks() as this.marks()
  }
  public static void main(String args[])
  {
  Student s = new Student();
  s.display();
  }
}
```

Syntax

```
You got A+
```

Rules to use this()

this() always should be the first statement of the constructor. One constructor can call only other single constructor at a time by using **this()**.



Tutorial4us.com
Call no parametrized Constructor
Call one parametrized Constructor

```
class  Sum
{
Sum()
{
System.out.println("No parametrized Constructor");
}
Sum( int a)
{
this();
System.out.println("One  parametrized Constructor");
}
Sum(int a, int b)
{
this(10);
System.out.println("Two parametrized Constructor");
}
}
class thisDemo
{
public static void main(String args[ ])
{
Sum obj=new Sum(10, 20);
}
}
```

Super keyword in java

Super keyword in java is a reference variable that is used to refer parent class object.**Super** is an implicit keyword create by JVM and supply each and every java program for performing important role in three places.

- At variable level
- At method level
- At constructor level

Need of super keyword:

Whenever the derived class is inherits the base class features, there is a possibility that base class features are similar to derived class features and JVM gets an ambiguity. In order to differentiate between base class features and derived class features must be preceded by super keyword.

Syntax

```
super.baseclass features.
```

Super at variable level:

Whenever the derived class inherit base class data members there is a possibility that base class data member are similar to derived class data member and JVM gets an ambiguity.

In order to differentiate between the data member of base class and derived class, in the context of derived class the base class data members must be preceded by super keyword.

Syntax

```
super.baseclass datamember name
```

if we are not writing super keyword before the base class data member name than it will be referred as current class data member name and base class data member are hidden in the context of derived class.

Program without using super keyword

Example

```java
class Employee
{
float salary=10000;
}
class HR extends Employee
{
float salary=20000;
void display()
{
System.out.println("Salary: "+salary);//print current class salary
}
}
class Supervarible
{
public static void main(String[] args)
{
HR obj=new HR();
obj.display();
}
}
```

Output

```
Salary: 20000.0
```

In the above program in Employee and HR class salary is common properties of both class the instance of current or derived class is referred by instance by default but here we want to refer base class instance variable that is why we use super keyword to distinguish between parent or base class instance variable and current or derived class instance variable.

Program using super keyword al variable level

Example

```
class Employee
{
float salary=10000;
}
class HR extends Employee
{
float salary=20000;
void display()
{
System.out.println("Salary: "+super.salary);//print base class salary
}
}
class Supervarible
{
public static void main(String[] args)
{
HR obj=new HR();
obj.display();
}
}
```

Output

```
Salary: 10000.0
```

Super at method level

The **super keyword** can also be used to invoke or call parent class method. It should be use in case of method overriding. In other word **super keyword** use when base class method name and derived class method name have same name.

Example of super keyword at method level

Example

```
class Student
{
void message()
{
System.out.println("Good Morning Sir");
}
}

class Faculty extends Student
{
void message()
{
System.out.println("Good Morning Students");
}

void display()
{
message();//will invoke or call current class message() method
super.message();//will invoke or call parent class message() method
}

public static void main(String args[])
{
Student s=new Student();
s.display();
}
}
```

Output

```
Good Morning Students

Good Morning Sir
```

In the above example Student and Faculty both classes have message() method if we call message() method from Student class, it will call the message() method of Student class not of Person class because priority of local is high.

In case there is no method in subclass as parent, there is no need to use super. In the example given below message() method is invoked from Student class but Student class does not have message() method, so you can directly call message() method.

Program where super is not required

Example

```
class Student
{
void message()
{
System.out.println("Good Morning Sir");
}
}

class Faculty extends Student
{

void display()
{
message();//will invoke or call parent class message() method
}

public static void main(String args[])
{
Student s=new Student();
s.display();
}
}
```

Output

```
Good Morning Sir
```

Super at constructor level

The super keyword can also be used to invoke or call the parent class constructor. Constructor are calling from bottom to top and executing from top to bottom.

To establish the connection between base class constructor and derived class constructors JVM provides two implicit methods they are:

- Super()
- Super(...)

Super()

Super() It is used for calling super class default constructor from the context of derived class constructors.

Super keyword used to call base class constructor

Syntax

```
class Employee
{
Employee()
{
System.out.println("Employee class Constructor");
}
}
class HR extends Employee
{
HR()
{
super(); //will invoke or call parent class constructor
System.out.println("HR class Constructor");
}
}
class Supercons
{
public static void main(String[] args)
{
HR obj=new HR();
}
}
```

Output

```
Employee class Constructor

HR class Constructor
```

Note: super() is added in each class constructor automatically by compiler.

In constructor, default constructor is provided by compiler automatically but it also adds**super()** before the first statement of constructor.If you are creating your own constructor and you do not have either this() or super() as the first statement, compiler will provide super() as the first statement of the constructor.

Super(...)

Super(...) It is used for calling super class parameterize constructor from the context of derived class constructor.

Important rules

Whenever we are using either super() or super(...) in the derived class constructors the**super** always must be as a first executable statement in the body of derived class constructor otherwise we get a compile time error.

The following diagram use possibilities of using super() and super(........)

Rule 1 and Rule 3

Whenever the derived class constructor want to call default constructor of base class, in the context of derived class constructors we write super(). Which is optional to write because every base class constructor contains single form of default constructor?

Rule 2 and Rule 4

Whenever the derived class constructor wants to call parameterized constructor of base class in the context of derived class constructor we must write super(...). which is mandatory to write because a base class may contain multiple forms of parameterized constructors.

Synchronized Keyword in Java

Synchronized Keyword is used for when we want to allowed only one thread at a time then use Synchronized modifier. If a method or block declared as a Synchronized then at a time only one thread is allowed to operate on the given object.

Synchronized is a Modifier which is applicable for the method or block, we can not declare class or variable with this modifier.

Advantage of Synchronized

The main advantage of Synchronized keyword is we can resolve data inconsistency problem.

Dis-Advantage of Synchronized

The main dis-advantage of Synchronized keyword is it increased the waiting time of thread and effect performance of the system, Hence if there is no specific requirement it is never recommended to use synchronized keyword.

Abstract class in Java

A class that is declared with abstract keyword, is known as **abstract class**. An abstract class is one which is containing some defined method and some undefined method. In java programming undefined methods are known as un-Implemented, or abstract method.

Syntax

```
abstract class className

{

......
}
```

Example

```
abstract class A
{
.....
}
```

If any class have any abstract method then that class become an abstract class.

Example

```
class Vachile
{
abstract void Bike();
}
```

Class Vachile is become an abstract class because it have abstract Bike() method.

Make class as abstract class

To make the class as abstract class, whose definition must be preceded by a abstract keyword.

Example

```
abstract class Vachile
{
......
}
```

Abstract method

An abstract method is one which contains only declaration or prototype but it never contains body or definition. In order to make any undefined method as abstract whose declaration is must be predefined by abstract keyword.

Syntax

```
abstract ReturnType methodName(List of formal parameter)
```

Example

```
abstract  void  sum();
abstract  void  diff(int, int);
```

Example of abstract class

```
abstract class Vachile
{
 abstract void speed();  // abstract method
}
class Bike extends Vachile
{
void speed()
{
System.out.println("Speed limit is 40 km/hr..");
}
public static void main(String args[])
{
 Vachile obj = new Bike(); //indirect object creation
 obj.speed();
 }
}
```

Output

```
Speed limit is 40 km/hr..
```

Create an Object of abstract class

An object of abstract class cannot be created directly, but it can be created indirectly. It means you can create an object of abstract derived class. You can see in above example

Example

```
Vachile obj = new Bike(); //indirect object creation
```

Important Points about abstract class

- Abstract class of Java always contains common features.
- Every abstract class participates in inheritance.
- Abstract class definitions should not be made as final because abstract classes always participate in inheritance classes.
- An object of abstract class cannot be created directly, but it can be created indirectly.
- All the abstract classes of Java makes use of polymorphism along with method overriding for business logic development and makes use of dynamic binding for execution logic.

Advantage of abstract class

- Less memory space for the application
- Less execution time
- More performance

Why abstract class have no abstract static method ?

In abstract classes we have the only abstract instance method, but not containing abstract static methods because every instance method is created for performing repeated operation where as static method is created for performing a one time operations in other word every abstract method is instance but not static.

Abstract base class

An abstract base class is one which is containing physical representation of abstract methods which are inherited by various sub classes.

Abstract derived class

An abstract derived class is one which is containing logic representation of abstract methods which are inherited from abstract base class with respect to both abstract base class and abstract derived class one can not create objects directly, but we can create their objects indirectly both abstract base class and abstract derived class are always reusable by various sub classes.

When the derived class inherits multiple abstract method from abstract base class and if the derived class is not defined at least one abstract method then the derived class is known as abstract derived class and whose definition must be made as abstract by using abstract keyword. (When the derived class becomes an abstract derived class).

If the derived class defined all the abstract methods which are inherited from abstract Base class, then the derived class is known as concrete derived class.

Example of abstract class having method body

```
abstract class Vachile
{
 abstract void speed();
 void mileage()
{
 System.out.println("Mileage is 60 km/ltr..");
}
}
class Bike extends Vachile
{
void speed()
{
System.out.println("Speed limit is 40 km/hr..");
}
public static void main(String args[])
{
 Vachile obj = new Bike();
 obj.speed();
 obj.mileage();
 }
}
```

Output

```
Mileage is 60 km/ltr..

Speed limit is 40 km/hr..
```

Example of abstract class having constructor, data member, methods

```
abstract class Vachile
{
 int limit=40;
 Vachile()
{
System.out.println("constructor is invoked");
}
 void getDetails()
{
System.out.println("it has two wheels");
}
 abstract void run();
}

class Bike extends Vachile
{
 void run()
{
System.out.println("running safely..");
}
 public static void main(String args[])
{
  Vachile obj = new Bike();
  obj.run();
  obj.getDetails();
  System.out.println(obj.limit);
 }
}
```

Output

```
constructor is invoked

running safely..

it has two wheels

40
```

Difference Between Abstract class and Concrete class

Concrete class	Abstract class
A Concrete class is used for specific requirement	Abstract class is used to fulfill a common requirement.
Object of concrete class can create directly.	Object of an abstract class can not create directly (can create indirectly).
Concrete class containing fully defined methods or implemented method.	Abstract class has both undefined method and defined method.

Download Code

Difference Between Static and non-Static Variable in Java

The variable of any class are classified into two types;

- Static or class variable
- Non-static or instance variable

Static variable in Java

Memory for static variable is created only one in the program at the time of loading of class. These variables are preceded by static keyword. tatic variable can access with class reference.

Non-static variable in Java

Memory for non-static variable is created at the time of create an object of class. These variable should not be preceded by any static keyword Example: These variables can access with object reference.

Difference between non-static and static variable

	Non-static variable	Static variable

1	These variable should not be preceded by any static keyword Example: `class A { int a; }`	These variables are preceded by static keyword. **Example** `class A { static int b; }`
2	Memory is allocated for these variable whenever an object is created	Memory is allocated for these variable at the time of loading of the class.
3	Memory is allocated multiple time whenever a new object is created.	Memory is allocated for these variable only once in the program.
4	Non-static variable also known as instance variable while because memory is allocated whenever instance is created.	Memory is allocated at the time of loading of class so that these are also known as class variable.
5	Non-static variable are specific to an object	Static variable are common for every object that means there memory location can be sharable by every object reference or same class.
6	Non-static variable can access with object reference. **Syntax** `obj_ref.variable_name`	Static variable can access with class reference. **Syntax** `class_name.variable_name`

Note: static variable not only can be access with class reference but also some time it can be accessed with object reference.

Example of static and non-static variable.

Example

```
class Student
{
int roll_no;
float marks;
String name;
static String College_Name="ITM";
}
class StaticDemo
{
public static void main(String args[])
{
Student s1=new Student();
s1.roll_no=100;
s1.marks=65.8f;
s1.name="abcd";
System.out.println(s1.roll_no);
System.out.println(s1.marks);
System.out.println(s1.name);
System.out.println(Student.College_Name);
//or System.out.println(s1. College_Name); but first is use in real time.
Student  s2=new  Student();
s2.roll_no=200;
s2.marks=75.8f;
s2.name="zyx";
System.out.println(s2.roll_no);
System.out.println(s2.marks);
System.out.println(s2.name);
System.out.println(Student.College_Name);
}
}
```

Output

```
100

65.8

abcd

ITM

200

75.8

zyx

ITM
```

Note: In the above example College_Name variable is commonly sharable by both S1 and S2 objects.

Understand static and non-static variable using counter

Program of counter without static variable

In this example, we have created an instance variable named count which is incremented in the constructor. Since instance variable gets the memory at the time of object creation, each object will have the copy of the instance variable, if it is incremented, it won't reflect to other objects. So each objects will have the value 1 in the count variable.

Example

```
class Counter
{
int count=0;//will get memory when instance is created
Counter()
{
count++;
System.out.println(count);
}
public static void main(String args[])
{
Counter c1=new Counter();
Counter c2=new Counter();
Counter c3=new Counter();
}
}
```

Output

```
1

1

1
```

Program of counter by static variable

As we have mentioned above, static variable will get the memory only once, if any object changes the value of the static variable, it will retain its value.

Example

```
class Counter
{
static int count=0; //will get memory only once
Counter()
{
count++;
System.out.println(count);
}
public static void main(String args[])
{
Counter c1=new Counter();
Counter c2=new Counter();
Counter c3=new Counter();
}
}
```

Output

```
1
```

```
2

3
```

Difference between Static and non-static method in Java

In case of non-static method memory is allocated multiple time whenever method is calling. But memory for static method is allocated only once at the time of class loading. Method of a class can be declared in two different ways

- Non-static methods
- Static methods

Difference between non-static and static Method

	Non-Static method	Static method

1	These method never be preceded by static keyword
	Example:
	`void fun1()`
	`{`
	`......`
	`......`
	`}`
		These method always preceded by static keyword
		Example:
		`static void fun2()`
		`{`
		`......`
		`......`
		`}`

2	Memory is allocated multiple time whenever method is calling.	Memory is allocated only once at the time of class loading.
3	It is specific to an object so that these are also known as instance method.	These are common to every object so that it is also known as member method or class method.
4	These methods always access with object reference Syntax: **Objref.methodname();**	These property always access with class reference Syntax: **className.methodname();**
5	If any method wants to be execute multiple time that can be declare as non static.	If any method wants to be execute only once in the program that can be declare as static .

Note: In some cases static methods not only can access with class reference but also can access with object reference.

Example of Static and non-Static Method

Example

```
class A
{
void fun1()
{
System.out.println("Hello I am Non-Static");
}
static void fun2()
{
System.out.println("Hello I am Static");
}
}
class Person
{
public static void main(String args[])
{
A  oa=new  A();
   oa.fun1(); // non static method
   A.fun2();  // static method
}
}
```

Output

```
Hello I am Non-Static

Hello I am Static
```

Following table represent how the static and non-static properties are accessed in the different static or non-static method of same class or other class.

Program to accessing static and non-static properties.

Example

```
class A
{
int y;
void f2()
{
System.out.println("Hello f2()");
}
}
class B
{
int z;
void f3()
{
System.out.println("Hello f3()");
A a1=new A();
a1.f2();
}
}
class Sdemo
{
static int x;
static void f1()
{
System.out.println("Hello f1()");
}
public static void main(String[] args)
{
x=10;
System.out.println("x="+x);
f1();
System.out.println("Hello main");
B b1=new B();
b1.f3();
}
}
```

Constructor in Java

constructor in Java is a special member method which will be called implicitly (automatically) by the JVM whenever an object is created for placing user or programmer defined values in place of default values. In a single word constructor is a special member method which will be called automatically whenever object is created.

The purpose of constructor is to initialize an object called object initialization. Constructors are mainly create for initializing the object. Initialization is a process of assigning user defined values at the time of allocation of memory space.

Syntax

```
className()
{
.......
.......
}
```

Advantages of constructors in Java

- A constructor eliminates placing the default values.
- A constructor eliminates calling the normal or ordinary method implicitly.

How Constructor eliminate default values ?

Constructor are mainly used for eliminate default values by user defined values, whenever we create an object of any class then its allocate memory for all the data members and initialize there default values. To eliminate these default values by user defined values we use constructor.

Constructor Example in Java

```
class Sum
{
int a,b;
Sum()
{
a=10;
b=20;
}
public static void main(String s[])
{
Sum s=new Sum();
c=a+b;
System.out.println("Sum: "+c);
}
}
```

Output

```
Sum: 30
```

In above example when we create an object of "Sum" class then constructor of this class call and initialize user defined value in a=10 and b=20. if we can not create constructor of Sum class then it print " Sum: 0 " because default values of integer is zero.

Rules or properties of a constructor

- Constructor will be called automatically when the object is created.
- Constructor name must be similar to name of the class.
- Constructor should not return any value even void also. Because basic aim is to place the value in the object. (if we write the return type for the constructor then that constructor will be treated as ordinary method).
- Constructor definitions should not be static. Because constructors will be called each and every time, whenever an object is creating.
- Constructor should not be private provided an object of one class is created in another class (Constructor can be private provided an object of one class created in the same class).

- Constructors will not be inherited from one class to another class (Because every class constructor is create for initializing its own data members).
- The access specifier of the constructor may or may not be private.
  1. If the access specifier of the constructor is private then an object of corresponding class can be created in the context of the same class but not in the context of some other classes.
  2. If the access specifier of the constructor is not private then an object of corresponding class can be created both in the same class context and in other class context.

Difference between Method and Constructor

	Method	**Constructor**
1	Method can be any user defined name	Constructor must be class name
2	Method should have return type	It should not have any return type (even void)
3	Method should be called explicitly either with object reference or class reference	It will be called automatically whenever object is created
4	Method is not provided by compiler in any case.	The java compiler provides a default constructor if we do not have any constructor.

Types of constructors

Based on creating objects in Java constructor are classified in two types. They are

- Default or no argument Constructor
- Parameterized constructor.

Default Constructor

A constructor is said to be default constructor if and only if it never take any parameters.

If any class does not contain at least one user defined constructor than the system will create a default constructor at the time of compilation it is known as system defined default constructor.

Syntax of Default Constructor

```
class className
{
.....        // Call default constructor
clsname ()
{
Block of statements;  // Initialization
}
.....
}
```

Note: System defined default constructor is created by java compiler and does not have any statement in the body part. This constructor will be executed every time whenever an object is created if that class does not contain any user defined constructor.

Example of default constructor.

In below example, we are creating the no argument constructor in the Test class. It will be invoked at the time of object creation.

Example

```
//TestDemo.java
class Test
{
int a, b;
Test ()
{
System.out.println("I am from default Constructor...");
a=10;
b=20;
System.out.println("Value of a: "+a);
System.out.println("Value of b: "+b);
}
};
class TestDemo
{
public static void main(String [] args)
{
Test t1=new Test ();
}
};
```

Output

```
Output:

I am from default Constructor...

Value of a: 10

Value of b: 20
```

Rule-1:

Whenever we create an object only with default constructor, defining the default constructor is optional. If we are not defining default constructor of a class, then JVM will call automatically system defined default constructor. If we define, JVM will call user defined default constructor.

Purpose of default constructor?

Default constructor provides the default values to the object like 0, 0.0, null etc. depending on their type (for integer 0, for string null).

Example of default constructor that displays the default values

```
class Student
{
int roll;
float marks;
String name;
void show()
{
System.out.println("Roll: "+roll);
System.out.println("Marks: "+marks);
System.out.println("Name: "+name);
}
}
class TestDemo
{
public static void main(String [] args)
{
Student s1=new Student();
s1.show();
}
}
```

Output

```
Roll: 0

Marks: 0.0

Name: null
```

Explanation: In the above class, we are not creating any constructor so compiler provides a default constructor. Here 0, 0.0 and null values are provided by default constructor.

parameterized constructor

If any constructor contain list of variable in its signature is known as paremetrized constructor. A parameterized constructor is one which takes some parameters.

Syntax

```
class ClassName
{
.......
ClassName(list of parameters) //parameterized constructor
{
.......
}
.......
}
```

Syntax to call parametrized constructor

```
ClassName  objref=new ClassName(value1, value2,.....);

                              OR

new ClassName(value1, value2,.....);
```

Example of Parametrized Constructor

```
class Test
{
int a, b;
Test(int n1, int n2)
{
System.out.println("I am from Parameterized Constructor...");
a=n1;
b=n2;
System.out.println("Value of a = "+a);
System.out.println("Value of b = "+b);
}
};
class TestDemo1
{
public static void main(String k [])
{
Test t1=new Test(10, 20);
}
};
```

Important points Related to Parameterized Constructor

- Whenever we create an object using parameterized constructor, it must be define parameterized constructor otherwise we will get compile time error. Whenever we define the objects with respect to both parameterized constructor and default constructor, It must be define both the constructors.
- In any class maximum one default constructor but 'n' number of parameterized constructors.

Example of default constructor, parameterized constructor and overloaded constructor

Example

```
class Test
{
int a, b;
Test ()
{
System.out.println("I am from default Constructor...");
a=1;
b=2;
System.out.println("Value of a ="+a);
System.out.println("Value of b ="+b);
}
Test (int x, int y)
{
System.out.println("I am from double Paraceterized Constructor");
a=x;
b=y;
System.out.println("Value of a ="+a);
System.out.println("Value of b ="+b);
}
Test (int x)
{
System.out.println("I am from single Parameterized Constructor");
a=x;
b=x;
System.out.println("Value of a ="+a);
System.out.println("Value of b ="+b);
}
Test (Test T)
{
System.out.println("I am from Object Parameterized Constructor...");
a=T.a;
b=T.b;
System.out.println("Value of a ="+a);
System.out.println("Value of b ="+b);
}
};
class TestDemo2
{
public static void main (String k [])
{
Test t1=new Test ();
Test t2=new Test (10, 20);
Test t3=new Test (1000);
Test t4=new Test (t1);
}
};
```

Note By default the parameter passing mechanism is call by reference.

Constructor Overloading

Constructor overloading is a technique in Java in which a class can have any number of constructors that differ in parameter lists.The compiler differentiates these constructors by taking the number of parameters, and their type. In other words whenever same constructor is existing multiple times in the same class with different number of parameters or order of parameters or type of parameters is known as**Constructor overloading**. In general constructor overloading can be used to initialized same or different objects with different values.

Syntax

```
class  ClassName
{
ClassName()
{
..........
..........
}
ClassName(datatype1 value1)
{.......}
ClassName(datatype1 value1, datatype2 value2)
{.......}
ClassName(datatype2 variable2)
{.......}
ClassName(datatype2 value2, datatype1 value1)
{.......}
........
}
```

Why overriding is not possible at constructor level.

The scope of constructor is within the class so that it is not possible to achieved overriding at constructor level.

Relationship in Java

Type of relationship always makes to understand how to reuse the feature from one class to another class. In java programming we have three types of relationship they are.

- Is-A Relationship
- Has-A Relationship
- Uses-A Relationship

Is-A relationship

In Is-A relationship one class is obtaining the features of another class by using inheritance concept with extends keywords.

In a IS-A relationship there exists logical memory space.

Example of Is-A Relation

Example

```
class Faculty
{
float salary=30000;
}
class Science extends Faculty
{
float bonous=2000;
public static void main(String args[])
{
Science obj=new Science();
System.out.println("Salary is:"+obj.salary);
System.out.println("Bonous is:"+obj.bonous);
}
}
```

Output

```
Salary is: 30000.0

Bonous is: 2000.0
```

Has-A relationship

In Has-A relationship an object of one class is created as data member in another class the relationship between these two classes is Has-A.

In Has-A relationship there existed physical memory space and it is also known as part of or kind of relationship.

Example of Has-A Relation

Example

```
class Employee
{
float salary=30000;
}
```

```
class Developer extends Employee
{
float bonous=2000;
public static void main(String args[])
{
Employee obj=new Employee();
System.out.println("Salary is:"+obj.salary);
}
}
```

Output

```
Salary is: 30000.0
```

Uses-A relationship

A method of one class is using an object of another class the relationship between these two classes is known as Uses-A relationship.

As long as the method is execution the object space (o1) exists and once the method execution is completed automatically object memory space will be destroyed.

Example of Uses-A Relation

Example

```
class Employee
{
float salary=30000;
}
class Salary extends Employee
{
void disp()
{
float bonous=1000;
Employee obj=new Employee();
float Total=obj.salary+bonous;
System.out.println("Total Salary is:"+Total);
}
}
class Developer
{
public static void main(String args[])
{
Salary s=new Salary();
s.disp();
}
}
```

Output

```
Total Salary is: 31000.0
```

Note 1: The default relationship in java is Is-A because for each and every class in java there exist an implicit predefined super class is java.lang.Object.

Note 2: The universal example for Has-A relationship is System.out (in System.out statement, out is an object of printStream class created as static data member in another system class and printStream class is known as Has-A relationship).

Note 3: Every execution logic method (main()) of execution logic is making use of an object of business logic class and business logic class is known as Uses-A relationship.

OOP's Concept in Java

Object-Oriented Programming is a methodology or paradigm to design a program using classes and objects.

- Object
- Class
- Inheritance
- Polymorphism
- Abstraction
- Encapsulation

Object

Object is the physical as well as logical entity where as **class** is the only logical entity. Object in Java

Class

Class: Class is a blue print which is containing only list of variables and method and no memory is allocated for them. A class is a group of objects that has common properties. Class in Java

Encapsulation

Encapsulation is a process of wrapping of data and methods in a single unit is called encapsulation. Encapsulation is achieved in C++ language by class concept. The main advantage of using of encapsulation is to secure the data from other methods, when we make a data private then these data only use within the class, but these data not accessible outside the class. Encapsulation in Java

Abstraction

Abstraction is the concept of exposing only the required essential characteristics and behavior with respect to a context. Abstraction in Java

Hiding of data is known as **data abstraction**. In object oriented programming language this is implemented automatically while writing the code in the form of class and object.

Inheritance

The process of obtaining the data members and methods from one class to another class is known as **inheritance**. It is one of the fundamental features of object-oriented programming.Inheritance in Java

Polymorphism

The process of representing one Form in multiple forms is known as **Polymorphism**. Here one form represent original form or original method always resides in base class and multiple forms represents overridden method which resides in derived classes. Polymorphism in Java

Note: All these concept I will discuss in detail later.

Inheritance in Java

The process of obtaining the data members and methods from one class to another class is known as **inheritance**. It is one of the fundamental features of object-oriented programming.

Important points

- In the inheritance the class which is give data members and methods is known as base or super or parent class.
- The class which is taking the data members and methods is known as sub or derived or child class.
- The data members and methods of a class are known as features.
- The concept of inheritance is also known as re-usability or extendable classes or sub classing or derivation.

Why use Inheritance ?

- For Method Overriding (used for Runtime Polymorphism).
- It's main uses are to enable polymorphism and to be able to reuse code for different classes by putting it in a common super class
- For code Re-usability

Syntax of Inheritance

```
class Subclass-Name extends Superclass-Name
{
  //methods and fields
}
```

Real Life Example of Inheritance in Java

The real life example of inheritance is child and parents, all the properties of father are inherited by his son.

Advantage of inheritance

If we develop any application using concept of Inheritance than that application have following advantages,

- Application development time is less.
- Application take less memory.
- Application execution time is less.
- Application performance is enhance (improved).
- Redundancy (repetition) of the code is reduced or minimized so that we get consistence results and less storage cost.

Note: In Inheritance the scope of access modifier increasing is allow but decreasing is not allow. Suppose in parent class method access modifier is default then it's present in child class with default or public or protected access modifier but not private(it decreased scope).

Tpyes of Inheritance

Based on number of ways inheriting the feature of base class into derived class we have five types of inheritance; they are:

- Single inheritance
- Multiple inheritance
- Hierarchical inheritance
- Multilevel inheritance
- Hybrid inheritance

Single inheritance

In single inheritance there exists single base class and single derived class.

Example of Single Inheritance

```
class Faculty
{
float salary=30000;
}
class Science extends Faculty
{
float bonous=2000;
public static void main(String args[])
{
Science obj=new Science();
System.out.println("Salary is:"+obj.salary);
System.out.println("Bonous is:"+obj.bonous);
}
}
```

Output

```
Salary is: 30000.0

Bonous is: 2000.0
```

Multilevel inheritances in Java

In Multilevel inheritances there exists single base class, single derived class and multiple intermediate base classes.

Single base class + single derived class + multiple intermediate base classes.

Intermediate base classes

An intermediate base class is one in one context with access derived class and in another context same class access base class.

Hence all the above three inheritance types are supported by both classes and interfaces.

Example of Multilevel Inheritance

```
class Faculty
{
float total_sal=0, salary=30000;
}

class HRA extends Faculty
{
float hra=3000;
}

class DA extends HRA
{
float da=2000;
}

class Science extends DA
{
float bonous=2000;
public static void main(String args[])
{
```

```
Science obj=new Science();
obj.total_sal=obj.salary+obj.hra+obj.da+obj.bonous;
System.out.println("Total Salary is:"+obj.total_sal);
}
}
```

Output

```
Total Salary is: 37000.0
```

Multiple inheritance

In multiple inheritance there exist multiple classes and singel derived class.

The concept of multiple inheritance is not supported in java through concept of classes but it can be supported through the concept of interface.

Hybrid inheritance

Combination of any inheritance type

In the combination if one of the combination is multiple inheritance then the inherited combination is not supported by java through the classes concept but it can be supported through the concept of interface.

Inheriting the feature from base class to derived class

In order to inherit the feature of base class into derived class we use the following syntax

Syntax

```
class ClassName-2 extends ClasssName-1
{
variable  declaration;
Method declaration;
}
```

Explanation

1. ClassName-1 and ClassName-2 represents name of the base and derived classes respectively.
2. extends is one of the keyword used for inheriting the features of base class into derived class it improves the functionality of derived class.

Important Points for Inheritance:

- In java programming one derived class can extends only one base class because java programming does not support multiple inheritance through the concept of classes, but it can be supported through the concept of Interface.
- Whenever we develop any inheritance application first create an object of bottom most derived class but not for top most base class.
- When we create an object of bottom most derived class, first we get the memory space for the data members of top most base class, and then we get the memory space for data member of other bottom most derived class.
- Bottom most derived class contains logical appearance for the data members of all top most base classes.
- If we do not want to give the features of base class to the derived class then the definition of the base class must be preceded by final hence final base classes are not reusable or not inheritable.
- If we are do not want to give some of the features of base class to derived class than such features of base class must be as private hence private features of base class are not inheritable or accessible in derived class.
- Data members and methods of a base class can be inherited into the derived class but constructors of base class can not be inherited because every constructor of a class is made for initializing its own data members but not made for initializing the data members of other classes.
- An object of base class can contain details about features of same class but an object of base class never contains the details about special features of its derived class (this concept is known as scope of base class object).
- For each and every class in java there exists an implicit predefined super class called java.lang.Object. because it providers garbage collection facilities to its sub classes for collecting un-used memory space and improved the performance of java application.

Example of Inheritance

```
class Faculty
{
float salary=30000;
}
class Science extends Faculty
{
float bonous=2000;
public static void main(String args[])
{
Science obj=new Science();
System.out.println("Salary is:"+obj.salary);
System.out.println("Bonous is:"+obj.bonous);
}
}
```

Output

```
Salary is: 30000.0
```

```
Bonous is: 2000.0
```

Why multiple inheritance is not supported in java?

Due to ambiguity problem java does not support multiple inheritance at class level.

Example

```
class A
{
void disp()
{
System.out.println("Hello");
}
}
class B
{
void disp()
System.out.println("How are you ?");
}
}
class C extends A,B  //suppose if it were
{
Public Static void main(String args[])
{
C obj=new C();
obj.disp();//Now which disp() method would be invoked?
}
}
```

In above code we call both class A and class B disp() method then it confusion which class method is call. So due to this ambiguity problem in java do not use multiple inheritance at class level, but it support at interface level.

Difference between Java Inheritance and C++ Inheritance

The main difference between java Inheritance and C++ Inheritance is; Java doesn't support multiple inheritance but C++ support.

Method Overloading in Java

Whenever same method name is exiting multiple times in the same class with different number of parameter or different order of parameters or different types of parameters is known as**method overloading**.

Why use method Overloading in Java ?

Suppose we have to perform addition of given number but there can be any number of arguments, if we write method such as a(int, int)for two arguments, b(int, int, int) for three arguments then it is very difficult for you and other programmer to understand purpose or behaviors of method they can not identify purpose of method. So we use method overloading to easily figure out the program. For example above two methods we can write sum(int, int) and sum(int, int, int) using method overloading concept.

Syntax

```
class  class_Name

{

Returntype  method()

{.........}

Returntype  method(datatype1 variable1)

{.........}

Returntype  method(datatype1 variable1, datatype2 variable2)

{.........}

Returntype  method(datatype2 variable2)

{.........}

Returntype  method(datatype2 variable2, datatype1 variable1)

{.........}

}
```

Different ways to overload the method

There are two ways to overload the method in java

- By changing number of arguments or parameters
- By changing the data type

By changing number of arguments

In this example, we have created two overloaded methods, first sum method performs addition of two numbers and second sum method performs addition of three numbers.

Example Method Overloading in Java

```
class Addition
{
void sum(int a, int b)
{
System.out.println(a+b);
}
void sum(int a, int b, int c)
{
System.out.println(a+b+c);
}
public static void main(String args[])
{
Addition obj=new Addition();
obj.sum(10, 20);
obj.sum(10, 20, 30);
}
}
```

Output

```
30

60
```

By changing the data type

In this example, we have created two overloaded methods that differs in data type. The first sum method receives two integer arguments and second sum method receives two float arguments.

Example Method Overloading in Java

```
class Addition
{
void sum(int a, int b)
{
System.out.println(a+b);
}
void sum(float a, float b)
{
System.out.println(a+b);
}
public static void main(String args[])
{
Addition obj=new Addition();
obj.sum(10, 20);
obj.sum(10.05, 15.20);
}
}
```

Output

```
30

25.25
```

Why Method Overloading is not possible by changing the return type of method?

In java, method overloading is not possible by changing the return type of the method because there may occur ambiguity. Let's see how ambiguity may occur:

because there was problem:

Example of Method Overloading

```
class Addition
{
 int sum(int a, int b)
 {
 System.out.println(a+b);
 }
 double sum(int a, int b)
 {
 System.out.println(a+b);
 }
 public static void main(String args[])
 {
 Addition obj=new Addition();
 int result=obj.sum(20,20); //Compile Time Error
 }
}
```

Explanation of Code

Example

```
int result=obj.sum(20,20);
```

Here how can java determine which sum() method should be called

Note: The scope of overloading is within the class.

Any object reference of class can call any of overloaded method.

Can we overload main() method ?

Yes, We can overload main() method. A Java class can have any number of main() methods. But run the java program, which class should have main() method with signature as "public static void main(String[] args). If you do any modification to this signature, compilation will be successful. But, not run the java program. we will get the run time error as main method not found.

Example of override main() method

Example

```
public class mainclass
{
public static void main(String[] args)
{
System.out.println("Execution starts from Main()");
}
void main(int args)
{
System.out.println("Override main()");
}
double main(int i, double d)
{
System.out.println("Override main()");
return d;
}
}
```

Outpot

```
Execution starts from Main()
```

Method Overriding in Java

Whenever same method name is existing in both base class and derived class with same types of parameters or same order of parameters is known as **method Overriding**. Here we will discuss about **Overriding in Java**.

Note: Without Inheritance method overriding is not possible.

Advantage of Java Method Overriding

- Method Overriding is used to provide specific implementation of a method that is already provided by its super class.
- Method Overriding is used for Runtime Polymorphism

Rules for Method Overriding

- method must have same name as in the parent class.
- method must have same parameter as in the parent class.

- must be IS-A relationship (inheritance).

Understanding the problem without method overriding

Lets understand the problem that we may face in the program if we do not use method overriding.

Example Method Overriding in Java

```
class Walking
{
void walk()
{
System.out.println("Man walking fastly");
}
}
class OverridingDemo
{
public static void main(String args[])
{
Man obj = new Man();
obj.walk();
}
}
```

Output

```
Man walking
```

Problem is that I have to provide a specific implementation of walk() method in subclass that is why we use method overriding.

Example of method overriding in Java

In this example, we have defined the walk method in the subclass as defined in the parent class but it has some specific implementation. The name and parameter of the method is same and there is IS-A relationship between the classes, so there is method overriding.

Example

```
class Walking
{
void walk()
{
System.out.println("Man walking fastly");
}
}
class Man extends walking
{
void walk()
{
System.out.println("Man walking slowly");
}
}

class OverridingDemo
{
public static void main(String args[])
{
Man obj = new Man();
obj.walk();
}
}
```

Output

```
Man walking slowly
```

Note: Whenever we are calling overridden method using derived class object reference the highest priority is given to current class (derived class). We can see in the above example high priority is derived class.

Note: super. (super dot) can be used to call base class overridden method in the derived class.

Accessing properties of base class with respect to derived class object

```
class A
{
int x;
void f1()
{
x=10;
System.out.println(x);
}
void f4()
{
System.out.println("this is f4()");
System.out.println("-----------------");
}
};
class B extends A
{
int y;
void f1()
{
int y=20;
System.out.println(y);
System.out.println("this is f1()");
System.out.println("------------------");
}
};
class C extends A
{
int z;
void f1()
{
z=10;
System.out.println(z);
System.out.println("this is f1()");
}
};
class Overide
{
public static void main(String[] args)
{
A a1=new B();
a1.f1();
a1.f4();
A c1=new C();
c1.f1();
c1.f4();
}
}
```

Example of Implement overriding concept

```
class Person
{
String name;
void sleep(String name)
{
this.name=name;
System.out.println(this.name +"is sleeping+8hr/day");
}
void walk()
{
System.out.println("this is walk()");
System.out.println("-----------------");
}
};
class Student extends Person
{
void writExams()
{
System.out.println("only student write the exam");
}
void sleep(String name)
{
super.name=name;
System.out.println(super.name +"is sleeping 6hr/day");
System.out.println("------------------");
}
};
class Developer extends Person
{
public void designProj()
{
System.out.println("Design the project");
}
void sleep(String name)
{
super.name=name;
System.out.println(super.name +"is sleeping 4hr/day");
System.out.println("------------------");
}
};
class OverideDemo
{
public static void main(String[] args)
{
Student s1=new Student();
s1.writExams();
s1.sleep("student");
s1.walk();
Developer d1=new Developer();
d1.designProj();
d1.sleep("developer");
}
}
```

Difference between Overloading and Overriding

	Overloading	**Overriding**
1	Whenever same method or Constructor is existing multiple times within a class either with different number of parameter or with different type of parameter or with different order of parameter is known as Overloading.	Whenever same method name is existing multiple time in both base and derived class with same number of parameter or same type of parameter or same order of parameters is known as Overriding.
2	Arguments of method must be different at least arguments.	Argument of method must be same including order.

3	Method signature must be different.	Method signature must be same.
4	Private, static and final methods can be overloaded.	Private, static and final methods can not be override.
5	Access modifiers point of view no restriction.	Access modifiers point of view not reduced scope of Access modifiers but increased.
6	Also known as compile time polymorphism or static polymorphism or early binding.	Also known as run time polymorphism or dynamic polymorphism or late binding.
7	Overloading can be exhibited both are method and constructor level.	Overriding can be exhibited only at method label.
8	The scope of overloading is within the class.	The scope of Overriding is base class and derived class.
9	Overloading can be done at both static and non-static methods.	Overriding can be done only at non-static method.
10	For overloading methods return type may or may not be same.	For overriding method return type should be same.

Note: In overloading we have to check only methods names (must be same) and arguments types (must be different) except these the remaining like return type access modifiers etc. are not required to check But in overriding every things check like method names arguments types return types access modifiers etc.

Interface in Java

Interface is similar to class which is collection of public static final variables (constants) and abstract methods.

The interface is a mechanism to achieve fully abstraction in java. There can be only abstract methods in the interface. It is used to achieve fully abstraction and multiple inheritance in Java.

Why we use Interface ?

- It is used to achieve fully abstraction.
- By using Interface, you can achieve multiple inheritance in java.
- It can be used to achieve loose coupling.

properties of Interface

- It is implicitly abstract. So we no need to use the abstract keyword when declaring an interface.
- Each method in an interface is also implicitly abstract, so the abstract keyword is not needed.
- Methods in an interface are implicitly public.
- All the data members of interface are implicitly public static final.

How interface is similar to class ?

Whenever we compile any Interface program it generate .class file. That means the bytecode of an interface appears in a .class file.

How interface is different from class ?

- You can not instantiate an interface.
- It does not contain any constructors.
- All methods in an interface are abstract.
- Interface can not contain instance fields. Interface only contains public static final variables.
- Interface is can not extended by a class; it is implemented by a class.
- Interface can extend multiple interfaces. It means interface support multiple inheritance

Behavior of compiler with Interface program

In the above image when we compile any interface program, by default compiler added public static final before any variable and public abstract before any method. Because **Interface** is design for fulfill universal requirements and to achieve fully abstraction.

Declaring Interfaces:

The **interface** keyword is used to declare an interface.

Example

```
interface Person
{
 datatype variablename=value;
 //Any number of final, static fields
 returntype methodname(list of parameters or no parameters)
 //Any number of abstract method declarations
}
```

Explanations

In the above syntax **Interface** is a keyword interface name can be user defined name the default signature of variable is public static final and for method is public abstract. JVM will be added implicitly public static final before data members and public abstract before method.

Example

```
public static final datatype variable name=value; ----> for data member

public abstract returntype methodname(parameters)---> for method
```

Implementing Interfaces:

A class uses the implements keyword to implement an interface. The implements keyword appears in the class declaration following the extends portion of the declaration.

Example

```
interface Person
{
void run();
}
class Employee implements Person
{
public void run()
{
System.out.println("Run fast");
}
}
```

When we use abstract and when Interface

If we do not know about any things about implementation just we have requirement specification then we should be go for **Interface**

If we are talking about implementation but not completely (partially implemented) then we should be go for **abstract**

Rules for implementation interface

- A class can implement more than one interface at a time.
- A class can extend only one class, but implement many interfaces.
- An interface can extend another interface, similarly to the way that a class can extend another class.

Relationship between class and Interface

- Any class can extends another class
- Any Interface can extends another Interface.
- Any class can Implements another Interface
- Any Interface can not extend or Implements any class.

Difference between Abstract class and Interface

	Abstract class	**Interface**
1	It is collection of abstract method and concrete methods.	It is collection of abstract method.
2	There properties can be reused commonly in a specific application.	There properties commonly usable in any application of java environment.
3	It does not support multiple inheritance.	It support multiple inheritance.
4	Abstract class is preceded by abstract keyword.	It is preceded by Interface keyword.
5	Which may contain either variable or constants.	Which should contains only constants.
6	The default access specifier of abstract class methods are default.	There default access specifier of interface method are public.
7	These class properties can be reused in other class using extend keyword.	These properties can be reused in any other class using implements keyword.
8	Inside abstract class we can take constructor.	Inside interface we can not take any constructor.
9	For the abstract class there is no restriction like initialization of variable at the time of variable declaration.	For the interface it should be compulsory to initialization of variable at the time of variable declaration.
10	There are no any restriction for abstract class variable.	For the interface variable can not declare variable as private, protected, transient, volatile.
11	There are no any restriction for abstract class method modifier that means we can use any modifiers.	For the interface method can not declare method as strictfp, protected, static, native, private, final, synchronized.

Example of Interface

```
interface Person
{
void run();  // abstract method
}
class A implements Person
{
public void run()
{
System.out.println("Run fast");
}
public static void main(String args[])
 {
 A obj = new A();
 obj.run();
 }
}
```

Output

```
Run fast
```

Multiple Inheritance using interface

Example

```
interface Developer
{
void disp();
}
interface Manager
{
void show();
}

class Employee implements Developer, Manager
{
public void disp()
{
System.out.println("Hello Good Morning");
}
public void show()
{
System.out.println("How are you ?");
}
public static void main(String args[])
{
Employee obj=new Employee();
obj.disp();
obj.show();
}
}
```

Output

```
Hello Good Morning

How are you ?
```

Marker or tagged interface

An interface that have no member is known as marker or tagged interface. For example: Serializable, Cloneable, Remote etc. They are used to provide some essential information to the JVM so that JVM may perform some useful operation.

Example

```
//Way of writing Serializable interface
```

```
public interface Serializable
{
}
```

Why interface have no constructor ?

Because, constructor are used for eliminate the default values by user defined values, but in case of interface all the data members are public static final that means all are constant so no need to eliminate these values.

Other reason because constructor is like a method and it is concrete method and interface does not have concrete method it have only abstract methods that's why interface have no constructor.

Abstraction in Java

Abstraction is the concept of exposing only the required essential characteristics and behavior with respect to a context.
Hiding of data is known as **data abstraction**. In object oriented programming language this is implemented automatically while writing the code in the form of class and object.

Real Life Example of Abstraction in Java

Abstraction shows only important things to the user and hides the internal details, for example, when we ride a bike, we only know about how to ride bikes but can not know about how it work? And also we do not know the internal functionality of a bike.

Another real life example of Abstraction is ATM Machine; All are performing operations on the ATM machine like cash withdrawal, money transfer, retrieve mini-statement…etc. but we can't know internal details about ATM.

Note: Data abstraction can be used to provide security for the data from the unauthorized methods.

Note: In Java language data abstraction can achieve using class.

Example of Abstraction

```
class Customer
{
int account_no;
float balance_Amt;
String name;
int age;
String address;
void balance_inquiry()
{
/* to perform balance inquiry only account number
is required that means remaining properties
are hidden for balance inquiry method */
}
void fund_Transfer()
{
/* To transfer the fund account number and
balance is required and remaining properties
are hidden for fund transfer method */
}
```

How to achieve Abstraction ?

There are two ways to achieve abstraction in java

- Abstract class (0 to 100%)
- Interface (Achieve 100% abstraction)

Read more about Interface and Abstract class in the previous section.

Difference between Encapsulation and Abstraction

Encapsulation is not providing full security because we can access private member of the class using reflection API, but in case of Abstraction we can't access static, abstract data member of a class.

Encapsulation in Java

Encapsulation is a process of wrapping of data and methods in a single unit is called encapsulation. Encapsulation is achieved in java language by class concept.
Combining of state and behavior in a single container is known as encapsulation. In java language encapsulation can be achieve using **class** keyword, state represents declaration of variables on attributes and behavior represents operations in terms of method.

Advantage of Encapsulation

The main advantage of using of encapsulation is to secure the data from other methods, when we make a data private then these data only use within the class, but these data not accessible outside the class.

Real life example of Encapsulation in Java

The common example of encapsulation is capsule. In capsule all medicine are encapsulated in side capsule.

Benefits of encapsulation

- Provides abstraction between an object and its clients.
- Protects an object from unwanted access by clients.

- Example: A bank application forbids (restrict) a client to change an Account's balance.

Let's see the Example of Encapsulation in java

Example

```
class Employee
{
private String name;

public String getName()
{
return name;
}
public void setName(String name){
this.name=name;
}
}

class Demo
{
public static void main(String[] args)
{
Employee e=new Employee();
e.setName("Harry");
System.out.println(e.getName());
}
}
```

Output

```
Harry
```

Polymorphism in Java

The process of representing one form in multiple forms is known as **Polymorphism**.

Polymorphism is derived from 2 greek words: **poly** and **morphs**. The word "poly" means many and "morphs" means forms. So polymorphism means many forms.

Polymorphism is not a programming concept but it is one of the principal of OOPs. For many objects oriented programming language polymorphism principle is common but whose implementations are varying from one objects oriented programming language to another object oriented programming language.

Real life example of polymorphism

Suppose if you are in class room that time you behave like a student, when you are in market at that time you behave like a customer, when you at your home at that time you behave like a son or daughter, Here one person present in different-different behaviors.

How to achieve Polymorphism in Java ?

In java programming the Polymorphism principal is implemented with method overriding concept of java.

Polymorphism principal is divided into two sub principal they are:

- Static or Compile time polymorphism
- Dynamic or Runtime polymorphism

Note: Java programming does not support static polymorphism because of its limitations and java always supports dynamic polymorphism.

Let us consider the following diagram

Here original form or original method always resides in base class and multiple forms represents overridden method which resides in derived classes.

In the above diagram the sum method which is present in BC class is called original form and the sum() method which are present in DC1 and DC2 are called overridden form hence Sum() method is originally available in only one form and it is further implemented in multiple forms. Hence Sum() method is one of the polymorphism method.

Example of Runtime Polymorphism in Java

In below example we create two class Person an Employee, Employee class extends Person class feature and override walk() method. We are calling the walk() method by the reference variable of Parent class. Since it refers to the subclass object and subclass method overrides the Parent class method, subclass method is invoked at runtime. Here method invocation is determined by the JVM not compiler, So it is known as runtime polymorphism.

Example of Polymorphism in Java

```
class Person
{
void walk()
{
System.out.println("Can Run....");
}
}
class Employee extends Person
{
```

```
void walk()
{
System.out.println("Running Fast...");
}
public static void main(String arg[])
{
Person p=new Employee(); //upcasting
p.walk();
}
}
```

Output

```
Running fast...
```

Dynamic Binding

Dynamic binding always says create an object of base class but do not create the object of derived classes. Dynamic binding principal is always used for executing polymorphic applications.

The process of binding appropriate versions (overridden method) of derived classes which are inherited from base class with base class object is known as dynamic binding.

Advantages of dynamic binding along with polymorphism with method overriding are.

- Less memory space
- Less execution time
- More performance

Static polymorphism

The process of binding the overloaded method within object at compile time is known as **Static polymorphism** due to static polymorphism utilization of resources (main memory space) is poor because for each and every overloaded method a memory space is created at compile time when it binds with an object. In C++ environment the above problem can be solve by using dynamic polymorphism by implementing with virtual and pure virtual function so most of the C++ developer in real worlds follows only dynamic polymorphism.

Dynamic polymorphism

In dynamic polymorphism method of the program binds with an object at runtime the advantage of dynamic polymorphism is allocating the memory space for the method (either for overloaded method or for override method) at run time.

Conclusion

The advantage of dynamic polymorphism is effective utilization of the resources, So Java always use dynamic polymorphism. Java does not support static polymorphism because of its limitation.

Exception Handling in Java

The process of converting system error messages into user friendly error message is known as **Exception handling**. This is one of the powerful feature of Java to handle run time error and maintain normal flow of java application.

Exception

An **Exception** is an event, which occurs during the execution of a program, that disrupts the normal flow of the program's Instructions.

Why use Exception Handling

Handling the exception is nothing but converting system error generated message into user friendly error message. Whenever an exception occurs in the java application, JVM will create an object of appropriate exception of sub class and generates system error message, these system generated messages are not understandable by user so need to convert it into user friendly error message. You can convert system error message into user friendly error message by using exception handling feature of java. For Example: when you divide any number by zero then system generate **/ by zero** so this is not understandable by user so you can convert this message into user friendly error message like **Don't enter zero for denominator.**

Hierarchy of Exception classes

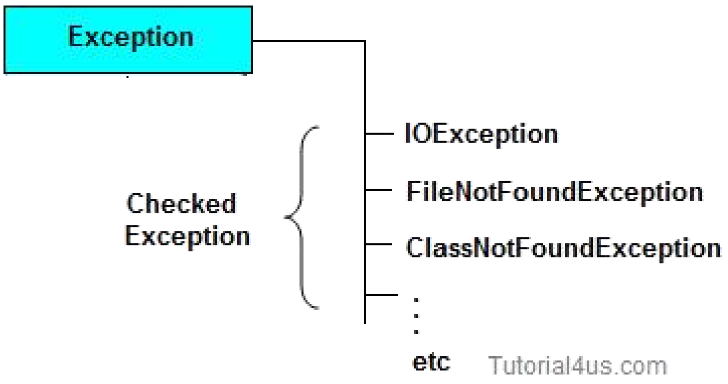
Type of Exception

- Checked Exception
- Un-Checked Exception

Checked Exception

Checked Exception are the exception which checked at compile-time. These exception are directly sub-class of java.lang.Exception class.

Only for remember: Checked means checked by compiler so checked exception are checked at compile-time.



Un-Checked Exception

Un-Checked Exception are the exception both identifies or raised at run time. These exception are directly sub-class of java.lang.RuntimeException class.

Note: In real time application mostly we can handle un-checked exception.

Only for remember: Un-checked means not checked by compiler so un-checked exception are checked at run-time not compile time.

Difference between checked Exception and un-checked Exception

	Checked Exception	**Un-Checked Exception**
1	checked Exception are checked at compile time	un-checked Exception are checked at run time
3	e.g. FileNotFoundException, NumberNotFoundException etc.	e.g. ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc.

Difference between Error and Exception

	Error	**Exception**
1	Can't be handle.	Can be handle.
2	Example: NoSuchMethodError OutOfMemoryError	Example: ClassNotFoundException NumberFormateException

Handling the Exception

Handling the exception is nothing but converting system error generated message into user friendly error message in others word whenever an exception occurs in the java application, JVM will create an object of appropriate exception of sub class and generates system error message, these system generated messages are not understandable by user so need to convert it into user-friendly error message. You can convert system error message into user-friendly error message by using exception handling feature of java.

Use Five keywords for Handling the Exception

- try
- catch
- finally
- throws
- throw

Syntax for handling the exception

Syntax

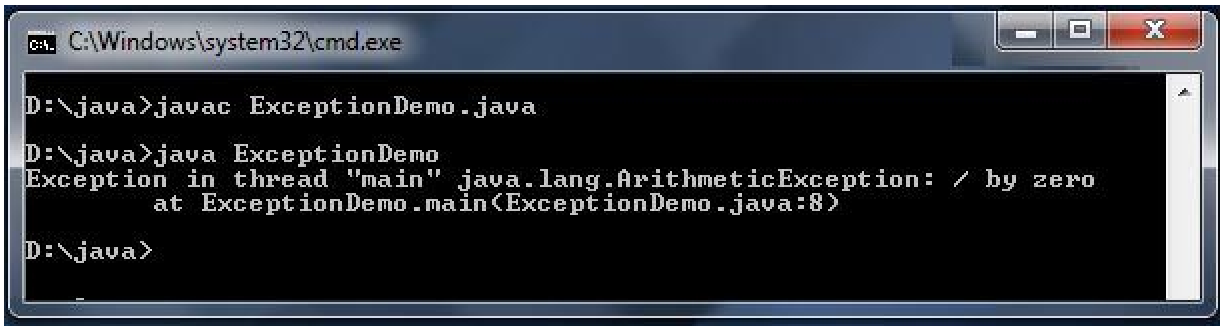
```
try
{
  // statements causes problem at run time
}
catch(type of exception-1 object-1)
{
  // statements provides user friendly error message
}
catch(type of exception-2 object-2)
{
  // statements provides user friendly error message
}
finally
{
  // statements which will execute compulsory
}
```

Example without Exception Handling

Syntax

```
class ExceptionDemo
{
public static void main(String[] args)
{
int a=10, ans=0;
ans=a/0;
System.out.println("Denominator not be zero");
}
}
```

Abnormally terminate program and give a message like below, this error message is not understandable by user so we convert this error message into user friendly error message, like "denominator not be zero".



```
D:\java>javac ExceptionDemo.java

D:\java>java ExceptionDemo
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at ExceptionDemo.main(ExceptionDemo.java:8)

D:\java>
```

Example of Exception Handling

Example

```
class ExceptionDemo
{
public static void main(String[] args)
{
int a=10, ans=0;
try
{
ans=a/0;
}
catch (Exception e)
{
System.out.println("Denominator not be zero");
}
}
}
```

Output

```
Denominator not be zero
```

Next -- try and catch block

try and catch block

try block

Inside **try block** we write the block of statements which causes executions at run time in other words try block always contains problematic statements.

Important points about try block

- If any exception occurs in try block then CPU controls comes out to the try block and executes appropriate catch block.
- After executing appropriate catch block, even through we use run time statement, CPU control never goes to try block to execute the rest of the statements.
- Each and every try block must be immediately followed by catch block that is no intermediate statements are allowed between try and catch block.

Syntax

```
try
{
 .....
}
/* Here no other statements are allowed
```

```
between try and catch block */
catch()
{
 ....
}
```

- Each and every try block must contains at least one catch block. But it is highly recommended to write multiple catch blocks for generating multiple user friendly error messages.
- One try block can contains another try block that is nested or inner try block can be possible.

Syntax

```
try
{
.......
try
{
.......
}
}
```

catch block

Inside **catch** block we write the block of statements which will generates user friendly error messages.

catch block important points

- Catch block will execute exception occurs in try block.
- You can write multiple catch blocks for generating multiple user friendly error messages to make your application strong. You can see below example.
- At a time only one catch block will execute out of multiple catch blocks.
- in catch block you declare an object of sub class and it will be internally referenced by JVM.

Example without Exception Handling

Example

```
class ExceptionDemo
{
public static void main(String[] args)
{
int a=10, ans=0;
ans=a/0;
System.out.println("Denominator not be zero");
}
}
```

Abnormally terminate program and give a message like below, this error message is not understandable by user so we convert this error message into user friendly error message, like "denominator not be zero".

Example of Exception Handling

Example

```
class ExceptionDemo
{
public static void main(String[] args)
{
int a=10, ans=0;
try
{
ans=a/0;
}
catch (Exception e)
{
System.out.println("Denominator not be zero");
}
}
}
```

Output

```
Denominator not be zero
```

Multiple catch block

You can write multiple catch blocks for generating multiple user friendly error messages to make your application strong. You can see below example.

Example

```
import java.util.*;
class ExceptionDemo
{
public static void main(String[] args)
{
int a, b, ans=0;
Scanner s=new Scanner(System.in);
System.out.println("Enter any two numbers: ");
try
{
        a=s.nextInt();
        b=s.nextInt();
        ans=a/b;
        System.out.println("Result: "+ans);
}
catch(ArithmeticException ae)
{
System.out.println("Denominator not be zero");
}
catch(Exception e)
{
System.out.println("Enter valid number");
}
}
}
```

Output

```
Enter any two number: 5 0

Denominator not be zero
```

finally Block in Exception Handling

Inside **finally**block we write the block of statements which will relinquish (released or close or terminate) the resource (file or database) where data store permanently.

finally block important points

- Finally block will execute compulsory
- Writing finally block is optional.
- You can write finally block for the entire java program
- In some of the circumstances one can also write try and catch block in finally block.

Example

```
class ExceptionDemo
{
public static void main(String[] args)
{
int a=10, ans=0;
try
{
ans=a/0;
}
catch (Exception e)
{
System.out.println("Denominator not be zero");
}
finally
{
System.out.println("I am from finally block");
}
}
}
```

Output

```
Denominator not be zero

I am from finally block
```

Exception Classes in Java

Exception are mainly classified into two type checked exception and un-checked exception.

Checked Exception Classes

- FileNotFoundException
- ClassNotFoundException
- IOException
- InterruptedException

Un-Checked Exception Classes

- ArithmeticException
- ArrayIndexOutOfBoundsException
- StringIndexOutOfBoundsException
- NumberFormateException
- NullPointerException
- NoSuchMethodException
- NoSuchFieldException

FileNotFoundException

If the given filename is not available in a specific location (in file handling concept) then FileNotFoundException will be raised. This exception will be thrown by the FileInputStream, FileOutputStream, and RandomAccessFile constructors.

ClassNotFoundException

If the given class name is not existing at the time of compilation or running of program then ClassNotFoundException will be raised. In other words this exception is occured when an application tries to load a class but no definition for the specified class name could be found.

IOException

This is exception is raised whenever problem occurred while writing and reading the data in the file. This exception is occurred due to following reason;

- When try to transfer more data but less data are present.
- When try to read data which is corrupted.
- When try to write on file but file is read only.

InterruptedException

This exception is raised whenever one thread is disturb the other thread. In other words this exception is thrown when a thread is waiting, sleeping, or otherwise occupied, and the thread is interrupted, either before or during the activity.

ArithmeticException

This exception is raised because of problem in arithmetic operation like divide by zero. In other words this exception is thrown when an exceptional arithmetic condition has occurred. For example, an integer "divide by zero".

Example

```
class ExceptionDemo
{
public static void main(String[] args)
{
int a=10, ans=0;
try
{
ans=a/0;
}
catch (Exception e)
{
System.out.println("Denominator not be zero");
}
}
}
```

ArrayIndexOutOfBoundsException

This exception will be raised whenever given index value of an array is out of range. The index is either negative or greater than or equal to the size of the array.

Example

```
int a[]=new int[5];
a[10]=100; //ArrayIndexOutOfBoundsException
```

StringIndexOutOfBoundsException

This exception will be raised whenever given index value of string is out of range. The index is either negative or greater than or equal to the size of the array.

Example

```
String s="Hello";
s.charAt(3);
s.charAt(10); // Exception raised
```

chatAt() is a predefined method of string class used to get the individual characters based on index value.

NumberFormateException

This exception will be raised whenever you trying to store any input value in the un-authorized datatype.

Example: Storing string value into int datatype.

Example

```
int a;
a="Hello";
```

Example

```
String s="hello";
int i=Integer.parseInt(s);//NumberFormatException
```

NoSuchMethodException

This exception will be raised whenever calling method is not existing in the program.

NullPointerException

A NullPointerException is thrown when an application is trying to use or access an object whose reference equals to null.

Example

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

StackOverFlowException

This exception throw when full the stack because the recursion method are stored in stack area.

Difference Between Throw and Throws Keyword

throw

throw is a keyword in java language which is used to throw any user defined exception to the same signature of method in which the exception is raised.

Note: throw keyword always should exist within method body.

whenever method body contain throw keyword than the call method should be followed by throws keyword.

Syntax

```
class className
{
returntype method(...) throws Exception_class
{
throw(Exception obj)
}
}
```

throws

throws is a keyword in java language which is used to throw the exception which is raised in the called method to it's calling method throws keyword always followed by method signature.

Example

```
returnType methodName(parameter)throws Exception_class....
{
.....
}
```

Difference between throw and throws

	throw	throws
1	throw is a keyword used for hitting and generating the exception which are occurring as a part of method body	throws is a keyword which gives an indication to the specific method to place the common exception methods as a part of try and catch block for generating user friendly error messages
2	The place of using throw keyword is always as a part of method body.	The place of using throws is a keyword is always as a part of method heading
3	When we use throw keyword as a part of method body, it is mandatory to the java programmer to write throws keyword as a part of method heading	When we write throws keyword as a part of method heading, it is optional to the java programmer to write throw keyword as a part of method body.

Example of throw and throws

Example

```
// save by DivZero.java

package pack;

public class DivZero
{
public void division(int a, int b)throws ArithmeticException
{
if(b==0)
{
ArithmeticException ae=new ArithmeticException("Does not enter zero for Denominator");
throw ae;
}
else
{
int c=a/b;
System.out.println("Result: "+c);
}
}
}
```

Compile: javac -d . DivZero.java

Example

```
// save by ArthException.java

import pack.DivZero;
import java.util.*;

class ArthException
```

```
{
public static void main(String args[])
{
System.out.println("Enter any two number: ");
Scanner s=new Scanner(System.in);
try
{
int a=s.nextInt();
int b=s.nextInt();
DivZero dz=new DivZero();
dz.division(a, b);
}
catch(Exception e)
{
System.err.println(e);
}
}
}
```

Compile: javac ArthException.java

Download Code Click

Steps to Compile and Run code

First you save throw-example files into you PC in any where, here i will save this file in C:\>

- C:\throw-example\>javac -d . DivZero.java
- C:\throw-example\>javac ArthException.java

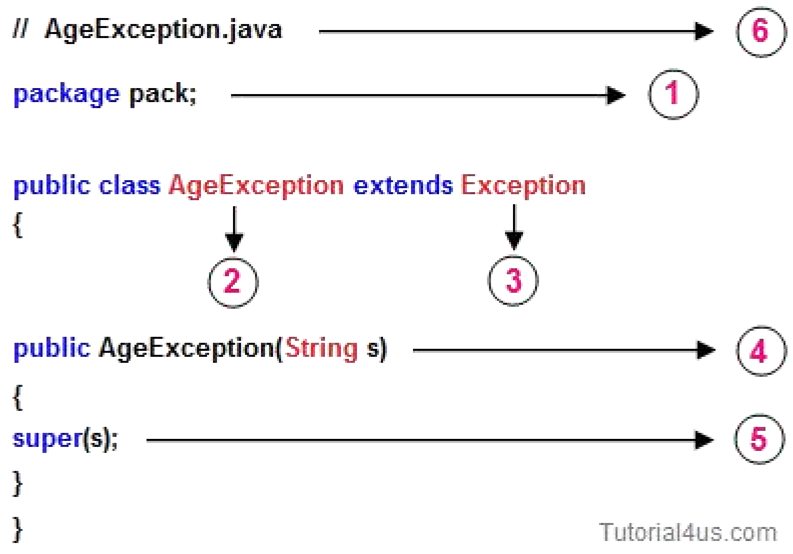
Note: First compile DivZero.java code then compile ArthException.java code.

Custom Exception in Java

If any exception is design by the user known as user defined or Custom Exception. Custom Exception is created by user.

Rules to design user defined Exception

1. Create a package with valid user defined name.
2. Create any user defined class.
3. Make that user defined class as derived class of Exception or RuntimeException class.
4. Declare parametrized constructor with string variable.
5. call super class constructor by passing string variable within the derived class constructor.
6. Save the program with public class name.java


// AgeException.java
6
package pack;
1
public class AgeException extends Exception
{
2
3
public AgeException(String s)
4
{
super(s);
5
}
}
Tutorial4us.com

Example

```
// save by AgeException.java
package nage;

public class AgeException extends Exception
{
public AgeException(String s)
{
super(s);
}
}
```

Compile: javac -d . AgeException.java

Example

```
// save by CheckAge.java
package nage;

public class CheckAge
{
public void verify(int age)throws AgeException
{
if (age>0)
{
System.err.print("valid age");
}
else
{
AgeException ae=new AgeException("Invalid age");
throw(ae);
}
}
}
```

Compile: javac -d . CheckAge.java

Example

```
// save by VerifyAgeException

import nage.AgeException;
import nage.CheckAge;
import java.util.*;

public class VerifyAgeException
{
public static void main(String args[])
{
int a;
System.out.println("Enter your age");
Scanner s=new Scanner(System.in);
a=s.nextInt();
try
{
CheckAge ca=new CheckAge();
ca.verify(a);
}
catch(AgeException ae)
{
System.err.println("Age should not be -ve");
}
catch(Exception e)
{
System.err.println(e);
}
}
}
```

Compile: javac VerifyAgeException.java

Steps to compile and run above program

First you save verify-age files into you PC in any where, here i will save this file in C:\>

- C:\verify-age\>javac -d . AgeException.java
- C:\verify-age\>javac -d . CheckAge.java
- C:\verify-age\>javac VerifyAgeException.java

Note: First compile AgeException.java code then CheckAge.java and at last compile VerifyAgeException.java code.

Multithreading in Java

Multithreading in java is a process of executing multiple threads simultaneously. The aim of multithreading is to achieve the concurrent execution.

Thread

Thread is a lightweight components and it is a flow of control. In other words a flow of control is known as thread.

State or Life cycle of thread

State of a thread are classified into five types they are

1. New State
2. Ready State
3. Running State
4. Waiting State
5. Halted or dead State

New State

If any new thread class is created that represent new state of a thread, In new state thread is created and about to enter into main memory. No memory is available if the thread is in new state.

Ready State

In ready state thread will be entered into main memory, memory space is allocated for the thread and 1st time waiting for the CPU.

Running State

Whenever the thread is under execution known as running state.

Halted or dead State

If the thread execution is stoped permanently than it comes under dead state, no memory is available for the thread if its comes to dead state.

Note: If the thread is in new or dead state no memory is available but sufficient memory is available if that is in ready or running or waiting state.

Achieve multithreading in java

In java language multithreading can be achieve in two different ways.

1. Using thread class

2. Using Runnable interface

Using thread class

In java language multithreading program can be created by following below rules.

1. Create any user defined class and make that one as a derived class of thread class.

```
class  Class_Name  extends  Thread
{
........
}
```

2. Override run() method of Thread class (It contains the logic of perform any operation)
3. Create an object for user-defined thread class and attached that object to predefined thread class object.

   Class_Name obj=new Class_Name Thread t=new Thread(obj);

4. Call start() method of thread class to execute run() method.
5. Save the program with filename.java

Example of multithreading using Thread class

Thread based program for displaying 1 to 10 numbers after each and every second.

```
// Threaddemo2.java

class Th1 extends Thread
{
public void run()
{
try
{
for(int i=1;i< =10;i++)
{
System.out.println("value of i="+i);
Thread.sleep(1000);
}
}
catch(InterruptedException ie)
{
System.err.println("Problem in thread execution");
}
}
}
class Threaddemo2
{
public static void main(String args[])
{
Th1   t1=new   Th1();
System.out.println("Execution status of t1 before start="+t1.isAlive());
t1.start();
System.out.println("Execution status of t1 before start="+t1.isAlive());
try
{
Thread.sleep(5000);
}
catch(InterruptedException ie)
{
System.out.println("Problem in thread execution");
}
System.out.println("Execution status of t1 during execution="+t1.isAlive());
try
{
Thread.sleep(5001);
}
catch(InterruptedException ie)
{
System.out.println("problem in thread execution");
}
System.out.println("Execution status of t1 after completation="+t1.isAlive());
}
}
```

Output

```
Execution status of t1 before start=false    //new state

Execution status of t1 after start=true       //ready state

1

2

3

4

5

6

Execution status of t1 during execution=true              //running state

7

8

9
```

```
10

Execution status of t1 after completation=false          //halted state
```

Thread class properties

Thread class contains constant data members, constructors, predefined methods.

Constant data members

- MAX-PRIORITY
- MIN-PRIORITY
- NORM-PRIORITY

MAX-PRIORITY

Which represent the minimum priority that a thread can have whose values is 10.

```
Syntax:

public static final int MAX-PRIORITY=10
```

MIN-PRIORITY

Which represents the minimum priority that a thread can have.

```
Syntax:

public static final int MIN-PRIORITY=0
```

NORM-PRIORITY

Which represent the default priority that is assigned to a thread.

```
Syntax:

public static final int NORM-PRIORITY=5
```

Constructors of Thread class

- Thread()
- Thread(String name)

- Thread(object)
- Thread(object, String name)

Thread()

Which will be execute to set the predefined name for newly created thread, these names are generally in the form of thread -0, thread -1, ...

Syntax to call constructor:

Syntax

```
Thread t=new Thread();
```

Thread(String name)

Which can be used to provide user defined name for newly created thread.

Syntax

```
Thread t=new Thread("newthread");
```

Thread(object)

Which can be used to provide default name for newly created user defined thread.

Syntax

```
UserdefinedThreadclass  obj=new UserdefinedThreadclass();
Thread t=new Thread("obj");
```

object, String name

Which will be used to provide user defined name for the newly created user defined thread.

Syntax

```
UserdefinedThreadclass  obj=new UserdefinedThreadclass();
Thread t=new Thread(object, "secondthread");
```

Methods of Thread class

- getPriority()
- setPriority()
- getName()
- setName()
- isDeamon()
- run()
- start()
- sleep()
- suspend()
- resume()
- stop()
- isAlive()
- currentThread()
- join()
- getState()
- yield()

getPriority()

This method is used to get the current priority of thread.

```
Thread t=new Thread();
int x=t.getPriority();
System.out.println(x);
```

setPriority()

This method is used to set the current priority of thread.

```
Thread t=new Thread();
t.setPriority(any priority number between o to 10)
or
t.setPriority(Thread.MAX-PRIORITY)
```

getName()

This method is used to get the current executing thread name.

```
Thread t=new Thread();
String s=t.getName();
System.out.println(s);
```

setName()

This method is used to set the userdefined name for the thread.

```
Thread t=new Thread();
t.setName("mythread");
```

isDeamon()

Which returns true if the current thread is background thread otherwise return false.

```
Thread t=new Thread();
boolean b=t.isDeamon();
```

run()

Which contains the main business logic that can be executed by multiple threads simultaneously in every user defined thread class run method should be overridden.

```
public Class_Name extends Thread
{
public void run()
{
.....
.....
}
}
```

start()

Used to convert ready state thread to running state.

```
Thread t=new Thread();
t.start();
```

sleep()

Used to change running state thread to ready state based on time period it is a static method should be called with class reference.

```
public static final sleep(long milisecond)throws InterruptedException
{
try
{
Thread.sleep(3000);
}
catch(InterruptedException ie)
{
........
........
}
}
```

Once the given time period is completed thread state automatically change from waiting to running state.

suspend()

Used to convert running state thread to waiting state, which will never come back to running state automatically.

```
Thread t=new Thread();
t.suspend();
```

resume()

Used to change the suspended thread state(waiting state) to ready state.

```
Thread t=new Thread();
t.resume();
```

Note: Without using suspend() method resume() method can not be use.

What is the difference between sleep() and suspend()

Sleep() can be used to convert running state to waiting state and automatically thread convert from waiting state to running state once the given time period is completed. Where as suspend() can be used to convert running state thread to waiting state but it will never return back to running state automatically.

stop()

This method is used to convert running state thread to dead state.

```
Thread t=new Thread();
t.stop();
```

isAlive()

Which is return true if the thread is in ready or running or waiting state and return false if the thread is in new or dead state.

```
Thread t=new Thread();
t.isAlive();
```

currentThread()

Used to get the current thread detail like thread name thread group name and priority

```
Thread t=new Thread();
t.currentThread();
```

Note:

- The default thread name is thread-0, (if it is a main thread default name is main)
- The default thread group name is main
- Default thread priority is "5" is normal priority.

join()

Which can be used to combined more than one thread into a single group signature is public final void join()throws InterruptedException

```
try
{
t.join();
t2.join();
.....
.....
}
```

getState()

This method is used to get the current state of thread.

```
Thread t=new Thread();
t.getState();
```

yield()

Which will keep the currently executing thread into temporarily pass and allows other threads to execute

Using Runnable Interface

Runnable is one of the predefined interface in java.lang package, which is containing only one method and whose prototype is " Public abstract void run "

The run() method of thread class defined with null body and run() method of Runnable interface belongs to abstract. Industry is highly recommended to override abstract run() method of Runnable interface but not recommended to override null body run() method of thread class.

In some of the circumstance if one derived class is extending some type of predefined class along with thread class which is not possible because java programming never supports multiple inheritance. To avoid this multiple inheritance problem, rather than extending thread class we implement Runnable interface.

Rules to create the thread using Runnable interface

- Create any user defined class and implements runnable interface within that
- Override run() method within the user defined class.
- call start() method to execute run() method of thread class
- Save the program with classname.java

```
class  Class_Name  implement Runnable
{
public void run()
{
........
}
}
Class_Name obj=new Class_name();
Thread t=new Thread();
t.start();
```

Note: While implementing runnable interface it is very mandatory to attach user defined thread class object reference to predefined thread class object reference. It is optional while creating thread by extending Thread class.

Thread Synchronization

Whenever multiple threads are trying to use same resource than they may be chance to of getting wrong output, to overcome this problem thread synchronization can be used.

Definition: Allowing only one thread at a time to utilized the same resource out of multiple threads is known as thread synchronization or thread safe.

In java language thread synchronization can be achieve in two different ways.

1. Synchronized block
2. Synchronized method

Note: synchronization is a keyword(access modifier in java)

Synchronized block

Whenever we want to execute one or more than one statement by a single thread at a time(not allowing other thread until thread one execution is completed) than those statement should be placed in side synchronized block.

```
class  Class_Name  implement Runnable or extends Thread
{
public void run()
{
synchronized(this)
{
.......
.......
}
}
}
```

Synchronized method

Whenever we want to allow only one thread at a time among multiple thread for execution of a method than that should be declared as synchronized method.

```
class  Class_Name  implement Runnable or extends Thread
{
public void run()
{
synchronized void fun()
{
.......
.......
}
public void run()
{
fun();
....
}
}
```

Interthread Communication

The process of execution of exchanging of the data / information between multiple threads is known as Interthread communication or if an output of first thread giving as an input to second thread the output of second thread giving as an input to third thread then the communication between first second and third thread known as Interthread communication.

In order to develop Interthread communication application we use some of the methods of java.lang.Object class and these methods are known as Interthread communication methods.

Interthread communication methods

1. public final void wait(long msec)
2. public final void wait()
3. public final void notify()
4. public final void notifyAll()

public final void wait(long msec)

public final void wait (long msec) is used for making the thread to wait by specifying the waiting time in terms of milliseconds. Once the waiting time is completed, automatically the thread will be interred into ready state from waiting state. This methods is not recommended to used to make next thread to wait on the basis of time because java programmer may not be able to decide or determine the CPU burst time of current thread and CPU burst time is decided by OS but not by the programmer.

public final void wait()

public final void wait() is used for making the thread to wait without specifying any waiting time this method is recommended to use to make the next thread to wait until current thread complete its execution.

public final void notify()

public final void notify() is used for transferring one thread at a time from waiting state to ready state.

public final void notifyAll()

public final void notifyAll() is used for transferring all the threads at a time from waiting state to ready state.

Note: public final void wait (long msec) and public final void wait() throws a predefined Exception called java.lang.InterruptedException.

String Handling in Java

The basic aim of **String Handling** concept is storing the string data in the main memory (RAM), manipulating the data of the String, retrieving the part of the String etc. **String Handling** provides a lot of concepts that can be performed on a string such as concatenation of string, comparison of string, find sub string etc.

Character

It is an identifier enclosed within single quotes (' '). Example: 'A', '$', 'p'

String:

String is a sequence of characters enclosed within double quotes (" ") is known as **String.** Example: "Java Programming".

In java programming to store the character data we have a fundamental datatype called **char**. Similarly to store the string data and to perform various operation on String data, we have three predefined classes they are:

- String
- StringBuffer
- StringBuilder

String class

It is a predefined class in java.lang package can be used to handle the String. String class is**immutable** that means whose content can not be changed at the time of execution of program.

String class object is immutable that means when we create an object of String class it never changes in the existing object. Example:

Example

```
class StringHandling
{
public static void main(String arg[])
{
String s=new String("java");
s.concat("software");
System.out.println(s);
}
}
```

Output

```
java
```

Explanation: Here we can not change the object of String class so output is only java not java software.

Methods of String class

length()

length(): This method is used to get the number of character of any string.

Example

```
class StringHandling
{
public static void main(String arg[])
{
int l;
String s=new String("Java");
l=s.length();
System.out.println("Length: "+l);
}
}
```

Output

```
Length: 4
```

charAt(index)

charAt(): This method is used to get the character at a given index value.

Example

```
class StringHandling
{
public static void main(String arg[])
{
char c;
String s=new String("Java");
c=s.charAt(2);
System.out.println("Character: "+c);
}
}
```

Output

```
Character: v
```

toUpperCase()

toUpperCase(): This method is use to convert lower case string into upper case.

Example

```
class StringHandling
{
public static void main(String arg[])
{
String s="Java";
System.out.println("String: "+s.toUpperCase());
}
}
```

Output

```
String: JAVA
```

toLowerCase()

toLowerCase(): This method is used to convert lower case string into upper case.

Example

```
class StringHandling
{
public static void main(String arg[])
{
String s="JAVA";
System.out.println("String: "+s.toLowerCase());
}
}
```

Output

```
String: java
```

concat()

concat(): This method is used to combined two string.

Example

```
class StringHandling
{
public static void main(String arg[])
{
String s1="Hitesh";
String s2="Raddy";
System.out.println("Combined String: "+s1.concat(s2));
}
}
```

Output

```
Combined String: HiteshRaddy
```

equals()

equals(): This method is used to compare two strings, It return true if strings are same otherwise return false. It is case sensitive method.

Example

```
class StringHandling
{
public static void main(String arg[])
{
String s1="Hitesh";
String s2="Raddy";
String s3="Hitesh";
System.out.println("Compare String: "+s1.equals(s2));
System.out.println("Compare String: "+s1.equals(s3));
}
}
```

Output

```
Compare String: false

Compare String: true
```

equalsIgnoreCase()

equalsIgnoreCase(): This method is case insensitive method, It return true if the contents of both strings are same otherwise false.

Example

```
class StringHandling
{
public static void main(String arg[])
{
String s1="Hitesh";
String s2="HITESH";
String s3="Raddy";
System.out.println("Compare String: "+s1.equalsIgnoreCase(s2));
System.out.println("Compare String: "+s1.equalsIgnoreCase(s3));
}
}
```

Output

```
Compare String: true

Compare String: false
```

compareTo()

compareTo(): This method is used to compare two strings by taking unicode values, It return 0 if the string are same otherwise return +ve or -ve integer values.

Example

```
class StringHandling
{
public static void main(String arg[])
{
String s1="Hitesh";
String s2="Raddy";
int i;
i=s1.compareTo(s2);
if(i==0)
{
System.out.println("Strings are same");
}
else
{
System.out.println("Strings are not same");
}
}
}
```

Output

```
Strings are not same
```

compareToIgnoreCase()

compareToIgnoreCase(): This method is case insensitive method, which is used to compare two strings similar to compareTo().

Example

```
class StringHandling
{
public static void main(String arg[])
{
String s1="Hitesh";
String s2="HITESH";
int i;
i=s1.compareToIgnoreCase(s2);
if(i==0)
{
System.out.println("Strings are same");
}
else
{
System.out.println("Strings are not same");
}
}
}
```

Output

```
Strings are same
```

startsWith()

startsWith(): This method return true if string is start with given another string, otherwise it returns false.

Example

```
class StringHandling
{
public static void main(String arg[])
{
String s="Java is programming language";
System.out.println(s.startsWith("Java"));
}
}
```

Output

```
true
```

endsWith()

endsWith(): This method return true if string is end with given another string, otherwise it returns false.

Example

```
class StringHandling
{
public static void main(String arg[])
{
String s="Java is programming language";
System.out.println(s.endsWith("language"));
}
}
```

Output

```
true
```

subString()

subString(): This method is used to get the part of given string.

Example

```
class StringHandling
{
public static void main(String arg[])
{
String s="Java is programming language";
System.out.println(s.substring(8)); // 8 is starting index
}
}
```

Output

```
programming language
```

Example

```
class StringHandling
{
public static void main(String arg[])
{
String s="Java is programming language";
System.out.println(s.substring(8, 12));
}
}
```

Output

```
prog
```

indexOf()

indexOf(): This method is used find the index value of given string. It always gives starting index value of first occurrence of string.

Example

```
class StringHandling
{
public static void main(String arg[])
{
String s="Java is programming language";
System.out.println(s.indexOf("programming"));
}
}
```

Output

```
8
```

lastIndexOf()

lastIndexOf(): This method used to return the starting index value of last occurence of the given string.

Example

```
class StringHandling
{
public static void main(String arg[])
{
String s1="Java is programming language";
String s2="Java is good programming language";
System.out.println(s1.lastIndexOf("programming"));
System.out.println(s2.lastIndexOf("programming"));
}
}
```

Output

```
8

13
```

trim()

trim(): This method remove space which are available before starting of string and after ending of string.

Example

```
class StringHandling
```

```
{
public static void main(String arg[])
{
String s="    Java is programming language    ";
System.out.println(s.trim());
}
}
```

Output

```
Java is programming language
```

split()

split(): This method is used to divide the given string into number of parts based on delimiter (special symbols like @ space ,).

Example

```
class StringHandling
{
public static void main(String arg[])
{
String s="contact@tutorial4us.com";
String[] s1=s.split("@");  // divide string based on @
for(String c:s1) //  foreach loop
{
System.out.println(c);
}
}
}
```

Output

```
contact

@tutorial4us.com
```

replace()

replace(): This method is used to return a duplicate string by replacing old character with new character.

Note: In this method data of original string will never be modify.

Example

```
class StringHandling
{
public static void main(String arg[])
{
```

```
String s1="java";
String s2=s1.replace('j', 'k');
System.out.println(s2);
}
}
```

Output

```
kava
```

StringBuffer Class in Java

It is a predefined class in java.lang package can be used to handle the String, whose object is mutable that means content can be modify.

StringBuffer class is working with thread safe mechanism that means multiple thread are not allowed simultaneously to perform operation of StringBuffer.

StringBuffer class object is **mutable** that means when we create an object of StringBulder class it can be change.

Example StringBuffer

```
class StringHandling
{
public static void main(String arg[])
{
StringBuffer sb=new StringBuffer("java");
sb.append("software");
System.out.println(sb);
}
}
```

Output

```
javasoftware
```

Explanation: Here we can changes in the existing object of StringBuffer class so output is javasoftware.

Difference Between String and StringBuffer

String	**StringBuffer**
The data which enclosed within double quote (" ") is by default treated as String class.	The data which enclosed within double quote (" ") is not by default treated as StringBuffer class
String class object is immutable	StringBuffer class object is mutable
When we create an object of String class by	When we create an object of StringBuffer class by

default no additional character memory space is created.	default we get 16 additional character memory space.

Similarities Between String and StringBuffer

- Both of them are belongs to public final. so that they never participates in inheritance that is is-A relationship is not possible but they can always participates in As-A and Uses-A relationship.
- We can not override the method of String and StringBuffer.

Methods of StringBuffer class

reverse()

reverse(): This method is used to reverse the given string and also the new value is replaced by the old string.

Example

```
class StringHandling
{
public static void main(String arg[])
{
StringBuffer sb=new StringBuffer("java code");
System.out.println(sb.reverse());
}
}
```

Output

```
edoc avaj
```

insert()

insert(): This method is used to insert either string or character or integer or real constant or boolean value at a specific index value of given string.

Example

```
class StringHandling
{
public static void main(String arg[])
{
StringBuffer sb=new StringBuffer("this is my java code");
System.out.println(sb.insert(11, "first "));
}
}
```

Output

```
this is my first java code
```

append()

append(): This method is used to add the new string at the end of original string.

Example

```
class StringHandling
{
public static void main(String arg[])
{
StringBuffer sb=new StringBuffer("java is easy");
System.out.println(sb.append(" to learn"));
}
}
```

Output

```
java is easy to learn
```

replace()

replace() This method is used to replace any old string with new string based on index value.

Example

```
class StringHandling
{
public static void main(String arg[])
{
StringBuffer sb=new StringBuffer("This is my code");
System.out.println(sb.replace(8, 10, "java"));
}
}
```

Output

```
This is java code
```

Explanation: In above example java string is replaced with old string (my) which is available between 8 to 10 index value.

deleteCharAt()

deleteCharAt(): This method is used to delete a character at given index value.

Example

```
class StringHandling
```

```
{
public static void main(String arg[])
{
StringBuffer sb=new StringBuffer("java");
System.out.println(sb.deleteCharAt(3));
}
}
```

Output

```
jav
```

delete()

delete(): This method is used to delete string form given string based on index value.

Example

```
class StringHandling
{
public static void main(String arg[])
{
StringBuffer sb=new StringBuffer("java is easy to learn");
StringBuffer s;
s=sb.delete(8, 13);
System.out.println(sb);
}
}
```

Output

```
java is to learn
```

Explanation: In above example string will be deleted which is existing between 8 and 13 index value.

toString()

toString(): This method is used to convert mutable string value into immutable string.

Example

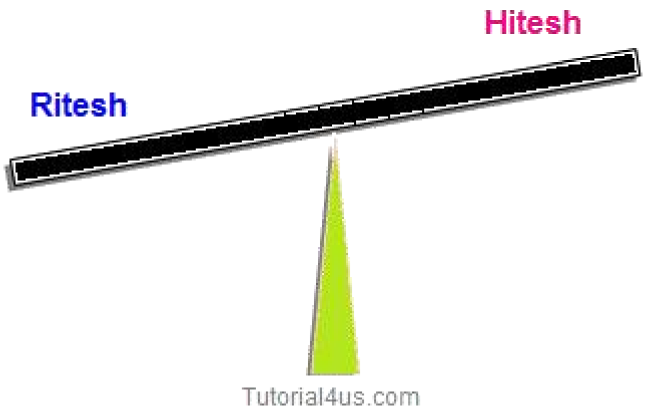
```
class StringHandling
{
public static void main(String arg[])
{
StringBuffer sb=new StringBuffer("java");
String s=sb.toString();
System.out.println(s);
s.concat("code");
}
}
```

Output

```
java
```

String Compare in Java

There are three way to compare string object in java:



- By equals() method
- By == operator
- By compreTo() method

equals() Method in Java

equals() method always used to comparing contents of both source and destination String. It return true if both string are same in meaning and case otherwise it returns false. It is case sensitive method.

Example

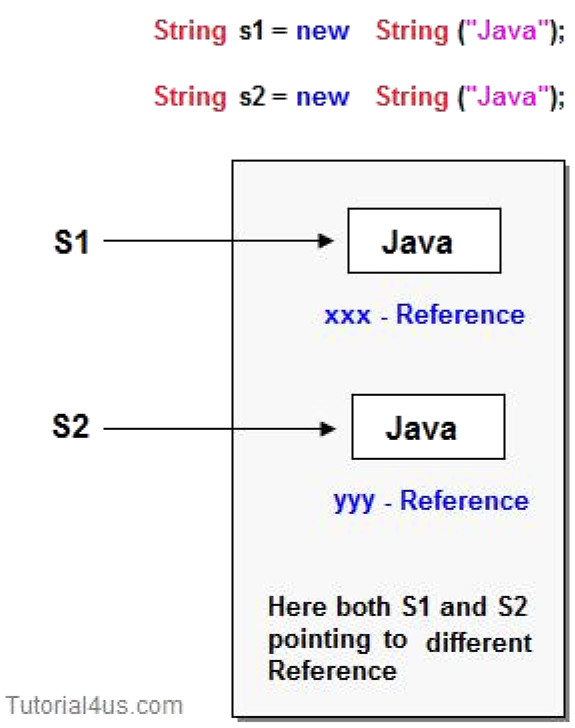
```
class StringHandling
{
public static void main(String arg[])
{
String s1="Hitesh";
String s2="Raddy";
String s3="Hitesh";
System.out.println("Compare String: "+s1.equals(s2));
System.out.println("Compare String: "+s1.equals(s3));
}
}
```

Output

```
Compare String: false

Compare String: true
```

== or Double Equals to Operator in Java

== Operator is always used for comparing references of both source and destination objects but not their contents.



```
String s1 = new String ("Java");
String s2 = new String ("Java");
```

Example

```
class StringHandling
{
public static void main(String arg[])
{
String s1=new String("java");
String s2=new String("java");
if(s1==s2)
{
System.out.println("Strings are same");
}
else
{
System.out.println("Strings are not same");
}
}
}
```

Output

```
Strings are not same
```

compareTo() Method in Java

comapreTo() method can be used to compare two string by taking Unicode values. It returns 0 if the string are same otherwise returns either +ve or -ve integer.

Example

```
class StringHandling
{
public static void main(String arg[])
{
String s1="Hitesh";
String s2="Raddy";
int i;
i=s1.compareTo(s2);
if(i==0)
{
System.out.println("Strings are same");
}
else
{
System.out.println("Strings are not same");
}
}
}
```

Output

```
Strings are not same
```

Difference between equals() method and == operator

equals() method always used to comparing contents of both source and destination String.

== Operator is always used for comparing references of both source and destination objects but not their contents.

String Concatenation

There are two way to concat string object in java:

- By + (string concatenation) operator
- By concat() method

By + operator

Using Java string concatenation operator (+) you can combined two or more strings.

Example

```
class StringHandling
{
public static void main(String arg[])
{
String s= "Java" + "Code";
System.out.println(s);
}
}
```

Output

```
JavaCode
```

By concat() method

concat() method is used to combined two strings.

Example

```
class StringHandling
{
public static void main(String arg[])
{
String s1="Java";
String s2="Code";
String s3=s1.concat(s2);
System.out.println(s3);
}
}
```

Output

```
JavaCode
```

StringBuilder

It is a predefined class in java.lang package can be used to handle the String. StringBuilder class is almost similar to to StringBuffer class. It is also a mutable object. The main difference StringBuffer and StringBuilder class is StringBuffer is thread safe that means only one threads allowed at a time to work on the String where as StringBuilder is not thread safe that means multiple threads can work on same String value.

Difference between StringBuffer and StringBuilder

All the things between StringBuffer and StringBuilder are same only difference is StringBuffer is synchronized and StringBuilder is not synchronized. synchronized means one thread is allow at a time so it thread safe. Not synchronized means multiple threads are allow at a time so it not thread safe.

	StringBuffer	**StringBuilder**
1	It is thread safe.	It is not thread safe.
2	Its methods are synchronized and provide thread safety.	Its methods are not synchronized and unable to provide thread safety.
3	Relatively performance is low because thread need to wait until previous process is complete.	Relatively performance is high because no need to wait any thread it allows multiple thread at a time.
1	Introduced in 1.0 version.	Introduced in 1.5 version.

When we use String, StringBuffer and StringBuilder

- If the content is fixed and would not change frequently then we use String.
- If content is not fixed and keep on changing but thread safety is required then we use StringBuffer
- If content is not fixed and keep on changing and thread safety is not required then we use StringBuilder

StringTokenizer in Java

It is a pre defined class in **java.util** package can be used to split the given string into tokens (parts) based on delimiters (any special symbols or spaces).

Suppose that we have any string like "Features of Java_Language" when we use stringTokenizer this string is split into tokens whenever spaces and special symbols present. After split string are :

Example

```
Features

of

Java

Language
```

Methods of StringTokenizer

- hasMoreTokens()
- nextToken()

hasMoreTokens()

It is predefined method of StringTokenizer class used to check whether given StringTokenizer having any elements or not.

nextToken()

Which can be used to get the element from the StringTokenizer.

Example of StringTokenizer:

Example of StringTokenizer

```
import java.util.*;
class Stringtokenizerdemo
{
public static void main(String args[])
{
String str="He is a gentle man";
StringTokenizer st=new StringTokenizer(str," ");
System.out.println("The tokens are: ");
while(st.hasMoreTokens())
{
String one=st.nextToken();
System.out.println(one);
}
}
}
```

Output

```
The tokens are:

He

is

a

gentle

man
```

Boxing and Unboxingin Java

Definition of Auto Boxing

The process of implicitly converting fundamental type values into the equivalent wrapper class object is known as auto boxing.

Converting fundamental type values into object type values:

In order to convert the fundamental data into equivalent wrapper class object type data we use the following generalized predefined parameterized constructor by taking fundamental data type as a parameter.

Example:

In JDK 1.4 converting fundamental data type values into wrapper class object is known as boxing. In the case of JDK 1.5 and in higher version it is optional to the Java programmer to convert the fundamental data type value into the equivalent wrapper class object. That is implicitly taken care by JVM and it is known as auto boxing.

Definition of auto Unboxing.

In process of implicitly conversion objects type data into fundamental type data is known as auto un-boxing.

Converting object type value into fundamental type value:

In order to convert wrapper class object data into fundamental type data, we use the following predefined instance method present in each and every wrapper class.

In case of JDK 1.5 and in higher version it is optional to the Java programmer to convert object data into fundamental type data and this process is known as auto un-boxing and it takes care by the JVM.

Java Scanner Class in Java

Scanner is one of the predefined class which is used for reading the data dynamically from the keyboard.

Import Scanner class

Import Scanner Class in Java

```
java.util.Scanner
```

Constructor of Scanner Class

Scanner(InputStream)

This constructor create an object of Scanner class by talking an object of InputStream class. An object of InputStream class is called **in** which is created as a static data member in the System class.

Syntax of Scanner Class in Java

```
Scanner  sc=new  Scanner(System.in);
```

Here the object **'in'** is use the control of keyboard

Instance methods of Scanner Class

	Method	Description
1	public byte nextByte()	Used for read byte value
2	public short nextShort()	Used for read short value
3	public int nextInt()	Used for read integer value
4	public long nextLong()	Used for read numeric value
5	public float nextLong()	Used for read numeric value
6	public double nextDouble()	Used for read double value
7	public char nextChar()	Used for read character
8	public boolean nextBoolean()	Used for read boolean value
9	public String nextLine()	Used for reading any kind of data in the form of String data.

Method 1 to 8 are used for reading fundamental values from the keyboard. Method 9 (public String nextLine()) is used for reading any kind of data in the form of String data.

For Remember all above methods

From method 1 to 8 combindly we represent as public xxx nextxxx(). Here xxx represents any fundamental data type. These methods are used for reading the fundamental data from keyboard.

Accept two values dynamically from the keyboard and compute sum.

Example of Scanner Class in Java

```
import java.util.Scanner
public class ScannerDemo
{
public static void main(String args[])
{
```

```
Scanner  s=new  Scanner(System.in);
System.out.println("Enter first no= ");
int num1=s.nextInt();
System.out.println("Enter second no= ");
int num2=s.nextInt();
System.out.println("Sum of no is= "+(num1+num2));
}
}
```

Output

```
Enter first no=4

Enter second no=5

Sum of no is=9
```

Program which is accept two number as a string and compute their sum.

Example

```
import java.util.Scanner;
class Dataread
{
public static void main(String[] args)
{
Scanner s=new Scanner(System.in);
System.out.println("Enter first number: ");
String s1=s.nextLine();
System.out.println("Enter second number: ");
String s2=s.nextLine();
int res=Integer.parseInt(s1) + Integer.parseInt(s2);
System.out.println("Sum= "+res);
}
}
```

Output

```
Enter first number: 5

Enter second number: 6

Sum= 11
```