

**BE POSITIVE.....**

**"If you are not willing to learn no one can help you,  
If you are determined to learn ,no one can stop you"**

**Think once.....**

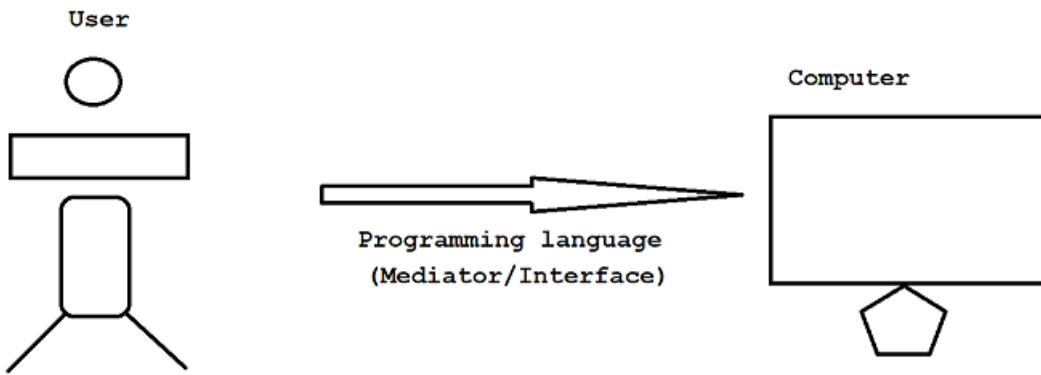


## Programming language:

A language which is used to communicate between user and computer is called programming language.

Programming language acts like a mediator or interface between user and computer.

Diagram: java2.1



### Programming language divided into two types:

1) Low Level Language.

2) High Level Language.

#### 1) Low Level Language:

A language which is understand by a computer easily is called low level language.

A language which is computer dependent is called low level language.

ex:

Machine Language

Assembly Language

#### Machine Language :

It is a fundamental language of a computer which is combination of 0's and 1's.

It is also known as binary language.

Our computer understands many languages but to understand machine language it does not required any translator.

#### Advantages:

- A program written in machine language consumes less memory.
- It does not require any translator.

- It is more efficient when compare to other languages.

### **Disadvantages:**

- It is a burdun on a programmer to remember dozen's of binary code.
- If anywhere error raised in our program then locating and handling that error becomes difficult.
- Modifications can't be done easily.

### **Assembly Language :**

- The second generation language came into an existence is called assembly language.
- Assembly language is a replacement of symbols and letters for mathematical programming code i.e opcode values.
- It is also known as symbolic language.
- Assembly language can't understand by a computer directly. We required translator.

### **We have three translators :**

1) Assembler

2) Compiler

3) Interpreter

#### **1) Assembler :**



It is one of the translator which converts assemblic code to machine code.

#### **Merits:**

- If anywhere error raised in our program then locating and handling that error become easy.
- Modifications can be done easily.

#### **Demerits:**

- It is a mind trick to remember all symbolic code.
- It requires translator.
- It is less efficient when compare to machine language.

### **Q)What is Debugging?**

Bug is also known as Error.

The process of eliminating the bugs from the application is called debugging.

#### **2)High Level Language :**

A language which is understand by a user easily is called high level language.

A language which is user dependent is called high level language.

**ex:**

C++,Java,.Net,Python,Perl and etc.

High level language can't understand by a computer directly. We required translators.

### Compiler :

It is used to compile and execute our program at a time.

### Interpreter :

It will execute our program line by line procedure.

### Advantages :

- It is easy to learn and easy to use because it is similar to English language.
- Debugging can be done easily.
- Modifications can be done easily when compare to low level language.



### Disadvantages :

- A program written in high level language consumes huge amount of memory.
- It requires translator.
- It is not efficient when compare to low level language.

### Project :

- Project Title : Employee Management System
- Stack of Technologies : React + Spring Boot + MySQL DB
- A project is a collection of modules.

We have different types of modules in project.

ex:

- login module
- customer module
- payment module
- report generation module
- registration module
- and etc.

**Every project contains two domains:**

**1) Technical Domain :**

- It describes using which technology we developed our project.

ex:

Java

**2) Functional Domain :**

It describes state of a project.

ex:

Healthcare domain,Banking domain ,Insuarance domain,ERP domain and etc.

**Q)Differences between C++ and Java?**

<b>C++</b>	<b>Java</b>
It is developed by Bjarne Stroustrup.	It is developed by James Gosling.
It is partial object oriented programming language.	It is a purely object oriented programming language.
It is a platform dependent.	It is a platform independent.
It supports multiple inheritance.	It does not support multiple inheritance.
It supports pointers.	It does not support pointers.
It supports goto statement.	It does not support goto statement.
It supports operator overloading.	It does not support operator overloading.
Memory allocation and deallocation will taken care by a programmer.	Memory allocation and deallocation will taken care by a JVM.
It supports Access Specifiers i.e:public, private and protected.	It supports Access Modifiers i.e:default,public,private and protected.
It supports three types of loops. i.e :do while , while and for loop.	It supports four types of loops. i.e:do while, while , for and for each loop.
It supports preprocessor directory(#).	It does not support preprocessor directory.
We can save C++ program by using .cpp extension.	We can save java program by using .java extension.

## Q) Differences between Python and Java?

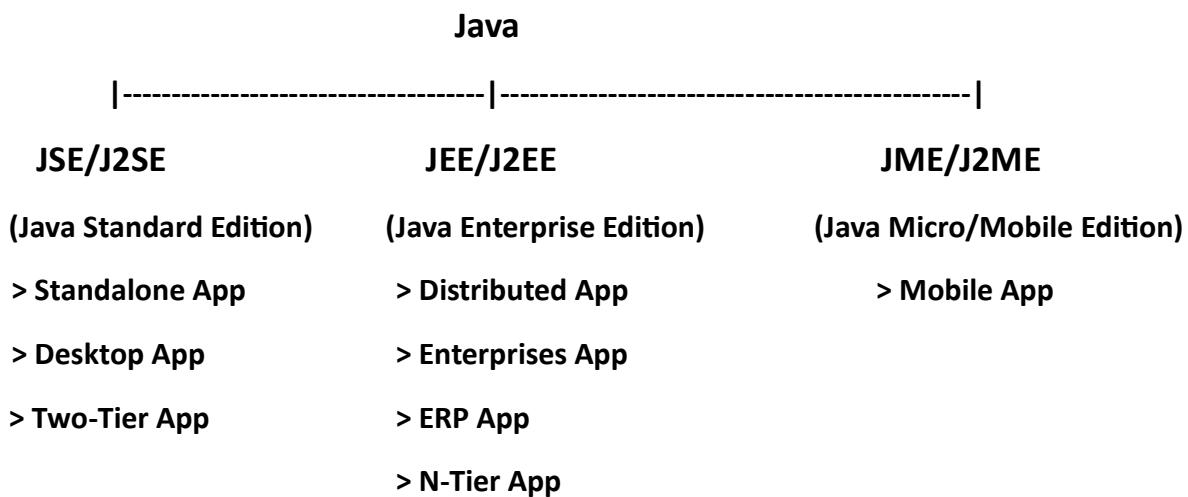
<u>PYTHON</u>	<u>JAVA</u>
It is a product of Microsoft.	It is a product of Oracle Corporation.
It is developed by Guido Van Rossum.	It is developed by James Gosling.
It is a scripting language.	It is a object oriented programming language.
It is a dynamically typed language.	It is a statically typed language.
It is a interpreted language.	It is a compiled language.
It contains less code then java.	It contains more code then python.
It is less secured.	It is highly secured.
Performance is not good.	Performance is better.

## Q) Differences between .Net and Java?

<u>.Net</u>	<u>Java</u>
It is a product of microsoft.	It is a product of oracle corporation.
It is a platform dependent.	It is a platform independent.
There is no security.	It is highly secured.
To develop medium scale projects we need to use .net.	To develop major scale projects we need to use java
Less collection of frameworks.  ex: Asp.net asp .net mvc	It contains large collection of frameworks.  ex: spring boot  microservices  spring security  Hibernate  Structs and etc.

## Modules in Java :

There are three modules in java.



### ➤ Standalone application :

A normal java program which contains main method is called standlone application.

ex:

```
class Test
{
    public static void main(String[] args)
    {
        -
        -
        - //code to be execute
        -
    }
}
```



### ➤ Desktop application :

It is a software application specially designed for desktops and laptops to perform perticular task.

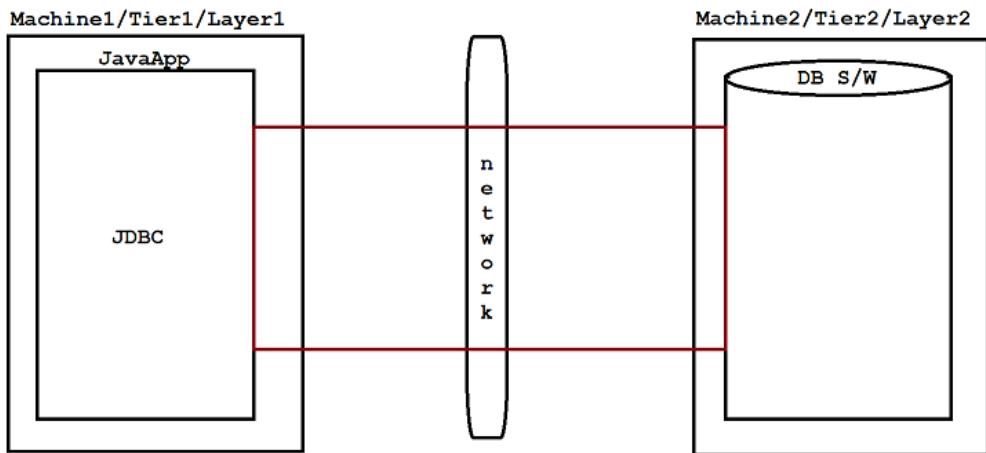
ex:

- ✓ control panel
- ✓ Recycle Bin
- ✓ VLC Media Player

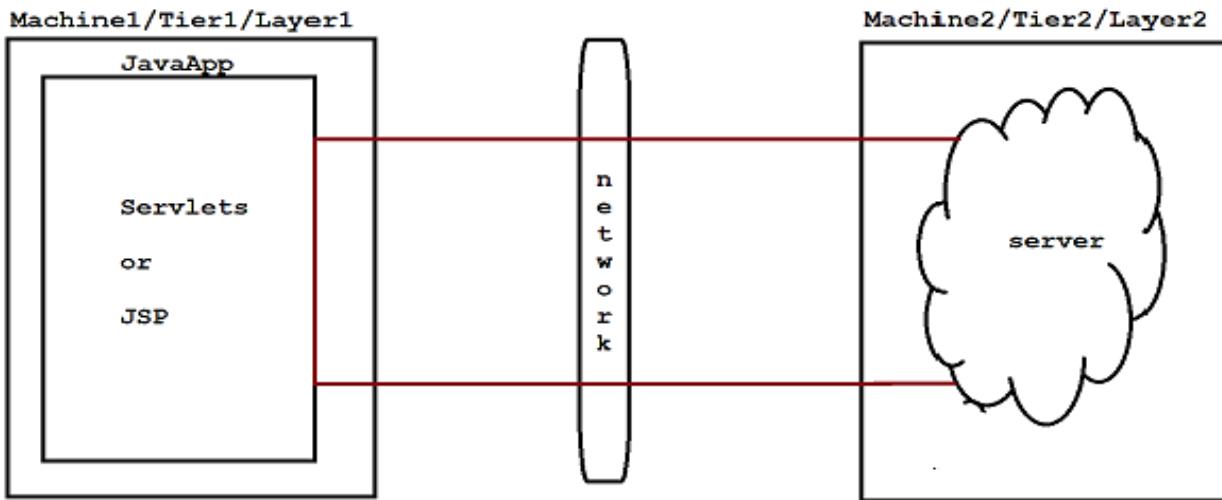
### ➤ Two-Tier application :

Having more then one tier is called two tier application.

### Diagram: java4.1

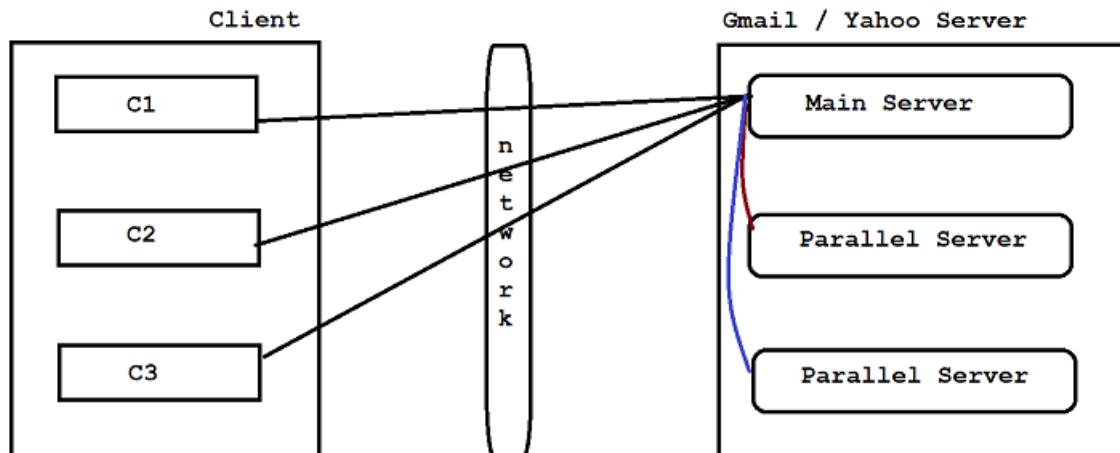


### Diagram: java4.2



### ➤ Distributed application:

### Diagram : java4.3



In client/server architecture , if multiple request goes to main server then main server will distributes request to it's parallel server to reduce the burdun of main server such type of application is called distributed application.

➤ **Enterprises application :**

An application which deals with large business complex logic by taking the support of middleware services is called enterprises application.

Here middleware services means authentication, autherization,malware production, firewall, security , transaction and etc.

ex:

- ✓ Facebook
- ✓ Online shopping website

➤ **ERP application :**

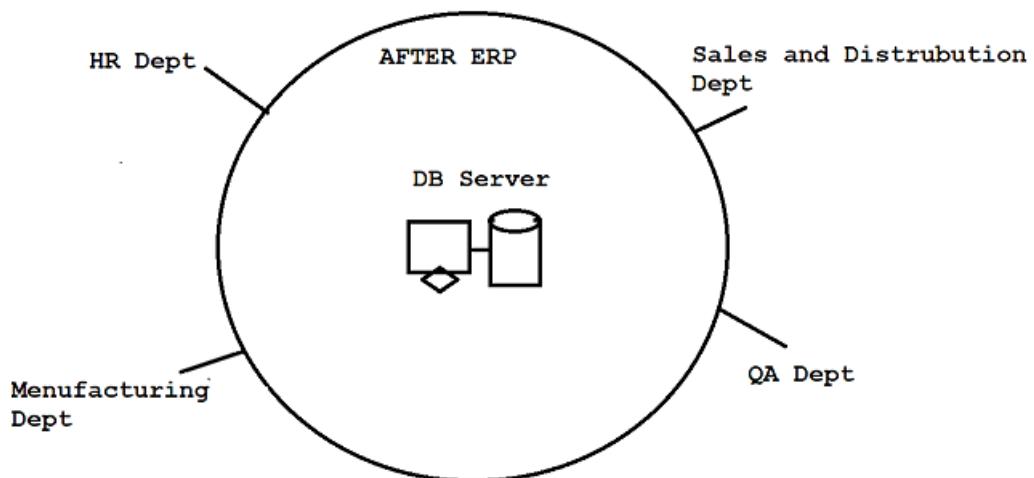
ERP stands for Enterprise Resource Planning.

ERP is used to maintain the data in the enterprise.

Diagram: java4.4



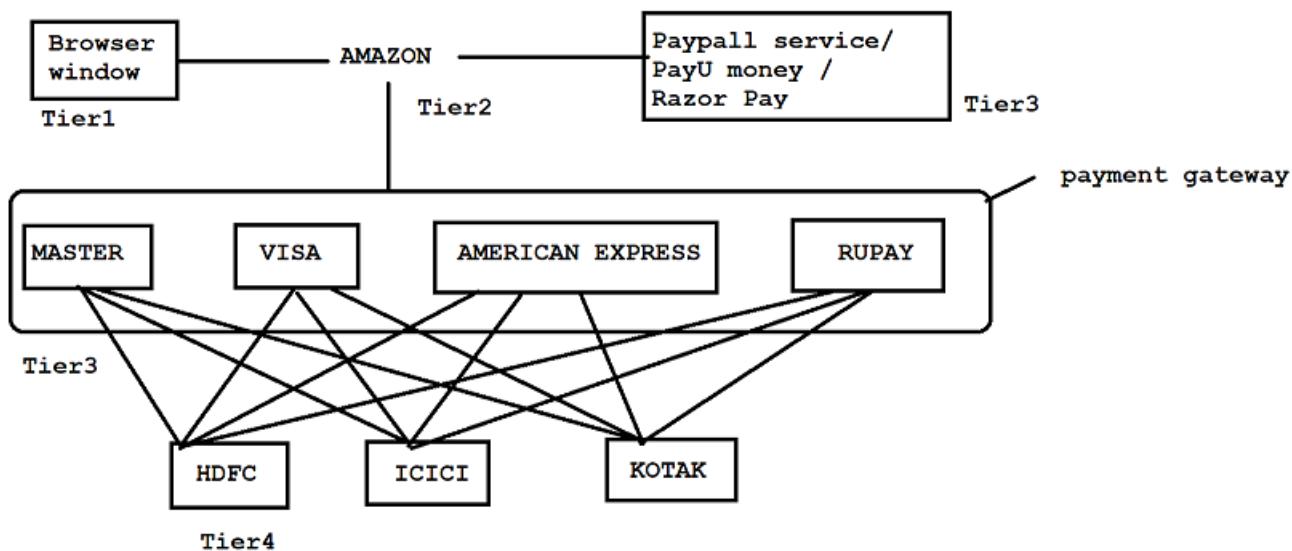
Amazon



## ➤ N-Tier application:

Having more than two tiers is called N-tier application.

### Diagram: java4.5

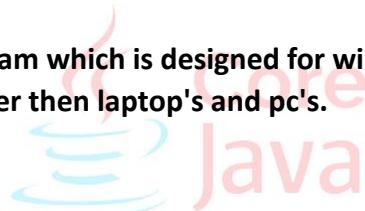


## ➤ Mobile application :

It is a software application or a program which is designed for wireless network devices like mobile, phone, cell, tab, cellular and etc rather than laptop's and pc's.

ex:

- ✓ Gpay
- ✓ PhonePay
- ✓ TempleRun
- ✓ and etc.



## Escape Characters or Escape Sequences :

Escape characters are used to design our output in neat and clean manner.

Escape characters always starts with back slash (\) followed by a single character.

ex:

\n

Mostly escape characters are used in a output statement in java.

ex:

```
System.out.println("\n");
```

We have following list of escape characters in java.

- \n (new line)
- \b (back space)
- \t (horizontal tab)
- \r (carriage return)
- \f (form feeding)
- \\" (back slash)
- \\" (double quote)
- \' (single quote)

and etc.

### **1) \n (new line) :**

```
class Test
{
    public static void main(String[] args)
    {
        System.out.println("IHUB\nTALENT");
    }
}
```

#### **Out put:**

IHUB

TALENT

### **2) \b (back space):**

```
class Test1
{
    public static void main(String[] args)
    {
        System.out.println("IHUBTA\bLENT");
    }
}
```

**Out put:**

IHUBTLENT

**ex:**

```
class Test2
{
    public static void main(String[] args)
    {
        System.out.println("IHUB\b\b\bTALENT");
    }
}
```

**Out put:**

ITALENT

**3) \t (horizontal tab) :**

```
class Test3
{
    public static void main(String[] args)
    {
        System.out.println("IHUB\tTALENT");
    }
}
```



**Out put:**

IHUB TALENT

**4) \r (carriage return) :**

```
class Test4
{
    public static void main(String[] args)
    {
```

```
        System.out.println("IHUB\rTALENT");

    }

}


```

**Out put:**

TALENT

**ex:**

```
class Teat5

{

    public static void main(String[] args)

    {

        System.out.println("TALENT\rIHUB");

    }

}
```

**Out put:**

IHUBNT

**6) \\ (back slash) :**



```
class Test6

{

    public static void main(String[] args)

    {

        System.out.println("IHUB\\TALENT");

    }

}
```

**Out put:**

IHUB\TALENT

**7) \" (double quote) :**

```
class Test7

{

    public static void main(String[] args)
```

```
{  
    System.out.println("IHUB\"TALENT");  
}  
}
```

**Out put:**

IHUB"TALENT

### **8) \' (single quote) :**

```
class Test8  
{  
    public static void main(String[] args)  
    {  
        System.out.println("IHUB'TALENT");  
        System.out.println("IHUB\'TALENT");  
    }  
}
```

**Out put:**

IHUB'TALENT

IHUB'TALENT



### **Interview Question**

**Q) Write a c program to display %d ?**

```
void main()  
{  
    clrscr();  
  
    printf("%d"); // 0  
    getch(); }  
  
void main()  
{  
    clrscr();
```

```
printf("%%d"); // %d  
  
getch();  
}
```

**Q)What will be the output of below code?**

ex:

```
class Demo  
{  
    public static void main(String[] args)  
    {  
        System.out.print("\n zx");  
        System.out.print("\n bp i");  
        System.out.print("\n rha");  
    }  
}
```

Out put:              hai



**Project :**

Technically, project is a collection of modules.

ex:

- registration module
- addtocart module
- payment module
- report generation module
- admin module
- login module and etc.

Every project contains two domains.

**1)Technical Domain :**

Using which technology we developed our project.

ex: Java

## **2)Functional Domain:**

It describes state of a project.

ex:

Healthcare domain

Banking domain

Insurance domain

ERP domain

and etc.



In java, uppercase letters will consider as different and lower case letters will consider as different that's why we consider java is a case sensitive programming language.

As java is a case sensitive , we must and should follow naming conventions for following things.

- classes
- interfaces
- variables
- methods
- keywords
- packages and
- constants

## **1) classes :**

In java,A class must and should starts with upper case letter and if it contains multiple words then each inner word must starts with initcap.

**ex:**

Predefined classes	Userdefined classes
System	IhubTalent
StringBuffer	QualityThought
FileWriter	Test
PrintWriter And Etc.	TestApp and etc

## **2) interfaces :**

In java, an interface must and should starts with upper case letter and if it contains multiple words then each inner word must starts with initcap.



**ex:**

Predefined interfaces	Userdefined interfaces
Runnable	ITest
Serializable	IExampleApp
ListIterator	IQualityThought
PreparedStatement	and etc.
Comparator and etc.	

## **3) variables :**

In java, a variable must and should starts with lowercase letter and if it contains multiple words then each inner word must starts with initcap.

**ex:**

<u>Predefined variables</u>	<u>Userdefined variables</u>
length	i
out	emplId
in	studName
err and etc.	deptNo and etc.

#### **4) methods:**

In java, a method name must and should starts with lower case letter and if it contains multiple words then each inner word must starts with initcap.

**ex:**

<u>Predefined methods</u>	<u>Userdefined methods</u>
HASHCODE()	getEmployeeDetails()
TOSTRING()	calculateBillAmount()
GETCLASS()	getDetails()
GETPRIORITY()	showInfo() and etc.
SETNAME() AND ETC.	

#### **5) keywords :**

In java, all keywords we need to declare under lower case letters.

**ex:**

<u>Predefined keywords</u>	
if	private
else	protected
do	and etc.
while	
for	
public	

## **6)packages :**

In java , a package name must and should declare under lower case letters.

**ex:**

<b><u>Predefined packages</u></b>	<b><u>Userdefined packages</u></b>
java.lang	ihub
java.io	qualitythought
java.util	com.ihubtalent.www
java.util.stream	com.google.www
java.text	and etc.
java.sql	
javax.servlet	
and etc.	



## **7) constants :**

In java, all constants we need to declare under uppercase letters.

**ex:**

<b><u>Predefined constants</u></b>	<b><u>Userdefined constants</u></b>
MAX_PRIORITY	LIMIT
MIN_PRIORITY	N
NORM_PRIORITY	and etc.
MAX_VALUE	
MIN_VALUE and etc.	

## **Interview Questions :**

**Q) How many classes are there in java?**

java 7 - 4024 classes

java 8	-	4240 classes
java 9	-	6005 classes
java 10	-	6002 classes

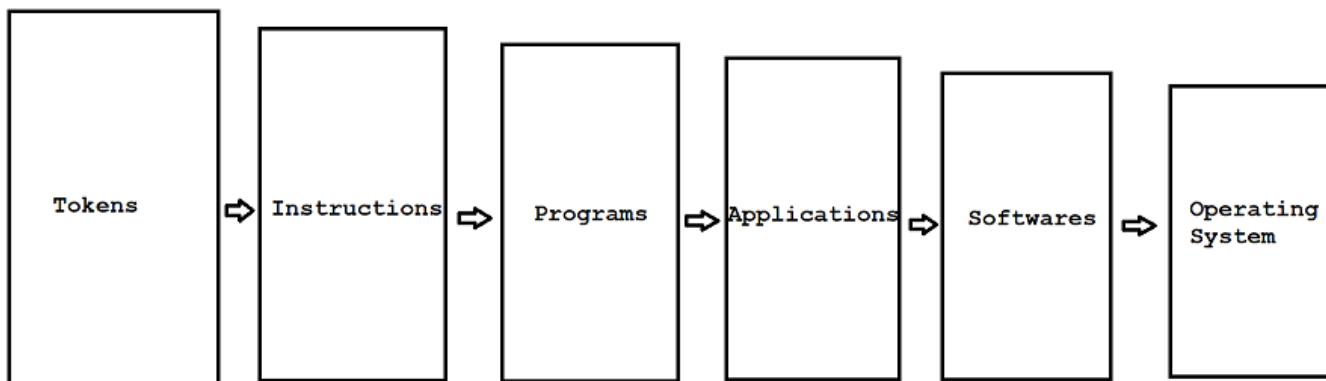
## Q) Which package is a default package in java?

java.lang package

## Q) What is Token?

It is a small unit of a program which consist identifiers, keywords, constants, variables, datatypes , operators and etc.

Diagram : java5.1



## History of Java :

In **1990**, Sun Micro System took one project to develop a software called consumer electronic device which can be controlled by a remote like setup box.That time project was called Stealth project and later they have renamed to Green project.

**James Gosling** , **Mike Sheradin** and **Patrick Naughton** were there to develop this project.They met in a place called Aspan/Colarado to start with work with graphic system.James gosling decided to use C and C++ languages to develop the project.But problem what they have faced is , they are system dependent.Then James Gosling decided why don't we create our own programming language which is system independent.

In **1991**, They have developed one programming language called an OAK.OAK means strength, itself is a coffee seed name and it is a national tree for many countries like **Germany,France,USA** and etc.Later in **1995**, they have renamed **OAK** to Java.

Java is a island of an Indonaisia where first coffee of seed was produced and during the development of project they were consuming lot of coffee's that's why symbol of java is a cup of coffee with saucer.

## Interview Questions

**Q) Java originally known as \_\_\_\_ ?**

OAK

**Q) In which year java was developed?**

In 1995.

**Q) What are the features of Java?**

- 1) Simple**
- 2) Object oriented**
- 3) High level**
- 4) Platform independent**
- 5) Architecture Neutral**
- 6) Portable**
- 7) Highly secured**
- 8) Multithreaded**
- 9) Distributed**
- 10) Dynamic**



**Q) What is the difference between JDK , JRE and JVM ?**

**JDK :**

JDK stands for Java Development Kit.

JDK is a installable software which consist of Java Runtime Environment (JRE), Compiler (javac) , Interpreter (java) , archiever (.jar) , document generator (javadoc) , Java Virtual Machine (JVM) and other tools need for java application development.

#### JRE :

JRE stands for Java Runtime Environment.

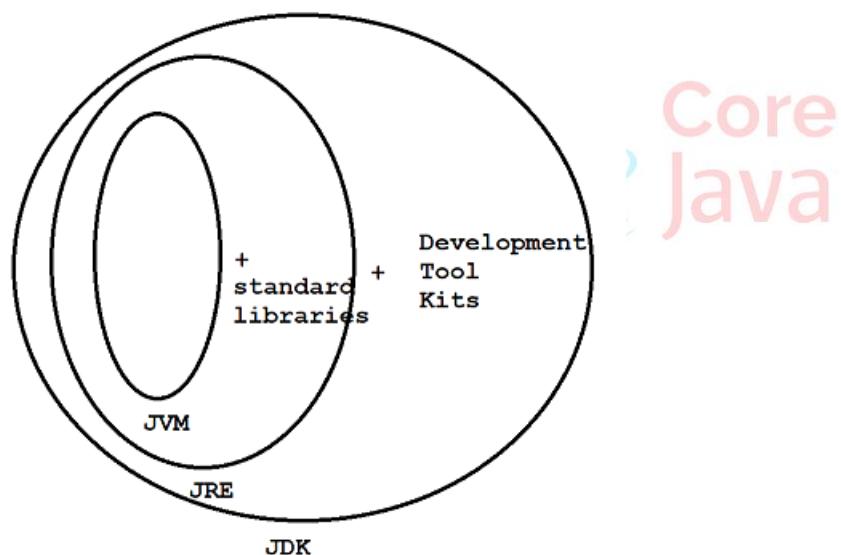
JRE provides very good environment to run java applications only.

#### JVM:

JVM stands for Java Virtual Machine.

JVM is an interpreter which is used to execute our program line by line procedure.

Diagram: java6.1



## HOW TO INSTALL JAVA

JDK	:	1.8v
Version	:	Java 8
Vendor	:	Oracle Corporation (Sun Micro System)
Open source	:	Open source

### Download:

[https://drive.google.com/file/d/16fr2McV\\_Bex0NYIOdcVfC4k2gwUUNqzq/view?usp=share\\_link](https://drive.google.com/file/d/16fr2McV_Bex0NYIOdcVfC4k2gwUUNqzq/view?usp=share_link)

### Environmental setup for Java :

#### step1:

Make sure JDK 1.8 installed successfully.

#### Step2:

Copy "lib" directory from java\_home folder.

ex:

C:\Program Files\Java\jdk1.8.0\_181\lib

#### Step3:

Paste java "lib" in environmental variables.

ex:

Right click to my computer --> properties -->

Advanced system settings --> environmental variables -->

User variables --> click to new button -->

variable Name : CLASSPATH

variable Value : C:\Program Files\Java\jdk1.8.0\_181\lib; --> ok.

System variables --> click to new button -->

variable Name : path

variable value : C:\Program Files\Java\jdk1.8.0\_181\bin;

--> ok --> ok --> ok.

#### Step4:

Open the command prompt and type below commands.

ex:

cmd>javap

cmd>java -version

## Steps to develop first Java application :

### Step1:

Make sure JDK software installed successfully.

### Step2:

Make sure environmental setup done perfectly.

### Step3:

Create a "javaprog" folder inside 'E' drive.

### Step4:

Open the notepad and develop simple Hello World program.

#### ex:

```
class Test
{
    public static void main(String[] args)
    {
        System.out.println("Hello World");
    }
}
```



### Step5:

Save above program inside javaprog folder with same name as class name.

### Step6:

Open the command prompt from "javaprog" location.

### Step7:

Compile above program with below command.

#### ex:

```
javaprog> javac Test.java
```

|

file name

### Step8:

Execute above program with below command.

ex:

javaprog> java Test

|

class name

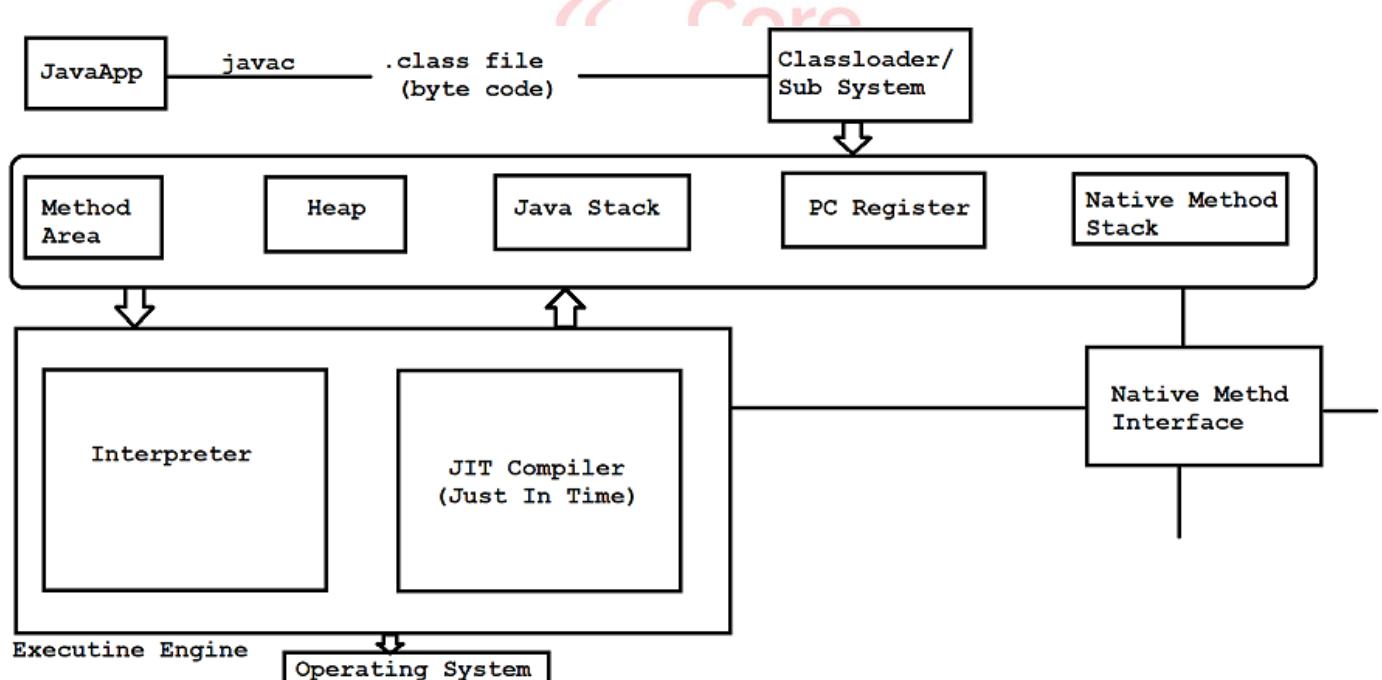
### Interview Question

Q) A .class file contains \_\_\_ ?

byte code instructions

### Internal Architecture of JVM

#### Diagram: java8.1



Java application contains java code instructions. Once if we compile java code instructions converts to byte code instructions in .class file.

JVM will invoke one module called classloader/Subsystem to load all the byte code instructions from .class file. The work of classloader is to check these byte code instructions are proper or not. If they are not proper , it will refuse the execution. If they are proper then it will allocate the memory.

We have five types of memories in java.

## **1) Method Area:**

It contains code of a class , code of a method or code of a variable.

## **2) Heap:**

Our object creations will store in heap area.

### **Note:**

Whenever JVM loads byte code instructions from .class file it will create method area and heap area automatically.

## **3) Java Stack :**

To execute java methods we required some memory that memory will be allocated in java stack.

## **4) PC Register :**

It is a program counter register which is used to track the address of an instruction.

## **5) Native Method Stack:**

Java methods will execute in method area.

Similarly Native methods will execute in native method stack.

But to execute native methods we required a program called native method interface.

## **Execution Engine:**

Execution engine contains interpreter and jit compiler.

Whenever JVM loads the byte code instructions from .class file , it uses interpreter and jit compiler simultaneously.

Interpreter is used to execute our program line by line procedure.

JIT compiler is used to increase the execution speed of our program.

## **Interview Questions**

### **Q) What is Native Method in java?**

Method which is developed by using some other language is called native method.

### **Q) How many memories are there in java?**

We have five memories in java.

#### **1) Method Area**

- 2) Heap
- 3) Java Stack
- 4) PC Register
- 5) Native Method Stack

### **Q)What is JIT compiler?**

It is a part of a JVM which is used to increase the execution speed of our program.

### **Q)How many classloader are there in java?**

We have three predefined classloaders in java.

- 1) Bootstrap classloader (It loads rt.jar file)
- 2) Extension classloader (It loads all the jar files from ext folder)
- 3) Application/System classloader (It loads .class file from classpath)

### **Q)What is garbage Collector?**

Garbage Collector is used to destroy unused or useless objects in java.

### **Q)In how many ways we can call garbage collector in java?**

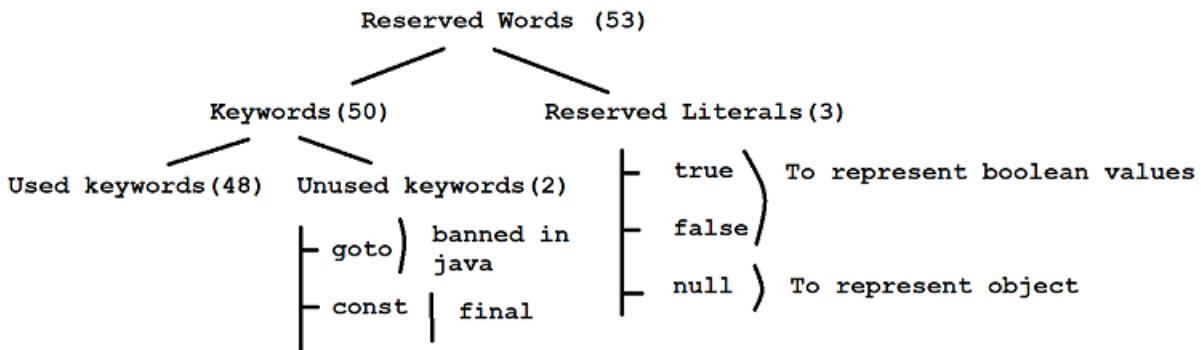
There are two ways to call the garbage collector in java.

- 1) System.gc();
- 2) Runnable.getRuntime().gc();

### **Reserved words**

- There are some identifiers which are reserved to associate some functionality or meaning such type of identifiers are called reserved words.
- Java supports 53 reserved words.
- All reserved words we need to declare under lower case letters.
- Reserved words are divided into two types.

### **Diagram: java8.2**



### Used keywords with respect to class :

package

import

class

enum

interface

extends

implements .



### Used keywords with respect to object :

this

super

new

typeof

### Used keywords with respect to datatype :

byte

short

int

long

float

double

**boolean**

**char**

### **Used keywords with respect to modifiers :**

**default**

**public**

**private**

**protected**

**static**

**final**

**abstract**

**strictfp**

**synchronized**

**transient**

**volatile**



### **Used keywords with respect to flow control :**

**if**

**else**

**do**

**while**

**break**

**continue**

**for**

**switch**

**case**

### **Used keywords with respect to returntype :**

**void**

### **Used keywords with respect to exception handling :**

try  
catch  
throw  
throws  
finally  
assert

### **Identifiers :**

- A name in java is called identifier.
- It can be class name , method name , variable name or label name.

**ex:**

```
class Test
{
    public static void main(String[] args)
    {
        int x=10;
        System.out.println(x);
    }
}
```



**Here :** Test, main, args and x are identifiers.

### **Rules to declare an identifier:**

#### **Rule1:**

Identifier will accept following characters.

**ex:**

A-Z

a-z

**0-9**

**-**

**\$**

### **Rule2:**

If we take other characters then we will get compile time error.

**ex:**

```
emp_no //valid  
stud#no //invalid  
emp$al //valid
```

### **Rule3:**

Identifier must and should starts with alphabet,underscore or dollar symbol but not with digits.

**ex:**

```
a1234 //valid  
_ab123 //valid  
$abcd //valid  
1abcd //invalid
```



### **Rule4:**

Every identifier is a case sensitive.

**ex:**

```
number;  
NUMBER;  
NumBer;
```

### **Rule5:**

We can't take reserved words as an identifier name.

ex:

```
class if //invalid  
void else() //invalid  
int switch; //invalid
```

### Rule6:

There is no length limit for an identifier but it is not recommended to take more than 15 characters.

### Rule7:

We can take predefined classes and interfaces as an identifier name but it is not good programming practice.

ex:

<u>Predefined classes</u>	<u>Predefined interfaces</u>
System	Runnable
String	Serializable
StringBuffer	Remote
File	Iterator
FileWriter and etc	Enumeration and etc.

ex:

```
int System; //valid  
int Runnable; //valid
```

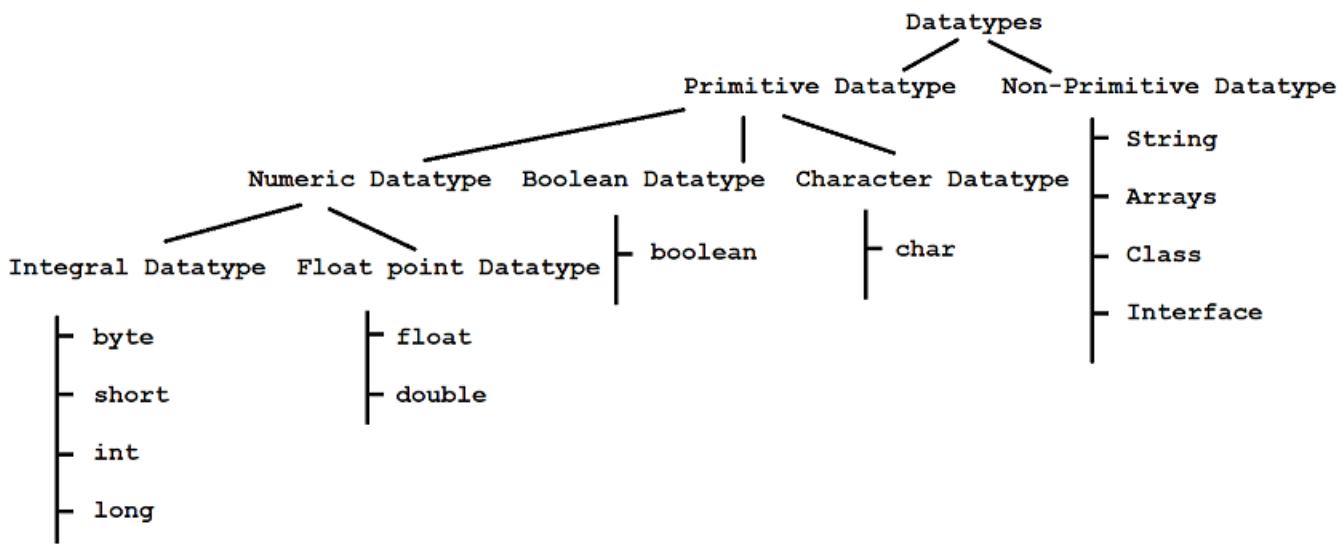
## Datatypes :

Datatype describes what type of value we want to store inside a variable.

Datatype also tells how much memory has to be created for a variable.

In java, datatypes are divided into two types.

### Diagram: java9.1



### Byte :

It is a smallest datatype in java.

**size:** 1 byte (8 bits)

**range:** -128 to 127 (- $2^7$  to  $2^7-1$ )

**ex:**

1) byte b=10;

```
System.out.println(b); //10
```

2) byte b=130;

```
System.out.println(b); //C.T.E
```

3) byte b=10.5;

```
System.out.println(b); //C.T.E
```



### Short :

It is a rarely used datatype in java.

This datatype is developed for 16-bit processor. But 16 bit processors are outdated. Hence short datatype is also outdated.

**size:** 2 bytes (16 bits)

**range: -32768 to 32767 (-2<sup>15</sup> to 2<sup>15</sup>-1)**

ex:

1) byte b=20;

short s=b;

System.out.println(s); // 20

2) short s=10.56;

System.out.println(s); // C.T.E

3) short s="hi";

System.out.println(s); // C.T.E

Int :

**It is mostly used datatype in java.**

**size: 4 bytes (32 bits)**

**range: -2147483648 to 2147483647 (-2<sup>31</sup> to 2<sup>31</sup>-1)**

ex:

1) int i=10.5;

System.out.println(i); //C.T.E

2) int i="bye";

System.out.println(i); // C.T.E

3) int i= true;

System.out.println(i); // C.T.E

4) int i = 'a';

System.out.println(i); // 97

**In java , For every character we have Universal Unicode value.**

ex:

a ---> 97

A ---> 65

## Long:

If int datatype is not enough to hold large value then we need to use long datatype.

size : 8 bytes (64 bits)

Range : (-2^63 to 2^63-1)

## ex:

1) long l="true";

```
System.out.println(l); //C.T.E
```

2) long l=10.56;

```
System.out.println(l); // C.T.E
```

3) long l="A";

```
System.out.println(l); //C.T.E
```

4) long l='d';

```
System.out.println(l); // 100
```



## **Q) What is the difference between float and double datatype?**

float	double
If we need 4 to 6 decimal point of accuracy then we need to use float.	If we need 14 to 16 decimal point of accuracy then we need to use double.
size : 4 bytes (32 bits)	size: 8 bytes (64 bits)
range: -3.4e38 to 3.4e38	range: -1.7e308 to 1.7e308
To represent float value we need to suffix with 'f'.	To represent double value we need to suffix with 'd'.
ex: 10.5f;	ex:10.5d

## ex:

1) float f=10;

```
System.out.println(f); //10.0
```

2) float i=10.5f;

```
System.out.println(i); //10.5
```

```
3) float f='a';  
  
System.out.println(f); //97.0
```

### Boolean :

It is used to represent boolean values either true or false.

Size : (Not Applicable)

Range: (Not Applicable)

ex:

```
1) boolean b="true";  
  
System.out.println(b); //C.T.E
```

```
2) boolean b=TRUE;  
  
System.out.println(b); //C.T.E cannot find symbol
```

```
3) boolean b=true;  
  
System.out.println(b); //true
```

### Char :

It is a single character which is enclosed in a single quotation.

size : 2 bytes (16 bits)

range : 0 to 65535

ex:

```
1) char ch='a';  
  
System.out.println(ch); // a  
  
2) char ch=65;  
  
System.out.println(ch); // A  
  
3) char ch="a";  
  
System.out.println(ch); // C.T.E  
  
4) char ch='ab';  
  
System.out.println(ch); // C.T.E
```

Diagram: java10.1

Datatype	Size	Range	Wrapper class	Default value
byte	1 byte	-128 to 127	Byte	0
short	2 bytes	-32768 to 32767	Short	0
int	4 bytes	-2147483648 to 2147483647	Integer	0
long	8 bytes	-2^63 to 2^63-1	Long	0
float	4 bytes	-3.4e38 to 3.4e38	Float	0.0
double	8 bytes	-1.7e308 to 1.7e308	Double	0.0
boolean	-	-	Boolean	false
char	2 bytes	0 to 65536	Character	0 (space)

Q) Write a java program to find out range of byte datatype?

byte : -128 to 127

ex:

```
class Test
{
    public static void main(String[] args)
    {
        System.out.println(Byte.MIN_VALUE);
        System.out.println(Byte.MAX_VALUE);
    }
}
```



Q) Write a java program to find out range of int datatype?

int : -2147483648 to 2147483647

ex:

```
class Test
{
    public static void main(String[] args)
```

```
{  
    System.out.println(Integer.MIN_VALUE);  
    System.out.println(Integer.MAX_VALUE);  
}  
}
```

## Java Source File structure

### case1:

A java program can have multiple classes.

### case2:

If java program contains multiple classes then we need to check which class contains main method. That class will consider as main class. We need to save our program with main class name only.

### ex:

```
B.java  
class A  
{  
    - // some logic  
}  
class B  
{  
    public static void main(String[] args)  
    {  
        - //some logic  
    }  
}
```



### case 3:

If a java program contains multiple classes with main method then we can save program name with any name.

ex:

```
Ihub.java

class A
{
    public static void main(String[] args)
    {
        System.out.println("A-class");
    }
}

class B
{
    public static void main(String[] args)
    {
        System.out.println("B-class");
    }
}

class C
{
    public static void main(String[] args)
    {
        System.out.println("C-class");
    }
}
```



Note:

If we compile above program we will get three .class files i.e

A.class, B.class and C.class file.

javaprog> javac Ihub.java

javaprog> java A (A class will execute)

javaprog> java B (B class will execute)

javaprog> java C (C class will execute)

case4:

If a java program contains multiple classes with main method then declare one class as public. That public class consider as main class and we need to save our program name with public class name only.

ex:

A.java

```
public class A
{
    public static void main(String[] args)
    {
        System.out.println("A-class");
    }
}

class B
{
    public static void main(String[] args)
    {
        System.out.println("B-class");
    }
}

class C
{
    public static void main(String[] args)
    {
        System.out.println("C-class");
    }
}
```

Note:

If we compile above program we will get three .class files i.e

A.class, B.class and C.class file.

```
javaprog> javac A.java  
javaprog> java A (A class will execute)  
javaprog> java B (B class will execute)  
javaprog> java C (C class will execute)
```

### Interview Question

**Q)What is the difference between default class and public class?**

**default class**

We can access that class within the package.

**ex:**

```
class Test  
{  
    -  
    -  
    -  
}
```



**public class :**

We can access that class within the package and outside of the package.

**ex:**

```
public class Test  
{  
    -  
    -  
    -  
}
```

### Types of variables

A name which is given to a memory location is called variable.

Purpose of variable is used to store the data.

We have two types of variables in java.

### **1) Primitive variables :**

It is used to represent primitive values.

### **2) Reference variables :**

It is used to represent object reference.

**ex:**

```
Student s=new Student();
```

|

reference variable

Based on the position and execution these variables are divided into three types.

**1) Instance variable / Non-Static variable**

**2) Static variable / Global variable**

**3) Local variable / Temporary variable / Automatic variable**

#### **1) Instance variable :**

A value of a variable which is varied(changes) from object to object is called instance variable.

Instance variable will be created at the time of object creation and it will destroy at the time of object destruction.Hence scope of instance variable is same as scope of an object.

Instance variable will store in heap area as a part of an object.

Instance variable must and should declare immediately after the class but not inside methods ,blocks and constructors.

Instance variable can access directly from instance area but we can't access directly from static area.

To access instance variable from static area we need to create object reference.

**ex:1**

```
class Test
```

```
{
```

```
//instance variable
```

```
int i=10;

public static void main(String[] args)
{
    System.out.println(i);//C.T.E
}

}
```

ex:2

```
class Test
{
    //instance variable
    int i=10;
```

```
public static void main(String[] args)
{
    Test t=new Test();
    System.out.println(t.i);//10
}
```

ex:3

```
class Test
{
    public static void main(String[] args)
    {
        //calling
        Test t=new Test();
        t.m1();
    }
    //non-static method
    public void m1()
```



```
{  
    System.out.println("M1-Method");  
}  
}
```

**Note:**

If we won't initialize any value to instance variable then JVM will initialize default values.

**ex:4**

```
class Test  
{  
    //instance variable  
    boolean b;  
  
    public static void main(String[] args)  
    {  
        Test t=new Test();  
        System.out.println(t.b); //false }}}
```



**2)Static variable:**

A value of a variable which is not varied from object to object is called static variable.

Static variable will be created at the time of classloading and it will destroy at the time of class unloading. Hence scope of static variable is same as scope of a .class file.

Static variable will store in method area.

Static variable must and should declare immediately after the class by using static keyword but not inside methods ,blocks and constructors.

Static variable can access directly from instance area as well as from static area.

Static variable we can access using object reference and classname.

**ex:1**

```
class Test  
{
```

```
//static variable  
  
static int i=10;  
  
  
public static void main(String[] args)  
{  
    System.out.println(i); // 10  
  
    Test t=new Test();  
    System.out.println(t.i);//10  
  
    System.out.println(Test.i);//10  
}  
}
```

ex:2



```
class Test  
{  
    //static method  
    public static void m1()  
    {  
        System.out.println("M1-Method");  
    }  
  
    public static void main(String[] args)  
    {  
        //calling  
        m1();  
  
        Test t=new Test();
```

```
t.m1();
```

```
Test.m1();
```

```
}
```

```
}
```

### Note:

If we won't initialize any value to static variable then JVM will initialized default value.

#### ex:3

```
class Test
```

```
{
```

```
//static variable
```

```
static String s;
```

```
public static void main(String[] args)
```

```
{
```

```
System.out.println(s); // null }}
```



### 3) Local variable :

To meet temporary requirements a programmer will declare some variables inside methods , blocks and constructors such type of variables are called local variables.

Local variable will be created at the time of execution block and it will destroy when execution block is executed.Hence scope of local variable is same as scope of execution block where it is declared.

Local variable will store java stack memory.

#### ex:1

```
class Test
```

```
{
```

```
public static void main(String[] args)
```

```
{
```

```
//local variable
```

```
int i=10;
```

```
System.out.println(i); //10
```

```
    }  
}
```

**Note:**

If we won't initialize any value to local variable then JVM will not initialize any default value.

**ex:2**

```
class Test
```

```
{
```

```
    public static void main(String[] args)  
    {  
        //local variable  
        int i;  
        System.out.println(i); //C.T.E  
    }
```

```
}
```



Error : variable i might not have been initialized

**Main method :**

Our program contains main method or not.

Either it is properly declare or not. It is not a responsibility of a compiler to check. It is a liability of a JVM to look for main method always at runtime.

If JVM won't found main method then it will throw one runtime error called main method not found.

JVM always look for main method with following signature.

**syntax:**

```
public static void main(String[] args)
```

If we perform any changes in above signature then JVM will throw one runtime error called main method not found.

**Public :**

JVM wants to call this method from anywhere.

**Static :**

JVM wants to call this method without using object reference.

### Void :

Main method does not return anything to JVM that's why return type is void.

### Main :

It is a identifier given to a main method.

### String[] args :

It is a command line argument.

We can do following changes in main method.

1) Order of modifiers is not important incase of public static we can declare static public also.

ex:

```
static public void main(String[] args)
```

2) We can change String[] in following acceptable formats.

ex:

```
public static void main(String[] args)
```

```
public static void main(String []args)
```

```
public static void main(String args[])
```

3) We can change String[] with var-arg parameter.

ex:

```
public static void main(String... args)
```

4) We can change "args" with any java valid identifier.

ex:

```
public static void main(String[] ihub)
```

5) Main method will accept following modifiers.

ex:

synchronized, strictfp and final .

## Command Line argument

Arguments which are passing through command prompt such type of arguments are called command line arguments.

In command line arguments we need to pass our input values at runtime command.

ex: javaprog> javac Test.java

javaprog> java Test 101 sandeep m 1000.0

```
|   |   |   |____ args[3]  
|   |   |_____ args[2]  
|   |_____ args[1]  
|_____ args[0]
```

ex:

```
class Test  
{  
    public static void main(String[] args)  
    {  
        System.out.println(args[0]);  
        System.out.println(args[1]);  
        System.out.println(args[2]);  
        System.out.println(args[3]);  
    }  
}
```

o/p:

```
javac Test.java  
java Test 101 raja m 1000.0
```

## System.out.println() :

It is a output statement in java.

If we want to display any data or userdefined statements then we need to use `System.out.println()` stmt.

syntax:

static variable

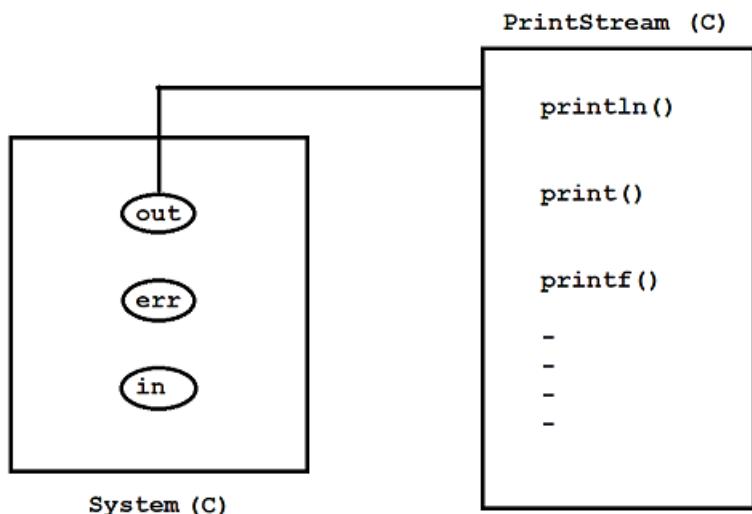
|

```

System.out.println();
|           |
predefined   predefined method
final
class

```

Diagram: java12.1



ex:

```

class Test
{
    public static void main(String[] args)
    {
        System.out.println("stmt1");
        System.out.print("stmt2");
        System.out.printf("stmt3");
    }
}

```

### various ways to display the data

1)

```
System.out.println("Welcome to Java World");
```

2)

```
int i=10;  
  
System.out.println(i);  
  
System.out.println("The value is =" + i);
```

3)

```
int i=10,j=20;  
  
System.out.println(i + " " + j);  
  
System.out.println(i + " and " + j);
```

4)

```
int i=10,j=20,k=30;  
  
System.out.println(i + " " + j + " " + k);
```

**Q) What is the difference between System.out.println() and System.err.println()?**

### System.out.println()

It is used to display the output on console.



ex:

```
class Test  
{  
    public static void main(String[] args)  
    {  
        System.out.println("Hello World");  
    }  
}
```

### System.err.println()

It is used to display the output on console as well as it will redirect to physical file.

ex:

```
class Test  
{  
    public static void main(String[] args)
```

```
{  
    System.out.println("Hello World");  
}  
}  
  
javac Test.java  
java Test >abc.txt
```

## Fully Qualified Name

Full Qualified Name means we can declare a class or interface along with package name.

It is used to improve readability of our code.

ex:

```
class Test
```

```
{
```

```
    public static void main(String[] args)  
    {  
        java.util.Date d=new java.util.Date();  
        System.out.println(d);  
    }  
}
```



## Import statements :

Whenever we use import statements we should not use fully qualified name.

Using short name also we can achieve.

In java, we have three import statements.

1) Explicit class import

2) Implicit class import

3) Static import

### 1) Explicit class import :

This type of import statement is recommended to use because it will improve the readability of our code.

Ex1:

```
import java.util.Date;  
  
class Test  
{  
  
    public static void main(String[] args)  
    {  
  
        Date d=new Date();  
  
        System.out.println(d);  
  
    }  
}
```

A `java.time` package is introduced in java 1.8 version.

It is used to display date and time.

Ex2:

```
import java.time.LocalDate;  
  
class Test  
{  
  
    public static void main(String[] args)  
    {  
  
        LocalDate date=LocalDate.now();  
  
        System.out.println(date);  
  
    }  
}
```

Ex3:

```
import java.time.LocalDate;  
  
class Test  
{  
  
    public static void main(String[] args)  
    {  
  
        LocalDate date=LocalDate.now();  
  
        int d=date.getDayOfMonth();  
    }  
}
```



```

        int m=date.getMonthValue();

        int y=date.getYear();

        System.out.println(d+"/"+m+"/"+y);

    }

}

```

Ex4:

```

import java.time.LocalTime;

class Test

{
    public static void main(String[] args)

    {

        LocalTime time=LocalTime.now();

        System.out.println(time);

    }
}

```

Ex5:

```

import java.time.LocalTime;

class Test

{
    public static void main(String[] args)

    {

        LocalTime time=LocalTime.now();

        int h=time.getHour();

        int m=time.getMinute();

        int s=time.getSecond();

        System.out.println(h+":"+m+":"+s);

    }
}

```



2)implicit class import :

This type of import statement is not recommended to use because it will reduce readability of our code.

**ex:1**

```
import java.util.*;  
  
class Test  
{  
  
    public static void main(String[] args)  
    {  
  
        Date d=new Date();  
  
        System.out.println(d);  
  
    }  
}
```

**ex:2**

```
import java.time.*;  
  
class Test  
{  
  
    public static void main(String[] args)  
    {  
  
        LocalDate date=LocalDate.now();  
  
        System.out.println(date);  
  
  
        LocalTime time=LocalTime.now();  
  
        System.out.println(time);  
  
    }  
}
```



**3)Static import :**

Using static import we can call static members(static variables and static methods) directly.

Often use of static import makes our program unreadable or complex.

### ex:1

```
import static java.lang.System.*;
class Test
{
    public static void main(String[] args)
    {
        out.println("Learn");
        out.println("Practice");
        out.println("Achieve");
    }
}
```

### ex:2

```
import static java.lang.System.*;
class Test
{
    public static void main(String[] args)
    {
        out.println("Learn");

        out.println("Practice");

        exit(0);

        out.println("Achieve");
    }
}
```



### var-arg method :

Until 1.4 version it is not possible to declare a method with variable number of arguments. But from 1.5 version onwards it is possible to declare a method with variable number of arguments.

We can declare var-arg method as follow.

### ex:1

#### var-arg parameter

```
methodOne(int... i)
```

|

ellipse

Here var-arg parameter is a replacements of single dimensional array.

ex:

```
... ----> []
```

We can invoke var-arg method with any number of integer values including zero.

### ex:2

#### var-arg parameter :



```
class Test
{
    public static void main(String[] args)
    {
        //calling
        methodOne();
        methodOne(10);
        methodOne(10,20);
        methodOne(10,20,30,40);

    }
    //static method

    public static void methodOne(int... i)
```

```
{  
    System.out.println("var-arg method");  
}  
  
}
```

## ex:2

### general parameter :

```
class Test  
{  
    public static void main(String[] args)  
    {  
        //calling  
        methodOne(10);  
    }  
    //static method  
    public static void methodOne(int i)  
    {  
        System.out.println("static method");  
    }  
}
```



### case1:

We can mix var-arg parameter with general parameters also.

ex:

```
methodOne(int i,int... j);
```

### case2:

If we mix var-arg parameter with general parameter then var-arg parameter must in last parameter.

ex:

```
methodOne(int i,int... j); // valid
```

```
methodOne(int... i,int j); // invalid
```

### case3:

A var-arg method can have only one var-arg parameter.

ex:

```
methodOne(int... i); //valid
```

```
methodOne(int... i,int... j); //Invalid
```



Download link: <https://www.editplus.com/download.html>

### Java Basic Programs :

**Q)Write a java program to perform sum of two numbers?**

```
import java.util.Scanner;
class Example1
{
    public static void main(String[] args)
    {
        Scanner sc=new Scanner(System.in);

        System.out.println("Enter the First Number :");
        int a=sc.nextInt();

        System.out.println("Enter the Second Number :");
```

```

int b=sc.nextInt();

//logic

int c=a+b;

System.out.println("sum of two numbers is =" +c);

}

}

```

**Q) Write a java program to perform sum of two numbers without using third variable?**

```

import java.util.Scanner;

class Example2

{
    public static void main(String[] args)
    {
        Scanner sc=new Scanner(System.in);

        System.out.println("Enter the First Number :");

        int a=sc.nextInt();

        System.out.println("Enter the Second Number :");

        int b=sc.nextInt();

        System.out.println("sum of two numbers is =" +(a+b));
    }
}

```

**Q) Write a java program to find out square of a given number ?**

```

import java.util.Scanner;

class Example3
{

```

```

public static void main(String[] args)
{
    Scanner sc=new Scanner(System.in);

    System.out.println("Enter the number :");
    int n=sc.nextInt();

    //logic
    int square=n*n;

    System.out.println("square of a given number is = "+square);

}
}

```

**Q)Write a java program to perform cube of a given number?**



```

import java.util.Scanner;
class Example4
{
    public static void main(String[] args)
    {
        Scanner sc=new Scanner(System.in);

        System.out.println("Enter the number :");
        int n=sc.nextInt();

        //logic
        int cube=n*n*n;

        System.out.println("Cube of a given number is =" +cube);
    }
}

```

```
    }  
}  
}
```

**Q)Write a java program to perform area of a circle ?**

```
import java.util.Scanner;  
  
class Example5  
{  
  
    public static void main(String[] args)  
    {  
  
        Scanner sc=new Scanner(System.in);  
  
        System.out.println("Enter the radius :");  
        int r=sc.nextInt();  
  
        //logic  
        float area=3.14f*r*r;  
  
        System.out.println("Area of a circle is =" +area);  
    }  
}
```

**Q)Write a java program to perform perimeter of a circle?**

```
import java.util.Scanner;  
  
class Example6  
{  
  
    public static void main(String[] args)  
    {  
  
        Scanner sc=new Scanner(System.in);  
  
        System.out.println("Enter the radius :");  
        int r=sc.nextInt();  
    }  
}
```

```

//logic

float perimeter= 2*3.14f*r;

System.out.println("Perimeter of a circle is =" +perimeter);

}

}

```

### Diagram: java15.1

programs	Logic	inputs	outputs
sum of two numbers	int c=a+b;	a,b	c
square of a given number	int square=n*n;	n	square
cube of a given number	int cube=n*n*n;	n	cube
area of a circle	float area=3.14f*r*r;	r	area
perimeter of a circle	float perimeter=2*3.14f*r;	r	perimeter

### Assignments

Q) Write a java program to find out area of a rectangle?

Q) Write a java program to find out area of a square?

Q) Write a java program to find area of a triangle?

Q) Write a java program to perform swapping of two numbers?

input:

a=10 , b=20

output:

a=20 , b=10

import java.util.Scanner;

class Example7

```

}

public static void main(String[] args)
{
    Scanner sc=new Scanner(System.in);

    System.out.println("Enter the First Number :");
    int a=sc.nextInt();

    System.out.println("Enter the Second Number :");
    int b=sc.nextInt();

    System.out.println("Before swapping A="+a+" and B="+b);

    //swapping logic
    int temp=a;
    a=b;
    b=temp;

    System.out.println("After swapping A="+a+" and B="+b);

}
}

```



**Q)Write a java program to perform swapping of two numbers without using third variable?**

```

import java.util.Scanner;
class Example8
{
    public static void main(String[] args)
    {
        Scanner sc=new Scanner(System.in);

```

```

System.out.println("Enter the First Number :");
int a=sc.nextInt();

System.out.println("Enter the Second Number :");
int b=sc.nextInt();

System.out.println("Before swapping A="+a+" and B="+b);

//swapping logic
a=a+b;
b=a-b;
a=a-b;

System.out.println("After swapping A="+a+" and B="+b);

}

}

```

**Q)Write a java program to accept one salary and find out 10% of TDS?**

```

import java.util.Scanner;

class Example9
{
    public static void main(String[] args)
    {
        Scanner sc=new Scanner(System.in);

        System.out.println("Enter the Salary :");
        int salary=sc.nextInt();

        float tds=(float)salary*10/100;
    }
}

```

```
        System.out.println("10 percent of TDS is =" +tds);  
    }  
}
```

**Q)Write a java program to convert CGPA to percentage?**

```
import java.util.Scanner;  
  
class Example10  
{  
    public static void main(String[] args)  
    {  
        Scanner sc=new Scanner(System.in);  
  
        System.out.println("Enter the CGPA :");  
        float cgpa=sc.nextFloat();  
  
        float percentage=cgpa*9.5f;  
  
        System.out.println("CGPA of "+cgpa+" to percentage is =" +percentage);  
    }  
}
```

**Q)Write a java program to accept 6 marks of a student then find out total and average?**

```
class Example11  
{  
    public static void main(String[] args)  
    {  
        int m1=87,m2=43,m3=39,m4=57,m5=65,m6=36;  
        int total=0;  
        float average;
```

```

total=m1+m2+m3+m4+m5+m6;

average=(float)total/6;

System.out.println("Total :" +total);
System.out.println("Average :" +average);

}
}

```

**Q)Write a java program to find out area of a triangle?**

```

import java.util.Scanner;

class Example12
{
    public static void main(String[] args)
    {
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter the base :");
        int base=sc.nextInt();
        System.out.println("Enter the height :");
        int height=sc.nextInt();

        //logic
        float area=0.5f*base*height;

        System.out.println("Area of a triangle is =" +area);
    }
}

```

## Typecasting

The process of converting from one datatype to another datatype is called typecasting.

In java, we can perform typecasting in two ways.

1) Implicit typecasting

2) Explicit typecasting

### **1) Implicit typecasting:**

- If we want to store small value into a bigger variable then we need to use implicit typecasting.
- A compiler is responsible to perform implicit typecasting.
- There is no possibility to loss the information.
- It is also known as Widening or Upcasting.
- We can perform implicit typecasting as follow.

ex:

byte --> short

-->



int --> long --> float --> double

-->

char

ex1:

```
class Test
{
    public static void main(String[] args)
    {
        byte b=10;

        int i=b;

        System.out.println(i);//10
    }
}
```

```
}
```

### Ex2:

```
class Test  
{  
    public static void main(String[] args)  
    {  
        char ch='a';  
  
        int i=ch;  
  
        System.out.println(i); // 97  
    }  
}
```

### Ex3:

```
class Test  
{  
    public static void main(String[] args)  
    {  
        int i =10;  
  
        float f=i;  
  
        System.out.println(f); //10.0  
    }  
}
```



## 2) Explicit typecasting:

- ✓ If we want store bigger value into a smaller variable then we need to use explicit typecasting.
- ✓ A programmer is responsible to perform explicit typecasting.
- ✓ There is a possibility to loss the information.
- ✓ It is also known as Narrowing or Downcasting.
- ✓ We can perform explicit typecasting as follow.

ex:

```
byte    <--    short  
          <--  
          int <-- long <-- float <-- double  
          <--  
          char
```

ex1:

```
class Test  
{  
    public static void main(String[] args)  
    {  
        float f=10.5f;  
        int i =(int)f;  
        System.out.println(i); //10  
    }  
}
```



Ex2:

```
class Test  
{  
    public static void main(String[] args)  
    {  
        int i=65;  
  
        char ch=(char)i;  
  
        System.out.println(ch); //A
```

```

    }
}

Ex3:

class Test
{
    public static void main(String[] args)
    {
        int i=130;
        byte b=(byte)i;

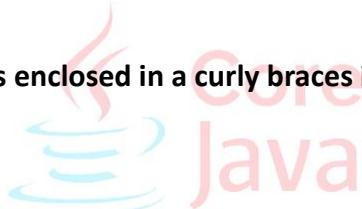
        System.out.println(b);//-126
    }
}

```

### Types of Blocks in Java:

A block is a set of statements which is enclosed in a curly braces i.e {}.

#### syntax:



```

{
    -
    - //set of statements
    -
}


```

We have three types of blocks in java.

**1)Instance block**

**2)Static block**

**3)Local block**

### **1)Instance block:**

- ✓ A instance block is used to initialize the instance variables.
- ✓ A instance block must and should declare immediately after the class but not inside methods and constructors.
- ✓ Instance block will execute at the time of object creation.Hence it is also known as lazy loading.

We can declare instance block as follow.

syntax:

```
//instance block  
{  
    -  
    - //set of statements  
    -  
}
```

ex1:

```
class Test  
{  
    //instance block  
    {  
        System.out.println("Instance-Block");  
    }  
  
    public static void main(String[] args)  
    {  
        System.out.println("Main-Method");  
    }  
}
```

**o/p:**

Main-Method

ex1:

```
class Test  
{  
    //instance block  
    {  
        System.out.println("Instance-Block");  
    }  
}
```



```
public static void main(String[] args)
{
    System.out.println("Main-Method");
    Test t=new Test();
}
}
```

**o/p:**

**Main-Method**

**Instance-Block**

**Ex3:**

```
class Test
```

```
{
    //instance block
{
    System.out.println("Instance-Block");
}
```



```
public static void main(String[] args)
{
    Test t1=new Test();
    System.out.println("Main-Method");
    Test t2=new Test();
}
}
```

**o/p:**

**Instance-Block**

**Main-Method**

**Instance-Block**

#### Ex4:

```
class Test
{
    //instance variable
    int i;

    //instance block
    {
        i=100;
    }

    public static void main(String[] args)
    {
        Test t=new Test();
        System.out.println(t.i);
    }
}
```



#### 2)Static block:

- A static block is used to initialize static variables.
- A static block must and should declare immediately after the class using static keyword but not inside methods,blocks and constructors.
- A static block will execute at the time of class loading.Hence it is called early loading.

We can declare static block as follow.

#### Syntax:

```
//static block
static
{
    -
    -
    - //set of statements
    -
}
```

ex1:

```
class Test
{
    //static block
    static
    {
        System.out.println("static-block");
    }

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

**o/p:**

static-block  
main-method



ex2:

```
class Test
{
    //static variable
    static int i;

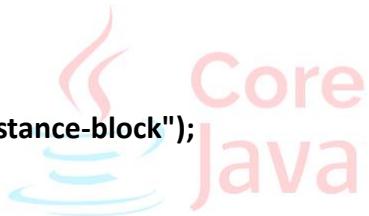
    //static block
    static
    {
        i=200;
    }

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

```
        System.out.println(i); // 200
    }
}
```

### Ex3 :

```
-----  
class Test  
{  
    //static block  
    static  
    {  
        System.out.println("static-block");  
    }  
    //instance block  
    {  
        System.out.println("instance-block");  
    }  
}
```



```
public static void main(String[] args)  
{  
    Test t=new Test();  
    System.out.println("main-method");  
}  
}
```

**o/p:**

```
static-block  
instance-block  
main-method
```

### 3)Local block:

- A local block is used to initialize the local variables.

- A local block must and should declare inside methods and constructors.

We can declare local block as follow.

### Syntax:

```
//local block  
{  
    -  
    - //set of statements  
    -  
}
```

### ex1:

```
class Test  
{  
  
    public static void main(String[] args)  
    {  
        System.out.println("stmt1");  
        //local block  
        {  
            System.out.println("stmt2");  
        }  
        System.out.println("stmt3");  
    }  
}
```

### o/p:

```
stmt1  
stmt2  
stmt3
```

### ex2:

```
class Test  
{
```



```
public static void main(String[] args)
{
    //local variable
    int i;

    //local block
    {
        i=300;
    }

    System.out.println(i);//300
}

}
```

### **Q)Is it possible to execute java program without main method?**

Till java 1.6 version it possible to execute java program without main method by using static block. But from java 1.7 version onwards it is not possible to execute java program without main method.

ex:

```
class Test
{
    //static block
    static
    {
        System.out.println("Hello World");
        System.exit(0);
    }
}
```

## Operators:

Operator is a symbol which is used to perform some operations on operands.

**ex:**

c = a + b;

Here = and + are operators.

Here a,b and c are operands.

It can be arithmetic operation, relational operation, logical operation , bitwise operation and etc.

We have following list of operators in java.

- 1)Assignment operators
- 2)Logical operators
- 3)Bitwise operators
- 4)Conditional / Ternary operators
- 5)Arithmetic operators
- 6)Relational operators
- 7)Unary operators



### **1)Assignment operators :**

**ex:**

```
class Test
{
    public static void main(String[] args)
    {
        int i=10;
        i=20;
        i=30;
        System.out.println(i);//30
    }
}
```

**Note:**

**Reinitialization is possible in java.**

**ex:**

```
class Test
{
    public static void main(String[] args)
    {
        final int i=10;
        i=20;
        i=30;
        System.out.println(i); // C.T.E
    }
}
```

**Note:**

Final variables we can't change

**ex:**

```
class Test
{
    public static void main(String[] args)
    {
        int i=(1,2,3,4,5);
        System.out.println(i); // C.T.E
    }
}
```

**ex:**

```
class Test
{
    //global variable
    static int i=10;
```



```
public static void main(String[] args)
{
    //local variable
    int i=20;

    System.out.println(i);//20
}
```

### Note:

Here priority goes to local variable.

### ex:

```
class Test
{

    public static void main(String[] args)
    {
        boolean b=5>2;

        System.out.println(b);//true
    }
}
```

### ex:

```
class Test
{

    public static void main(String[] args)
    {
        boolean b=5>20;
```



```
        System.out.println(b);//false
    }
}
```

**ex:**

```
class Test
{
    public static void main(String[] args)
    {
        int i=10;

        i+=5; // i = i + 5
    }
}
```



**ex:**

```
class Test
{
    public static void main(String[] args)
    {
        int i=10;

        i*=3; // i = i * 3

        System.out.println(i);// 30
    }
}
```

**ex:**

```
class Test
{
    public static void main(String[] args)
    {
        int i=10;

        i-=3; // i = i - 3

        System.out.println(i); // 7
    }
}
```

**ex:**

```
class Test
{
    public static void main(String[] args)
    {
        int i=10;

        i/=3;

        System.out.println(i); // 3
    }
}
```



```
public static void main(String[] args)
{
    int i=10;

    i/=30;

    System.out.println(i); //0
}
```

**ex:**

```
class Test
{
```

```
    public static void main(String[] args)
    {
        int i=10;
```



```
        i%=3;

        System.out.println(i); //1
    }
}
```

**ex:**

```
class Test
{
```

```
    public static void main(String[] args)
    {
        int i=20;
```

```
i%=500;  
  
        System.out.println(i);//20  
    }  
}
```

## How to convert decimal to binary

10 - decimal number

1010 - binary number

2|10  
--- 0  
2|5  
--- 1  
2|2        ^  
--- 0        |  
1



1010

decimal number : 2

binary number : 10

2|2  
--- 0 ^  
1
0010

## How to convert binary to decimal

**1010 - binary number**

**10 - decimal numbers**

**1010**

<---

**0\*1 + 1\*2 + 0\*4 + 1\*8**

**0 + 2 + 0 + 8**

**10**

## **2)Logical operators:**

### **Logical AND operator (&&)**

**truth table**

T	T	= T
T	F	= F
F	T	= F
F	F	= F



**ex:**

```
class Test
{
    public static void main(String[] args)
    {
        boolean b= (5>2) && (10<5);

        System.out.println(b); //false
    }
}
```

**ex:**

```
class Test
{
    public static void main(String[] args)
    {
        boolean b= (5>2) && (10<15);

        System.out.println(b); //true
    }
}
```

**ex:**

```
class Test
{
    public static void main(String[] args)
    {
        boolean b= (5>20) && false;

        System.out.println(b); //false
    }
}
```

**Logical OR operator (||):**

**truth table**

T	T	= T
T	F	= T
F	T	= T
F	F	= F

**ex:**

```
class Test
```



```
{  
  
    public static void main(String[] args)  
    {  
  
        boolean b= (5>3) || (6<2);  
  
        System.out.println(b); //true  
    }  
}
```

ex:

```
class Test  
{  
  
    public static void main(String[] args)  
    {  
  
        boolean b= (5>30) || (6<2);  
  
        System.out.println(b); //false  
    }  
}
```

ex:

```
class Test  
{  
  
    public static void main(String[] args)  
    {  
  
        boolean b= (6>1) && (4<1) || (6>9);  
  
        System.out.println(b); //false
```



```
    }  
}
```

### Logical NOT operator (!) :

ex:

```
class Test  
{  
    public static void main(String[] args)  
    {  
        boolean b= !(5>2);  
  
        System.out.println(b); //false  
    }  
}
```

ex:

```
class Test  
{  
    public static void main(String[] args)  
    {  
        boolean b= !(6>10);  
  
        System.out.println(b); //true  
    }  
}
```



### 3)Bitwise operators:

#### Bitwise AND operator (&)

Bitwise AND operator deals with binary numbers.

**truth table**

T	T	= T
---	---	-----

T      F      = F  
F      T      = F  
F      F      = F

**ex:**

```
class Test
{
    public static void main(String[] args)
    {
        int a=10,b=15;

        int c= a & b;

        System.out.println(c); //10
    }
}

/*
10 - 1010
15 - 1111
-----
& - 1010
<---
0*1 + 1*2 + 0*4 + 1*8

0 + 2 + 0 + 8 =10
*/
```

**ex:**

```
class Test
{
    public static void main(String[] args)
    {
```



```

int a=10,b=5;

int c= a & b;

System.out.println(c); //0

}

}

/*
10 - 1010
5 - 0101
-----
& - 0000
<---
0*1 + 0*2 + 0*4 + 0*8
0 + 0 + 0 + 0 =10
*/

```



### Bitwise OR operator (|):

Bitwise OR operator deals with binary numbers.

#### truth table

T	T	= T
T	F	= T
F	T	= T
F	F	= F

#### ex:

```

class Test
{
    public static void main(String[] args)
    {
        int a=10,b=5;

```

```

int c= a | b;

        System.out.println(c); //15

    }

}

/*
10 - 1010
5 - 0101
-----
| - 1111
<---
1*1 + 1*2 + 1*4 + 1*8
1+2+4+8
15
*/

```



## Bitwise XOR operator (^):

Bitwise XOR operator deals with binary numbers.

### Truth table

T	T	= F
T	F	= T
F	T	= T
F	F	= F

### ex:

```
class Test
```

```
{
```

```

public static void main(String[] args)
{
    int a=2 , b=3;

    int c = a ^ b;

    System.out.println(c); //1
}

}
/*

```

**2 - 0010**

**3 - 0011**

-----

**^ - 0001**

<---

**1\*1 + 0\*2 + 0\*4 + 0\*8**

**1 + 0 + 0 + 0 = 1**

\*/



#### 4) Conditional / Ternary operators:

##### Syntax:

**(condition)?value1:value2;**

##### Ex:

```

class Test
{
    public static void main(String[] args)

```

```
{  
    boolean b=(5>2)?true:false;  
  
    System.out.println(b);//true  
}  
}
```

**ex:**

```
class Test  
{  
    public static void main(String[] args)  
    {  
        boolean b=(!(5>2))?true:false;  
  
        System.out.println(b);//false } }
```

**Q)Write a java program to find out greatest of two numbers?**



```
import java.util.Scanner;  
class Test  
{  
    public static void main(String[] args)  
    {  
  
        Scanner sc=new Scanner(System.in);  
  
        System.out.println("Enter the First number :");  
        int a=sc.nextInt(); // 10  
  
        System.out.println("Enter the Second number :");  
        int b=sc.nextInt(); // 40  
  
        int max=(a>b)?a:b;
```

```
        System.out.println("Greatest of two numbers is =" + max);  
    }  
}
```

**Q)Write a java program to find out greatest of three numbers?**

```
import java.util.Scanner;  
  
class Test  
{  
  
    public static void main(String[] args)  
    {  
  
        Scanner sc=new Scanner(System.in);  
  
        System.out.println("Enter the First number :");  
        int a=sc.nextInt(); // 10  
  
        System.out.println("Enter the Second number :");  
        int b=sc.nextInt(); // 40  
  
        System.out.println("Enter the Third number :");  
        int c=sc.nextInt(); // 60  
  
        int max=(a>b)?((a>c)?a:c):((b>c)?b:c);  
  
        System.out.println("Greatest of three numbers is =" + max);  
    }  
}
```

## Right Shift Operator (>>)

**10 >> 1 = 10/2**

**10 >> 2 = 10/4**

**10 >> 3 = 10/8**

**10 >> 4 = 10/16**

**10 >> 5 = 10/32**

**ex:**

```
class Test
{
    public static void main(String[] args)
    {
        int i= 10 >> 3;
        System.out.println(i); // 10/8
    }
}
```

**ex:**

```
class Test
{
    public static void main(String[] args)
    {
        int i= 30 >> 5;
        System.out.println(i); // 30/32 =0
    }
}
```



## Left Shift Operator (<<)

**10 << 1 = 10\*2**

**10 << 2 = 10\*4**

**10 << 3 = 10\*8**

**10 << 4 = 10\*16**

**10 << 5 = 10\*32**

**ex:**

```
class Test
{
    public static void main(String[] args)
    {
        int i= 10 << 2;
        System.out.println(i); // 10*4 = 40
    }
}
```

**ex:**

```
class Test
{
    public static void main(String[] args)
    {
        int i= 100 << 6;
        System.out.println(i); // 100 * 64 = 6400
    }
}
```

**ex:**

```

class Test
{
    public static void main(String[] args)
    {
        System.out.println(10 > 20); // false
        System.out.println(10 < 20); // true

        System.out.println(10 >= 10 );// true
        System.out.println(10 <= 20); // true
    }
}

```

### Arithmetic Operators:

% - modules

/ - division

\* - multiplication

+ - addition

- - subtraction



### ex:

```

class Test
{
    public static void main(String[] args)
    {
        int i= 6+5%3+7/2*4+8-6;

        System.out.println(i);
    }
}

/*
   6 + 5%3 + 7/2*4 + 8-6;

```

**6 + 2 + 3\*4 + 8 - 6**

**6 + 2 + 12 + 8 - 6**

**28-6 = 22**

**\*/**

## **Increment / Decrement operators (++/--)**

We have two types of increment operators.

### **1)Pre-Increment**

**ex:**

**++i;**

### **2)Post-Increment**

**ex:**



**i++;**

We have two types of decrement operators.

### **1)Pre-Decrement**

**ex:**

**--i;**

### **2)Post-Decrement**

**ex:**

**i--;**

## Post Increment and Decrement

Rule1: First take

Rule2: Then change

ex:

```
class Test
{
    public static void main(String[] args)
    {
        int i=10;

        i++;

        System.out.println(i); // 11
    }
}
```



ex:

```
class Test
{
    public static void main(String[] args)
    {
        int i=10;

        System.out.println(i++); // 10
    }
}
```

ex:

```
class Test
```

```
{
```

```
public static void main(String[] args)
{
    int i=10;

    int j=i++;

    System.out.println(i+" "+j);//11 10
}
```

**ex:**

```
class Test
{
    public static void main(String[] args)
    {
        int i=10;

        int j=i--;

        System.out.println(i+" "+j);//9 10
    }
}
```

**ex:**

```
class Test
{
    public static void main(String[] args)
    {
        int i=10;

        int j=i++ + i++; // 10 + 11
```



```
        System.out.println(i+" "+j);//12 21  
    }  
}
```

ex:

```
class Test  
{  
    public static void main(String[] args)  
    {  
        int i=10;  
  
        int j=i-- + i--;  
    }
```

```
        System.out.println(i+" "+j);//8 19
```

## Pre-increment or pre-decrement



Rule1: first change

Rule2: then take

ex:

```
class Test  
{  
    public static void main(String[] args)  
    {  
        int i=10;  
  
        ++i;  
  
        System.out.println(i); // 11  
    }
```

```
}
```

**ex:**

```
class Test  
{  
    public static void main(String[] args)  
    {  
        int i=10;  
  
        System.out.println(++i);// 11  
    }  
}
```

**ex:**

```
class Test  
{  
    public static void main(String[] args)  
    {  
        int i=10;  
  
        int j=++i;  
  
        System.out.println(i+" "+j);//11 11  
    }  
}
```

**ex:**

```
class Test  
{  
    public static void main(String[] args)  
    {  
        int i=10;
```



```
int j=--i + --i; // 9 + 8

System.out.println(i+" "+j);//8 17
}

}
```

**ex:**

```
class Test

{
    public static void main(String[] args)
    {
        int i=10;

        int j=i++ + ++i;
```



```
System.out.println(i+" "+j);//12  22
```

```
}
```

**ex:**

```
class Test

{
    public static void main(String[] args)
    {
        int i=100;

        100++;
```

```
        System.out.println(i);//C.T.E  
    }  
}
```

**ex:**

```
class Test  
{  
    public static void main(String[] args)  
    {  
        int i=10;  
  
        System.out.println(++i++);// C.T.E  
    }  
}
```

**“Don’t tell sorry anyone without your mistake”**



- A control statement enables the programmer to control flow of our program.
- A control statement allows to make decisions, to jump from one section of code to another section and to execute the code repeatedly.

**we have four types of control statements in java.**

- 1) Decision Making Statement
- 2) Selection Statement
- 3) Iteration Statement
- 4) Jump Statement

## **1) Decision Making Statement**

It is used to create conditions in our program.

Decision making statement is possible by using following ways.

- i) if stmt
- ii) if else stmt
- iii) if else if ladder

iv) nested if stmt

### i) if stmt

It will execute the source code only if our condition is true.

**syntax:**

```
if(condition/expression)
{
    -
    - //code to be execute
    -
}
```

**ex:**

```
class Test
{
    public static void main(String[] args)
    {
        System.out.println("stmt1");
        if(5>2)
        {
            System.out.println("stmt2");
        }
        System.out.println("stmt3");
    }
}
```

**o/p:**

```
stmt1
stmt2
stmt3
```

ex:

```
class Test
{
    public static void main(String[] args)
    {
        System.out.println("stmt1");

        if(5>20)
        {
            System.out.println("stmt2");
        }

        System.out.println("stmt3");
    }
}
```

o/p:

stmt1

stmt3



ex:

```
class Test
{
    public static void main(String[] args)
    {
        if(!(5>2))
        {
            System.out.println("stmt1");
            System.out.println("stmt2");
            System.out.println("stmt3");
        }
    }
}
```

**o/p:**

stmt2

stmt3

**ex:**

class Test

{

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

{

**if(!false)**

**System.out.println("stmt1");**

**System.out.println("stmt2");**

**System.out.println("stmt3");**

}

}

**o/p:**

stmt1

stmt2

stmt3



**Q)Write a java program to find out greatest of two numbers?**

**import java.util.Scanner;**

**class Test**

{

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

{

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

**System.out.println("Enter the First Number :");**

**int a=sc.nextInt(); // 10**

```

System.out.println("Enter the second number :");

int b=sc.nextInt(); // 16

if(a>b)
    System.out.println(a+" is greatest");

if(b>a)
    System.out.println(b+" is greatest");

}

}

```

## **Q)Write a java program to find out greatest of three numbers?**

```

import java.util.Scanner;

class Test

{
    public static void main(String[] args)
    {
        Scanner sc=new Scanner(System.in);

        System.out.println("Enter the First Number :");

        int a=sc.nextInt(); // 10

        System.out.println("Enter the second number :");

        int b=sc.nextInt(); // 16

        System.out.println("Enter the third number :");

        int c=sc.nextInt(); // 6

        if((a>b) && (a>c))
            System.out.println(a+" is greatest");

        if((b>a) && (b>c))
            System.out.println(b+" is greatest");
    }
}

```



```

        if((c>a) && (c>b))

            System.out.println(c+" is greatest");

    }

}

```

## **ii) if else stmt :**

It will execute the source code either our condition is true or false.

### **syntax:**

```

if(condition/expression)

{
    - // code to be execute if cond is true

}

else

{
    - // code to be execute if cond is false
}

```



### **ex:**

```

class Test

{
    public static void main(String[] args)
    {
        System.out.println("stmt1");
        if(5>2)
            System.out.println("TRUE");
        else
            System.out.println("FALSE");
    }
}

```

```
        System.out.println("stmt2");  
    }  
}
```

**o/p:**

stmt1

TRUE

stmt2

**ex:**

```
class Test
```

```
{
```

```
    public static void main(String[] args)
```

```
{
```

```
        System.out.println("stmt1");
```

```
        if(5>20)
```

```
            System.out.println("TRUE");
```

```
        else
```

```
            System.out.println("FALSE");
```

```
        System.out.println("stmt2");
```

```
}
```

**o/p:**

stmt1

FALSE

stmt2

**Q)Write a java program to check given age is eligible to vote or not?**

```
import java.util.Scanner;
```

```
class Test
```

```
{
```

```

public static void main(String[] args)
{
    Scanner sc=new Scanner(System.in);

    System.out.println("Enter the Age :");
    int age=sc.nextInt();//7

    if(age>=18)
        System.out.println("U r eligible to vote");
    else
        System.out.println("U r not eligible to vote");
}
}

```

**Q)Write a java program to check given number is even or odd?**



```

import java.util.Scanner;
class Test
{
    public static void main(String[] args)
    {
        Scanner sc=new Scanner(System.in);

        System.out.println("Enter the number :");
        int n=sc.nextInt();//7

        if(n%2==0)
            System.out.println("It is even number");
        else
            System.out.println("It is odd number");
    }
}

```

**Q)Write a java program to find out given number is odd or not?**

```
import java.util.Scanner;  
  
class Test  
{  
  
    public static void main(String[] args)  
    {  
  
        Scanner sc=new Scanner(System.in);  
  
        System.out.println("Enter the number :");  
        int n=sc.nextInt(); //7  
  
        if(n%2!=0)  
            System.out.println("It is odd number");  
        else  
            System.out.println("It is not odd number");  
    }  
}
```

**Q)Write a java program to find out given year is a leap year or not?**

```
import java.util.Scanner;  
  
class Test  
{  
  
    public static void main(String[] args)  
    {  
  
        Scanner sc=new Scanner(System.in);  
  
        System.out.println("Enter the year :");  
        int year=sc.nextInt(); //
```

```

        if(year%4==0)

            System.out.println("It is a leap year");

        else

            System.out.println("It is not a leap year");

    }

}

```

**Q)Write a java program to check given number is +ve or -ve?**

**ex:**

```

import java.util.Scanner;

class Test

{

    public static void main(String[] args)

    {

        Scanner sc=new Scanner(System.in);

        System.out.println("Enter the number :");

        int n=sc.nextInt(); //6

        if(n==0)

        {

            System.out.println("It is not a positive or negative number");

            System.exit(0);

        }

        if(n>0)

            System.out.println("It is positive number ");

        else

            System.out.println("It is negative number ");

    }

}

```

### iii)if else if ladder

It is used to execute the source code based on multiple conditions.

syntax:

```
if(cond1)
{
    - //code to be execute if cond1 is true
}
else if(cond2)
{
    - //code to be execute if cond2 is true
}
else if(cond3)
{
    - //code to be execute if cond3 is true
}
else
{
    - //code to be execute if all conditions are false.
}
```



ex:

```
import java.util.Scanner;

class Test
{
    public static void main(String[] args)
    {
        Scanner sc=new Scanner(System.in);

        System.out.println("Enter the option :");
        int option=sc.nextInt(); // 10
```

```

if(option==100)
    System.out.println("It is police number");

else if(option==103)
    System.out.println("It is enquiry number");

else if(option==108)
    System.out.println("It is emergency number");

else
    System.out.println("Invalid option");

}
}

```

**Q)Write a java program to check given alphabet is a vowel or not?**

vowels : a e i o u

```

import java.util.Scanner;

class Test
{
    public static void main(String[] args)
    {
        Scanner sc=new Scanner(System.in);

        System.out.println("Enter the Alphabet :");
        char ch=sc.next().charAt(0);

        if(ch=='a')
            System.out.println("It is a vowel");

        else if(ch=='e')
            System.out.println("It is a vowel");

        else if(ch=='i')
            System.out.println("It is a vowel");

        else if(ch=='o')
            System.out.println("It is a vowel");
    }
}

```



```

        System.out.println("It is a vowel");

    else if(ch=='u')

        System.out.println("It is a vowel");

    else

        System.out.println("It is not a vowel");

    }

}

```

**Q)Write a java program to find out given alphabet is a upper case letter , lower case letter , digit or a special symbol?**

**ex:**

```

import java.util.Scanner;

class Test

{

    public static void main(String[] args)

    {

        Scanner sc=new Scanner(System.in);

        System.out.println("Enter the alphabet :");

        char ch=sc.next().charAt(0);

        if(ch>='A' && ch<='Z')

            System.out.println("It is a upper case letter");

        else if(ch>='a' && ch<='z')

            System.out.println("It is a lower case letter");

        else if(ch>='0' && ch<='9')

            System.out.println("It is a digit");

        else

            System.out.println("It is special symbol");

    }

}

```



**Q)Write a java program to accept 6 marks of a student then find out total ,average and grade?**

- i. if average is greater then equals to 75 then A grade.
- ii. if average is greater then equals to 50 then B grade.
- iii. if average is greater then equals to 35 then C grade.
- iv. if average is less then 35 then failed.

**ex:**

```
class Test
{
    public static void main(String[] args)
    {
        int m1=56,m2=39,m3=56,m4=87,m5=48,m6=62;
        int total=0;
        float average;

        total=m1+m2+m3+m4+m5+m6;
        average=(float)total/6;

        System.out.println("Total :" +total);
        System.out.println("Average :" +average);

        if(average>=75)
            System.out.println("Grade : A grade");
        else if(average>=50)
            System.out.println("Grade : B grade");
        else if(average>=35)
```



```

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

    else

        System.out.println("Grade : Failed");


    }

}

```

#### **iv)nested if stmt :**

If stmt contains another if stmt is called nested if stmt.

##### **syntax:**

```

if(condition)
{
    if(condition)
    {
        -
        -
        - //code to be execute
        -
    }
}

```



##### **ex:**

```

class Test

{
    public static void main(String[] args)
    {
        System.out.println("stmt1");
        if(100==100)

```

```
{  
    System.out.println("stmt2");  
    if(5>3)  
    {  
        System.out.println("stmt3");  
    }  
    System.out.println("stmt4");  
}  
System.out.println("stmt5");  
}  
}  
stmt1  
stmt2  
stmt3  
stmt4  
stmt5
```



ex:

```
class Test  
{  
    public static void main(String[] args)  
    {  
        System.out.println("stmt1");  
        if(100==100)  
        {  
            System.out.println("stmt2");  
            if(5>30)  
            {  
                System.out.println("stmt3");  
            }  
            System.out.println("stmt4");  
    }
```

```
        }
        System.out.println("stmt5");
    }
}
```

**o/p:**

stmt1

stmt2

stmt4

stmt5

**ex:**

```
class Test
{
    public static void main(String[] args)
    {
        System.out.println("stmt1");
        if(100!=100)
        {
            System.out.println("stmt2");
            if(5>30)
            {
                System.out.println("stmt3");
            }
            System.out.println("stmt4");
        }
        System.out.println("stmt5");
    }
}
```



**Q)Write a java program to find to given number is +ve or -ve ?**

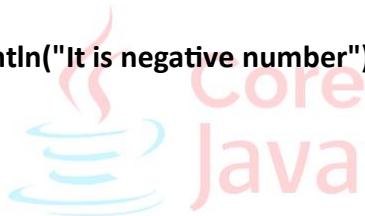
```
import java.util.Scanner;
class Test
```

```

{
    public static void main(String[] args)
    {
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter the number :");
        int n=sc.nextInt();//7

        if(n!=0)
        {
            if(n>0)
            {
                System.out.println("It is positive number");
                System.exit(0);
            }
            System.out.println("It is negative number");
        }
    }
}

```



## 2) Selection Statement:

### switch case

It is used execute the source code based on multiple conditions.

It is similar to if else if ladder.

### Syntax:

```

switch(condition/expression)
{
    case value1: //code to be execute
        break stmt;
}

```

```

        case value2: //code to be execute
            break stmt;

        default: //code to be execute if all cases are failed
    }

```

**Declaration of break stmt is optional in switch case.**

If we won't declare break stmt then from where our condition is satisfied from there all cases will be executed that state is called "Fall Through State of Switch Case".

**ex:**

```

import java.util.Scanner;
class Test
{
    public static void main(String[] args)
    {
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter the option :");
        int option=sc.nextInt(); // 103

        switch(option)
        {
            case 100 : System.out.println("It is police number");
                        break;
            case 103 : System.out.println("It is enquiry number");
                        break;
            case 108 : System.out.println("It is emergency number");
                        break;
            default: System.out.println("Invalid option");
        }
    }
}

```



```

    }
}

ex:

import java.util.Scanner;

class Test
{
    public static void main(String[] args)
    {
        Scanner sc=new Scanner(System.in);

        System.out.println("Enter the option :");
        int option=sc.nextInt(); // 103

        switch(option)
        {
            case 100 : System.out.println("It is police number");
            case 103 : System.out.println("It is enquiry number");

            case 108 : System.out.println("It is emergency number");

            default: System.out.println("Invalid option");
        }
    }
}

```

**Q)Write a java program to check given alphabet is a vowel or consonent?**

**vowels : a e i o u**

**ex:**

```

import java.util.Scanner;

class Test

```

```
{  
  
    public static void main(String[] args)  
    {  
  
        Scanner sc=new Scanner(System.in);  
  
        System.out.println("Enter the Alphabet :");  
        char ch=sc.next().charAt(0);  
  
        switch(ch)  
        {  
  
            case 'a': System.out.println("It is a vowel"); break;  
            case 'e': System.out.println("It is a vowel"); break;  
            case 'i': System.out.println("It is a vowel"); break;  
            case 'o': System.out.println("It is a vowel"); break;  
            case 'u': System.out.println("It is a vowel"); break;  
            default : System.out.println("It is a consonent");  
        }  
  
    }  
}
```

ex:

```
import java.util.Scanner;  
  
class Test  
{  
  
    public static void main(String[] args)  
    {  
  
        Scanner sc=new Scanner(System.in);  
  
        System.out.println("Enter the number in word :");
```

```

String str=sc.next();

switch(str)
{
    case "one": System.out.println("January"); break;
    case "two": System.out.println("February"); break;
    case "three": System.out.println("March"); break;
    case "four": System.out.println("April"); break;
    case "five": System.out.println("May"); break;
    default : System.out.println("Coming Soon....");
}

}

```

**Switch contains integers, Strings and characters.**

**ex:**



```

class Test
{
    public static void main(String[] args)
    {
        float f=10.3f;

        switch(f)
        {
            case 10.0: System.out.println("stmt1"); break;
            case 10.1: System.out.println("stmt2"); break;
            case 10.2: System.out.println("stmt3"); break;
            case 10.3: System.out.println("stmt4"); break;
            case 10.4: System.out.println("stmt5"); break;
            default : System.out.println("Not Found");
        }
    }
}

```

```
    }  
  
}  
}  
o/p: C.T.E
```

### **3) Iteration Statement:**

**Iteration statement is used to execute the code repeatedly.**

**Iteration statement is possible using LOOPS.**

**We have four types of LOOPS.**

- i) do while loop**
- ii) while loop**
- iii) for loop**
- iv) for each loop**



#### **i) do while loop**

**It will execute the source code until our condition is true.**

**syntax:**

```
do  
{  
-  
- //code to be execute
```

```
 }while(condition);
```

In do while loop, our code will execute atleast for one time either our condition is true or false.

ex:

```
class Test
{
    public static void main(String[] args)
    {
        int i=1;

        do
        {
            System.out.print(i+" ");
        }
        while (i<=10);
    }
}
```



ex:

```
class Test
{
    public static void main(String[] args)
    {
        int i=11;

        do
        {
```

```
        System.out.print(i+" ");//11
    }
    while (i<=10);

}
}
```

**Q)Write a java program to display 10 natural numbers ?**

```
class Test
{
    public static void main(String[] args)
    {
        int i=1;

        do
        {
            System.out.print(i+" ");
            i++;
        }
        while (i<=10);

    }
}
```



**Q)Write a java program to display 10 natural numbers in descending order ?**

```
class Test
{
    public static void main(String[] args)
    {
        int i=10;
```

```

    do
    {
        System.out.print(i+" ");
        i--;
    }
    while (i>=1);

}

```

**Q)Write a java program to perform sum of 10 natural numbers ?**

$$1+2+3+4+5+6+7+8+9+10 = 55$$

```

class Test
{
    public static void main(String[] args)
    {
        int i=1,sum=0;

        do
        {
            sum=sum+i;
            i++;
        }
        while (i<=10);

        System.out.println("Sum of 10 natural numbers is =" +sum);
    }
}

```



**Q)Write a java program to find out factorial of a given number?**

$$n = 5 \times 4 \times 3 \times 2 \times 1 = 120$$

ex:

```
import java.util.Scanner;

class Test

{

    public static void main(String[] args)

    {

        Scanner sc=new Scanner(System.in);

        System.out.println("Enter the number :");

        int n=sc.nextInt();

        int i=n,fact=1;

        do

        {

            fact=fact*i;

            i--;

        }

        while (i>=1);

        System.out.println("Factorial of a given number is =" +fact);

    }

}
```



**Q) Write a java program to display multiplication table of a given number?**

```
import java.util.Scanner;

class Test

{

    public static void main(String[] args)

    {

        Scanner sc=new Scanner(System.in);

        System.out.println("Enter the number :");
```

```

int n=sc.nextInt() // 6

int i=1;
do
{
    System.out.println(n+" * "+i+" = "+n*i);
    i++;
}
while (i<=10);

}

```

## **ii) while loop**

**It will execute the source code untill our condition is true.**

### **Syntax :**

```
while(condition/expression)
```

```
{
```

```
-
```

```
- //code to be execute
```

```
-
```

```
}
```



### **Ex:**

```

class Test

{
    public static void main(String[] args)
    {
        int i=1;

        while(i<=10)
        {

```

```
        System.out.print(i+" ");//infinite 1
    }
}
}
```

ex:

```
class Test
{
    public static void main(String[] args)
    {
        int i=11;

        while(i<=10)
        {
            System.out.print(i+" ");// nothing
        }
    }
}
```



**Q)Write a java program to display 10 natural numbers ?**

```
class Test
{
    public static void main(String[] args)
    {
        int i=1;

        while(i<=10)
        {

```

```

        System.out.print(i+" ");// 1 2 3 4 5 6 7 8 9 10

    i++;

}

}

}

```

**Q)Write a java program to display A to Z alphabets ?**

```

class Test

{
    public static void main(String[] args)
    {
        char i='A';

        while(i<='Z')
        {
            System.out.print(i+" ");// A B C D ... Z

            i++;
        }
    }
}

```



**Q)Write a java program to display 10 natural numbers ?**

```

class Test

{
    public static void main(String[] args)
    {
        int i=1,sum=0;

        while(i<=10)
        {
            sum=sum+i;
        }
    }
}

```

```

        i++;
    }

    System.out.println("sum of 10 natural numbers is =" + sum);
}

}

```

**Q)Write a java program to find out factorial of a given number ?**

```

import java.util.Scanner;

class Test
{
    public static void main(String[] args)
    {
        Scanner sc=new Scanner(System.in);

        System.out.println("Enter the number :");
        int n=sc.nextInt();

        int i=n,fact=1;

        while(i>=1)
        {
            fact=fact*i;
            i--;
        }

        System.out.println("Factorial of a given number is =" + fact);
    }
}

```



**Q)Write a java program to display multiplication table of a given number?**

```

import java.util.Scanner;

class Test
{

```

```
public static void main(String[] args)
{
    Scanner sc=new Scanner(System.in);

    System.out.println("Enter the number :");
    int n=sc.nextInt();

    int i=1;
    while(i<=10)
    {
        System.out.println(n+" * "+i+" = "+n*i);
        i++;
    }
}
```



**Q) Write a java program to perform sum of digits of a given number?**

**input:**

123

**output:**

6

**ex:**

```

import java.util.Scanner;

class Test

{
    public static void main(String[] args)
    {
        Scanner sc=new Scanner(System.in);

        System.out.println("Enter the number :");
        int n=sc.nextInt();

        int rem,sum=0;

        while(n>0)
        {
            rem = n % 10;
            sum = sum + rem;
            n = n / 10;
        }

        System.out.println("Sum of digits of a given number is =" +sum);
    }
}

```



**Q) Write a java program to find out reverse of a given number?**

**input:**

123

**output:**

321

**ex:**

import java.util.Scanner;

class Test

{

```
public static void main(String[] args)
{
    Scanner sc=new Scanner(System.in);

    System.out.println("Enter the number :");
    int n=sc.nextInt();

    int rem,rev=0;

    while(n>0)
    {
        rem = n % 10;
        rev = rev * 10 + rem;
        n = n / 10;
    }
    System.out.println("Reverse of a given number is =" +rev);
}
}
```

**Q)Write a java program to find out given number is palindrome or not?**

**input:**

**121**

**output:**

**It is palindrome number**

**ex:**

```
import java.util.Scanner;

class Test
{
    public static void main(String[] args)
    {
```

```

Scanner sc=new Scanner(System.in);

System.out.println("Enter the number :");
int n=sc.nextInt();

int temp=n;

int rem,rev=0;

while(n>0)
{
    rem = n % 10;
    rev = rev * 10 + rem;
    n = n / 10;
}
if(temp==rev)
    System.out.println("It is a palindrome number ");
else
    System.out.println("It is not a palindrome number ");
}
}

```

**Q)Write a java program to find out given number is armstrong or not?**

**input:**

**121 (1\*1\*1 + 2\*2\*2 + 1\*1\*1)**

**output:**

**It is not a armstrong number**

**input:**

**153 (1\*1\*1 + 5\*5\*5 + 3\*3\*3) (1+125+27)**

**output:**

**It is a armstrong number**

**ex:**

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



```
    Scanner sc=new Scanner(System.in);  
  
    System.out.println("Enter the number :");  
    int n=sc.nextInt();
```

```
    int temp=n;
```

```
    int rem,sum=0;
```

```
    while(n>0)  
    {  
        rem = n % 10;  
        sum = sum + rem * rem * rem;  
        n = n / 10;  
    }  
    if(temp==sum)  
        System.out.println("It is a armstrong number ");
```

```
        else  
            System.out.println("It is not a armstrong number ");  
    }  
}
```

## Ways to write java methods :

There are four ways to write java methods.

- 1) No returntype with No argument method
- 2) No returntype with argument method
- 3) With returntype with No argument method
- 4) With returntype with argument method

### 1) No returntype with No argument method:

If we don't have arguments then we need to ask input values inside callie method.

**Q)Write a java program to perform sum of two numbers using no returntype with no argument method?**

ex:

```
import java.util.Scanner;  
  
class Test  
{  
    public static void main(String[] args)  
    {  
        //caller method  
        m1();  
    }  
  
    //static method  
    //callie method
```

```

public static void m1()

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

    System.out.println("Enter the First Number :");

    int a=sc.nextInt();

    System.out.println("Enter the Second Number :");

    int b=sc.nextInt();


    //logic

    int c=a+b;

    System.out.println("sum of two numbers is =" +c);

}
}

```

**Q)Write a java program to find out square of a given number by using no returntype with no argument method ?**



ex:

```

import java.util.Scanner;

class Test

{
    public static void main(String[] args)
    {
        //caller method

        square();
    }

    //callie method

```

```

//static method

public static void square()

{

    Scanner sc=new Scanner(System.in);

    System.out.println("Enter the number :");

    int n=sc.nextInt();


    //logic

    int result=n*n;

    System.out.println("square of a given number is =" +result);

}

}

}

```

## **2) No returntype with argument method :**

IF we have arguments then we need to ask input values inside main method.

Here arguments are completely depends upon number of inputs.

**Q) Write java program to perform sum of two numbers by using no returntype with argument method?**

```

import java.util.Scanner;

class Test

{

    public static void main(String[] args)

    {

        Scanner sc=new Scanner(System.in);

        System.out.println("Enter the First Number :");

        int a=sc.nextInt();

```

```

System.out.println("Enter the Second Number :");

int b=sc.nextInt();

//caller method

sum(a,b);

}

//callie method

//static method

public static void sum(int a,int b)

{

    int c=a+b;

    System.out.println("sum of two numbers is =" +c);

}

}

```

**Q)Write a java program to find out square of a given number by using no returntype with argument method?**

ex:

```

import java.util.Scanner;

class Test

{

    public static void main(String[] args)

    {

        Scanner sc=new Scanner(System.in);

        System.out.println("Enter the number :");

        int n=sc.nextInt();

        //caller method

        square(n);
    }
}

```

```

}

//callie method

//static method

public static void square(int n)

{

    int result=n*n;

    System.out.println("square of a given number is "+result);

}

}

}

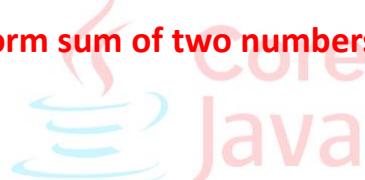
```

### 3)With returntype with No argument method:

A returntype is completely depends upon output.

**Q)Write a java program to perform sum of two numbers by using With returntype with No argument method ?**

ex:



```

import java.util.Scanner;

class Test

{

    public static void main(String[] args)

    {

        //caller method

        int k=sum();

        System.out.println("Sum of two numbers is "+k);

    }

}

//callie method

//static method

public static int sum()

```

```

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

    System.out.println("Enter the First Number :");

    int a=sc.nextInt();

    System.out.println("Enter the Second Number :");

    int b=sc.nextInt();

    //logic

    int c=a+b;

    return c;
}

}

```

**Q)Write a java program to find out square of a given number using with returntype with no argument method?**



```

import java.util.Scanner;

class Test
{
    public static void main(String[] args)
    {
        //caller

        int k=square();

        System.out.println("Square of a given number is =" +k);
    }

    //callie method

    //static method

    public static int square()
    {

```

```

Scanner sc=new Scanner(System.in);

System.out.println("Enter the Number :");

int n=sc.nextInt();

//logic

int result=n*n;

return result;

}

}

```

#### 4)With returntype with argument method:

Q)Write a java program to perform sum of two numbers by using with returntype with argument method?

ex:

```

import java.util.Scanner;

class Test

{
    public static void main(String[] args)
    {
        Scanner sc=new Scanner(System.in);

        System.out.println("Enter the First Number :");

        int a=sc.nextInt();

        System.out.println("Enter the Second Number :");

        int b=sc.nextInt();

        //caller method

        System.out.println("sum of two numbers is =" +sum(a,b));

    }
}

```



```

//callie method

//static method

public static int sum(int a,int b)

{

    int c=a+b;

    return c;

}

}

```

**Q)Write a java program to perform square of a given number by using with returntype with argument method?**

```

import java.util.Scanner;

class Test

{

    public static void main(String[] args)

    {

        Scanner sc=new Scanner(System.in);

        System.out.println("Enter the Number :");

        int n=sc.nextInt();




        //caller method

        System.out.println("square of a given number is =" +square(n));

    }






    //callie method

    //static method

    public static int square(int n)

    {

        //logic

```



```

        int result=n*n;

        return result;
    }

}

import java.util.Scanner;
class Test
{
    public static void main(String[] args)
    {
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter the Number :");
        int n=sc.nextInt();

        //caller method
        System.out.println(find(n));
    }

    //callie method
    //static method
    public static String find(int n)
    {
        if(n%2==0)
            return "It is even number";
        else
            return "It is odd number";
    }
}

```



## Assignment

Q)Write a java program to perform reverse of a given number using 4 ways?

Q)Write a java program to check given number is armstrong or not using 4 ways?

### Java Recursion:

A method which called itself for many number of times is called recursion.

Recursion is similar to loopings.

Whenever we implement recursion we should not use loops.

Q)Write a java program to find out factorial of a given number using recursion?

ex:1

```
import java.util.Scanner;  
  
class Test  
{  
    public static void main(String[] args)  
    {  
        Scanner sc=new Scanner(System.in);  
        System.out.println("Enter the Number :");  
        int n=sc.nextInt();  
  
        //caller method  
        System.out.println("Factorial of a given number is =" +factorial(n));  
    }  
    //callie method  
    //static method  
    public static int factorial(int n)  
    {  
        if(n<0)  
            return -1;
```

```

        if(n==0)
            return 1;

        return n*factorial(n-1);
    }
}

```

**Q)Write a java program to display sum of two numbers without using arithmetic operator?**

```

import java.util.Scanner;

class Test
{
    public static void main(String[] args)
    {
        Scanner sc=new Scanner(System.in);

        System.out.println("Enter the First Number :");
        int a=sc.nextInt();

        System.out.println("Enter the Second Number :");
        int b=sc.nextInt();

        //caller method
        System.out.println("sum of two numbers is =" +sum(a,b));

    }
    //callie method
    //static method
    public static int sum(int a,int b)

```



```

{
    if(a==0)
        return b;
    else
        return sum(--a,++b);
}
}

```

**Q)Write a java program to display 10 naturals without using loops?**

```

import java.util.Scanner;

class Test
{
    public static void main(String[] args)
    {
        //caller method
        display(1);
    }

    //callie method
    //static method
    public static void display(int i)
    {
        if(i<=10)
        {
            System.out.print(i+" ");
            display(i+1);
        }
    }
}

```



**Q)Write a java program to display Nth element of fibonacci series?**

**fibonacci series : 0 1 1 2 3 5 8**

**input:**

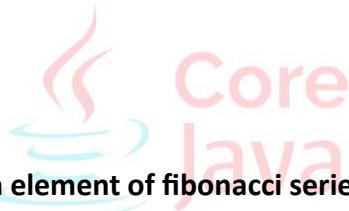
4

**output:**

2

**ex:**

```
import java.util.Scanner;  
  
class Test  
{  
  
    public static void main(String[] args)  
    {  
  
        Scanner sc=new Scanner(System.in);  
  
        System.out.println("Enter the Number :");  
        int n=sc.nextInt();  
  
        //caller method  
        System.out.println("Nth element of fibonacci series is =" +fib(n));  
    }  
  
    //callie method  
    //static method  
  
    public static int fib(int n)  
    {  
  
        if(n==0 || n==1)  
            return 0;  
        else if(n==2)  
            return 1;  
        else  
            return fib(n-1)+fib(n-2);  
    }  
}
```



### iii)for loop

It will execute the source code untill our condition is true.

**syntax:**

```
for(declaration and initialization;condition;incrementation/decremenatation)
{
    -
    -
    - //code to be execute
    -
}

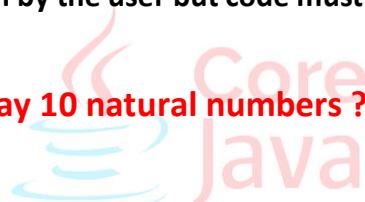
}
```

If number of iterations are known by the user then we need to use for loop.

If number of iterations are not known by the user then we need to use while loop.

If number of iterations are not known by the user but code must execute atleast for one time then we need to use do while loop.

**Q)Write a java program to display 10 natural numbers ?**



```
class Test
{
    public static void main(String[] args)
    {
        for(int i=1;i<=10;i++)
        {
            System.out.print(i+" "); //1 2 3 4 5 6 7 8 9 10
        }
    }
}
```

**ex:**

```
class Test
{
    public static void main(String[] args)
```

```

{
    for(;;)
    {
        System.out.print("Hello "); //infinite Hello
    }
}

```

**Q)Write a java program to display even numbers from 1 to 10?**

```

class Test
{
    public static void main(String[] args)
    {
        for(int i=1;i<=10;i++)
        {
            if(i%2==0)
                System.out.print(i+" "); //2 4 6 8 10
        }
    }
}

```

**Q)Write a java program to display factorial of a given number?**

ex:

```

import java.util.Scanner;
class Test
{
    public static void main(String[] args)
    {
        Scanner sc=new Scanner(System.in);

        System.out.println("Enter the Number :");
        int n=sc.nextInt();
    }
}

```

```

int fact=1;

for(int i=n;i>=1;i--)
{
    fact=fact*i;
}

System.out.println("Factorial of a given number is =" +fact);
}
}

```

**Q) Write a java program to display fibonacci series of a given number?**

**fibonacci series : 0 1 1 2 3 5 8**

**ex:**



```

import java.util.Scanner;

class Test
{
    public static void main(String[] args)
    {
        Scanner sc=new Scanner(System.in);

        System.out.println("Enter the Number :");
        int n=sc.nextInt(); //5

        int a=0,b=1,c;
        System.out.print(a+" "+b+" "); // 0 1

        for(int i=2;i<=n;i++)
        {
            c=a+b;

```

```
        System.out.print(c+" ");
        a=b;
        b=c;
    }

}

}
```

**Q)Write a java program to check given number is prime or not?**

**input:**

5

**output:**

It is prime number



**input:**

10

**output:**

It is not prime number

**ex:**

```
import java.util.Scanner;
```

```
class Test
```

```
{
```

```
    public static void main(String[] args)
```

```
{
```

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

```
        System.out.println("Enter the Number :");
```

```
        int n=sc.nextInt(); //5
```

```

boolean flag=true;

for(int i=2;i<=n/2;i++)
{
    if(n%i==0)
    {
        flag=false;
        break;
    }
}

if(flag==true)
    System.out.println("It is prime number ");

else
    System.out.println("It is not prime number");}

```

**Q)Write a java program to find out given number is perfect or not?**



```

import java.util.Scanner;

class Test
{

    public static void main(String[] args)
    {

        Scanner sc=new Scanner(System.in);

        System.out.println("Enter the Number :");
        int n=sc.nextInt(); //5

        int sum=0;
        for(int i=1;i<n;i++)
        {
            if(n%i==0)
            {

```

```

        sum=sum+i;

    }

}

if(n==sum)

    System.out.println("It is a perfect number ");

else

    System.out.println("It is not a perfect number ");}}
```

**Q)Write a java program to display all the prime numbers from 1 to Nth element?**

**input:**

**20**

**output:**

**2 3 5 7 11 13 17 19**

**ex:**

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



```
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter the Number :");
        int n=sc.nextInt();
```

```
    for(int i=2;i<n;i++)
    {
```

```
        boolean flag=true;
```

```
        for(int j=2;j<i;j++)
        {
```

```
            if(i%j==0)
```

```
{
```

```

        flag=false;

        break;
    }

}

if(flag==true)

    System.out.print(i+" ");

}

}

}

```

## Loop Patterns

1)  
1 1 1 1  
2 2 2 2  
3 3 3 3  
4 4 4 4



ex:  
class Test

```

{
    public static void main(String[] args)
    {
        //rows
        for(int i=1;i<=4;i++)
        {
            //columns
            for(int j=1;j<=4;j++)
            {

```

```
        System.out.print(i+" ");
    }
    //new line
    System.out.println("");
}}
```

2)

1 2 3 4

1 2 3 4

1 2 3 4

1 2 3 4

class Test

```
{
    public static void main(String[] args)
    {
        //rows
        for(int i=1;i<=4;i++)
        {
            //columns
            for(int j=1;j<=4;j++)
            {
                System.out.print(j+" ");
            }
            //new line
            System.out.println("");
        }
    }
}
```

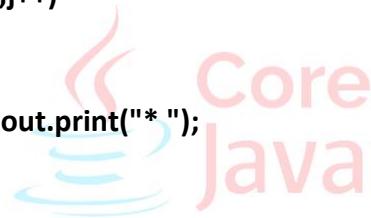


```
* * * *
* * * *
* * * *
* * * *

class Test
{
    public static void main(String[] args)
    {
        //rows
        for(int i=1;i<=4;i++)
        {
            //columns
            for(int j=1;j<=4;j++)
            {
                System.out.print("* ");
            }
            //new line
            System.out.println("");
        }
    }
}

4)
4 4 4 4
3 3 3 3
2 2 2 2
1 1 1 1

class Test
{
    public static void main(String[] args)
    {
```



```

//rows
for(int i=4;i>=1;i--)
{
    //cols
    for(int j=1;j<=4;j++)
    {
        System.out.print(i+" ");
    }
    //new line
    System.out.println("");
}
}

5)
A A A A
B B B B
C C C C
D D D D

```



```

class Test
{
    public static void main(String[] args)
    {
        //rows
        for(char i='A';i<='D';i++)
        {
            //cols
            for(char j='A';j<='D';j++)
            {
                System.out.print(i+" ");
            }
        }
    }
}

```

```

        }
        //new line
        System.out.println("");
    }
}

6)
D D D D
C C C C
B B B B
A A A A

```

```

class Test
{
    public static void main(String[] args)
    {
        //rows
        for(char i='D';i>='A';i--)
        {
            //cols
            for(char j='A';j<='D';j++)
            {
                System.out.print(i+" ");
            }
            //new line
            System.out.println("");
        }
    }
}

```

7)

\* \* \* \*



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

class Test
{
    public static void main(String[] args)
    {
        //rows
        for(int i=1;i<=4;i++)
        {
            //cols
            for(int j=1;j<=4;j++)
            {
                if(i==1 || i==4 || j==1 || j==4)
                    System.out.print("* ");
                else
                    System.out.print("  ");
            }
            //new line
            System.out.println("");
        }
    }
}
```

8)

```
* - -
-* - -
-- * -
--- *

class Test
{
    public static void main(String[] args)
```

```

{
    //rows
    for(int i=1;i<=4;i++)
    {
        //cols
        for(int j=1;j<=4;j++)
        {
            if(i==j)
                System.out.print("* ");
            else
                System.out.print("- ");
        }
        //new line
        System.out.println("");
    }
}
9)
* --- *
-* - * -
-- * --
-* - * -
* --- *

class Test
{
    public static void main(String[] args)
    {
        //rows
        for(int i=1;i<=5;i++)

```



```

    {
        //cols
        for(int j=1;j<=5;j++)
        {
            if(i==j || i+j==6)
                System.out.print("* ");
            else
                System.out.print("- ");
        }
        //new line
        System.out.println("");
    }
}

```

10)

1 1 1  
1 0 1  
1 1 1



```

class Test
{
    public static void main(String[] args)
    {
        //rows
        for(int i=1;i<=3;i++)
        {
            //cols
            for(int j=1;j<=3;j++)
            {
                if(i==2 && j==2)

```

```

        System.out.print("0 ");
    else
        System.out.print("1 ");
    }
    //new line
    System.out.println("");
}
}
}

```

### Left Side Loop Patterns

1)  
1  
2 2  
3 3 3  
4 4 4 4



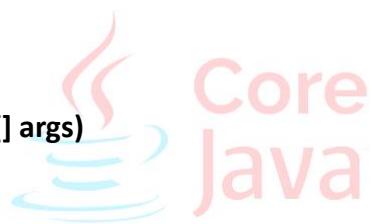
```

class Test
{
    public static void main(String[] args)
    {
        //rows
        for(int i=1;i<=4;i++)
        {
            //cols
            for(int j=1;j<=i;j++)
            {

```

```
        System.out.print(i+" ");
    }
    //new line
    System.out.println("");
}
}
2)
4 4 4 4
3 3 3
2 2
1

class Test
{
    public static void main(String[] args)
    {
        //rows
        for(int i=4;i>=1;i--)
        {
            //cols
            for(int j=1;j<=i;j++)
            {
                System.out.print(i+" ");
            }
            //new line
            System.out.println("");
        }
    }
}
3)
*
```



```
* *  
* * *  
* * * *  
  
class Test  
{  
    public static void main(String[] args)  
    {  
        //rows  
        for(int i=1;i<=4;i++)  
        {  
            //cols  
            for(int j=1;j<=i;j++)  
            {  
                System.out.print("* ");  
            }  
            //new line  
            System.out.println("");  
        }  
    }  
}
```

4)

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

```
class Test  
{  
    public static void main(String[] args)  
    {  
        //rows
```



```

for(int i=4;i>=1;i--)
{
    //cols
    for(int j=1;j<=i;j++)
    {
        System.out.print("* ");
    }
    //new line
    System.out.println("");
}

}
5)
1
2 3
4 5 6
7 8 9 0

```



```

class Test
{
    public static void main(String[] args)
    {
        int k=1;
        //rows
        for(int i=1;i<=4;i++)
        {
            //cols
            for(int j=1;j<=i;j++)
            {

```

```

        if(k<=9)

            System.out.print(k++ +" ");

        else

            System.out.print("0 ");

        }

        //new line

        System.out.println("");

    }

}

}

```

6)  
\*  
\* \*  
\* \* \*  
\* \* \* \*  
\* \* \*  
\* \*  
\*



```

class Test

{
    public static void main(String[] args)
    {
        //ascending
        //rows
        for(int i=1;i<=4;i++)
        {

```

```

//cols

for(int j=1;j<=i;j++)
{
    System.out.print("* ");
}

//new line

System.out.println("");

}

//descending

//rows

for(int i=3;i>=1;i--)
{
    //cols

    for(int j=1;j<=i;j++)
    {
        System.out.print("* ");
    }

    //new line

    System.out.println("");
}
}

```

7)

A

A B

A B C

A B C D

ex:

class Test

{

public static void main(String[] args)

```

{
    //rows
    for(char i='A';i<='D';i++)
    {
        //cols
        for(char j='A';j<=i;j++)
        {
            System.out.print(j+" ");
        }
        //new line
        System.out.println("");
    }
}

```

8)

D D D D

C C C

B B

A



class Test

```

{
    public static void main(String[] args)
    {
        //rows
        for(char i='D';i>='A';i--)
        {
            //cols
            for(char j='A';j<=i;j++)
            {
                System.out.print(i+" ");
            }
        }
    }
}
```

```
    }  
    //new line  
    System.out.println("");  
}  
}
```

## Interview Question

**Q)Write a java program to display reverse of a given number in words?**

**input:**

123

**output:**

ThreeTwoOne

**ex:**

```
import java.util.Scanner;  
  
class Test  
{  
    public static void main(String[] args)  
    {  
        Scanner sc=new Scanner(System.in);  
  
        System.out.println("Enter the number :");  
        int n=sc.nextInt();  
        String rev="";  
        int rem;  
        while(n>0 || n!=0)  
        {  
            rem=n%10;  
            switch(rem)  
            {  
                case 0 : rev+="Zero"; break;  
                case 1 : rev+="One"; break;
```



```

        case 2 : rev+="Two"; break;
        case 3 : rev+="Three"; break;
        case 4 : rev+="Four"; break;
        case 5 : rev+="Five"; break;
        case 6 : rev+="Six"; break;
        case 7 : rev+="Seven"; break;
        case 8 : rev+="Eight"; break;
        case 9 : rev+="Nine"; break;
    }
    n=n/10;
}
System.out.println(rev);
}

```

### Right Side patterns

1)  
1  
2 2  
3 3 3  
4 4 4 4



```

class Test
{
    public static void main(String[] args)
    {
        //rows
        for(int i=1;i<=4;i++)
        {
            //space
            for(int j=4;j>i;j--)
            {

```

```

        System.out.print(" ");
    }

    //elements

    for(int j=1;j<=i;j++)
    {
        System.out.print(i+" ");
    }

    //new line

    System.out.println("");
}

}

```

2)

4 4 4 4  
3 3 3  
2 2  
1



ex:

```

class Test
{
    public static void main(String[] args)
    {
        //rows

        for(int i=4;i>=1;i--)
        {
            //space

            for(int j=4;j>i;j--)
            {

```

```

        System.out.print(" ");
    }

    //elements

    for(int j=1;j<=i;j++)
    {
        System.out.print(i+" ");
    }

    //new line

    System.out.println("");
}

}

```

3)

```

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



```

class Test
{
    public static void main(String[] args)
    {
        //ascending
        //rows
        for(int i=1;i<=4;i++)
        {

```

```
//space  
  
for(int j=4;j>i;j--)  
{  
    System.out.print(" ");  
}  
  
//elements  
  
for(int j=1;j<=i;j++)  
{  
    System.out.print("* ");  
}  
  
//new line  
  
System.out.println("");  
}
```



```
//descending  
  
//rows  
for(int i=3;i>=1;i--)  
{  
    //space  
  
    for(int j=4;j>i;j--)  
    {  
        System.out.print(" ");  
    }  
  
    //elements  
  
    for(int j=1;j<=i;j++)  
    {  
        System.out.print("* ");  
    }  
  
    //new line  
  
    System.out.println("");
```

```
        }
    }
}
```

## Pyramids

1)

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

```
class Test
```

```
{
    public static void main(String[] args)
    {
        //rows
        for(int i=1;i<=4;i++)
        {
            //space
            for(int j=4;j>i;j--)
            {
                System.out.print(" ");
            }
            //left elements
            for(int j=1;j<=i;j++)
            {
                System.out.print(j+" ");
            }
            //right side
            for(int j=i-1;j>=1;j--)
```



```

        {
            System.out.print(j+" ");
        }

        //new line
        System.out.println("");
    }

}

2)
1 2 3 4 3 2 1
1 2 3 2 1
1 2 1
1

```



```

class Test
{
    public static void main(String[] args)
    {
        //rows
        for(int i=4;i>=1;i--)
        {
            //space
            for(int j=4;j>i;j--)
            {
                System.out.print(" ");
            }
            //left elements
            for(int j=1;j<=i;j++)

```

```

        {
            System.out.print(j+" ");
        }

        //right side
        for(int j=i-1;j>=1;j--)
        {
            System.out.print(j+" ");
        }

        //new line
        System.out.println("");
    }
}

```

3)

```

*
* *
* * *
* * * *
* * * * *

```

ex:

```

class Test
{
    public static void main(String[] args)
    {
        //rows
        for(int i=1;i<=4;i++)
        {
            //space
            for(int j=4;j>i;j--)
            {
                System.out.print(" ");
            }
        }
    }
}

```



```

    }

    //left elements

    for(int j=1;j<=i;j++)

    {

        System.out.print("* ");

    }

    //right side

    for(int j=i-1;j>=1;j--)

    {

        System.out.print("* ");

    }

    //new line

    System.out.println("");
}

}
}

```



## Assignment

```

1     1
1 2   2 1
1 2 3  3 2 1
1 2 3 4 3 2 1

```

## 4)Jump Statement

Jump statement is used to jump from one section of code to another section.

We have two types of jump statements in java.

i)break stmt

ii)continue stmt

### i)break stmt

It is used to break the execution of loops and switch case.

For conditional statements we can use if condition.

**syntax:**

**break;**

**ex:**

**class Test**

{

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

{

**System.out.println("stmt1");**

**break;**

**System.out.println("stmt2");**

}

}

**o/p: C.T.E break outside switch or loop**



**ex:**

**class Test**

{

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

{

**System.out.println("stmt1");**

**if(true)**

{

**break;**

}

**System.out.println("stmt2");**

}

}

**o/p: C.T.E break outside switch or loop**

**ex:**

```

class Test
{
    public static void main(String[] args)
    {
        for(int i=1;i<=10;i++)
        {
            if(i==5)
            {
                break;
            }
            System.out.print(i+" ");
        }
    }
}

```

## ii) continue stmt

It is used to continue the execution of loops.

For conditional statements we can use if condition.

**syntax:**

**continue;**

**ex:**

```

class Test
{
    public static void main(String[] args)
    {
        System.out.println("stmt1");

        continue;

        System.out.println("stmt2");
    }
}

```



```
}
```

**o/p: C.T.E continue outside of loop**

ex:

```
class Test
```

```
{
```

```
    public static void main(String[] args)
```

```
{
```

```
        System.out.println("stmt1");
```

```
        if(true)
```

```
{
```

```
        continue;
```

```
}
```

```
        System.out.println("stmt2");
```

```
}
```

```
}
```

**o/p: C.T.E continue outside of loop**

ex:

```
class Test
```

```
{
```

```
    public static void main(String[] args)
```

```
{
```

```
        for(int i=1;i<=10;i++)
```

```
{
```

```
        if(i==5)
```

```
{
```

```
        continue;
```

```
}
```

```
        System.out.print(i+" "); //1 2 3 4 6 7 8 9 10 }}}
```



**“Whatever happens,happens for good”**

-With love sandeep-

## **Arrays:**

- A normal value can store only one value at a time.
- To store more than one value at a time we need to use arrays.
- Array is a collection of homogeneous data elements.
- The main advantage of arrays are

**1) We can represent multiple elements using single variable name.**

**ex:**

```
int[] arr={10,20,30};
```

**2) Performance point of view arrays is recommended to use.**

The main disadvantage of arrays are



**1) Ararys are fixed in size.Once if we create an array there is no chance of increasing and decreasing the size of an array.**

**2) To use arrays concept in advanced we should know what is the size of an array which is always not possible.**

In java, Arrays are categories into three types.

- ✓ Single Dimensional Array
- ✓ Double Dimensional Array / Two Dimensional Array
- ✓ Multi-Dimensional Array / Three Dimensional Array

## **Array Declaration**

At the time of array declaration we should not specify array size.

## Arrays

--	--	--

Single Dimensional Array

Double Dimensional Array

Multi-Dimensional Array

int[] arr;

int[][] arr;

int[][][] arr;

int []arr;

int [][][]arr;

int [][][][]arr;

int arr[];

int arr[][];

int arr[][][];

int[] []arr;

int[][] []arr;

int[] arr[];

int[][] arr[];

int []arr[];

int[] [][]arr;

int[] arr[][],

int[] []arr[],

int [][][]arr[],

int [][][][]arr[],

### Array creation

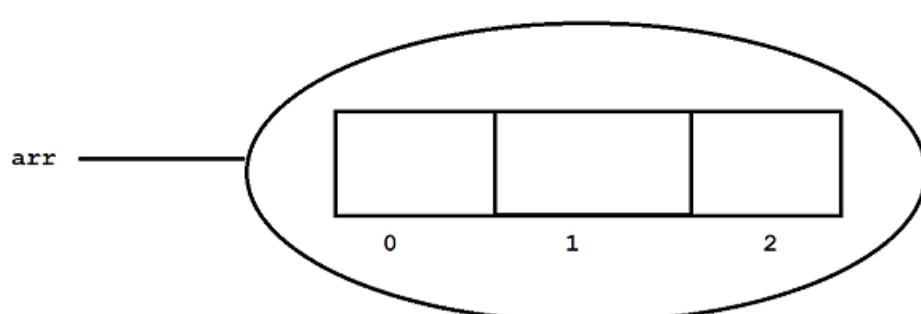


In java, every array consider an object.Hence we will use new operator to construct an array.

ex:

```
int[] arr=new int[3];
```

Diagram: java25.1



## Rules to constructor an array:

### Rule1:

**At the time of array creation compulsory we need to specify array size.**

ex:

```
int[] arr=new int[3];
```

```
int[] arr=new int[]; // C.T.E array dimision is missing
```

### Rule2:

**It is legal to have an array size with zero.**

ex:

```
int[] arr=new int[0];
```

```
System.out.println(arr.length); //0
```

### Rule3:

We can't give negative number as an array size otherwise we will get runtime exception called **NegativeArraySizeException**.

ex:

```
int[] arr=new int[3];
```

```
int[] arr=new int[-3]; // R.E NegativeArraySizeException
```

### Rule4:

**The allowed datatype of an array size is byte,short,int and char.**

**If we take other datatype then we will get compile time error.**

ex:

```
byte b=10;
```

```
int[] arr=new int[b];
```

```
int[] arr=new int['a'];
```

```
int[] arr=new int[10.5d]; //invalid
```

**Rule5:**

The maximum length we can give to an array size is maximum length of integer.

ex:

```
int[] arr=new int[2147483647];
```

**Array Initialization:**

Once if we create an array , every array element will be initialized with default values.

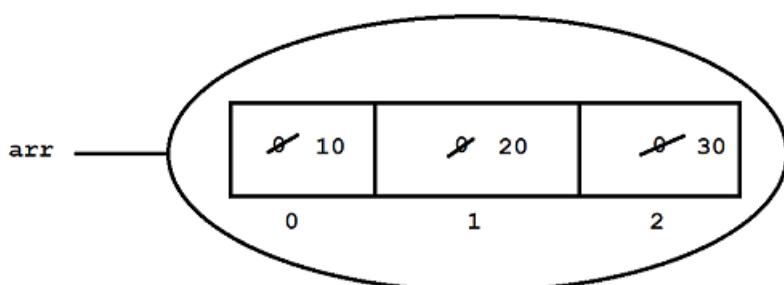
If we are not happy with default values then we can change with our customized values.

ex:

```
int[] arr=new int[3];
arr[0]=10;
arr[1]=20;
arr[2]=30;
arr[3]=40; // R.E ArrayIndexOutOfBoundsException
```



Diagram: java25.2



**Array Declaration, Creation and Initialization using single line:**

```
int[] arr;
arr=new int[3];
```

```

arr[0]=10;
arr[1]=20;
arr[2]=30;           ===> int[] arr={10,20,30};

====> char[] carr={'a','b','c'};

====> String[] sarr={"hi","hello","bye"};

```

**Q)What is the difference between length and length() method?**

### **length:**

**It is a final variable which is applicable only for arrays.**

**It will return size of an array.**

**ex:**

```

class Test
{
    public static void main(String[] args)
    {
        int[] arr=new int[4];

        System.out.println(arr.length);//4

        System.out.println(arr.length()); // C.T.E cannot find symbol
    }
}

```

### **length():**

**It is a final method which is applicable only for String objects.**

**It will return number of characters present in a String.**

**ex:**

```
class Test
```

```
{  
    public static void main(String[] args)  
    {  
        String str="bhaskar";  
  
        System.out.println(str.length()); // 7  
  
        System.out.println(str.length); // C.T.E cannot find symbol  
    }  
}
```

**Q)Write a java program to accept array elements and display them?**

```
import java.util.Scanner;  
  
class Test  
{  
    public static void main(String[] args)  
    {  
        Scanner sc=new Scanner(System.in);  
  
        System.out.println("Enter the Array Size :");  
        int size=sc.nextInt(); //5  
  
        int[] arr=new int[size];  
  
        //inserting elements  
        for(int i=0;i<arr.length;i++)  
        {  
            System.out.println("Enter the element : ");  
            arr[i]=sc.nextInt();  
        }  
  
        //displaying elements
```

```
        for(int i=0;i<arr.length;i++)  
        {  
            System.out.print(arr[i]+" ");  
        }  
    }  
}
```

### approach2

```
class Test  
{  
    public static void main(String[] args)  
    {  
        int[] arr={10,20,50,30};  
        //displaying elements  
        for(int i=0;i<arr.length;i++)  
        {  
            System.out.print(arr[i]+" ");  
        }  
    }  
}
```

### approach3

```
class Test  
{  
    public static void main(String[] args)  
    {  
        int[] arr={10,20,50,30};  
    }  
}
```



```
//displaying elements  
//for each loop  
for(int i:arr)  
{  
    System.out.print(i+" ");  
}  
}}
```

**Q)Write a java program to display array elements in reverse order?**

**input:**

5 2 8 1 9 6

**output:**

6 9 1 8 2 5

**ex:**

class Test

{

```
public static void main(String[] args)
```

{

```
    int[] arr={5,2,8,1,9,6};
```

```
    //reverse order
```

```
    for(int i=arr.length-1;i>=0;i--)
```

{

```
        System.out.print(arr[i]+" ");
```

}

}

}



**Q)Write a java program to perform sum of array elements ?**

**input:**

5 2 8 1 9 6

**output:**

31

**ex:**

**class Test**

{

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

{

**int[] arr={5,2,8,1,9,6};**

**//for each loop**

**int sum=0;**

**for(int i:arr)**

    {

**sum=sum+i;**

    }

**System.out.println(sum);**

}

}



**Q)Write a java program to display array elements in ascending order?**

**input:**

5 2 8 1 9 6

**output:**

1 2 5 6 8 9

**approach1**

**import java.util.Arrays;**

**class Test**

{

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

```
{  
    int[] arr={5,2,8,1,9,6};  
  
    Arrays.sort(arr);  
  
    //for each loop  
    for(int i:arr)  
    {  
        System.out.print(i+" ");  
    }  
}
```

## approach2

```
class Test  
{  
    public static void main(String[] args)  
    {  
        int[] arr={5,2,8,1,9,6};  
  
        //ascending order  
        for(int i=0;i<arr.length;i++)  
        {  
            for(int j=0;j<arr.length;j++)  
            {  
                if(arr[i]<arr[j])  
                {  
                    int temp=arr[i];  
                    arr[i]=arr[j];  
                    arr[j]=temp;  
                }  
            }  
        }  
    }  
}
```



```

    //display element
    //for each loop
    for(int i:arr)
    {
        System.out.print(i+" ");
    }
}

```

**Q)Write a java program to display array elements in descending order?**

**input:**

5 2 8 1 9 6

**output:**

9 8 6 5 2 1

**approach1**

```

import java.util.Arrays;
class Test
{
    public static void main(String[] args)
    {
        int[] arr={5,2,8,1,9,6};

        Arrays.sort(arr);// 1 2 5 6 8 9
    }
}

```



```

//read reverse
for(int i=arr.length-1;i>=0;i--)
{
    System.out.print(arr[i]+" ");
}

```

```
    }
}
```

### approach2

```
class Test
{
    public static void main(String[] args)
    {
        int[] arr={5,2,8,1,9,6};

        //descending order
        for(int i=0;i<arr.length;i++)
        {
            for(int j=0;j<arr.length;j++)
            {
                if(arr[i]>arr[j])
                {
                    int temp=arr[i];
                    arr[i]=arr[j];
                    arr[j]=temp;
                }
            }
        }

        //display elements
        for(int i:arr)
        {
            System.out.print(i+" ");
        }
    }
}
```



```
    }  
}
```

**Q)Write a java program to display highest elements from given array?**

**input:**

5 2 8 1 9 6

**output:**

9

### approach1

```
import java.util.Arrays;  
  
class Test  
{  
  
    public static void main(String[] args)  
    {  
  
        int[] arr={5,2,8,1,9,6};  
  
        Arrays.sort(arr); // 1 2 5 6 8 9  
  
        System.out.println(arr[arr.length-1]);  
  
    }  
}
```



### approach2

```
class Test  
{  
  
    public static void main(String[] args)  
    {  
  
        int[] arr={5,2,8,1,9,6};  
  
    }  
}
```

```

int max=arr[0];

//reading the elements

for(int i=0;i<arr.length;i++)
{
    if(arr[i]>max)
    {
        max=arr[i];
    }
}
System.out.println(max);
}}
```

**Q)Write a java program to display least elements from given array?**

**input:**

5 2 8 1 9 6

**output:**

1



### approach1

```

import java.util.Arrays;

class Test

{
    public static void main(String[] args)
    {
        int[] arr={5,2,8,1,9,6};

        Arrays.sort(arr);//1 2 5 6 8 9

        System.out.println(arr[0]);//1
    }
}
```

```
}
```

**approach2**

```
class Test
{
    public static void main(String[] args)
    {
        int[] arr={5,2,8,1,9,6};

        int min=arr[0];

        //reading elements
        for(int i=0;i<arr.length;i++)
        {
            if(arr[i]<min)
            {
                min=arr[i];
            }
        }
        System.out.println(min);
    }
}
```



**Q)Write a java program to display three highest elements from given array?**

**input:**

5 2 8 1 9 6

**output:**

9 8 6

**ex:**

```
import java.util.Arrays;
```

```
class Test
```

```
{
```

```

public static void main(String[] args)
{
    int[] arr={5,2,8,1,9,6};

    Arrays.sort(arr);//1 2 5 6 8 9

    System.out.print(arr[arr.length-1]+" ");
    System.out.print(arr[arr.length-2]+" ");
    System.out.print(arr[arr.length-3]+" ");
}

```

**Q)Write a java program to display duplicate elements from given array?**

**input:**

3 5 1 2 1 9 6 5 7 3

**output:**

3 5 1

**ex:**

**class Test**

{

```
public static void main(String[] args)
```

{

int[] arr={3,5,1,2,1,9,6,5,7,3};

//duplicate elements

for(int i=0;i<arr.length;i++)

{

for(int j=i+1;j<arr.length;j++)

{

if(arr[i]==arr[j])

{

System.out.print(arr[i]+" ");

}}}

**Q)Write a java program to display unique/distinct elements from given array?**

**input:**

3 5 1 2 1 9 6 5 7 3

**output:**

2 9 6 7

**ex:**

**class Test**

{

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

{

**int[] arr={3,5,1,2,1,9,6,5,7,3};**

**//unique elements**

**for(int i=0;i<arr.length;i++)**

{

**int cnt=0;**

**for(int j=0;j<arr.length;j++)**

{

**if(arr[i]==arr[j])**

{

**cnt++;**

}

}

**if(cnt==1)**

**System.out.print(arr[i]+ " ");**

}}}



**Q)Write a java program to find out leader element from given array?**

**input:**

7 2 8 23 19 5 15

**output:**

**15 19 23**

```
class Test
{
    public static void main(String[] args)
    {
        int[] arr={7,2,8,23,19,5,15};

        int max=arr[arr.length-1];

        System.out.print(max+" ");

        for(int i=arr.length-2;i>=0;i--)
        {
            if(arr[i]>max)
            {
                max=arr[i];
                System.out.print(max+" ");
            }
        }
    }
}
```



**Q)Write a java program to find out triplet of an element from given sum value?**

**input:**

**arr = 7 2 8 1 6 5 4**

**sum = 10**

**output:**

**7 2 1**

**1 5 4**

**ex:**

```
class Test
```

```
{
```

```
public static void main(String[] args)
```

```
{
```

```

int[] arr ={7,2,8,1,6,5,4};

int sum = 10;

for(int i=0;i<arr.length;i++)
{
    for(int j=i+1;j<arr.length;j++)
    {
        for(int k=j+1;k<arr.length;k++)
        {
            if(arr[i]+arr[j]+arr[k]==sum)
            {
                System.out.println(arr[i]+" "+arr[j]+" "+arr[k]);
            }
        }
    }
}

```

Q) Write a java program to find out missing element from given array?

input:

3 2 1 5



output:

4

ex:

class Test

{

public static void main(String[] args)

{

int[] arr ={3,2,1,5};

int sum\_of\_arr\_ele= arr.length+1;

int sum=(sum\_of\_arr\_ele\*(sum\_of\_arr\_ele+1))/2;

```

        for(int i=0;i<arr.length;i++)
        {
            sum=sum-arr[i];
        }

        System.out.println("Missing element is =" +sum);
    }
}

```

**Q)Write a java program to display prime elements from given array?**

**input:**

5 10 12 15 17 3

**output:**

5 17 3

**ex:**

**class Test**

{

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

{

int[] arr={5,10,12,15,17,3};



**for(int i=0;i<arr.length;i++)**

{

**boolean flag=true;**

**for(int j=2;j<=arr[i]/2;j++)**

{

**if(arr[i]%j==0)**

{

**flag=false;**

**break;**

```

        }
    }

    if(flag==true)
        System.out.print(arr[i]+" ");
    }
}
}

```

**Q)Write a java program to concatinate two arrays and display them in sorting order?**

**input:**

arr1 = 5 1 3 4 2

arr2 = 9 6 7 8 10

**output:**

1 2 3 4 5 6 7 8 9 10

**ex:**

import java.util.Arrays;

class Test

{

public static void main(String[] args)

{

int[] arr1 = {5,1,3,4,2};

int[] arr2 = {9,6,7,8,10};

int size1=arr1.length;

int size2=arr2.length;

arr1=Arrays.copyOf(arr1,size1+size2);



```

int j=0;

for(int i=size1;i<arr1.length;i++)
{
    arr1[i]=arr2[j++];

}

//ascending order
Arrays.sort(arr1);

//displaying elements
for(int i:arr1)
{
    System.out.print(i+" ");
}

```

**Q)Write a java program remove first occurrence of a given element?**

**input:**

**arr = 5 4 2 1 6 4 9 7 4**

**element : 4**

**output:**

**5 2 1 6 4 9 7 4**

**ex:**

**class Test**

**{**

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

**{**

**int[] arr = {5,4,2,1,6,4,9,7,4};**

**int element= 4;**

```

int[] resArr=new int[arr.length-1];

int j=0 ,cnt=0;

for(int i=0;i<arr.length;i++)
{
    if(arr[i]==element && cnt==0)
    {
        cnt++;
        continue;
    }

    resArr[j++]=arr[i];
}

//displaying the elements
for(int i:resArr)
{
    System.out.print(i+" ");
}

```



**Q)Write a java program to segregate the array elements?**

**input:**

1 1 0 0 1 0 0 1 0 1

**output:**

0 0 0 0 0 1 1 1 1 1

**ex**

```

import java.util.Arrays;

class Test

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

```

```
int[] arr = {1,1,0,0,1,0,0,1,0,1};
```

```
Arrays.sort(arr);
```

```
for(int i:arr)
{
    System.out.print(i+" ");
}
}
```

**Q)Write a java program to insert the element on given index/position?**

**input:**

```
arr = 4 7 1 6 2 9
```

```
position = 3
```

```
element = 10
```

**output:**

```
4 7 1 10 6 2 9
```

**ex:**



```
import java.util.Arrays;
class Test
{
    public static void main(String[] args)
    {
        int[] arr = {4,7,1,6,2,9};
        int position = 3;
        int element = 10;
        arr=Arrays.copyOf(arr,arr.length+1);

        for(int i=arr.length-1;i>=position;i--)
        {
            arr[i]=arr[i-1];
        }
    }
}
```

```

    }

    arr[position]=element;

    //display elements
    for(int i:arr)
    {
        System.out.print(i+" ");
    }
}

```

**Q)Write a java program to check array contains odd or not?**

**input:**

arr = 3 7 9 1 13 5

**output:**

Array contains odd

**ex:**

class Test

{

public static void main(String[] args)

{

int[] arr={3,7,9,1,13,5};

boolean flag=true;

for(int i:arr)

{

if(i%2==0)

{

flag=false;

break;

}

}



```

        if(flag==true)
            System.out.println("Array contains odd");
        else
            System.out.println("Array does not contains odd");
    }
}

```

## Double Dimensional Array

Double dimensional array is a combination of rows and columns.

The main objective of two dimensional array is just for instance use.

We can use double dimensional array when we want to create matrix type of applications, business oriented applications, gaming applications and etc.

We can declare double dimensional array as follow.

syntax:

datatype[][] variable\_name=new int[rows][cols];

ex:

int[][] arr=new int[3][3];

Here we can store 9 elements.

**Q)Write a java program to accept some array elements and display them in matrix form?**

ex:

```

import java.util.Scanner;

class Test
{
    public static void main(String[] args)
    {
        Scanner sc=new Scanner(System.in);

        System.out.println("Enter the Rows :");
    }
}

```

```

int rows=sc.nextInt();

System.out.println("Enter the columns :");
int cols=sc.nextInt();

int[][] arr=new int[rows][cols];

//insert elements
for(int i=0;i<arr.length;i++)
{
    for(int j=0;j<arr.length;j++)
    {
        System.out.println("Enter the element :");
        arr[i][j]=sc.nextInt();
    }
}

//display elements
for(int i=0;i<arr.length;i++)
{
    for(int j=0;j<arr.length;j++)
    {
        System.out.print(arr[i][j]+" ");
    }
    //new line
    System.out.println("");
}
}

```



**Q)Write a java program to display square of a matrix ?**

```

import java.util.Scanner;
class Test

```

```
{  
  
    public static void main(String[] args)  
    {  
  
        Scanner sc=new Scanner(System.in);  
  
        System.out.println("Enter the Rows :");  
        int rows=sc.nextInt();  
  
        System.out.println("Enter the columns :");  
        int cols=sc.nextInt();  
  
        int[][] arr=new int[rows][cols];  
  
        //insert elements  
        for(int i=0;i<arr.length;i++)  
        {  
            for(int j=0;j<arr.length;j++)  
            {  
                System.out.println("Enter the element :");  
                arr[i][j]=sc.nextInt();  
            }  
        }  
  
        //display elements  
        for(int i=0;i<arr.length;i++)  
        {  
            for(int j=0;j<arr.length;j++)  
            {  
                System.out.print(arr[i][j] * arr[i][j] + " ");  
            }  
        }  
    }  
}
```



```

        //new line
        System.out.println("");
    }

}

```

**Q)Write a java program to perform sum of diagonal elements?**

```

import java.util.Scanner;
class Test
{
    public static void main(String[] args)
    {
        Scanner sc=new Scanner(System.in);
        System.out.println("Enter the Rows :");
        int rows=sc.nextInt();

        System.out.println("Enter the columns :");
        int cols=sc.nextInt();

        int[][] arr=new int[rows][cols];

        //insert elements
        for(int i=0;i<arr.length;i++)
        {
            for(int j=0;j<arr.length;j++)
            {

```



```

        System.out.println("Enter the element :");

        arr[i][j]=sc.nextInt();

    }

}

//display elements

int sum=0;

for(int i=0;i<arr.length;i++)

{

    for(int j=0;j<arr.length;j++)

    {

        if(i==j)

        {

            sum=sum+arr[i][j];

        }

    }

}

System.out.println("sum of diagonal elements is =" +sum);

}

```



**Q) Write a java program to perform sum of upper triangle elements?**

```

import java.util.Scanner;

class Test

{

    public static void main(String[] args)

    {

        Scanner sc=new Scanner(System.in);

        System.out.println("Enter the Rows :");

        int rows=sc.nextInt();
    }
}

```

```

System.out.println("Enter the columns :");

int cols=sc.nextInt();

int[][] arr=new int[rows][cols];

//insert elements

for(int i=0;i<arr.length;i++)

{

    for(int j=0;j<arr.length;j++)

    {

        System.out.println("Enter the element :");

        arr[i][j]=sc.nextInt();

    }

}

//display elements

int sum=0;

for(int i=0;i<arr.length;i++)

{

    for(int j=0;j<arr.length;j++)

    {

        if(i<j)

        {

            sum=sum+arr[i][j];

        }

    }

}

System.out.println("sum of upper triangle elements is =" +sum);

}

```



```
}
```

**Q)Write a java program to perform lower triangle elements?**

```
import java.util.Scanner;  
  
class Test  
{  
  
    public static void main(String[] args)  
    {  
  
        Scanner sc=new Scanner(System.in);  
  
        System.out.println("Enter the Rows :");  
        int rows=sc.nextInt();  
  
        System.out.println("Enter the columns :");  
        int cols=sc.nextInt();  
  
        int[][] arr=new int[rows][cols];  
  
        //insert elements  
        for(int i=0;i<arr.length;i++)  
        {  
            for(int j=0;j<arr.length;j++)  
            {  
                System.out.println("Enter the element :");  
                arr[i][j]=sc.nextInt();  
            }  
        }  
  
        //display elements  
        int sum=0;  
        for(int i=0;i<arr.length;i++)
```



```

    {
        for(int j=0;j<arr.length;j++)
        {
            if(i>j)
            {
                sum=sum+arr[i][j];
            }
        }
        System.out.println("sum of lower triangle elements is =" +sum);
    }
}

```

### Multi-Dimensional Array

If array contains three dimensions is called multi-dimensional array.

We can declare three dimensional array as follow.

ex:

```
int[][][] arr=new int[3][3][3];
```

Here we can store 27 elements.

ex:

```
import java.util.Scanner;
```

```
class Test
```

```
{
```

```
public static void main(String[] args)
```

```
{
```

```
int[][][] arr=new int[3][3][3];
```

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

```
//insert elements
```

```
for(int i=0;i<arr.length;i++)
```

```

    {
        for(int j=0;j<arr.length;j++)
        {
            for(int k=0;k<arr.length;k++)
            {
                System.out.println("Enter the element :");
                arr[i][j][k]=sc.nextInt();
            }
        }
    }

//display elements
for(int i=0;i<arr.length;i++)
{
    for(int j=0;j<arr.length;j++)
    {
        for(int k=0;k<arr.length;k++)
        {
            System.out.print(arr[i][j][k]+" ");
        }
    }
    //new line
    System.out.println("");
}
}

```

## **Oops:**

**OOPS stands for Object Oriented Programming System or Structure.**

## Object oriented technology:

A technology which provides very good environment to represent the data in the form of objects is called object oriented technology.

A technology is said to be object oriented if it supports following features.

ex:

class  
object  
Abstraction  
Encapsulation  
Inheritance  
and  
polymorphism

### class:

- A class is a collection of data members and behaviours.
- Here data members means variables or fields or properties.
- Here behaviours means methods or actions or characteristics.
- In general , a class is a collection of variables and methods.
- A class is a blue print of an object.

We can declare a class as follow.

### syntax:

optional  
|  
Modifier class class\_name <extends> Parent\_classname  
                  <implements> Interface\_name  
{

```
- //variables and methods  
-  
}
```

A class will accept following modifiers.

ex:

**default**  
**public**  
**final**  
**abstract**

### Interview Question :

Q)What is final class?

If we declare any class as final then creating child class is not possible.



Q)What is abstract class?

If we declare any class as abstract then creating object for that class is not possible.

Q)What is singleton class?

A class which allows us to create only one object is called singleton class.

### object

- ✓ Object is a instance of a class.
- ✓ Here instance means allocating memory for our data members.
- ✓ Object is a outcome of a blue print.
- ✓ Memory space will be created when we create an object.
- ✓ We can create an object as follow.

syntax:

```
Class_Name Reference_variable=new Constructor();
```

**ex:**

```
Test t=new Test();
```

**It is possible to create more then one object in a single class.**

**ex:**

```
class Test
```

```
{
```

```
public static void main(String[] args)
```

```
{
```

```
Test t1=new Test();
```

```
Test t2=new Test();
```

```
Test t3=new Test();
```

```
System.out.println(t1.hashCode());
```

```
System.out.println(t2.hashCode());
```

```
System.out.println(t3.hashCode());
```

```
System.out.println(t1.toString()); //Test@Hexadecimalno
```

```
System.out.println(t2);
```

```
System.out.println(t3);
```

```
}
```

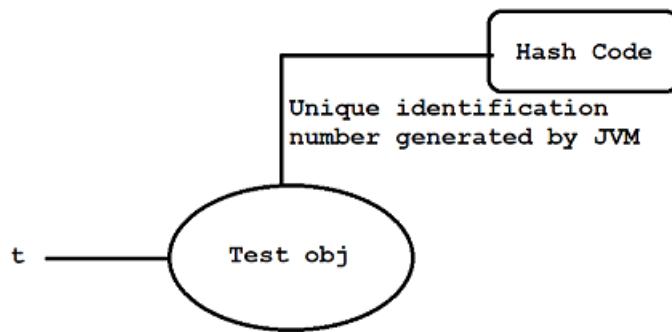
```
}
```

## **hashCode()**

- ✓ It is a method present in Object class.
- ✓ For every object, jvm will create a unique identification number i.e hash code.
- ✓ In order to read hash code we need to use hashCode() method.

**Diagram: java28.1**

```
Test t=new Test();
```



### toString():

It is a method present in Object class.

Whenever we trying to display any object reference, directly or indirectly `toString()` method will be executed.

### Interview Question

Q) What is Object class?



A Object class present in `java.lang` package.

It is a parent class for every java class.

It contains following methods.

ex:

`hashCode()`

`toString()`

`getClass()`

`wait()`

`notify()`

`notifyAll()`

`equals()`

and etc.

## Data Hiding:

- ✓ Our internal data should not go out directly.
- ✓ It means outside perform must not access our data directly such concept is called data hiding.
- ✓ The main object of data hiding is to provide security.
- ✓ By using private modifier we can achieve data hiding concept.

ex:

```
class Account  
{  
    private double balance;  
  
    -  
    -  
    -  
  
}
```

## Abstraction:



- ✓ Hiding the internal implementation and highlighting the set of services is called abstraction.
- ✓ The best example of Abstraction is GUI(Graphical User Interface) ATM machine. Where bank people will hide internal implementation and highlights the set of services like banking, withdrawl, ministatement and etc.

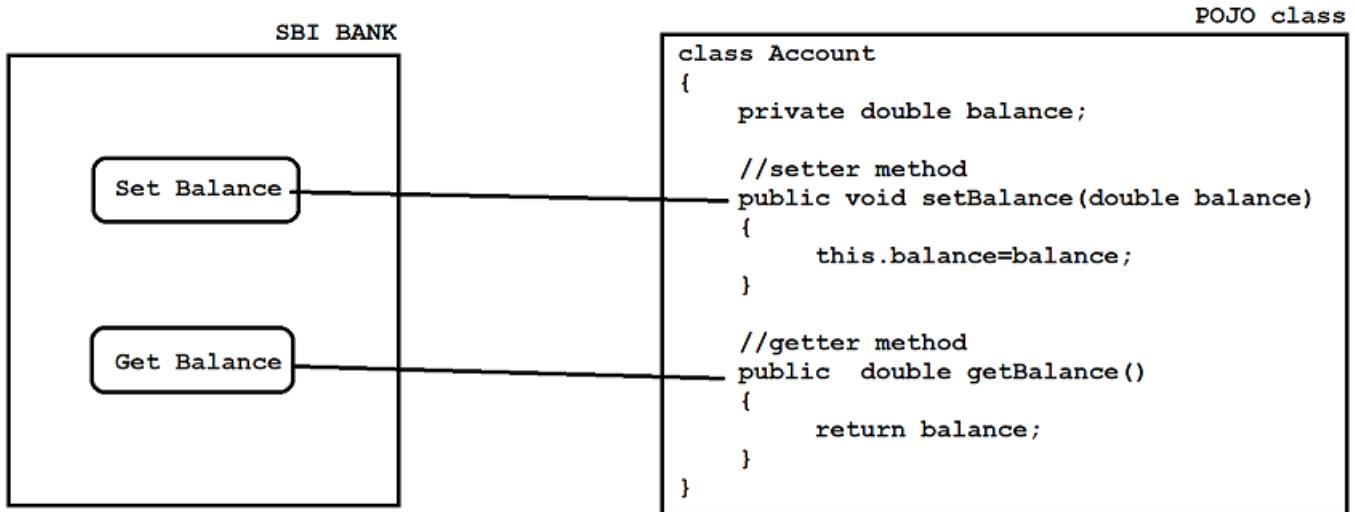
## The main advantages of abstraction are.

- 1)It gives security because it will hide internal data from the outsider.
- 2)Enhancement becomes more easy because without effecting enduser they can perform any changes in our internal system.
- 3)It provides flexibility to the enduser to use the system.
- 4)It improves maintainability of an application.

## Encapsulation:

- ✓ The process of encapsulating or grouping variables and its associate methods in a single entity is called encapsulation.
- ✓ A class is said to be encapsulated class if it supports data hiding and abstraction.
- ✓ In encapsulation for every variable we need to write setter and getter method.

Diagram: java28.2



The main advantages of encapsulation are.

- ✓ It gives security.
- ✓ Enhancements becomes more easy.
- ✓ It provides flexibility to the enduser to use the system.
- ✓ It improves maintainability of an application.

The main disadvantage of encapsulation is it will increase the length of our code and slow down the execution process.

ex:

```

class Student
{
    //instance variables
    private int studId;
    private String studName;
    private double studFee;

    //setter methods
  
```

```
public void setStudId(int studId)
{
    this.studId=studId;
}

public void setStudName(String studName)
{
    this.studName=studName;
}

public void setStudFee(double studFee)
{
    this.studFee=studFee;
}

//getter methods
public int getStudId()
{
    return studId;
}

public String getStudName()
{
    return studName;
}

public double getStudFee()
{
    return studFee;
}

public static void main(String[] args)
```



```

{
    Student s=new Student();
    s.setStudId(101);
    s.setStudName("Alan");
    s.setStudFee(10000d);
    System.out.println("Student Id :" + s.getStudId());
    System.out.println("Student Name :" + s.getStudName());
    System.out.println("Student Fee :" + s.getStudFee());
}

```

### approach2

```

class Student
{
    //instance variables
    private int studId;
    private String studName;
    private double studFee;
}

```



```

//setter methods
public void setStudId(int studId)
{
    this.studId=studId;
}

public void setStudName(String studName)
{
    this.studName=studName;
}

```

```
public void setStudFee(double studFee)
{
    this.studFee=studFee;
}

//getter methods

public int getStudId()
{
    return studId;
}

public String getStudName()
{
    return studName;
}

public double getStudFee()
{
    return studFee;
}

}

class Test
{
    public static void main(String[] args)
    {
        Student s=new Student();
        s.setStudId(101);
        s.setStudName("Alan");
    }
}
```



```

        s.setStudFee(10000d);

        System.out.println("Student Id :" + s.getStudId());

        System.out.println("Student Name :" + s.getStudName());

        System.out.println("Student Fee :" + s.getStudFee());

    }

}

```

**Q)What is the difference between POJO class and Java Bean class?**

#### POJO class

A class is said to be pojo class if it supports following two properties.

- 1) All variables must be private.
- 2) All variables must have setter and getter methods.

#### Java Bean class

A class is said to be java bean class if it supports following four properties.

- 1)A class should be public.
- 2)A class should have atleast zero argument constructor.
- 3)All variables must be private.
- 4)All variables must have setter and getter methods.

#### Note:

Every java bean class is a pojo class.But every pojo class is not a java bean class.

#### Is-A relationship:

- ✓ Is-A relationship is also known as inheritance.
- ✓ Using extends keyword we can implements Is-A relationship.
- ✓ The main objective of Is-A relationship is to achieve reusability.

ex:

```

class Parent
{

```

```
public void m1()
{
    System.out.println("Parent-M1 Method");
}

}

class Child extends Parent

{
    public void m2()
    {
        System.out.println("Child-M2 Method");
    }
}

class Test
{
    public static void main(String[] args)
    {
        Parent p=new Parent();
        p.m1();

        Child c=new Child();
        c.m1();
        c.m2();

        Parent p1=new Child();
        p1.m1();

        //Child c1=new Parent(); //invalid
    }
}
```



```
    }  
}
```

### **conclusion:**

Whatever our parent contains properties by default it goes to child. But whatever a child contains properties ,it never goes back to parent.

A parent reference can hold child object. But child reference can't hold parent object.

### **Inheritance:**

Inheritance is a mechanism where we declare a new class in the presence of existing class.

In general, A parent and child relationship is called inheritance.

We have five types of inheritance.

- 1) Single Level Inheritance
- 2) Multi Level inheritance
- 3) Multiple inheritance
- 4) Hierarchical inheritance
- 5) Hybrid inheritance



### **1) Single Level Inheritance:**

If a class is derived from one base class is called single level inheritance.

ex:

A (Parent class / Super class / Base class)

|

|

|

B (Child class / Sub class / Derived class)

**ex:**

---

**class A**

{

**public void m1()**  
    {  
        **System.out.println("M1-Method");**  
    }

}

**class B extends A**

{

**public void m2()**  
    {  
        **System.out.println("M2-Method");**  
    }

}

**class Test**

{

**public static void main(String[] args)**  
    {  
        **A a=new A();**  
        **a.m1();**

**B b=new B();**  
        **b.m1();**  
        **b.m2();**

```
    }  
}  
}
```

## **2)Multi Level inheritance:**

If a class is derived from one base class and that class is derived from another base class is called multilevel inheritance.

ex:

```
A  
|  
|  
|  
|  
B  
|  
|  
|  
|  
C
```



ex:

```
class A  
{  
    public void m1()  
    {  
        System.out.println("M1-Method");  
    }  
}  
  
class B extends A
```

```
{  
    public void m2()  
    {  
        System.out.println("M2-Method");  
    }  
}  
  
class C extends B  
{  
    public void m3()  
    {  
        System.out.println("M3-Method");  
    }  
}  
  
class Test  
{  
    public static void main(String[] args)  
    {  
        A a=new A();  
        a.m1();  
  
        B b=new B();  
        b.m1();  
        b.m2();  
  
        C c=new C();  
        c.m1();  
        c.m2();  
    }  
}
```



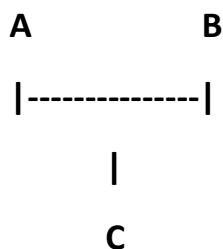
```
c.m3();
```

```
}
```

### **3)multiple inheritance:**

If a class is derived from more then one base class is called multiple inheritance.

ex:



In java, we can't extends more then one class simultaneously because java does not support multiple inheritance.



ex:

```
class A  
{  
}  
  
class B  
{  
}  
  
class C extends A,B  --->invalid  
{  
}
```

But interface can extends more then one interface so we can achieve multiple inheritance concept through interfaces.

ex:

```
interface A  
{  
}  
  
interface B  
{  
}  
  
interface C extends A,B  
{  
}
```

If our class does not extends any other class then our class is a direct child class of Object class.

ex:

Diag:

```
class A  
{  
}  
  
}  
A
```



If our class extends some other class then our class is a indirect child class of Object class.

ex:

Diag:

```
class A  
{  
}  
  
class B extends A  
{  
}
```

Object

|

|

A // multi-level inheritance

|

```
 }  
 |  
 B
```

**Java does not support cyclic inheritance.**

ex:

```
class A extends B  
{  
}  
  
class B extends A  
{  
}
```

**Q) Why java does not support multiple inheritance?**

There is a chance of raising ambiguity problem that's why java does not support multiple inheritance.

ex:

```
P1.m1()          p2.m1()  
|-----|  
 |  
 C.m1()
```



**4) Hierarchical inheritance:**

If multiple classes are derived from one base class is called hierarchical inheritance.

ex:

```
A  
|  
|-----|  
B      C
```

ex:

```
class A
{
    public void m1()
    {
        System.out.println("M1-Method");
    }
}

class B extends A
{
    public void m2()
    {
        System.out.println("M2-Method");
    }
}

class C extends A
{
    public void m3()
    {
        System.out.println("M3-Method");
    }
}

class Test
{
    public static void main(String[] args)
    {
```



```
A a=new A();  
a.m1();  
  
B b=new B();  
b.m1();  
b.m2();  
  
C c=new C();  
c.m1();  
c.m3();  
  
}  
}
```

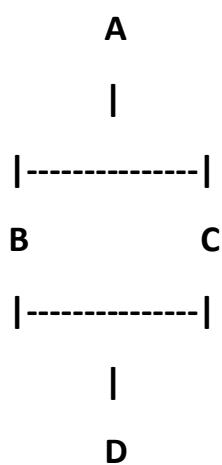
### 5) Hybrid inheritance:



Hybrid inheritance is a combination of more than one inheritance.

Java does not support hybrid inheritance.

ex:



### Has-A relationship:

- ✓ Has-A relationship is also known as composition and aggregation.
- ✓ There is no specific keyword to implement Has-A relationship but mostly we will use new operator.
- ✓ The main objective of Has-A relationship is to provide reusability.
- ✓ Has-A relationship will increase dependency between two components.

ex:

```
class Engine
{
    -
    - //engine specific functionality
    -
}

class Car
{
    Engine e=new Engine();
    -
}
```



ex:

```
class IhubTalent
{
    public String courseName()
    {
        return "Full Stack Java Course";
    }

    public double courseFee()
    {
        return 25000d;
```

```

    }

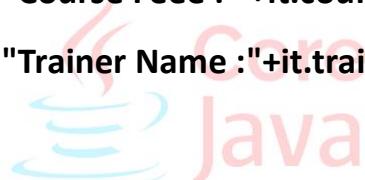
    public String trainerName()
    {
        return "Niyaz Sir";
    }
}

class Usha
{
    public void courseDetails()
    {
        IhubTalent it=new IhubTalent();

        System.out.println("Course Name : "+it.courseName());
        System.out.println("Course Fee : "+it.courseFee());
        System.out.println("Trainer Name : "+it.trainerName());
    }
}

class Student
{
    public static void main(String[] args)
    {
        Usha u=new Usha();
        u.courseDetails();
    }
}

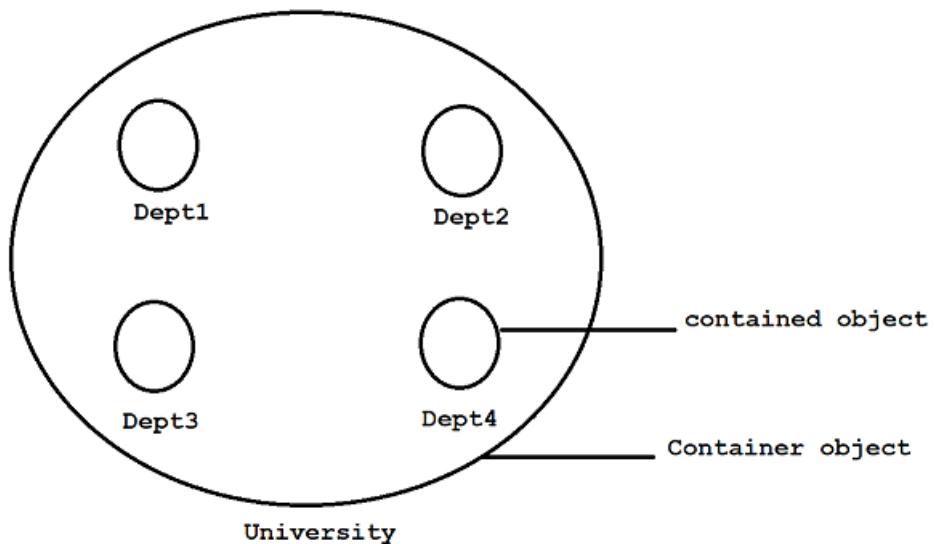
```



## Composition:

Without existing container object there is a no chance of having contained object then the relationship between container and contained object is called composition which is strongly association.

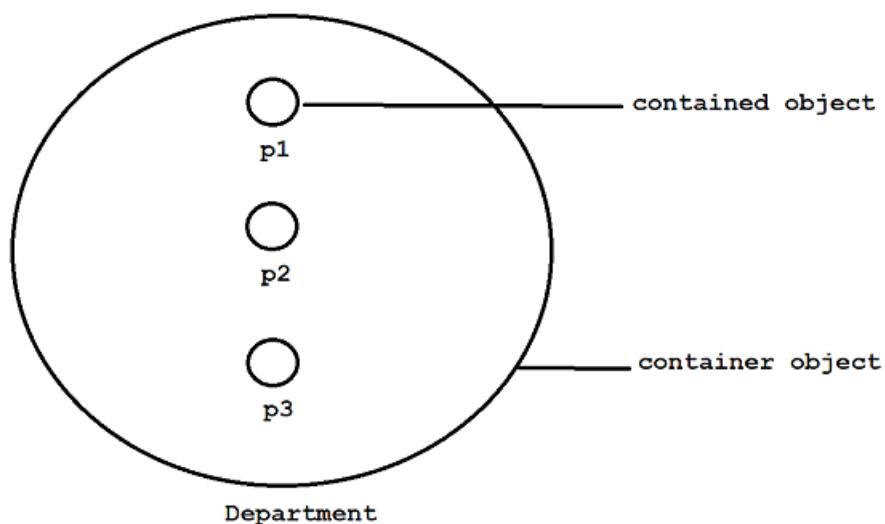
Diagram: java29.1



## Aggregation:

Without existing container object there is a chance of having contained object then the relationship between container and contained object is called aggregation which is loosely association.

Diagram: java29.2



## Method Overloading

- ✓ Having same method name with different parameters in a single class is called method overloading.
- ✓ All the methods present in a class are called overloaded methods.
- ✓ Method overloading will reduce complexity of the programming.

ex:

```

class A
{
    //overloaded methods

    public void m1()
    {
        System.out.println("0-arg method");
    }

    public void m1(int i)
    {
        System.out.println("int-arg method");
    }

    public void m1(double d)
    {
        System.out.println("double-arg method");
    }
}

class Test
{
    public static void main(String[] args)
    {
        A a=new A();
        a.m1();
        a.m1(10);
    }
}

```



```
a.m1(10.5d);  
}  
}
```

## **Method overriding:**

**Having same method name with same parameters in a two different classes is called method overriding.**

**Methods which are present in a parent class are called overridden methods.**

**Methods which are present in a child class are called overriding methods.**

**ex:**

```
class Parent  
{  
    public void property()  
    {  
        System.out.println("Cash+Gold+Land");  
    }  
    //overridden method  
    public void marry()  
    {  
        System.out.println("subhalakshmi");  
    }  
}  
  
class Child extends Parent  
{  
    //overriding method  
    public void marry()  
    {
```



```

        System.out.println("Rashmika");

    }

}

class Test

{

    public static void main(String[] args)

    {

        Parent p=new Parent();

        p.property(); //Cash+Gold+Land

        p.marry(); //subhalakshmi


        Child c=new Child();

        c.property(); // Cash+Gold+Land

        c.marry(); // Rashmika

        Parent p1=new Child();

        p1.property(); //Cash+Gold+Land

        p1.marry(); //Rashmika

    }

}

```

**Note:**

If we declare any method as final then overriding of that method is not possible.

ex:

```

class Parent

{

    public void property()

{

```

```
        System.out.println("Cash+Gold+Land");

    }

//overridden method

public final void marry()

{

    System.out.println("subhalakshmi");

}

}

class Child extends Parent

{

//overriding method

public final void marry()

{

    System.out.println("Rashmika");

}

}

class Test

{

public static void main(String[] args)

{

    Parent p=new Parent();

    p.property(); //Cash+Gold+Land

    p.marry(); //subhalakshmi


    Child c=new Child();

    c.property(); // Cash+Gold+Land

    c.marry(); // Rashmika
```



```

        Parent p1=new Child();

        p1.property(); //Cash+Gold+land

        p1.marry(); //Rashmika

    }

}

```

**Q)Write a java program for below code?**

```

1      1
1 2    2 1
1 2 3  3 2 1
1 2 3 4 4 3 2 1

```

ex:

```

public class Test {

    public static void main(String[] args) {
        int rows = 4;
    }
}
```



```

for (int i = 1; i <= rows; i++) {
    {
        // Print numbers in ascending order
        for (int j = 1; j <= i; j++) {
            System.out.print(j + " ");
        }
    }

    // Print spaces
    for (int j = 1; j <= (rows - i) * 2; j++) {
        System.out.print(" ");
    }
}
```

```

}

// Print numbers in descending order

for (int j = i; j >= 1; j--) {
    System.out.print(j + " ");
}

System.out.println();
}
}
}

```

### Method Hiding:

Method Hiding is exactly the same as method overriding with following differences.



Method Overriding	Method Hiding
Methods present in method overriding must be non-static.	Methods present in method hiding must be static.
Method resolution will be taken care by JVM based on runtime object.	Method resolution will be taken care by compiler based on reference type.
It is also known as Dynamic	It is also known as static
polymorphism, Runtime polymorphism or late binding.	polymorphism, compile time polymorphism or early binding.

ex:

```

class Parent
{
    public static void property()
    {

```

```

        System.out.println("Cash+Gold+Land");

    }

    public static void marry()

    {

        System.out.println("Subhalakshmi");

    }

}

class Child extends Parent

{

    public static void marry()

    {

        System.out.println("Rashmika");

    }

}

class Test

{

    public static void main(String[] args)

    {

        Parent p=new Parent();

        p.property(); //cash+gold+land

        p.marry(); // subhalakshmi


        Child c=new Child();

        c.property(); // cash+gold+land

        c.marry(); // Rashmika


        Parent p1=new Child();
    }
}

```



```
    p1.property(); // cash+gold+land  
  
    p1.marry(); // Subhalakshmi  
  
}  
}
```

### Q)Can we override static methods in java?

No, we can't override static methods in java.

### Q)Can we overload main method in java?

Yes, we can overload main method in java. But JVM will execute main method with String[] argument only.

ex:

```
class Test  
{  
    public static void main(String[] args)  
    {  
        System.out.println("string-arg method");  
    }  
    public static void main(int[] iargs)  
    {  
        System.out.println("int-arg method");  
    }  
}
```



### Polymorphism:

- ✓ Polymorphism has taken from Greek Word.
- ✓ Here poly means many and morphism means forms.
- ✓ The ability to represent in a different forms is called polymorphism.

The main objective of polymorphism is to provide flexibility.

We have two types of polymorphism.

1)Compile time polymorphism

2)Runtime polymorphism

**1)Compile time polymorphism:**

A polymorphism which exhibits at compile time is called compile time polymorphism.

ex:

Method overloading

Method Hiding

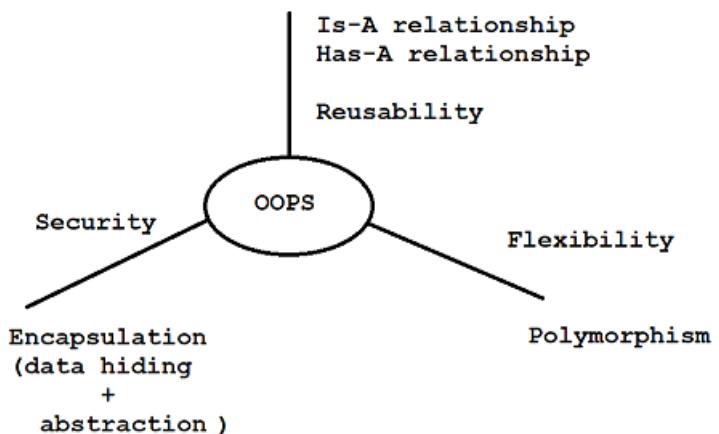
**2)Runtime polymorphism:**

A polymorphism which exhibits at runtime is called runtime polymorphism.

ex:

Method overriding.

Summary Diagram : java30.1



**Constructor:**

A constructor is a special type of method which is used to initialize an object.

ex:

```
Test t=new Test();
```

Having same name as class name is called constructor.

Constructor will execute at the time of object creation only.

**Constructor does not allowed any return type , if we take we won't get any compile time error or runtime error.**

**A constructor will accept following modifiers.**

**ex:**

**default**

**public**

**private**

**protected**

**In java , constructors are divided into two types.**

**1) Userdefined constructor**

**2) Default constructor**

### **1) Userdefined constructor:**



**A constructor which is created by the user based on application requirement is called user defined constructor.**

**We have two types of userdefined constructors.**

**i) Zero-Argument constructor**

**ii) Parameterized constructor**

### **i) Zero-Argument constructor**

**Suppose if we are not passing any argument to userdefined constructor is called zero -argument constructor.**

**ex:**

```
class Test
```

```
{
```

```
    Test()
```

```
{
```

```
    System.out.println("constructor");
```

```
}
```

```
public static void main(String[] args)
{
    System.out.println("Main-Method");
}
```

```
}
```

**o/p:**

**Main-Method**

**ex:**

```
class Test
```

```
{
```

```
Test()
{
```



```
    System.out.println("constructor");
}
```

```
public static void main(String[] args)
```

```
{
```

```
    System.out.println("Main-Method");

```

```
    Test t=new Test();

```

```
}
```

**o/p:**

**Main-Method**

**constructor**

**ex:**

**class Test**

{

**public Test()**

{

**System.out.println("constructor");**

}

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

{

**Test t1=new Test();**

**System.out.println("Main-Method");**

**Test t2=new Test();**

}



}

**o/p:**

**constructor**

**Main-Method**

**constructor**

## **ii)Parameterized constructor :**

**Suppose if we are passing atleast one argument to userdefined constructor is called parameterized constructor.**

**ex:**

**class Employee**

{

```
//instance variables

private int empId;

private String empName;

private double empSal;

public Employee(int empId,String empName,double empSal)

{

    this.empId=empId;

    this.empName=empName;

    this.empSal=empSal;

}

public void getEmployeeDetails()

{

    System.out.println("Employee Id :" +empId);

    System.out.println("Employee Name :" +empName);

    System.out.println("Employee Salary :" +empSal);

}

}

class Test

{

    public static void main(String[] args)

    {

        Employee e=new Employee(201,"Alan Morries",10000d);

        e.getEmployeeDetails();

    }

}
```

## 2) Default constructor

It is a compiler generated constructor for every java program where we are not declaring any constructor.

We can see default constructor by using below command.

ex:

```
javap -c Test
```

Diagram: java30.2

```
class Test
{
    public static void main(String[] args)
    {
        System.out.println("IHUB TALENT");
    }
}

class Test
{
    //default constructor
    Test();
    public static void main(String[] args)
    {
        System.out.println("IHUB TALENT");
    }
}
```

Default constructor

## Constructor overloading:

Having same constructor name with different parameters in a single class is called constructor overloading.

ex:

class A

```
{  
    A()  
    {  
        System.out.println("0-arg const");  
    }  
}
```

```

A(int i)
{
    System.out.println("int-arg const");
}

A(float f)
{
    System.out.println("float-arg const");
}

}

class Test
{
    public static void main(String[] args)
    {
        A a1=new A();
        A a2=new A(10);
        A a3=new A(10.5f);
    }
}

```



## this keyword

A "this" keyword is a java keyword which is used to represent current class object reference

We can utilize this keyword in following ways.

- 1) To refer current class variables
- 2) To refer current class methods
- 3) To refer current class constructors

### 1) To refer current class variables

class A

```

{
    int i=10;
    int j=20;

    A(int i,int j)
    {
        System.out.println(i+" "+j); // 100 200
        System.out.println(this.i+" "+this.j);//10 20
    }
}

class Test
{
    public static void main(String[] args)
    {
        A a=new A(100,200);
    }
}

```



## 2)To refer current class method

```

class A
{
    public void m1()
    {
        System.out.println("m1-method");
        this.m2();
    }

    public void m2()
    {

```

```
        System.out.println("m2-method");

    }

}

class Test

{

    public static void main(String[] args)

    {

        A a=new A();

        a.m1();

    }

}
```

### 3)To refer current class constructor

```
class A

{

    A()

    {

        System.out.println("0-arg const");

    }

    A(int i)

    {

        this();

        System.out.println("int-arg const");

    }

    A(double d)

    {

        this(10);

        System.out.println("double-arg const");

    }

}
```



```

    }
}

class Test
{
    public static void main(String[] args)
    {
        A a=new A(10.5d);
    }
}

```

## **super keyword:**

A "super" keyword is a java keyword which is used to refer super class object reference

We can utilize super keyword in following ways

- 1) To refer super class variables
- 2) To refer super class methods
- 3) To refer super class constructors

### **1) To refer super class variables**

```

class A
{
    int i=1;
    int j=2;
}

class B extends A
{
    int i=10;
    int j=20;
    B(int i,int j)

```



```

{
    System.out.println(this.i+" "+this.j); // 10 20
    System.out.println(super.i+" "+super.j); //1 2
    System.out.println(i+" "+j); // 100 200
}
}

class Test
{
    public static void main(String[] args)
    {
        B b=new B(100,200);
    }
}

```

## 2)To refer super class methods

```

class A
{
    public void m1()
    {
        System.out.println("m1-method");
    }
}

class B extends A
{
    public void m2()
    {
        super.m1();
        System.out.println("m2-method");
    }
}

```



```
    }
}

class Test
{
    public static void main(String[] args)
    {
        B b=new B();
        b.m2();
    }
}
```

### 3)To refer super class constructor

```
class A
{
    A()
    {
        System.out.println("A-constructor");
    }
}

class B extends A
{
    B()
    {
        super();
        System.out.println("B-constructor");
    }
}

class Test
```



```
{  
    public static void main(String[] args)  
    {  
        B b=new B();  
    }  
}
```

## Interface

Interface is a collection of zero or more abstract methods.

Abstract methods are incomplete methods , they ends with semicolon and do not have any body.

ex:

```
void m1();
```

It is not possible to create object for interfaces.

To implement abstract methods of an interface we need to use implementation class.

It is possible to create object for implementation class because it contains method with body.

By default every abstract method is a public and abstract.

Interface contains only constants i.e public static final.

syntax:

```
interface <interface_name>  
{  
    -  
    - //constants  
    - //abstract methods  
    -  
}
```

ex:

```
interface A
{
    //abstract method
    public abstract void m1();
}

class B implements A
{
    public void m1()
    {
        System.out.println("M1-Method");
    }
}

class Test
{
    public static void main(String[] args)
    {
        A a=new B();
        a.m1();
    }
}

ex:2

interface A
{
    //abstract method
    public abstract void m1();
}
```



```

class Test
{
    public static void main(String[] args)
    {
        //anonymous inner class
        A a=new A()
        {
            public void m1()
            {
                System.out.println("M1-Method");
            }
        };
        a.m1();
    }
}

```



If interface contains four methods then we need to write implementation/override all methods

ex:

**interface A**

```

{
    void show();
    public void display();
    abstract void view();
    public abstract void see();
}

```

```
class B implements A

{
    public void show()
    {
        System.out.println("show-method");
    }

    public void display()
    {
        System.out.println("display-method");
    }

    public void view()
    {
        System.out.println("view-method");
    }

    public void see()
    {
        System.out.println("see-method");
    }
}
```



```
class Test

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

        A a=new B();
        a.show();
    }
}
```

```
    a.display();

    a.view();

    a.see();

}

}
```

**In java, a class can't extends more then one class simultenously.**

**But interface can extends more then one interface simultenously.**

**ex:**

**interface A**

```
{  
    void m1();  
}
```

**interface B**

```
{  
    void m2();  
}
```

**interface C extends A,B**

```
{  
    void m3();  
}
```

**class D implements C**

```
{  
    public void m1()  
    {  
        System.out.println("m1-method");  
    }  
    public void m2()
```



```
{  
    System.out.println("m2-method");  
}  
  
public void m3()  
{  
    System.out.println("m3-method");  
}  
}  
  
class Test  
{  
    public static void main(String[] args)  
    {  
        C c=new D();  
        c.m1();  
        c.m2();  
        c.m3();  
    }  
}
```



**A class can implements more then one interface.**

**ex:**

```
interface Father  
{  
    float HT=6.2f;  
    void height();
```

```

}

interface Mother

{
    float HT=5.8f;

    void height();
}

class Child implements Father,Mother
{
    public void height()
    {
        float height=(Father.HT+Mother.HT)/2;

        System.out.println("Child Height is =" +height);
    }
}

```



```

class Test
{
    public static void main(String[] args)
    {
        Child c=new Child();

        c.height();
    }
}
```

### Java 1.7 version

Interface is a collection of zero or more abstract methods.

### Java 1.8 version

Interface is a collection of abstract methods ,default methods and static methods.

### java 1.9 version

Interface is a collection of abstract methods, default methods, static methods and

**private methods.**

## **Abstract class**

- ✓ It is a collection of zero or more abstract methods and concrete methods.
  - ✓ A abstract keyword is applicable for class and method but not for variable.
  - ✓ It is not possible to create object for abstract class.
- 
- ✓ To write the implementation of abstract method of an abstract class we will use sub classes.

**By default every abstract method is a public and abstract.**

**Abstract class contains only instance variables.**

**syntax:**

```
abstract class <class_name>
```

```
{
```

```
-  
-//instance variables  
-//abstract methods  
-//concrete methods
```

```
-
```

```
}
```



**ex:**

```
abstract class Plan
```

```
{
```

```
protected double rate;
```

```
//abstract method
```

```
public abstract void getRate();
```

```
//concrete method

public void calculateBillAmt(int units)

{
    System.out.println("Total Units : "+units);
    System.out.println("Total Bill : "+rate*units);
}

class DomesticPlan extends Plan

{
    public void getRate()

    {
        rate=2.5d;
    }
}

class CommercialPlan extends Plan

{
    public void getRate()

    {
        rate=5.0d;
    }
}

class Test

{
    public static void main(String[] args)

    {
        DomesticPlan dp=new DomesticPlan();
        dp.getRate();
    }
}
```



```

        dp.calculateBillAmt(250);

        CommercialPlan cp=new CommercialPlan();

        cp.getRate();

        cp.calculateBillAmt(250);

    }

}

```

### **Q)Differences between Interface and Abstract class?**

Interface	Abstract class
To declare interface we will use interface keyword.	To declare abstract class we will use abstract keyword.
It is a collection of abstract methods, default methods, static methods.	It is a collection of abstract methods and concrete methods.
It contains constants.	It contains instance variables.
To write the implementation for abstract methods of an interface we will use implementation class.	To write the implementation for abstract methods of an abstract class we will use sub classes.
Multiple inheritance is possible.	Multiple inheritance is not possible.
If we know only specification then we need to use interface.	If we know partial implementation then we need to use abstract class.

## **API**

- API stands for Application Programming Interface.
- API is a collection of packages.
- It is a base for the programmer to develop software applications.
- In java API's are divided into three types.

### **1)Predefined API**

Built in API is called predefined API.

ex:

<https://docs.oracle.com/javase/8/docs/api/>

## 2)Userdefined API

API which is created by the user based on the application requirements is called userdefined API.

## 3)Third party API

API which is given by third party vendor is called third party API.

ex:

javazoom api  
itext api  
and etc.

## Package

- A package is a collection of classes , interfaces , enums and Annotations.
- Here enum is a special class and annotation is a special interface.
- In general, a package is a collection of classes and interfaces.
- A package is also known as folder or a directory.
- In java packages are divided into two types.

## 1)Predefined packages

Built-In packages are called predefined packages.

ex:

java.lang  
java.io  
java.util  
java.util.stream  
java.time  
java.text  
java.sql  
javax.servlet  
javax.jsp

and etc.

## 2)Userdefined packages

A package which is created by the user based on the application requirement is called userdefined package.

To declare userdefined package we need to use package keyword.

syntax:

```
package <package_name>;
```

ex:

```
package com.ihub.www;
import java.util.Calendar;
class Test
{
    public static void main(String[] args)
    {
        Calendar c=Calendar.getInstance();
        int hour=c.get(Calendar.HOUR_OF_DAY);
        if(hour<=12)
            System.out.println("Good Morning");
        else if(hour<=16)
            System.out.println("Good Afternoon");
        else if(hour<=20)
            System.out.println("Good Evening");
        else
            System.out.println("Good Night");
    }
}
```

We can compile above program as follow.

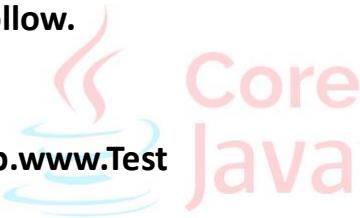
ex:

```
current directory
|
javaprog> javac -d . Test.java
|
destination
folder
```

We can run above program as follow.

ex:

```
javaprog> java com.ihub.www.Test
```



The logo for Core Java features the word "Core" in a light blue sans-serif font above the word "Java" in a larger, pinkish-red sans-serif font. To the left of "Core" is a stylized graphic of a flame or liquid rising from a blue base.

|
|
package name classname

## Inner classes

Sometimes we will declare a class inside another class such concept is called inner class.

syntax:

```
class Outer_Class
{
    class Inner_Class
    {
        -
    }
}
```

- //code to be execute }

Inner classes introduced as a part of event handling to remove GUI bugs.

But because of power features and benefits of inner classes , programmer started to use inner classes in our regular programming.

In inner class we can't declare static members.

### Access inner class data from static area of a outer class

```
class Outer
{
    class Inner
    {
        public void m1()
        {
            System.out.println("M1-Method");
        }
    }

    public static void main(String[] args)
    {
        Outer.Inner i=new Outer().new Inner();
        i.m1();

        //or

        new Outer().new Inner().m1();
    }
}
```

If we compile above program we will get two .class files i.e Outer.class and Outer\$Inner.class.

### Accessing inner class data from non-static area of outer class

```
class Outer
{
    class Inner
    {
        public void m1()
        {
            System.out.println("M1-Method");
        }
    }

    public void m2()
    {
        Inner i=new Inner();
        i.m1();
    }

    public static void main(String[] args)
    {
        Outer o=new Outer();
        o.m2();
    }
}
```



If we compile above program we will get two .class files i.e Outer.class and Outer\$Inner.class.

### **Note:**

We can't declare static members in inner class otherwise we will get illegal static declaration in inner class.

ex:

```
class Outer
{
    class Inner
    {
        public static void m1()
        {
            System.out.println("M1-Method");
        }
    }

    public void m2()
    {
        Inner i=new Inner();
        i.m1();
    }

    public static void main(String[] args)
    {
        Outer o=new Outer();
        o.m2();
    }
}

o/p:
```



## enum

- Enum is a group of named constants
- Enum concept is introduced in 1.5 version.
- Using enum we can create our own datatype called enumerated datatype.
- When compare to old language enum, java enum is more powerful.

Enum is a special class.

syntax:

```
enum <enum_type>
{
    val1,val2,...,valN
}
```

ex:

```
enum Months
{
    JAN,FEB,MAR
}
```



### Internal implementation of enum

Every enum internally consider as final class and extends with `java.lang.Enum` class.

Every enum constant is a reference variable of enum type.

ex:

```
enum Months           public final class Months extends java.lang.Enum
{
    {                   {
        JAN,FEB,MAR   ==>     public static final Months JAN=new Months();
    }                   public static final Months FEB=new Months();
                        public static final Months MAR=new Months();
```

```
}
```

## Declaration and Usage of enum

```
enum Months  
{  
    JAN,FEB,MAR  
}  
  
class Test  
{  
    public static void main(String[] args)  
    {  
        Months m=Months.JAN;  
        System.out.println(m); // JAN  
    }  
}
```

ex:

```
enum Months  
{  
    JAN,FEB,MAR  
}  
  
class Test  
{  
    public static void main(String[] args)  
    {  
        Months m=Months.FEB;  
        switch(m)  
        {  
            case JAN: System.out.println("January"); break;  
        }  
    }  
}
```



```
        case FEB: System.out.println("February"); break;
        case MAR: System.out.println("March"); break;
    }
}
```

## java.lang.Enum

- Power to enum will be inherited from java.lang.Enum class.
- A java.lang.Enum class contains following two methods.

### 1)values()

It is a static method which is used to read all the constants from enum.

### 2)ordinal()

It is used return ordinal number.

ex:

```
enum Months
```

```
{  
    JAN,FEB,MAR  
}
```



```
class Test
```

```
{  
  
    public static void main(String[] args)  
    {  
  
        Months[] m=Months.values();
```

```
        //for each loop
```

```
        for(Months m1:m)
```

```
{  
  
        System.out.println(m1+" ----- "+m1.ordinal());
```

```
    }}}
```

**When compare to old language enum ,java enum is more powerful because in addition to constants we can declare variables , methods and constructors.**

**ex:**

**enum Months**

**{**

**JAN,FEB,MAR;**

**Months()**

**{**

**System.out.println("constructor");**

**}**

**}**

**class Test**

**{**

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

**{**

**Months m=Months.JAN;**

**}**

**}**

**ex:**

**enum Months**

**{**

**JAN,FEB,MAR;**

**static int i=100;**



```

public static void main(String[] args)
{
    System.out.println(i); //100
}

```

## Wrapper classes

- The main objective of wrapper classes are
- To wrap primitive type to wrapper object and vice versa.
- To define several utility methods.

Primitive type	Wrapper class
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
boolean	Boolean
char	Character



## constructor

Every wrapper class contains two constructors. One will take corresponding primitive as an argument and second will take corresponding String as an argument.

ex:

Wrapper class	constructor
Byte	byte or String
Short	short or String
Integer	int or String
Long	long or String
Float	float or String
Double	double or String
Boolean	boolean or String

ex:1

```
class Test
{
    public static void main(String[] args)
    {
        Integer i1=new Integer(10);
        Integer i2=new Integer("20");
        System.out.println(i1+" "+i2);//10 20
    }
}
```

ex:

```
class Test
{
    public static void main(String[] args)
    {
        Boolean b1=new Boolean(true);
        Boolean b2=new Boolean("false");
        System.out.println(b1+" "+b2);
    }
}
```

ex:

```
class Test
{
    public static void main(String[] args)
    {
```



```
Character c=new Character('a');

System.out.println(c);//a

}

}
```

## Utility methods

### 1) valueOf()

It is exactly same as constructor.

It will convert primitive types to wrapper object.

ex:

```
class Test

{
    public static void main(String[] args)
    {
        Integer i1=Integer.valueOf(10);
        Integer i2=Integer.valueOf("20");
        System.out.println(i1+" "+i2);
    }
}
```

### 2) xxxValue()

It is used to convert Wrapper object to primitive type.

ex:

```
class Test

{
    public static void main(String[] args)
    {
        Integer i=new Integer(10);
```



```
byte b=i.byteValue();

short s=i.shortValue();

System.out.println(b+" "+s);

}
```

### **3) parseXxx()**

**It is used to convert string type to primitive type.**

**ex:**

```
class Test

{
    public static void main(String[] args)
    {
        String str="32";
        int i1=Integer.parseInt(str);
        System.out.println(i1);//32

        long l1=Long.parseLong(str);
        System.out.println(l1);//32

        float f1=Float.parseFloat(str);
        System.out.println(f1); //32.0
    }
}
```

### **4) toString()**

**It is used to convert wrapper type to string.**

**ex:**

```

class Test
{
    public static void main(String[] args)
    {
        Integer i=new Integer(10);
        String s=i.toString();
        System.out.println(s);//10
    }
}

```

### **Q)Types of objects in java?**

We have two types of objects in java.

1)Immutable object

2)Mutable object

#### **1)Immutable object**

After object creation if we perform any changes then for every change a new object will be created such type of object is called immutable object.

ex:

String

Wrapper classes

#### **2)Mutable object**

After object creation if we perform any changes then all the changes will be reflected in a single object only such type of object is called mutable object.

ex:

StringBuffer

StringBuilder

**“Everyone’s life depends upon their thoughts only,what you think that’s matter”**

-With love sandeep-

## Strings

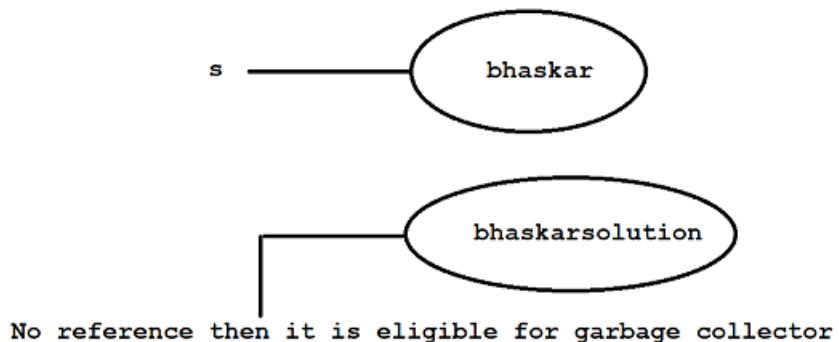
It is a collection of characters which is enclosed in a double quotation.

### case1:

Once if we create a String object we can't perform any changes. If we perform any changes then for every change a new object will be created such behaviour is called immutability of an object.

Diagram: java33.1

```
String s=new String("bhaskar");
s.concat("solution");
System.out.println(s); // bhaskarsolution
```



### case2:



Q) What is the difference between == and .equals() method?

==

It is a equality operator or comparision operator.

It used for reference comparision or address comparision.

ex:

```
class Test
```

```
{
```

```
    public static void main(String[] args)
    {
        String s1=new String("bhaskar");
        String s2=new String("bhaskar");
        System.out.println(s1==s2);//false
    }
}
```

```
    }  
}  
}
```

### .equals() method

It is method present in String class.

It is used for content comparision and it is case sensitive.

ex:

```
class Test  
{  
    public static void main(String[] args)  
    {  
        String s1=new String("bhaskar");  
        String s2=new String("bhaskar");  
        System.out.println(s1.equals(s2));//true  
    }  
}
```



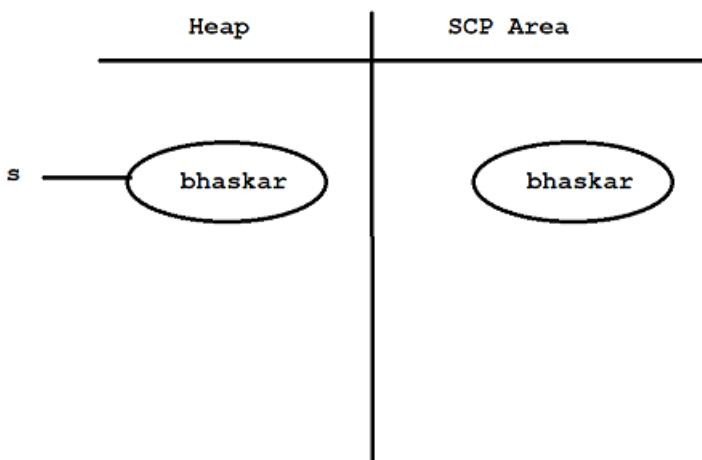
### case3:

Once if we create String object two objects will be created , one is on heap and another is on SCP (String Constant Pool) area.But 's' always points to heap area.

ex:

```
String s=new String("bhaskar");
```

Diagram: java33.2



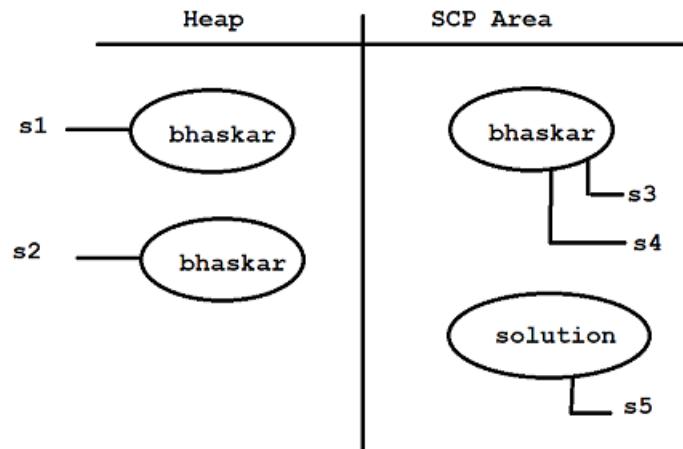
**Object creation in SCP area is always optional.** First JVM will check is there any object is created with same content or not. If it is created then it simply refers to that object. If it is not created then JVM will create a new object. Hence there is no chance of having duplicate objects in SCP area.

Even though SCP objects do not have have object reference, garbage collector can't access them.

SCP objects will be destroyed when JVM shutdowns or terminated.

Diagram: java33.3

```
String s1=new String("bhaskar");
String s2=new String("bhaskar");
String s3="bhaskar";
String s4="bhaskar";
String s5="solution";
```

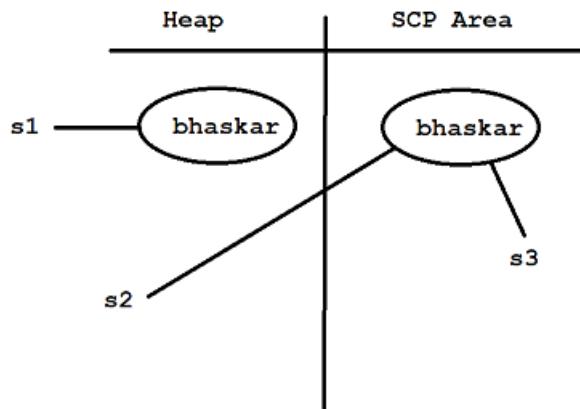


## Interning of String object

With the help of heap object reference if we need corresponding SCP object reference then we need to use `intern()` method.

#### Diagram: java33.4

```
String s1=new String("bhaskar");
String s2=s1.intern();
String s3="bhaskar";
System.out.println(s2==s3); //true
```



#### String important methods

```
class Test
{
    public static void main(String[] args)
    {
        String str="bhaskar";
        System.out.println(str.length());//7
        System.out.println(str.charAt(3));//s
        System.out.println(str.toUpperCase());//BHASKAR
        System.out.println(str.toLowerCase());//bhaskar
        System.out.println(str.equals("bhaskar"));//true
        System.out.println(str.equalsIgnoreCase("BHASKAR"));//true
        System.out.println(str.replace('a','A'));//bhAskAr
        System.out.println(str.concat("solution"));//bhaskarsolution
        System.out.println(str.indexOf('a'));//2
        System.out.println(str.lastIndexOf('a'));//5
        System.out.println(str.substring(4));//kar
        System.out.println(str.replaceAll(str,"gogo"));//gogo}
    }
}
```

**Q)Write a java program to accept one string and display it ?**

**approach1**

```
import java.util.Scanner;  
class Test  
{  
    public static void main(String[] args)  
    {  
        Scanner sc=new Scanner(System.in);  
  
        System.out.println("Enter the string :");  
        String str=sc.next(); //James Gosling  
  
        System.out.println("welcome : "+str); // welcome : James  
    }  
}
```



**approach2**

```
import java.util.Scanner;  
class Test  
{  
    public static void main(String[] args)  
    {  
        Scanner sc=new Scanner(System.in);  
  
        System.out.println("Enter the string :");  
        String str=sc.nextLine();
```

```
        System.out.println("welcome : "+str);
    }
}
```

**Q)Write a java program to display the string in reverse order?**

**input:**

hello

**output:**

olleh

**ex:**

**class Test**

{

```
    public static void main(String[] args)
```

{

```
        String str="hello";
```

```
        //converting string to char array
```

```
        char[] carr=str.toCharArray();
```

```
        String rev="";
```

```
        //reading reverse
```

```
        for(int i=carr.length-1;i>=0;i--)
```

{

```
            rev=rev+carr[i];
```

}

```
        System.out.println(rev);
```

}

}

**Q)Write a java program to find out given string is palindrome or not?**

**input:** madam

**output:**

**It is a palindrome string**

**ex:**

**class Test**

**{**

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

**{**

**String str="madam";**

**//converting string to char array**

**char[] carr=str.toCharArray();**



**String rev="";**

**//reading reverse**

**for(int i=carr.length-1;i>=0;i--)**

**{**

**rev=rev+carr[i];**

**}**

**if(str.equals(rev))**

**System.out.println("It is a palindrome string");**

**else**

**System.out.println("It is not a palindrome string");}}**

**Q)Write a java program to display reverse of a sentence?**

**input:**

This is java class

**output:**

class java is This

**ex:**

**class Test**

{

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

{

**String str="This is java class";**

**String[] sarr=str.split(" ");**



**//reading reverse**

**for(int i=sarr.length-1;i>=0;i--)**

    {

**System.out.print(sarr[i]+" ");**

    }

}

}

**Q)Write a java program to display reverse of each word in a sentence?**

**input:**

This Is Java Class

**output:**

sihT sl avaJ ssalC

**ex:**

**class Test**

{

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

{

**String str="This Is Java Class";**

**String[] sarr=str.split(" ");**



**//for each loop**

**for(String s:sarr)**

{

**char[] carr=s.toCharArray(); // T h i s**

**//reverse order**

**for(int i=carr.length-1;i>=0;i--)**

{

**System.out.print(carr[i]);**

}

**//space**

**System.out.print(" ");**

**} }**

**Q)Write a java program to display duplicate characters from given string?**

**input:**

googl

**output:**

og

**ex:**

**class Test**

{

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

{

**String str="google";**

**String characters="";**

**String duplicates="";**



**for(int i=0;i<str.length();i++)**

{

**//converting character to String**

**String current=Character.toString(str.charAt(i));**

**if(characters.contains(current))**

{

**if(!duplicates.contains(current))**

{

**duplicates+=current;**

**continue;**

}

```

    }
    characters+=current;
}
System.out.println(duplicates);
}
}

```

**Q)Write a java program to display unique/distinct characters from given string?**

**input:**

google

**output:**

gole

**ex:**

**class Test**

{

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

{

**String str="google";**

**String characters="";**

**String duplicates="";**

**for(int i=0;i<str.length();i++)**

{

**//converting character to String**

**String current=Character.toString(str.charAt(i));**

```

        if(characters.contains(current))
        {
            if(!duplicates.contains(current))
            {
                duplicates+=current;
                continue;
            }
            characters+=current;
        }
        System.out.println(characters);
    }
}

```

**Q)Write a java program to remove special characters from given string?**

**input:**



he@l#l\_o\$

**output:**

hello

**ex:**

**class Test**

{

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

{

**String str="he@l#l\_o\$";**

**str=str.replaceAll("[^a-zA-Z0-9]","");**

**System.out.println(str); // hello}}**

**Q)Write a java program to accept one long string and display those strings which are palindrome ?**

**input:**

racar is in madam

**output:**

racar madam

**ex:**

**class Test**

{

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

{

**String str="racar is in madam ";**

**String[] sarr=str.split(" ");**

**//for each loop**

**for(String s:sarr)**

{

**char[] carr=s.toCharArray(); // r a c a r**

**//reverse logic**

**String rev="";**

**for(int i=carr.length-1;i>=0;i--)**

{

**rev+=carr[i];**

}



```

        if(s.equals(rev))
            System.out.print(s+" ");
    }
}

```

**Q)Write a java program to check given strings are Anagram or not?**

```

import java.util.Arrays;
class Test
{
    public static void main(String[] args)
    {
        String str1="silent";
        String str2="listen1";
        //convert string to char array
        char[] ch1=str1.toCharArray();
        char[] ch2=str2.toCharArray();

        //sorting the characters
        Arrays.sort(ch1);
        Arrays.sort(ch2);

        boolean flag=true;
        for(int i=0;i<ch1.length;i++)
        {
            if(ch1[i]!=ch2[i])

```



```

    {
        flag=false;
        break;
    }
}

if(flag)
    System.out.println("It is Anagram String");
else
    System.out.println("It is not Anagram String");

}
}

```

## StringBuffer

If our content change frequently then it is not recommended to use String object because for every change a new object will be created.

To overcome this limitation we need to use StringBuffer.

In StringBuffer all the required changes is done in a single object.

### constructor

#### 1)StringBuffer sb=new StringBuffer()

It will create a empty StringBuffer object with default initial capacity of 16.

If capacity reaches to maximum capacity then new capacity will be created with below formulae.

ex:

```
capacity = currentcapacity+1*2;
```

ex:

```
class Test
{
    public static void main(String[] args)
    {
        StringBuffer sb=new StringBuffer();

        System.out.println(sb.capacity()); // 16

        sb.append("abcdefghijklnop");
        System.out.println(sb.capacity());// 16

        sb.append("qrst");
        System.out.println(sb.capacity());//16+1*2 = 34
    }
}
```

## 2)StringBuffer sb=new StringBuffer(int initialCapacity)

It will create StringBuffer object with specified initial capacity.

ex:

```
class Test
{
    public static void main(String[] args)
    {
        StringBuffer sb=new StringBuffer(19);

        System.out.println(sb.capacity()); // 19 {}
    }
}
```

### **3)StringBuffer sb=new StringBuffer(String s)**

It will create StringBuffer object equivalent to String.

Here capacity will be created with below formula.

ex:

```
capacity = s.length+16.
```

ex:

```
class Test
```

```
{
```

```
    public static void main(String[] args)
```

```
{
```

```
        StringBuffer sb=new StringBuffer("bhaskar");
```

```
        System.out.println(sb.capacity()); // s.length()+16 = 23
```



```
}
```

```
}
```

### **Q)Write a java program to display reverse of a string?**

```
class Test
```

```
{
```

```
    public static void main(String[] args)
```

```
{
```

```
        String str="hello";
```

```
        StringBuffer sb=new StringBuffer(str);
```

```
        String rev=sb.reverse().toString();
```

```
        System.out.println(rev);

    }

}
```

**Q)Write a java program to check given string is palindrome or not?**

```
class Test

{
    public static void main(String[] args)
    {
        String str="ECE";

        StringBuffer sb=new StringBuffer(str);
        String rev=sb.reverse().toString();

        if(str.equals(rev))
            System.out.println("It is a palindrome");
        else
            System.out.println("It is not a palindrome");
    }
}
```

## StringBuilder

StringBuilder is exactly same as StringBuffer with following differences.

StringBuffer	StringBuilder
All the methods present in StringBuffer are synchronized.	No method present in StringBuilder is synchronized.
At a time only one thread is allowed to access StringBuffer object.Hence we can achieve thread safety.	Multiple threads are allowed to access StringBuilder object.Hence we can't achieve thread safety.
Waiting time of a thread will be increases , effectively performance is low.	There is no waiting threads , effectively performance is high.
It is introduced in 1.0v.	It is introduced in 1.5v.

## StringTokenizer:

- A StringTokenizer is a class which is present in java.util package.
- It is used to tokenize the String based on regular expression.

We can create StringTokenizer object as follow.

ex:

```
 StringTokenizer st=new StringTokenizer(string,regularExp);
```

StringTokenizer class contains following methods.

ex:

```
 public boolean hasMoreTokens()  
 public String nextToken()  
 public boolean hasMoreElements()  
 public Object nextElement()  
 public int countTokens()
```

ex:1

```
import java.util.StringTokenizer;
```

```
class Test
{
    public static void main(String[] args)
    {
        StringTokenizer st=new StringTokenizer("This is java class");
        System.out.println(st.countTokens());
    }
}
```

**Note:**

Default regular expression is space.

ex:

```
import java.util.StringTokenizer;
```

```
class Test
```

```
{
    public static void main(String[] args)
    {
        StringTokenizer st=new StringTokenizer("This is java class","");
        System.out.println(st.countTokens());
    }
}
```

ex:

```
import java.util.StringTokenizer;
```

```
class Test
```

```
{
    public static void main(String[] args)
    {
        StringTokenizer st=new StringTokenizer("This is java class","");
    }
}
```



```
while(st.hasMoreTokens())
{
    String s=st.nextToken();
    System.out.println(s);
}
```

ex:

```
import java.util.StringTokenizer;
class Test
{
    public static void main(String[] args)
    {
        StringTokenizer st=new StringTokenizer("This is java class","");
    }
}
```

Core Java

```
while(st.hasMoreElements())
{
    String s=(String)st.nextElement();
    System.out.println(s);
}
```

ex:

```
import java.util.StringTokenizer;
class Test
{
    public static void main(String[] args)
    {
        StringTokenizer st=new StringTokenizer("9,99,999","","");
    }
}
```

```
while(st.hasMoreElements())
{
    String s=(String)st.nextElement();
    System.out.println(s);
}
```

## Exception Handling

Q) What is the difference between Exception and Error?

### Exception

Exception is a problem for which we can provide solution programmatically.

Exception will raise due to syntax errors.

ex:

**ArithmaticException**

**IllegalArgumentException**

**FileNotFoundException**



### Error

Error is a problem for which we can't provide solution programmatically.

Error will raise due to lack of system resources.

ex:

**OutOfMemoryError**

**StackOverFlowError**

**LinkageError**

    and etc.

As a part of java application development it is a responsibility of a programmer to provide smooth termination for every java program.

We have two types of terminations.

### **1)Smooth termination / Graceful termination**

During the program execution suppose if we are not getting any interruption in the middle of the program such type of termination is called smooth termination.

### **2)Abnormal termination**

During the program execution suppose if we are getting some interruption in the middle of the program such type of termination is called abnormal termination.

ex:

```
class Test
```

```
{
```

```
    public static void main(String[] args)
```

```
{
```

```
        System.out.println(10/0);
```

```
}
```

```
}
```



If any exception raised in our program so we must and should handle that exception otherwise our program will terminate abnormally.

Here exception will display name of the exception , description of the exception and line number of exception.

## Exception

- It is a unwanted , unexpected event which disturbs normal flow of our program.
- Exceptions always raised at runtime so they are also known as runtime events.
- The main objective of exception handling is to provide graceful termination.
- In java exceptions are divided into two types.

1)Predefined exceptions

2)Userdefined exceptions

**1)Predefined exceptions:**

Built-In exceptions are called predefined exceptions.

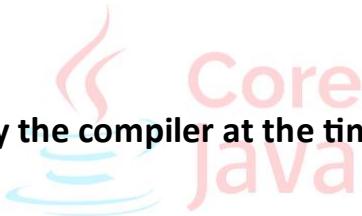
It is categories into two types.

i)Checked Exceptions

ii)Unchecked Exceptions

**i)Checked Exceptions**

Exceptions which are checked by the compiler at the time of compilation is called checked exceptions.



ex:

InterruptedException

IOException

FileNotFoundException

EOFException

**ii)Unchecked Exceptions**

Exceptions which are checked by the JVM at the time of runtime is called unchecked exceptions.

ex:

ArithmaticException

ClassCastException

IllegalArgumentException

## Diagram: java35.1

If any checked exception raised in our program then we must and should handle that exception by using try and catch block.

### try block

- It is a block which contains Risky Code.
- A try block always associate with catch block.
- A try block is used to throw the exception to catch block.
- If exception raise in try block then try block won't be executed.
- If exception raise in the middle of the program then rest of code won't be executed.

### Catch block

- It is a block which contains Error Handling code.
- A catch block always associate with try block.
- A catch block is used to catch the exception which is thrown by try block.
- A catch block won't be execute if we won't get any exception in try block.
- A catch block will take exception name as a parameter and that name must match with exception class name.

**syntax:**

```
try
{
    -
    - // Risky code
    -
}

catch(ArithmeticException ae)
{
    -
    - // Error Handling code
}
```

```
-  
}  
  
ex:1  
  
class Test  
{  
  
    public static void main(String[] args)  
    {  
  
        try  
        {  
  
            System.out.println("try-block");  
        }  
  
        catch(Exception e)  
        {  
  
            System.out.println("catch-block");  
        }  
    }  
}
```



o/p:  
try-block

ex:2

```
class Test  
{  
  
    public static void main(String[] args)  
    {  
  
        try  
        {  
  
            System.out.println(10/0);  
        }  
    }  
}
```

```
    }

    catch(ArithmeticException ae)
    {
        System.out.println("catch-block");
    }
}

o/p:
```

**catch-block**

**ex:3**

```
class Test
{
    public static void main(String[] args)
    {
        try
        {
            System.out.println("stmt1");
            System.out.println(10/0);
            System.out.println("stmt2");
        }
        catch(ArithmeticException ae)
        {
            System.out.println("catch-block");
        }
    }
}
```



**o/p:**

```
stmt1  
catch-block
```

### **finally block**

- It is a block which contains cleanup code.
- It is never recommended to write cleanup code in try block because if there is a exception in try block then try block won't be executed.
- It is never recommended to write cleanup code in catch block because if there is no exception in try block then catch block won't be executed.
- But we need a place where we can maintain cleanup code and it should execute irrespective of exception raised or not .such block is called finally block.

**syntax:**

```
try  
{  
    -  
    - // Risky code  
    -  
}  
catch(ArithmeticException ae)  
{  
    -  
    - // Error Handling code  
    -  
}  
finally  
{  
    -
```



- // finally block

-

}

ex:1

class Test

{

    public static void main(String[] args)

    {

        try

        {

            System.out.println("try-block");

        }

    catch(ArithmeticException ae)

    {

        System.out.println("catch-block");

    }

    finally

    {

        System.out.println("finally-block");

    }

}

o/p:

try-block

finally-block

ex:2

```
class Test
{
    public static void main(String[] args)
    {
        try
        {
            System.out.println(10/0);
        }
        catch(ArithmeticException ae)
        {
            System.out.println("catch-block");
        }
        finally
        {
            System.out.println("finally-block");
        }
    }
}

o/p:
catch-block
finally-block
```

A try with finally combination is valid in java.

ex:

```
class Test
{
    public static void main(String[] args)
```

```

{
    try
    {
        System.out.println("try-block");
    }
    finally
    {
        System.out.println("finally-block");
    }
}

```

**"If you develop any skill yourselves,society looks and wait for you"**

-With love sandeep-

**Q)What is the difference between final, finally and finalized method?**

### final



- It is a modifier which is applicable for variables, methods and classes.
- If we declare any variable as final then reassignment of that variable is not possible.
- If we declare any method as final then overriding of that method is not possible.
- If we declare any class as final then creating child class is not possible.

### finally

- It is a block which contains cleanup code and it should execute irrespective of exception raised or not.

### finalized method

- It is a method called by garbage collector just before destroying an object for cleanup activity.

### various methods to display exception details

- Throwable class is a root/parent class for Exception and Error class.

- Throwable class defines following methods to display exception details.

### **1) printStackTrace()**

It will display name of the exception , description of the exception and line number of the exception.

### **2) toString()**

It will display name of the exception and description of the exception .

### **3) getMessage()**

It will display description of the exception.

```
class Test
```

```
{
```

```
    public static void main(String[] args)
```

```
{
```

```
    try
```

```
{
```

```
        System.out.println(10/0);
```

```
}
```

```
    catch(Exception e)
```

```
{
```

```
        e.printStackTrace();
```

```
        System.out.println("-----");
```

```
        System.out.println(e.toString());
```

```
        System.out.println("-----");
```

```
        System.out.println(e.getMessage());
```

```
    }
}
}
```

## **throw stmt**

- Sometimes we will create exception object explicitly and handover to JVM manually by using throw statement.

ex:

```
throw new ArithmeticException("don't divide by zero");
```

ex:

```
class Test
```

```
{
    public static void main(String[] args)
    {
        throw new ArithmeticException("don't divide by zero");
    }
}
```



## **throws statement**

- Whenever checked exception raised in our program then we must and should handle that exception by using try and catch block or by using throws statement.

ex:1

```
class Test
```

```
{
    public static void main(String[] args)
    {
        try
        {
            Thread.sleep(3000);
        }
    }
}
```

```
        System.out.println("Welcome to java world");

    }

    catch (InterruptedException ie)

    {

        ie.printStackTrace();

    }

}}
```

ex:2

```
class Test

{

    public static void main(String[] args)throws InterruptedException

    {

        Thread.sleep(5000);

        System.out.println("Welcome to java world");

    }

}
```



## 2)Userdefined exceptions

- Exceptions which are created by the user based on the application requirements are called customized exceptions.

ex:

```
StudentNotPractisingException

NoInterestInJavaException

JustTimepassClassesException

TooYoungException

TooOldException
```

ex:

```
class TooYoungException extends RuntimeException
```

```

{
    TooYoungException(String s)
    {
        super(s);
    }
}

class TooOldException extends RuntimeException
{
    TooOldException(String s)
    {
        super(s);
    }
}

class Test
{
    public static void main(String[] args)
    {
        //convert string input to int
        int age=Integer.parseInt(args[0]);

        if(age<18)
            throw new TooYoungException("U r not eligible to vote");

        else
            throw new TooOldException("U r eligible to vote");
    }
}
o/p:

```



```
javac Test.java
```

```
java Test 25
```

```
java Test 5
```

### **Note:**

- A try block can have multiple catch blocks. If try block contains multiple catch blocks then order of catch block is very important , it should be from child to parent.

```
class Test
```

```
{  
    public static void main(String[] args)  
    {  
        try  
        {  
            System.out.println(10/0);  
        }  
        catch (ArithmaticException ae)  
        {  
            ae.printStackTrace();  
        }  
        catch (Runtimeexception re)  
        {  
            re.printStackTrace();  
        }  
    }  
}
```



### **java.io package**

#### **File**

- ```
File f=new File("abc.txt");
```
- File will check is there any abc.txt file already created or not.
  - If it is available it simply refers to that file. If it is not created then
  - it won't create any new file.

ex:

```
import java.io.*;  
  
class Test  
{  
  
    public static void main(String[] args)  
    {  
  
        File f=new File("abc.txt");  
  
        System.out.println(f.exists());//false  
    }  
}
```

A File object can be used to create a physical file.

ex:

```
import java.io.*;  
  
class Test  
{  
  
    public static void main(String[] args) throws IOException  
    {  
  
        File f=new File("abc.txt");  
  
        System.out.println(f.exists());//false  
  
        f.createNewFile();  
  
        System.out.println(f.exists());//true } }
```

A File object can be used to create a directory also.

ex:

```

import java.io.*;
class Test
{
    public static void main(String[] args) throws IOException
    {
        File f=new File("bhaskar123");
        System.out.println(f.exists());//false

        f.mkdir();
        System.out.println(f.exists());//true }
    }
}

```

**Q) Write a java program to Create a "cricket123" folder and inside that folder create "abc.txt" file?**

```

import java.io.*;
class Test
{
    public static void main(String[] args) throws IOException
    {
        File f1=new File("cricket123");
        f1.mkdir();

        File f2=new File("cricket123","abc.txt");
        f2.createNewFile();

        System.out.println("Please check the location");}}

```



## FileWriter

FileWriter is used to write character oriented data into a file.

## constructor

```
FileWriter fw=new FileWriter(String s);
```

```
FileWriter fw=new FileWriter(File f);
```

ex:

```
FileWriter fw=new FileWriter("aaa.txt");
```

or

```
File f=new File("aaa.txt");
```

```
FileWriter fw=new FileWriter(f);
```

If file does not exist then FileWriter will create a physical file.

## Methods

### 1)write(int ch)

It will insert single character into a file.



### 2)write(char[] ch)

It will insert array of characters into a file.

### 3)write(String s)

It will insert String into a file.

### 4)flush()

It gives guarantee that last character of a file is also inserted.

### 5)close()

It is used to close the FileWriter object.

ex:

```
import java.io.*;
```

```
class Test
{
    public static void main(String[] args) throws IOException
    {
        FileWriter fw=new FileWriter("aaa.txt");

        fw.write(98);// b
        fw.write("\n");

        char[] ch={'a','b','c'};
        fw.write(ch);
        fw.write("\n");

        fw.write("bhaskar\nsolution");
        fw.flush();
        fw.close();

        System.out.println("Please check the location");
    }
}
```



## FileReader

It is used to read character oriented data from a file.

### constructor

```
FileReader fr=new FileReader(String s);
```

```
FileReader fr=new FileReader(File f);
```

ex:

```
FileReader fr=new FileReader("aaa.txt");
```

or

```
File f=new File("aaa.txt");
FileReader fr=new FileReader(f);
```

## Methods

### 1)read()

It will read next character from a file and return unicode value.

If next character is not available then it will return -1.

### 2)read(char[] ch)

It will read collection of characters from a file.

### 3)close()

It is used to close FileReader object.

ex:1

```
import java.io.*;
class Test
{
    public static void main(String[] args)throws IOException
    {
        FileReader fr=new FileReader("aaa.txt");

        int i=fr.read();
        while(i!=-1)
        {
            System.out.print((char)i);
            i=fr.read();
        }
        fr.close();
    }
}
```



```
}
```

ex:2

```
import java.io.*;  
  
class Test  
{  
  
    public static void main(String[] args) throws IOException  
{  
  
        FileReader fr=new FileReader("aaa.txt");  
  
        char[] carr=new char[255];  
  
        //load the data from file to char array  
        fr.read(carr);  
  
        //reading the data from char array  
        for(char c:carr)  
        {  
            System.out.print(c);  
        }  
  
        fr.close();  
    }  
}
```



### Usage of FileWriter and FileReader is not recommended to use

- While inserting the data by using FileWriter ,we need to insert line

separator(\n) which is very headache for the programmer.

- While reading the data by using FileReader object ,we need to read character by character which is not convenient to the programmer.
- To overcome this limitation Sun micro system introduced BufferedWriter and BufferedReader.

## **BufferedWriter**

- It is used to insert character oriented data into a file.

### **constructor**

**BufferedWriter bw=new BufferedWriter(Writer w);**

**BufferedWriter bw=new BufferedWriter(Writer w,int buffersize);**

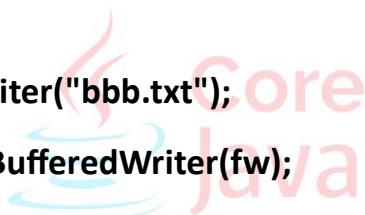
**BufferedWriter object does not communicate with files directly.**

**It will take the support of some writer objects.**

**ex:**

**FileWriter fw=new FileWriter("bbb.txt");**

**BufferedWriter bw=new BufferedWriter(fw);**



**or**

**BufferedWriter bw=new BufferedWriter(new FileWriter("bbb.txt"));**

## **Methods**

### **1)write(int ch)**

- It will insert single character into a file.

### **2)write(char[] ch)**

- It will insert array of characters into a file.

### **3)write(String s)**

- It will insert String into a file.

#### 4)flush()

- It gives guarantee that last character of a file is also inserted.

#### 5)close()

- It is used to close the BufferedWriter object.

#### 6)newLine()

- It will insert new line into a file.

ex:

```
import java.io.*;  
  
class Test  
  
{  
  
    public static void main(String[] args)throws IOException  
    {
```

BufferedWriter bw=new BufferedWriter(new FileWriter("bbb.txt"));

```
        bw.write(98);//b
```

```
        bw.newLine();
```

```
        char[] ch={'a','b','c'};
```

```
        bw.write(ch);
```

```
        bw.newLine();
```

```
        bw.write("bhaskar");
```

```
        bw.newLine();
```

```
        bw.flush();
```

```
        bw.close();
```

```
        System.out.println("Please check the location");
```

```
    }  
}
```

## BufferedReader

It is enhanced reader to read character oriented data from a file.

### constructor

- **BufferedReader br=new BufferedReader(Reader r);**
- **BufferedReader br=new BufferedReader(Reader r,int buffersize);**
- **BufferedReader object can't communicate with files directly. IT will take support of some reader objects.**

ex:

```
FileReader fr=new FileReader("bbb.txt");  
BufferedReader br=new BufferedReader(fr);
```

or



```
BufferedReader br=new BufferedReader(new FileReader("bbb.txt"));
```

The main advantage of BufferedReader over FileReader is we can read character line by line instead of character by character.

### methods

#### 1)read()

- It will read next character from a file and return unicode value.
- If next character is not available then it will return -1.

#### 2)read(char[] ch)

- It will read collection of characters from a file.

### 3)close()

- It is used to close BufferedReader object.

### 4)nextLine()

- It is used to read next line from the file.If next line is not available then it will return null.

ex:

```
import java.io.*;  
  
class Test  
{  
  
    public static void main(String[] args)throws IOException  
{  
  
        BufferedReader br=new BufferedReader(new FileReader("bbb.txt"));  
  
        String line=br.readLine();  
        while(line!=null)  
        {  
  
            System.out.println(line);  
  
            line=br.readLine();  
        }  
  
        br.close();  
    }  
}
```

## PrintWriter

- It is enhanced write to write character oriented data into a file.

## constructor

**PrintWriter pw=new PrintWriter(String s);**

**PrintWriter pw=new PrintWriter(File f);**

**PrintWriter pw=new PrintWriter(Writer w);**

**PrintWriter can communicate with files directly and it will take the support of some writer objects.**

**ex:**

**PrintWriter pw=new PrintWriter("ccc.txt");**

**or**

**PrintWriter pw=new PrintWriter(new File("ccc.txt"));**

**or**

**PrintWriter pw=new PrintWriter(new FileWriter("ccc.txt"));**

The main advantage of PrintWriter over FileWriter and BufferedWriter is we can insert any type of data.

Assume if we want insert primitive values then PrintWriter is best choice.

## methods

**write(int ch)**

**write(char[] ch)**

**write(String s)**

**flush()**

**close()**

**writeln(int i)**

**writeln(float f)**

**writeln(double d)**

**writeln(String s)**

```
writeln(char c)  
writeln(boolean b)
```

```
write(int i)  
write(float f)  
write(double d)  
write(String s)  
write(char c)  
write(boolean b)
```

ex:

```
import java.io.*;  
  
class Test  
{  
    public static void main(String[] args) throws IOException  
    {  
        PrintWriter pw=new PrintWriter("ccc.txt");  
  
        pw.write(100);// d  
        pw.println(100);// 100  
        pw.print('a');  
        pw.println(true);  
        pw.println("hi");  
        pw.println(10.5d);  
  
        pw.flush();
```



```
        pw.close();

        System.out.println("Please check the location");

    }

}
```

## **various ways to provide input values from the keyboard**

There are various ways to provide input values from keyboard.

- 1)Command line argument
- 2)BufferedReader class
- 3)Console class
- 4)Scanner class

### **1)Command line argument**

In command line argument we need to pass our inputs at runtime.

ex:

```
class Test
```

```
{
```

```
    public static void main(String[] args)
```

```
{
```

```
        String name=args[0];
```

```
        System.out.println("Welcome : "+name);
```

```
}
```

```
}
```



o/p:

```
javac Test.java
```

```
java Test Alan
```

### **2)BufferedReader class**

- BufferedReader class present in java.io package.

- **BufferedReader** class will take **InputStreamReader** object as a parameter which is embedded with **System.in**.

ex:

```
BufferedReader br=  
    new BufferedReader  
        (new InputStreamReader(System.in));
```

To read input values from console we need to **readLine()** method.

ex:

```
import java.io.*;  
  
class Test  
{  
  
    public static void main(String[] args) throws IOException  
    {  
  
        BufferedReader br=new BufferedReader(new  
InputStreamReader(System.in));
```

```
        System.out.println("Enter the Name :");  
        String name=br.readLine();
```

```
        System.out.println("Welcome : "+name);
```

```
    }  
}
```

### **3)Console class**

- Console class present in java.io package.
- We can create Console class object by using console() method of System class.

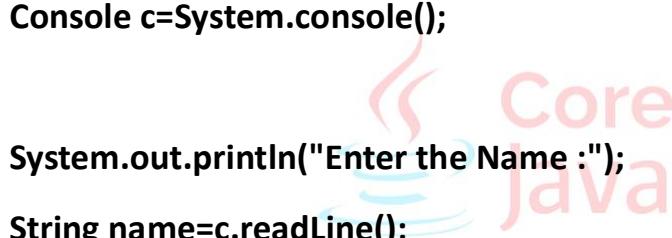
ex:

```
Console c=System.console();
```

To read inputs from console we need to use readLine() method.

ex:

```
import java.io.*;  
  
class Test  
{  
  
    public static void main(String[] args)throws IOException  
    {  
  
        Console c=System.console();  
  
        System.out.println("Enter the Name :");  
        String name=c.readLine();  
  
        System.out.println("Welcome : "+name);  
  
    }  
}
```



#### 4)Scanner class

- Scanner class present java.util package.
- We can create Scanner object class as follow.

ex:

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

We can read inputs from the console by using following methods.

ex:

```
next()  
nextLine()  
nextInt()  
nextFloat()  
nextDouble()  
next().charAt(0);  
and etc.
```

ex:

```
import java.util.*;  
  
class Test  
{  
  
    public static void main(String[] args)  
    {  
        Scanner sc=new Scanner(System.in);  
  
        System.out.println("Enter the No :");  
        int no=sc.nextInt();  
  
        System.out.println("Enter the Name :");  
        String name=sc.next();  
  
        System.out.println("Enter the Fee :");  
        double fee=sc.nextDouble();  
  
        System.out.println(no+" "+name+" "+fee);  
    }  
}
```

}

## **Generics**

- Array is a typesafe.
- We can provide guarantee that what type of elements are present in array.
- If requirement is there to store String values then we need to use String[] array.

ex:

```
String[] sarr=new String[100];  
  
sarr[0]="hi";  
  
sarr[1]="bye";  
  
sarr[2]=10; //invalid
```

At the time retrieve the data from array we don't need to perform any typecasting.

ex:

```
String[] sarr=new String[100];  
  
sarr[0]="hi";  
  
sarr[1]="bye";  
-  
-  
-  
  
String val=sarr[0];
```



Collection is not a typesafe.

We can't give guarantee that what type of elements are present in Collections.

If requirement is there to store String values then it is never recommended to use ArrayList. Here we won't get any compile time error or runtime error but sometimes our program will get failure.

ex:

```
ArrayList al=new ArrayList();
al.add("hi");
al.add("bye");
al.add(10);
```

**At the time of retrieving the data compulsary we need to perform typecasting.**

**ex:**

```
ArrayList al=new ArrayList();
al.add("hi");
al.add("bye");
al.add(10);
```

-  
-

```
String val=(String)al.get(0);
```



**To overcome above limitations sun micro system introduced Generics concepts in 1.5 version.**

**The main objective of generics are.**

- 1) To make Collections as typesafe.**
- 2) To avoid typecasting problem.**

## **Q)Differences between Array and Collection?**

| Array                                                   | Collection                                                                |
|---------------------------------------------------------|---------------------------------------------------------------------------|
| <b>It is a collection of homogeneous data elements.</b> | <b>It is a collection of homogeneous and heterogeneous data elements.</b> |
| <b>Arrays are fixed in size.</b>                        | <b>Collections are growable in nature.</b>                                |

|                                                                                                                             |                                                                                                             |
|-----------------------------------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------|
| <b>Performance point of view array is recommended to use.</b>                                                               | <b>Memory point of view Collection is recommended to use.</b>                                               |
| <b>It is typesafe.</b>                                                                                                      | <b>It is not typesafe.</b>                                                                                  |
| <b>Arrays are not implemented based on expect any ready made methods. every logic we need to write the code explicitly.</b> | <b>Collections are implemented based on data structure concept so we can expect For ready made methods.</b> |
| <b>Arrays will accept primitive type and object type.</b>                                                                   | <b>Collections will accept only object type.</b>                                                            |

## Collection

- A Collection is an interface which is present in `java.util` package
- It is a root interface for entire Collection Framework.
- If we want to represent group of individual objects in a single entity then we need to use Collection interface.

Collection interface contains following methods.

ex:

```

public abstract boolean add(E);
public abstract boolean remove(java.lang.Object);
public abstract boolean containsAll(java.util.Collection<?>);
public abstract boolean addAll(java.util.Collection<? extends E>);
public abstract boolean removeAll(java.util.Collection<?>);
public boolean removeIf(java.util.function.Predicate<? super E>);
public abstract boolean retainAll(java.util.Collection<?>);
public abstract void clear();
public abstract boolean equals(java.lang.Object);
public abstract int hashCode();

```

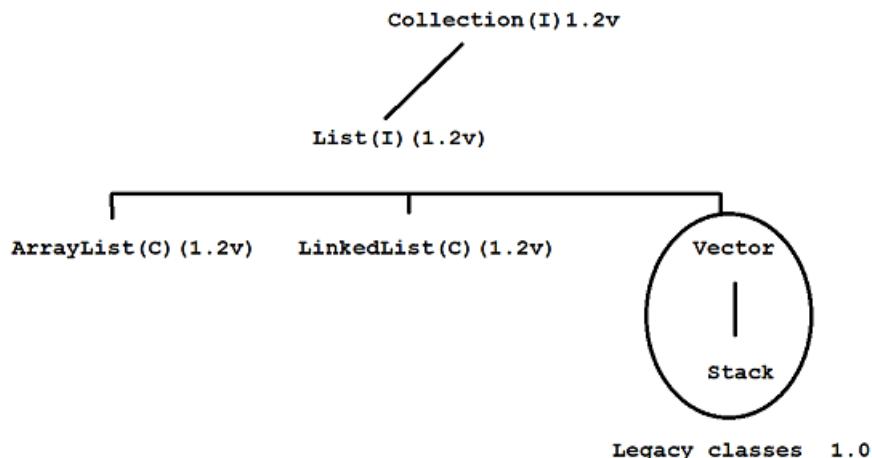
and etc

## List:

It is a child interface of Collection interface.

If we want to represent group of individual objects in a single entity where duplicates are allowed and order is preserved then we need to use List interface.

Diagram: java37.1



## ArrayList

- The underlying data structure is resizable array or growable array.
- Duplicate objects are allowed.
- Insertion order is preserved.
- Heterogenous objects are allowed.
- Null insertion is possible.
- ArrayList implements Serializable,Cloneable and RandomAccess interface.
- If our frequent operation is retrieval operation then ArrayList is a best choice.

ex:1

```
import java.util.*;  
  
class Test  
{  
  
    public static void main(String[] args)  
    {  
  
        ArrayList al=new ArrayList();  
  
        al.add("one");
```

```
    al.add("two");
    al.add("three");
    System.out.println(al); // [one, two, three]
    al.add("one");
    System.out.println(al); // [one, two, three, one]
    al.add(10);
    System.out.println(al); // [one, two, three, one, 10]
    al.add(null);
    System.out.println(al); // [one, two, three, one, 10, null]
}
}
```

ex:2

```
import java.util.*;
class Test
{
    public static void main(String[] args)
    {
        ArrayList<String> al=new ArrayList<String>();
        al.add("one");
        al.add("two");
        al.add("three");
        System.out.println(al); // [one, two, three]
        al.add("one");
        System.out.println(al); // [one, two, three, one]
        al.add(null);
        System.out.println(al); // [one, two, three, one, null]
    }
}
```



```
}

ex:3import java.util.*;

class Test

{

    public static void main(String[] args)

    {

        ArrayList<String> al=new ArrayList<String>();

        al.add("one");

        al.add("two");

        al.add("three");

        System.out.println(al);//[one, two, three]

        al.remove("one");

        System.out.println(al);//[two,three]

        System.out.println(al.contains("two"));//true

        al.clear();

        System.out.println(al); //[]

    }

}

ex:

import java.util.*;

class Test

{

    public static void main(String[] args)
```

```

{
    ArrayList<String> al=new ArrayList<String>();
    al.add("one");
    al.add("two");
    al.add("three");

    for(int i=0;i<al.size();i++)
    {
        String s=al.get(i);
        System.out.println(s);
    }
}

```



## LinkedList

- The underlying data structure is doubly LinkedList
- Duplicate objects are allowed.
- Insertion order is preserved.
- Heterogenous objects are allowed.
- Null insertion is possible.
- ArrayList implements Serializable,Cloneable and Deque interface.
- If our frequent operation is adding and removing in the middle then LinkedList is a best choice.

## Note:

LinkedList contains following methods.

ex:

```

addFirst(Object o)
removeFirst()
addLast(Object o)

```

```
removeLast()
```

```
getFirst()
```

```
getLast()
```

ex:1

```
import java.util.*;  
  
class Test  
{  
  
    public static void main(String[] args)  
    {  
  
        LinkedList<String> ll=new LinkedList<String>();  
  
        ll.add("one");  
  
        ll.add("two");  
  
        ll.add("three");  
  
        System.out.println(ll); // [one,two,three]  
  
        ll.add("one");  
  
        ll.add(null);  
  
        System.out.println(ll); // [one,two,three,one,null]  
    }  
}
```

ex:2

```
import java.util.*;  
  
class Test  
{  
  
    public static void main(String[] args)  
    {  
  
        LinkedList<String> ll=new LinkedList<String>();
```

```
ll.add("one");
ll.add("two");
ll.add("three");
System.out.println(ll);//[one,two,three]
ll.addFirst("gogo");
ll.addLast("jojo");
System.out.println(ll);//[gogo,one,two,three,jojo]
```

```
System.out.println(ll.getFirst());//gogo
```

```
System.out.println(ll.getLast());//jojo
```

```
ll.removeFirst();
```

```
ll.removeLast();
```

```
System.out.println(ll);//[one,two,three]
```

```
}
```

ex:3

```
import java.util.*;
class Test
{
    public static void main(String[] args)
    {
        LinkedList<String> ll1=new LinkedList<String>();
        ll1.add("one");
        ll1.add("two");
        ll1.add("three");
        System.out.println(ll1);//[one,two,three]
```

```

LinkedList<String> ll2=new LinkedList<String>();
ll2.add("raja");
System.out.println(ll2);//[raja]

ll2.addAll(ll1);
System.out.println(ll2);//[raja,one,two,three]

System.out.println(ll2.containsAll(ll1));//true

ll2.removeAll(ll1);
System.out.println(ll2);//[raja]

}

```



### 3)Vector

- The underlying data structure is resizable array or growable array.
- Duplicate objects are allowed.
- Insertion order is preserved
- Heterogenous objects are allowed.
- Null insertion is possible.
- ArrayList implements Serializable,Cloneable and RandomAccess interface.

**Note:**

All the methods present in Vector are called synchronized.

We have following methods in Vector class.

**ex:**

```

addElement()
removeElementAt()
removeAllElements()
firstElement()

```

**lastElement()**

and etc.

ex:

```
import java.util.*;
```

```
class Test
```

```
{
```

```
    public static void main(String[] args)
```

```
{
```

```
        Vector<Integer> v=new Vector<Integer>();
```

```
        System.out.println(v.capacity());//10
```

```
        for(int i=1;i<=10;i++)
```

```
{
```

```
        v.addElement(i);
```

```
}
```

```
        System.out.println(v);//[1,2,3,4,5,6,7,8,9,10]
```

```
        System.out.println(v.firstElement());//1
```

```
        System.out.println(v.lastElement());//10
```

```
        v.removeElementAt(5);
```

```
        System.out.println(v);//[1, 2, 3, 4, 5, 7, 8, 9, 10]
```

```
        v.removeAllElements();
```

```
        System.out.println(v);
```

```
    }  
}
```

## Q) Difference between ArrayList and Vector?

| ArrayList                    | Vector                        |
|------------------------------|-------------------------------|
| It is introduced in 1.2v.    | It is introduced in 1.0v.     |
| It is a non-legacy class.    | It is a legacy class.         |
| No methods are synchronized. | All methods are synchronized. |

## Stack

- It is a child class of Vector class
- If we depend upon Last In First Out order(LIFO) then we need to use stack.

## constructor

We can create a stack object as follow.



ex:

```
Stack s =new Stack();
```

## methods

### 1)push(Object o)

- It is used to insert the element to stack.

### 2)pop()

- It is used to remove the element from stack.

### 3)peek()

- It is used to read toppest element from stack.

### 4)isEmpty()

- It is used to check stack is empty or not.

## 5)search(Object o)

- It will return offset value if element is found in stack otherwise it will return -1.

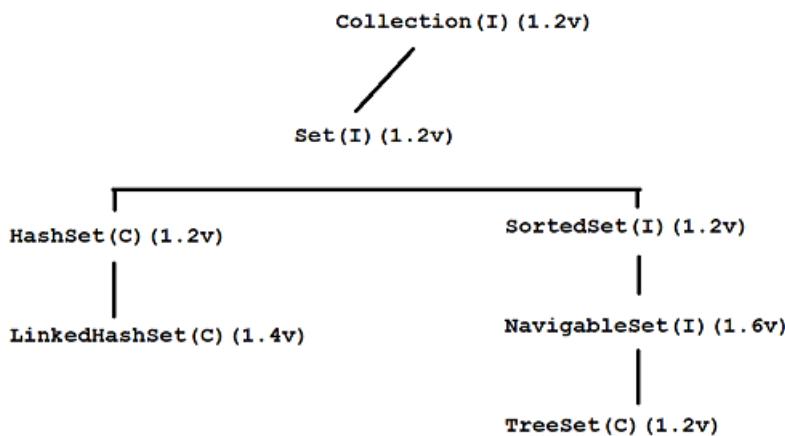
ex:

```
import java.util.*;  
  
class Test  
{  
  
    public static void main(String[] args)  
{  
  
        Stack<String> s=new Stack<String>();  
  
        s.push("A");  
  
        s.push("B");  
  
        s.push("C");  
  
        System.out.println(s);//[A,B,C]  
  
        Core Java  
  
        s.pop();  
  
        System.out.println(s);//[A,B]  
  
  
        System.out.println(s.peek());//B  
  
  
        System.out.println(s.isEmpty());//false  
  
  
        System.out.println(s.search("Z"));//-1  
  
  
        System.out.println(s.search("A"));//2  
    }  
}
```

## Set

- It is a child interface of Collection interface.
- If we want to represent group of individual objects in a single entity where duplicates are not allowed and order is not preserved then we need to use Set interface.

Diagram : java37.2



### HashSet

- The underlying data structure is Hashtable.
- Insertion order is not preserved because it is based on hash code of an object.
- Duplicate objects are not allowed.
- Heterogenous objects are allowed.
- Null insertion is possible.

ex:

```
import java.util.*;  
  
class Test  
{  
  
    public static void main(String[] args)  
    {  
  
        HashSet hs=new HashSet();  
  
        hs.add("one");  
  
        hs.add("nine");  
    }  
}
```

```

        hs.add("six");
        hs.add("four");
        System.out.println(hs); // [nine, six, four, one]
        hs.add("one");
        System.out.println(hs); // [nine, six, four, one]
        hs.add(null);
        System.out.println(hs); // [null, nine, six, four, one]
        hs.add(10);
        System.out.println(hs); // [null, nine, six, four, one, 10]
    }
}

```

## LinkedHashSet

- It is a child class of HashSet class.
- LinkedHashSet is exactly same as HashSet class with following differences.

| HashSet                                     | LinkedHashSet                                              |
|---------------------------------------------|------------------------------------------------------------|
| The underlying data structure is Hashtable. | The underlying data structure is Hashtable and LinkedList. |
| Insertion order is not preserved.           | Insertion order is preserved.                              |
| It is introduced in 1.2v.                   | It is introduced in 1.5v.                                  |

ex:

```

import java.util.*;
class Test
{
    public static void main(String[] args)
    {
        LinkedHashSet lhs=new LinkedHashSet();
        lhs.add("one");
        lhs.add("nine");
    }
}

```

```

    lhs.add("six");
    lhs.add("four");
    System.out.println(lhs); // [one, nine, six, four]
    lhs.add("one");
    System.out.println(lhs); // [one, nine, six, four]
    lhs.add(null);
    System.out.println(lhs); // [one, nine, six, four, null]
    lhs.add(10);
    System.out.println(lhs); // [one, nine, six, four, null, 10]
}
}

```

## TreeSet

- The underlying data structure is Balanced Tree.
- Insertion order is not preserved because it will take sorting order of object.
- Duplicate objects are not allowed.
- Heterogeneous objects are not allowed if we insert then we will get ClassCastException.
- For empty TreeSet if we are trying to insert null then we will get NullPointerException.
- After inserting some elements , if we are trying to insert null then we will get NullPointerException.

ex:

```

import java.util.*;
class Test
{
    public static void main(String[] args)
    {
        TreeSet ts=new TreeSet();

```

```

        ts.add(10);

        ts.add(1);

        ts.add(5);

        ts.add(3);

        ts.add(7);

        System.out.println(ts);//[1, 3, 5, 7, 10]

        ts.add(1);

        System.out.println(ts);//[1, 3, 5, 7, 10]

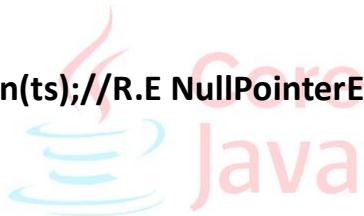
        //ts.add("hi");

        //System.out.println(ts);//R.E ClassCastException:

        //ts.add(null);

        //System.out.println(ts);//R.E NullPointerException
    }
}

```



**Q)What is the difference between Comparable and Comparator interface?**

### **Comparable**

A Comparable interface present in java.lang package.

If we dependent upon default natural sorting order then we need to use Comparable interface.

It contains only one method i.e compareTo() method.

ex:

**object1.compareTo(object2)**

**It will return -ve if object1 comes before object2.**

**It will return +ve if object1 comes after object2.**

**It will return 0 if both are same.**

**ex:**

```
class Test
```

```
{
```

```
    public static void main(String[] args)
```

```
{
```

```
        System.out.println("A".compareTo("Z")); // -25
```

```
        System.out.println("Z".compareTo("A")); // 25
```

```
        System.out.println("K".compareTo("K")); // 0
```

```
}
```

```
}
```

## Comparator



A Comparator interface present in java.util package.

If we depend upon customized sorting order then we need to use Comparator interface.

Comparator interface contains following two methods i.e compare() method and equals() method.

**ex:**

```
public int compare(obj1,obj2)
```

**It will return +ve if obj1 comes before object2.**

**It will return -ve if obj1 comes after object2.**

**It will return 0 if both are same.**

**Implementation of equals() method is optional because it is available in Object class which is inherited automatically through inheritance.**

**ex:**

```
import java.util.*;  
  
class Test  
{  
  
    public static void main(String[] args)  
    {  
  
        TreeSet<Integer> ts=new TreeSet<Integer>(new MyComparator());  
  
        ts.add(1);  
  
        ts.add(10);  
  
        ts.add(7);  
  
        ts.add(3);  
  
        System.out.println(ts);//[1, 3, 7, 10]  
    }  
}  
  
class MyComparator implements Comparator  
{  
  
    public int compare(Object obj1, Object obj2)  
    {  
  
        Integer i1=(Integer)obj1;  
  
        Integer i2=(Integer)obj2;  
  
  
        if(i1<i2)  
  
            return 1;  
  
        else if(i1>i2)  
  
            return -1;  
    }  
}
```



```

        else
            return 0;
    }
}

ex:
import java.util.*;
class Test
{
    public static void main(String[] args)
    {
        TreeSet<Integer> ts=new TreeSet<Integer>(new MyComparator());
        ts.add(1);
        ts.add(10);
        ts.add(7);
        ts.add(3);
        System.out.println(ts);//[1, 3, 7, 10]
    }
}

class MyComparator implements Comparator
{
    public int compare(Object obj1, Object obj2)
    {
        Integer i1=(Integer)obj1;
        Integer i2=(Integer)obj2;

        if(i1<i2)
            return -1;
    }
}

```



```

        else if(i1>i2)

            return 1;

        else

            return 0;

    }

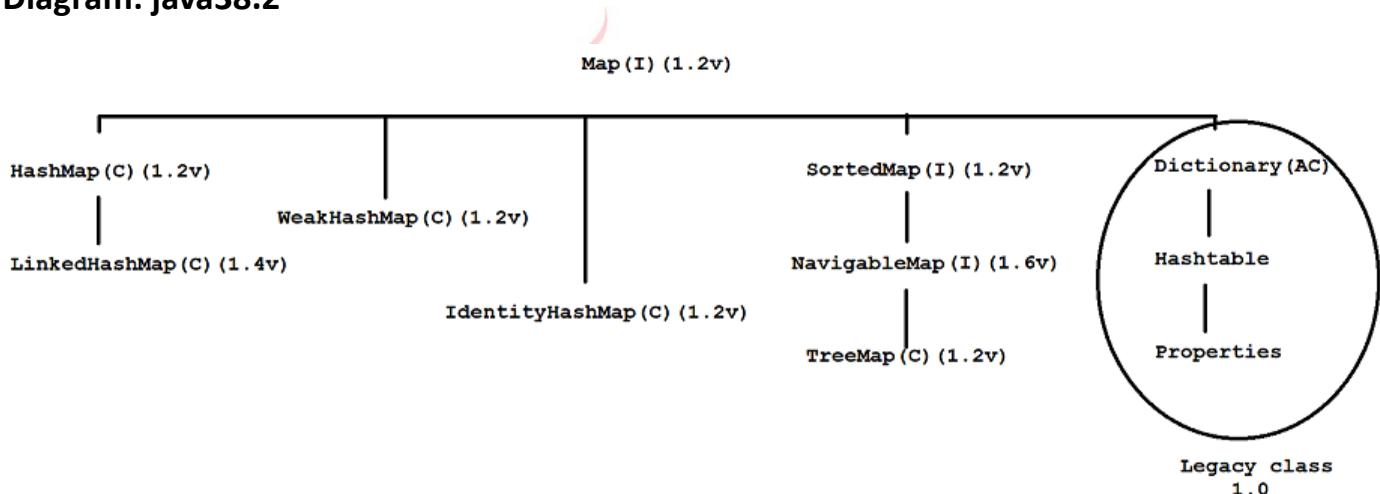
}

```

## Map

- It is not a child interface of Collection interface.
- If we want to represent group of individual objects in key-value pair then we need to use Map interface.
- Both key and value must be objects.
- A key can't be duplicate but value can be duplicate.
- Each key and value pair is called single entry / one entry.

Diagram: java38.2



## HashMap

- The underlying the data structure is Hashtable.
- Insertion order is not preserved because it is based on hash code of the keys.
- key can't be duplicate but value can be duplicate.
- Heterogenous objects are allowed for both keys and values.
- Null insertion is possible for both keys and values.

ex:

```

import java.util.*;
class Test
{
    public static void main(String[] args)
    {
        HashMap hm=new HashMap();
        hm.put("one","raja");
        hm.put("nine","alan");
        hm.put("six","jose");
        hm.put("five","clerk");
        System.out.println(hm);//{nine=alan, six=jose, one=raja, five=clerk}
        hm.put("one","gogo");
        System.out.println(hm);//{nine=alan, six=jose, one=gogo, five=clerk}
        hm.put("four","jose");
        System.out.println(hm);//{nine=alan, six=jose, four=jose, one=gogo,
five=clerk}
        hm.put(1,10);
        System.out.println(hm);//{nine=alan, 1=10, six=jose, four=jose, one=gogo,
five=clerk}
        hm.put(null,null);
        System.out.println(hm);//{null=null, nine=alan, 1=10, six=jose, four=jose,
one=gogo, five=clerk}}

```

## LinkHashMap

It is a child class of HashMap class.

LinkedHashMap is exactly same as HashMap class with following differences.

| HashMap                                     | LinkedHashMap                                              |
|---------------------------------------------|------------------------------------------------------------|
| The underlying data structure is Hashtable. | The underlying data structure is Hashtable and LinkedList. |
| Insertion order is not preserved.           | Insertion order is preserved.                              |

Introduced in 1.2v.

Introduced in 1.4v.

ex:

```
import java.util.*;  
  
class Test  
{  
  
    public static void main(String[] args)  
    {  
  
        LinkedHashMap lhm=new LinkedHashMap();  
  
        lhm.put("one","raja");  
  
        lhm.put("nine","alan");  
  
        lhm.put("six","jose");  
  
        lhm.put("five","clerk");  
  
        System.out.println(lhm);//{one=raja, nine=alan, six=jose, five=clerk}  
  
        lhm.put("one","gogo");  
  
        System.out.println(lhm);//{one=gogo, nine=alan, six=jose, five=clerk}  
  
        lhm.put("four","jose");  
  
        System.out.println(lhm);//{one=gogo, nine=alan, six=jose, five=clerk,  
four=jose}  
  
        lhm.put(1,10);  
  
        System.out.println(lhm);//{one=gogo, nine=alan, six=jose, five=clerk,  
four=jose, 1=10}  
  
        lhm.put(null,null);  
  
        System.out.println(lhm);//{one=gogo, nine=alan, six=jose, five=clerk,  
four=jose, 1=10, null=null}  
  
    }  
}
```

ex:

```

import java.util.*;
class Test
{
    public static void main(String[] args)
    {
        LinkedHashMap<String,String> lhm=new LinkedHashMap<String,String>();
        lhm.put("one","raja");
        lhm.put("nine","alan");
        lhm.put("six","jose");
        lhm.put("five","clerk");

        Set s=lhm.keySet();
        System.out.println(s);//[one,nine,six,five]

        Collection c=lhm.values();
        System.out.println(c);//[raja,alan,jose,clerk]

        Set s1=lhm.entrySet();
        System.out.println(s1);//[one=raja, nine=alan, six=jose, five=clerk]
    }
}

```



## TreeMap

- The underlying data structure is Red Black Tree.
- Insertion order is not preserved because it will sorting order of a key.
- Duplicate keys are not allowed but values can be duplicated.
- If we depend upon default natural sorting order then keys must be homogeneous and Comparable.
- If we depend upon customized sorting order then keys must be heterogeneous and Non Comparable.

- Key can't be null but value can be null.

ex:

```
import java.util.*;  
  
class Test  
{  
  
    public static void main(String[] args)  
    {  
  
        TreeMap<Integer,String> tm=new TreeMap<Integer,String>();  
  
        tm.put(1,"one");  
  
        tm.put(4,"four");  
  
        tm.put(2,"two");  
  
        tm.put(8,"eight");  
  
        System.out.println(tm); // {1=one, 2=two, 4=four, 8=eight}  
  
        tm.put(7,null);  
  
        System.out.println(tm); // {1=one, 2=two, 4=four, 7=null, 8=eight}  
  
        tm.put(null,"nine");  
  
        System.out.println(tm); // R.E NullPointerException  
  
    }  
  
}
```

## Hashtable

- The underlying data structure is a Hashtable.
- Insertion order is sorting order of the keys.
- Duplicate keys are not allowed but values can be duplicate.
- Hetrogeneous objects are allowed for both keys and values.
- Null key and value both are not allowed otherwise we will get NullPointerException.

ex:

```
import java.util.*;  
  
class Test
```

```

{
    public static void main(String[] args)
    {
        Hashtable<Integer, String> ht = new Hashtable<Integer, String>();
        ht.put(1, "one");
        ht.put(10, "ten");
        ht.put(3, "three");
        ht.put(7, "seven");
        System.out.println(ht); // {10=ten, 7=seven, 3=three, 1=one}

        // ht.put(null, "eight");
        // System.out.println(ht); // R.E NullPointerException

        // ht.put(8, null);
        // System.out.println(ht); // R.E NullPointerException
    }
}

```



## Types of cursors in java

- Cursor is used to read object one by one from Collections.
- We have three types of cursors in java.

1) Enumeration

2) Iterator

3) ListIterator

### 1) Enumeration

- It is used to read objects one by one from legacy collection objects.
- We can create Enumeration object as follow.

ex:

```
Enumeration e=v.elements();
```

Enumeration interface contains following two methods.

ex:

```
public boolean hasMoreElements();
```

```

public Object nextElement();

ex:

import java.util.*;

class Test

{
    public static void main(String[] args)
    {
        Vector v=new Vector();

        for(int i=1;i<=10;i++)
        {
            v.add(i);
        }
    }
}

```

System.out.println(v); // [1,2,3,4,5,6,7,8,9,10]

```
Enumeration e=v.elements();
```

```
while(e.hasMoreElements())
{
    Integer i=(Integer)e.nextElement();
    System.out.println(i);
}}
```

### Limitations with Enumeration

- Enumeration object is used to read objects only from legacy collections. Hence it is not a universal cursor.
- Using Enumeration we can perform read operation but not remove operation.
- To overcome this limitation Sun Micro system introduced Iterator interface.

### 2) Iterator

- Iterator interface is used to read objects one by one from any collection objects. Hence it is a universal cursor.
- Using Iterator interface we can perform read and remove operations.

We can create Iterator object as follow.

ex:

```
Iterator itr=al.iterator();
```

**Iterator interface contains following three methods.**

ex:

```
public boolean hasNext()  
public Object next()  
public void remove()
```

ex:

```
import java.util.*;  
  
class Test  
{  
  
    public static void main(String[] args)  
    {  
  
        ArrayList al=new ArrayList();  
  
        for(int i=1;i<=10;i++)  
        {  
            al.add(i);  
        }  
  
        System.out.println(al);//[1,2,3,4,5,6,7,8,9,10]  
    }  
}
```

```
Iterator itr=al.iterator();
```

```
while(itr.hasNext())  
{  
    Integer i=(Integer)itr.next();  
    if(i%2==0)
```

```

        {
            itr.remove();
        }
        else
        {
            System.out.println(i);
        }
    }

    System.out.println(al);//[1,3,5,7,9]
}

}

```

### Limitations with Iterator

Using Enumeration and Iterator we can read objects only in forward direction but in backward direction.Hence they are not bi-directional cursors.



Using Iterator interface we can perform read and remove operation but not adding and replacement of new objects.

To overcome this limitation sun micro system introduced ListIterator interface.

### 3)ListIterator

- It is a child interface of Iterator interface.
- If we want to read objects one by one from List Collection objects then we need to use ListIterator.
- Using ListIterator interface we can perform read, remove ,adding and replacement of new objects.

We can create ListIterator object as follow.

ex:

```
ListIterator litr=al.listIterator();
```

ListIterator interface contains following 9 methods.

ex:

```
public boolean hasNext();
public Object next();
```

```
public boolean hasPrevious();

public Object previous();

public int nextIndex();

public int previousIndex();

public void remove();

public void add(Object);

public void set(Object);
```

ex:

```
import java.util.*;

class Test

{

    public static void main(String[] args)

    {

        ArrayList al=new ArrayList();

        al.add("chiru");

        al.add("venki");

        al.add("bala");

        al.add("nag");

        System.out.println(al);//[chiru, venki, bala, nag]
```



```
ListIterator litr=al.listIterator();

while(litr.hasNext())

{

    String s=(String)litr.next();

    System.out.println(s);

}

}}
```

ex:

```
import java.util.*;

class Test

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

```

{
    ArrayList al=new ArrayList();
    al.add("chiru");
    al.add("venki");
    al.add("bala");
    al.add("nag");
    System.out.println(al);//[chiru, venki, bala, nag]
    ListIterator litr=al.listIterator();
    while(litr.hasNext())
    {
        String s=(String)litr.next();
        if(s.equals("bala"))
        {
            litr.remove();
        }
    }
    System.out.println(al);//[chiru, venki, nag]}
}

```



ex:

```

import java.util.*;
class Test
{
    public static void main(String[] args)
    {
        ArrayList al=new ArrayList();
        al.add("chiru");
        al.add("venki");
        al.add("bala");
        al.add("nag");
        System.out.println(al);//[chiru, venki, bala, nag]
    }
}

```

```

ListIterator litr=al.listIterator();

while(litr.hasNext())
{
    String s=(String)litr.next();
    if(s.equals("venki"))
    {
        litr.add("pavan");
    }
}

System.out.println(al); // [chiru, venki, pavan, bala, nag]}

```

ex:

```

import java.util.*;
class Test
{
    public static void main(String[] args)
    {
        ArrayList al=new ArrayList();
        al.add("chiru");
        al.add("venki");
        al.add("bala");
        al.add("nag");
        System.out.println(al); // [chiru, venki, bala, nag]
    }
}

```

```

ListIterator litr=al.listIterator();

while(litr.hasNext())
{
    String s=(String)litr.next();
    if(s.equals("venki"))

```

```

    {
        litr.set("pavan");
    }
}

System.out.println(al);//[chiru, pavan, bala, nag]{}}

```

## Interview Programs

**Q) Write a java program to display number of occurrence of a word in a string?**

**Input:**

I am am learning java java

**output:**

I-1, am-2 , learning-1, java-2



**ex:**

```
import java.util.*;
```

```
class Test
```

```
{
```

```
public static void main(String[] args)
```

```
{
```

```
String str="I am am learning java java";
```

```
LinkedHashMap<String,Integer> lhm=new LinkedHashMap<String,Integer>();
```

```
String[] sarr=str.split(" ");
```

```
//for each loop
```

```
for(String s:sarr)
```

```
{
```

```

        if(lhm.get(s)!=null)
        {
            lhm.put(s,lhm.get(s)+1);
        }
        else
        {
            lhm.put(s,1);
        }
    }
    System.out.println(lhm);
}
}

```

**Q) Write a java program to display number of occurrence of a characters in a string?**

**Input:**

java

**output:**

j-1, a-2, v-1



**ex:**

```

import java.util.*;
class Test
{
    public static void main(String[] args)
    {
        String str="java";
        LinkedHashMap<Character,Integer> lhm=new LinkedHashMap<Character,Integer>();
        char[] carr=str.toCharArray();
        for(int i=0;i<carr.length;i++)
        {
            if(lhm.get(carr[i])!=null)
            {
                lhm.put(carr[i],lhm.get(carr[i])+1);
            }
            else
            {
                lhm.put(carr[i],1);
            }
        }
        System.out.println(lhm);
    }
}

```

```

//for each loop

for(char c:carr)
{
    if(lhm.get(c)!=null)
    {
        lhm.put(c,lhm.get(c)+1);
    }
    else
    {
        lhm.put(c,1);
    }
}
System.out.println(lhm);
}
}

```



**Q)Write a java program to perform sum of two binary number?**

**input:**

```

1010
0101

```

**output:**

```
1111
```

**ex:**

```

import java.util.*;
class Test
{
    public static void main(String[] args)
    {
        Scanner sc=new Scanner(System.in);

```

```

System.out.println("Enter the First Binary number :");

String bin1=sc.next()//1010

System.out.println("Enter the Second Binary number :");

String bin2=sc.next()//0101

//converting binary to decimal

int a=Integer.parseInt(bin1,2);

int b=Integer.parseInt(bin2,2);

int c=a+b;

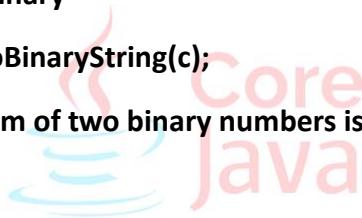
//convert decimal to binary

String result=Integer.toBinaryString(c);

System.out.println("sum of two binary numbers is =" +result);

}

```



## Assignment

**Q) Write a java program for below code?**

**input:**

A1B2C3D4

**output:**

ABBCCCDDDD

```

import java.util.*;

class Test

{
    public static void main(String[] args)
    {
        String str="A2B3C1";
    }
}

```

```

for(int i=0;i<str.length();i++)
{
    if(Character.isAlphabetic(str.charAt(i)))
    {
        System.out.print(str.charAt(i));
    }
    else
    {
        int a=Character.getNumericValue(str.charAt(i));
        for(int j=1;j<a;j++)
        {
            System.out.print(str.charAt(i-1));
        }
    }
}

```

**Q)Write a java program for below code?**

**input:**

ABBCCCDDDD



**output:**

A1B2C3D4

### Multi-threading

**Q)What is the difference between Thread and Process?**

#### Thread

- A thread is a light weight sub process.
- We can run multiple threads concurrently.
- One thread can communicate with another thread.

**ex:**

**class is one thread**

**constructor is one thread**

**block is one thread**

**and etc.**

## Process

- A process is a collection of threads.
- We can run multiple process concurrently.
- One process can't communicate with another process.

ex:

Taking a class using zoom meeting is one process.

Typing notes in notepad is one process.

Downloading a file from internet is one process.

## Multitasking

Executing several task simultaneously such concept is called multitasking.

We have two types of multitasking.

### **1) Thread based multitasking**

- Executing several task simultaneously where each task is a same part of a program.
- It is best suitable for programmatic level.

### **2) Process based multitasking**

- Executing several task simultaneously where each task is an independent process.
- It is best suitable for OS level.



## Multithreading

- Executing several threads simultaneously such concept is called multithreading.
- In multithreading only 10% of work should be done by the programmer and 90% of work will be done by JAVA API.
- The main important application area of multithreading are.

1) To implements multimedia graphics.

2) To develop animations.

3) To develop video games.

### **How to start a thread in java**

There are two ways to start / create a thread in java.

1) By extending Thread class

2) By implementing Runnable interface

### **1) By extending Thread class**

class MyThread extends Thread

```

}

//work of a thread

public void run()
{
    for(int i=1;i<=5;i++)
    {
        System.out.println("Child-Thread");
    }
}

class Test
{
    public static void main(String[] args)
    {
        MyThread t=new MyThread();

        //start a thread
        t.start();

        for(int i=1;i<=5;i++)
        {
            System.out.println("Parent-Thread");
        }
    }
}

```



### case1: Thread Schedular

- If multiple threads are waiting for execution which thread should execute will decide by thread scheduler.
- What algorithm, mechanism or behaviour used by thread scheduler is depends upon JVM vendor.
- Hence we can't expect any execution order or exact output in multithreading.

### case2: Difference between t.start() and t.run()

If we call `t.start()` method then a new thread will be created which is responsible to execute `run()` method automatically.

ex:

```
class MyThread extends Thread  
{  
    //work of a thread  
    public void run()  
    {  
        for(int i=1;i<=5;i++)  
        {  
            System.out.println("Child-Thread");  
        }  
    }  
}  
  
class Test  
{  
    public static void main(String[] args)  
    {  
        MyThread t=new MyThread();  
  
        //start a thread  
        t.start();  
  
        for(int i=1;i<=5;i++)  
        {  
            System.out.println("Parent-Thread");  
        }  
    }  
}
```



If we call t.run() method then no new thread will be created but run() method will execute just like a normal method.

ex:

```
class MyThread extends Thread  
{  
    //work of a thread  
    public void run()  
    {  
        for(int i=1;i<=5;i++)  
        {  
            System.out.println("Child-Thread");  
        }  
    }  
}  
  
class Test  
{  
    public static void main(String[] args)  
    {  
        MyThread t=new MyThread();  
  
        //no new thread  
        t.run();  
  
        for(int i=1;i<=5;i++)  
        {  
            System.out.println("Parent-Thread");  
        }  
    }  
}
```

**case3: If we won't override run() method**

If we won't override run() method then t.start() method will execute Thread class run() method automatically.

Thread class run() method is empty implementation so we won't get any output from child thread.

ex:

```
class MyThread extends Thread
{
}

class Test
{
    public static void main(String[] args)
    {
        MyThread t=new MyThread();
        //new thread
        t.start();

        for(int i=1;i<=5;i++)
        {
            System.out.println("Parent-Thread");
        }
    }
}
```



#### case4: If we overload run() method

If we overload run() method then Thread class start() method always execute run() method with no argument method.

ex:

```
class MyThread extends Thread
{
    public void run()
```

```

{
    System.out.println("0-arg method");
}
public void run(int i)
{
    System.out.println("int-arg method");
}
}

class Test
{
    public static void main(String[] args)
    {
        MyThread t=new MyThread();

        //new thread
        t.start();

        for(int i=1;i<=5;i++)
        {
            System.out.println("Parent-Thread");
        }
    }
}

```



### case5: If we override start() method

If we override start() method then no new thread will be created but start() method will execute just like normal method.

ex:

```

class MyThread extends Thread
{
    public void run()

```

```

{
    System.out.println("run method");
}

public void start()
{
    System.out.println("start method");
}

}

class Test
{

    public static void main(String[] args)
    {
        MyThread t=new MyThread();

        //no new thread
        t.start();

        for(int i=1;i<=5;i++)
        {
            System.out.println("Parent-Thread");
        }
    }
}

```

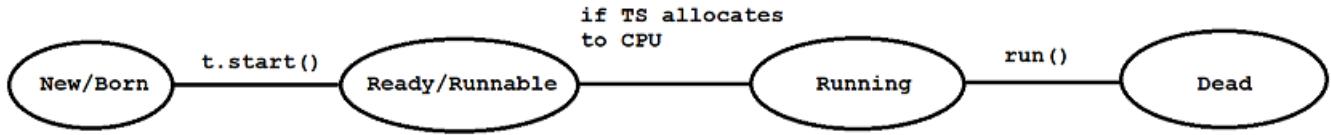


### case6: Life cycle of a thread

- In java, all the threads will store in java stack memory.

Diagram: java40.1

```
MyThread t=new MyThread(); // intantiate a thread  
t.start(); // start a thread
```



- Once if we create a thread then our thread will be in New/Born state.
- Once if we call t.start() method then our thread goes to Ready/Runnable state.
- If thread scheduler allocates to CPU then our thread will enter to Running state.
- Once the run() method execution is completed then our thread will goes to dead state.

## 2)By implementing Runnable interface

```
class MyRunnable implements Runnable  
{  
    public void run()  
    {  
        for(int i=1;i<=5;i++)  
        {  
            System.out.println("Child-Thread");  
        }  
    }  
}  
  
class Test  
{  
    public static void main(String[] args)  
    {  
        MyRunnable r = new MyRunnable();  
  
        //create a Thread class object
```



```

Thread t=new Thread(r); // r is a targatable interface

//start a thread

t.start();

for(int i=1;i<=5;i++)
{
    System.out.println("Parent-Thread");
}
}

```

### Setting and Getting the name of a thread

In java, every thread has a name. Explicitly provided by the programmer or automatically generated by JVM.

We have following methods to set and get name of a thread.

ex:

```

public final void setName(String name);

public final String getName();

```



ex:

---

```
class MyThread extends Thread
```

```
{
```

```
}
```

```
class Test
```

```
{
```

```
    public static void main(String[] args)
```

```
{
```

```
        System.out.println(Thread.currentThread().getName());//main
```

```

MyThread t=new MyThread();

System.out.println(t.getName());//Thread-0

Thread.currentThread().setName("Parent-Thread");

System.out.println(Thread.currentThread().getName());//Parent-Thread

t.setName("Child-Thread");

System.out.println(t.getName());

}

}

```

### Thread priority

- In java, every thread has a priority. Explicitly provided by the programmer or automatically generated by JVM.
- The valid range of thread priority is 1 to 10. Where 1 is a least priority and 10 is a highest priority.
- If we take more than 10 priority then we will get IllegalArgumentException.
- Thread class defines following standard constants as thread priority.

ex:

Thread.MAX\_PRIORITY - 10

Thread.MIN\_PRIORITY - 1

Thread.NORM\_PRIORITY - 5

- We don't have such constants like LOW\_PRIORITY and HIGH\_PRIORITY.
- Thread scheduler uses thread priority while allocating to CPU.
- A thread which is having highest priority will be executed first.
- If multiple threads having same priority then we can't expect any execution order.

We have following methods to set and get thread priority.

ex:

```

public final void setPriority(int newpriority);

public final int getPriority();

```

ex:

class MyThread extends Thread

```

}

}

class Test
{
    public static void main(String[] args)
    {
        System.out.println(Thread.currentThread().getPriority());// 5

        MyThread t=new MyThread();
        System.out.println(t.getPriority());//5

        Thread.currentThread().setPriority(9);
        System.out.println(Thread.currentThread().getPriority());//9

        System.out.println(t.getPriority());//5

        t.setPriority(4);
        System.out.println(t.getPriority());//4

        t.setPriority(11); // R.E IllegalArgumentException}
    }
}

```



### In how many ways we can prevent a thread from execution

We can prevent(stop) a thread from execution in three ways.

- 1) yield()
- 2) join()
- 3) sleep()

#### 1) yield()

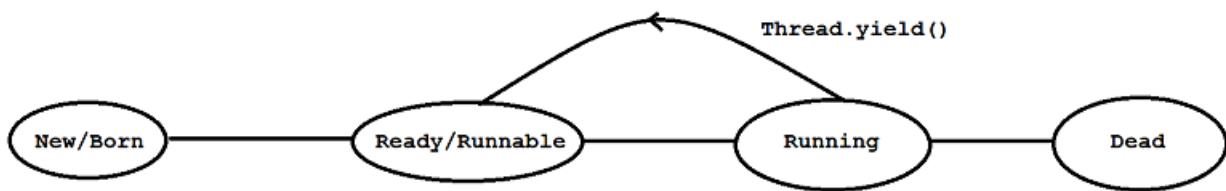
- It will pause current execution thread and gives the chance to other threads having same priority.
- If there is no waiting threads then same thread will continue it's execution.
- If multiple threads are waiting for execution then we can't expect any execution order.

- A thread which is yielded when it will get a chance for execution is depends upon mercy of thread scheduler.

ex:

```
public static native void yield()
```

Diagram: java40.2



ex:

```
class MyThread extends Thread
```

```
{
    public void run()
    {
        for(int i=1;i<=5;i++)
        {
            System.out.println("Child-Thread");
        }
    }
}

class Test
{
    public static void main(String[] args)
    {
        MyThread t=new MyThread();
        t.start();
        for(int i=1;i<=5;i++)
        {
    }
```



```

        Thread.currentThread().yield();
        System.out.println("Parent-Thread");
    }}}
```

## 2)join()

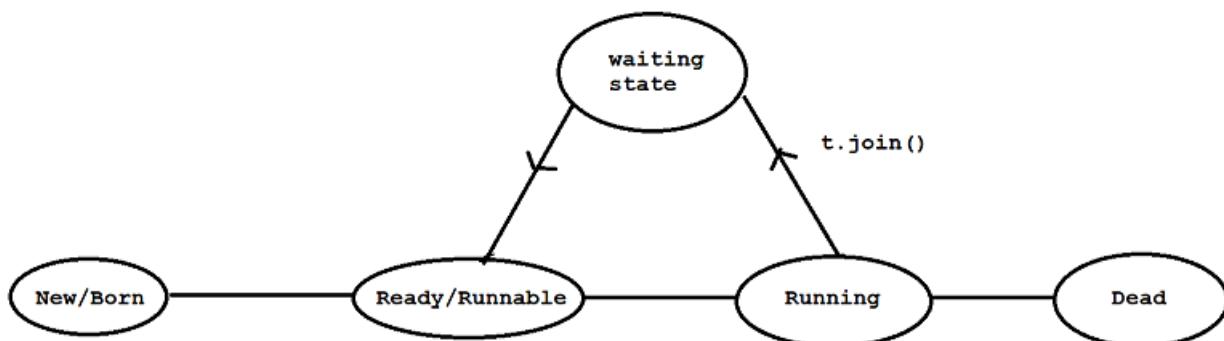
- If a thread wants to wait until the completion of some other thread then we need to use join() method.
- A join() method will throw one checked exception called InterruptedException so we must and should handle that exception by using try and catch block or by using throws statement.

ex:

```

public final void join()throws InterruptedException
public final void join(long ms)throws InterruptedException
public final void join(long ms,int ns)throws InterruptedException
```

Diagram: java40.3



ex:

```

class MyThread extends Thread
{
    public void run()
    {
        for(int i=1;i<=5;i++)
        {
            System.out.println("Child-Thread");
        }
    }
}
```

```

class Test
{
    public static void main(String[] args) throws InterruptedException
    {
        MyThread t=new MyThread();
        t.start();
        t.join();
        for(int i=1;i<=5;i++)
        {
            System.out.println("Parent-Thread");
        }
    }
}

```

### 3)sleep()

- If a thread don't want to perform any operation on particular amount of time then we need to use sleep() method.
- A sleep() method will throw one checked exception called InterruptedException so we must and should handle that exception by using try and catch block or by using throws statement.

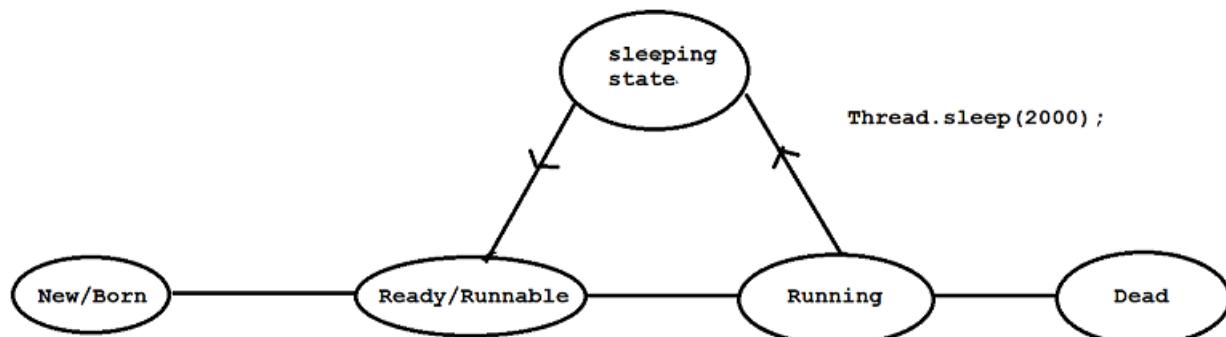
ex:

```

public static native void sleep()throws InterruptedException
public static native void sleep(long ms)throws InterruptedException
public static native void sleep(long ms,int ns)throws InterruptedException

```

Diagram: java40.4



ex:

```
class MyThread extends Thread
{
    public void run()
    {
        for(int i=1;i<=5;i++)
        {
            System.out.println("Child-Thread");
            try
            {
                Thread.sleep(2000);
            }
            catch (InterruptedException ie)
            {
                ie.printStackTrace();
            }
        }
    }
}

class Test
{
    public static void main(String[] args)
    {
        MyThread t=new MyThread();
        t.start();
        for(int i=1;i<=5;i++)
        {
            System.out.println("Parent-Thread");
        }
    }
}
```



## Daemon Thread

- Daemon thread is a service provider thread which provides services to user threads.
- There are many deamon thread are running internally i. garbage collector , finalizer and etc.
- Life of daemon thread is depends upon user threads.When user threads dies then daemon thread will terminate automatically.
- We can start a deamon thread by using setDaemon(true) method.
- To check , a thread is a daemon or not we need to use isDaemon() method.

ex:

```
class MyThread extends Thread  
{  
    public void run()  
    {  
        for(int i=1;i<=5;i++)  
        {  
            System.out.println(Thread.currentThread().isDaemon());  
            System.out.println("Child-Thread");  
        }  
    }  
}  
  
class Test  
{  
    public static void main(String[] args)  
    {  
        MyThread t=new MyThread();  
        t.setDaemon(true);  
        t.start();  
        for(int i=1;i<=5;i++)  
        {  
            System.out.println("Parent-Thread");  
        }  
    }  
}
```



## Problems without Synchronization

If there is no synchronization then we will face following problems.

1) Data inconsistency

2) Thread interference

ex:

class Table

{

```
void printTable(int n)
{
    for(int i=1;i<=5;i++)
    {
        System.out.println(n*i);
        try
        {
            Thread.sleep(2000);
        }
        catch (InterruptedException ie)
        {
            ie.printStackTrace();
        }
    }
}
```



class MyThread1 extends Thread

{

Table t;

MyThread1(Table t)

{

this.t=t;

}

public void run()

```

    {
        t.printTable(5);
    }
}

class MyThread2 extends Thread
{
    Table t;
    MyThread2(Table t)
    {
        this.t=t;
    }
    public void run()
    {
        t.printTable(10);
    }
}
class Test
{
    public static void main(String[] args)
    {
        Table obj=new Table();
        MyThread1 t1=new MyThread1(obj);
        MyThread2 t2=new MyThread2(obj);
        t1.start();
        t2.start();
    }
}

```



## synchronization

- A synchronized keyword is applicable for methods and blocks.
- A synchronization is allowed one thread to execute given object.Hence we achieve thread safety.
- The main advantage of synchronization is we solve data inconsistence problem.

- The main disadvantage of synchronization is ,it will increase waiting time of a thread which reduce the performance of the system.
- If there is no specific requirement then it is never recommended to use synchronization concept.
- synchronization internally uses lock mechanism.
- Whenever a thread wants to access object , first it has to acquire lock of an object and thread will release the lock when it completes it's task.
- When a thread wants to execute synchronized method.It automatically gets the lock of an object.
- When one thread is executing synchronized method then other threads are not allowed to execute other synchronized methods in a same object concurrently.But other threads are allowed to execute non-synchronized method concurrently.

ex:

```

class Table
{
    synchronized void printTable(int n)
    {
        for(int i=1;i<=5;i++)
        {
            System.out.println(n*i);
            try
            {
                Thread.sleep(2000);
            }
            catch (InterruptedException ie)
            {
                ie.printStackTrace();
            }
        }
    }
}

class MyThread1 extends Thread
{
    Table t;
    MyThread1(Table t)
}

```

```

    {
        this.t=t;
    }

    public void run()
    {
        t.printTable(5);
    }
}

class MyThread2 extends Thread
{
    Table t;

    MyThread2(Table t)
    {
        this.t=t;
    }

    public void run()
    {
        t.printTable(10);
    }
}

class Test
{
    public static void main(String[] args)
    {
        Table obj=new Table();
        MyThread1 t1=new MyThread1(obj);
        MyThread2 t2=new MyThread2(obj);
    }
}

```



```
        t1.start();
        t2.start();
    }
}
```

## **synchronized block**

- If we want to perform synchronization on specific resource of a program then we need to use synchronization.

ex:

If we have 100 lines of code and if we want to perform synchronization only for 10 lines then we need to use synchronized block.

If we keep all the logic in synchronized block then it will act as a synchronized method.

ex:

```
class Table
{
    void printTable(int n)
    {
        synchronized(this)
        {
            for(int i=1;i<=5;i++)
            {
                System.out.println(n*i);
                try
                {
                    Thread.sleep(2000);
                }
                catch (InterruptedException ie)
                {
                    ie.printStackTrace();
                }
            }
        }
    }
}
```



```
        }

    }

}//sync

}

class MyThread1 extends Thread

{

    Table t;

    MyThread1(Table t)

    {

        this.t=t;

    }

    public void run()

    {

        t.printTable(5);

    }

}
```



```
class MyThread2 extends Thread

{

    Table t;

    MyThread2(Table t)

    {

        this.t=t;

    }

    public void run()

    {

        t.printTable(10);

    }

}
```

```

class Test
{
    public static void main(String[] args)
    {
        Table obj=new Table();
        MyThread1 t1=new MyThread1(obj);
        MyThread2 t2=new MyThread2(obj);

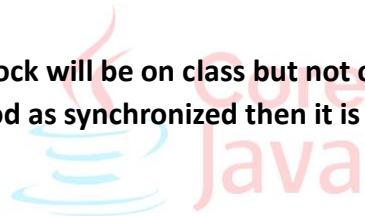
        t1.start();
        t2.start();
    }
}

```

### 3)Static synchronization

- In static synchronization the lock will be on class but not on object.
- If we declare any static method as synchronized then it is called static synchronization method.

ex:



```

class Table
{
    static synchronized void printTable(int n)
    {

        for(int i=1;i<=5;i++)
        {
            System.out.println(n*i);
            try
            {
                Thread.sleep(2000);
            }
            catch (InterruptedException ie)
        }
    }
}

```

```
        {
            ie.printStackTrace();
        }
    }

}

class MyThread1 extends Thread
{

    public void run()
    {
        Table.printTable(5);
    }
}

class MyThread2 extends Thread
{

    public void run()
    {
        Table.printTable(10);
    }
}

class Test
{
    public static void main(String[] args)
    {
```



```

    MyThread1 t1=new MyThread1();

    MyThread2 t2=new MyThread2();

    t1.start();

    t2.start();

}
}

```

## Marker interface

- Interface which does not have any methods or constants is called marker interface.
- Empty interface is called marker interface.
- By using marker interface we will get some ability to do.

ex:

Serializable

Remote

Cloneable



## Java 1.8 Features

### Functional Interface

An interface that contains exactly one abstract method is known as functional interface.

ex:

Runnable -> run()

Comparable -> compareTo()

ActionListener -> actionPerformed()

It can have any number of default and static methods.

Function interface is also known as Single Abstract Method Interface or SAM interface.

It is a new feature in java which helps in to achieve functional programming .

ex:

a=f1(){}

```
f1(f2(){})

{
}
```

**@FunctionalInterface** is a annotation which is used to declare functional interface and it is optional.

ex:1

```
interface A

{
    void show();
}

class B implements A

{
    public void show()
    {
        System.out.println("A- show method");
    }
}

class Test

{
    public static void main(String[] args)
    {
        B b=new B();
        b.show();
    }
}
```

ex:2



```
interface A
{
    void show();
}

class Test
{
    public static void main(String[] args)
    {
        A a=new A()
        {
            public void show()
            {
                System.out.println("From show method");
            }
        };
        a.show();
    }
}
```

ex:3

@FunctionalInterface

interface A

```
{

    void show();
}
```

```

class Test
{
    public static void main(String[] args)
    {
        A a=new A()
        {
            public void show()
            {
                System.out.println("From show method");
            }
        };
        a.show();
    }
}

```



## Lamda Expression

- Lamda Expression introduced in java 1.8v.
- Lamda Expression is used to enable functional programming.
- It is used to concise the code (To reduce the code).
- Lamda Expression can be used when we have functional interface.
  
- Lamda expression considered as a method not a class.
- Lamda expression does not support modifier,return type and method name.

ex:

### without lamda expression

```

public void m1()
{

```

```
        System.out.println("Hello');  
    }  
  
}
```

### with lamda expression

```
()->  
{  
    System.out.println("Hello');  
};
```

ex:

```
@FunctionalInterface
```

```
interface A
```

```
{  
    void show();  
}
```



```
class Test
```

```
{  
    public static void main(String[] args)  
    {  
        A a=()->  
        {  
            System.out.println("From show method");  
        };  
        a.show();  
    }  
}
```

ex:2

```
@FunctionalInterface  
  
interface A  
{  
    String show();  
}  
  
class Test  
{  
    public static void main(String[] args)  
    {  
        A a=()->  
        {  
            return "hello world";  
        };  
        System.out.println(a.show());  
    }  
}
```

ex:3

```
@FunctionalInterface  
  
interface A  
{  
    void show(int i,int j);  
}  
  
class Test  
{  
    public static void main(String[] args)  
    {  
    }
```

```

A a=(int i,int j)->

{
    System.out.println("sum of two number is =" +(i+j));
}

a.show(10,20);

}

```

ex:4

**@FunctionalInterface**

**interface A**

```
{
    int show(int i,int j);
}
```

**class Test**

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

A a=(int i,int j)->

```
{
    return i+j;
};
```

System.out.println("sum of two number is =" +a.show(10,20));

```
}
```

## Stream API

- If we want to process the objects from Collections then we need to use Stream API.

- Stream is an interface which is present in java.util.stream package.
- Stream is use to perform bulk operations on Collections.
- We can create Stream object as follow.

### **syntax**

```
Stream s=c.stream();
```

**ex:1**

```
import java.util.*;
import java.util.stream.*;
class Test
{
    public static void main(String[] args)
    {
        List<Integer> l=new ArrayList<Integer>();
        l.add(10);
        l.add(4);
        l.add(1);
        l.add(7);
        l.add(6);
        l.add(9);
        System.out.println(l);//[10,4,1,7,6,9]
```



```
//to display even elements
```

```
List<Integer> list=l.stream().filter(i->i%2==0).collect(Collectors.toList());
System.out.println(list);
}
```

**ex:2**

```
import java.util.*;
```

```
import java.util.stream.*;
class Test
{
    public static void main(String[] args)
    {
        List<Integer> l=new ArrayList<Integer>();
        l.add(10);
        l.add(4);
        l.add(1);
        l.add(7);
        l.add(6);
        l.add(9);
        System.out.println(l);//[10,4,1,7,6,9]
    }
}
```

 Core Java

```
//to display even elements
List<Integer> list=l.stream().filter(i->i%2!=0).collect(Collectors.toList());
System.out.println(list);
}
```

ex:3

```
import java.util.*;
import java.util.stream.*;
class Test
{
    public static void main(String[] args)
```

```

{
    List<Integer> l=new ArrayList<Integer>();
    l.add(45);
    l.add(89);
    l.add(76);
    l.add(49);
    l.add(65);
    l.add(59);
    System.out.println(l);//[45,89,76,49,65,59]

    //Add 10 grace marks
    List<Integer> list=l.stream().map(i->i+10).collect(Collectors.toList());
    System.out.println(list);//[55,99,86,59,75,69]
}

```



ex:4

```

import java.util.*;
import java.util.stream.*;
class Test
{
    public static void main(String[] args)
    {
        List<Integer> l=new ArrayList<Integer>();
        l.add(45);
        l.add(89);
        l.add(76);
        l.add(25);
    }
}

```

```
I.add(65);  
I.add(15);  
System.out.println(I);//[45,89,76,25,65,15]
```

```
//How many students are failed  
long failed=I.stream().filter(i->i<35).count();  
System.out.println(failed);  
  
}  
}
```

ex:5

```
import java.util.*;  
import java.util.stream.*;  
class Test  
{  
    public static void main(String[] args)  
    {  
        List<Integer> I=new ArrayList<Integer>();  
        I.add(45);  
        I.add(89);  
        I.add(76);  
        I.add(25);  
        I.add(65);  
        I.add(15);  
        System.out.println(I);//[45,89,76,25,65,15]
```



//sort the elements

```
        List<Integer> list=l.stream().sorted().collect(Collectors.toList());  
        System.out.println(list);//[15,25,45,65,76,89]  
    }  
}
```

ex:6

```
import java.util.*;  
import java.util.stream.*;  
class Test  
{  
    public static void main(String[] args)  
    {  
        List<Integer> l=new ArrayList<Integer>();  
        l.add(45);  
        l.add(89);  
        l.add(76);  
        l.add(25);  
        l.add(65);  
        l.add(15);  
        System.out.println(l);//[45,89,76,25,65,15]  
        //max element  
        Integer max=l.stream().max((i1,i2)->i1.compareTo(i2)).get();  
        System.out.println(max);//89 }}  
}
```

ex:7

```
import java.util.*;  
import java.util.stream.*;  
class Test  
{  
    public static void main(String[] args)
```



```
{  
    List<Integer> l=new ArrayList<Integer>();  
    l.add(45);  
    l.add(89);  
    l.add(76);  
    l.add(25);  
    l.add(65);  
    l.add(15);  
    System.out.println(l);//[45,89,76,25,65,15]
```

//min element

```
Integer max=l.stream().min((i1,i2)->i1.compareTo(i2)).get();  
System.out.println(max);//15
```

```
}  
}
```



ex:8

```
import java.util.*;  
import java.util.stream.*;  
class Test  
{  
    public static void main(String[] args)  
    {  
        List<Integer> list=Arrays.asList(10,5,8,9,1);  
  
        System.out.println("Reading Elements one by one ");
```

```
        list.forEach(System.out::println);

    }

}

ex:9

import java.util.*;
import java.util.stream.*;
class Test
{
    public static void main(String[] args)
    {
        List<Integer> al=Arrays.asList(5,7,1,2,3,9);

        List<Integer> list =
al.stream().sorted(Comparator.reverseOrder()).collect(Collectors.toList());
        System.out.println(list);
    }
}
```

### Program to convert array to collection

ex:

```
import java.util.*;
import java.util.stream.*;
class Test
{
```

```

public static void main(String[] args)
{
    String[] strarr={"hi","hello","bye"};
    List<String> list=Arrays.asList(strarr);

    list.forEach(System.out::println);
}

}

```

### Program to convert collection to array

ex:

```

import java.util.*;
import java.util.stream.*;
class Test
{
    public static void main(String[] args)
    {
        List<Integer> list=Arrays.asList(2,7,8,1,9);
        System.out.println(list);

        Integer[] iarr=list.toArray(new Integer[0]);
        for(int i:iarr)
        {
            System.out.println(i);
        }
    }
}

```



## Default methods in Interface

- It is introduced in java 8 version.
- To declare default method we need to use default keyword.
- It is a non-abstract method.
- Default method can override.

ex:1

**interface A**

{

//abstract method

**public abstract void m1();**

//default method

**default void m2()**

{

**System.out.println("M2-Method");**

}

}

**class B implements A**

{

**public void m1()**

{

**System.out.println("M1-Method");**

}

}

**class Test**

{

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

{

```
    A a=new B();  
    a.m1();  
    a.m2();  
}  
}
```

ex:2

**interface A**

```
{  
    //abstract method  
    public abstract void m1();
```

//default method

default void m2()

```
{  
    System.out.println("M2-Method");  
}
```

```
}
```

**class B implements A**

```
{  
    public void m1()  
    {  
        System.out.println("M1-Method");  
    }  
    public void m2()  
    {  
        System.out.println("Override M2-Method");  
    }  
}
```



```

}

class Test

{
    public static void main(String[] args)
    {
        A a=new B();
        a.m1();
        a.m2();
    }
}

```

We can achieve multiple inheritance through default methods of an interface.

ex:3

**interface Right**

```

{
    default void m1()
    {
        System.out.println("Right-M1-Method");
    }
}
```

**interface Left**

```

{
    default void m1()
    {
        System.out.println("Left-M1-Method");
    }
}
```

**class Middle implements Right,Left**



```

{
    public void m1()
    {
        System.out.println("Middle-M1-Method");
    }
}

class Test
{
    public static void main(String[] args)
    {
        Middle m=new Middle();
        m.m1();
    }
}

ex:4

interface Right
{
    default void m1()
    {
        System.out.println("Right-M1-Method");
    }
}

interface Left
{
    default void m1()
    {
        System.out.println("Left-M1-Method");
    }
}

```



```
    }
}

class Middle implements Right,Left
{
    public void m1()
    {
        Right.super.m1();
    }
}

class Test
{
    public static void main(String[] args)
    {
        Middle m=new Middle();
        m.m1();
    }
}
```

ex:5

```
interface Right
{
    default void m1()
    {
        System.out.println("Right-M1-Method");
    }
}

interface Left
{
```



```
default void m1()
{
    System.out.println("Left-M1-Method");
}

}

class Middle implements Right,Left
{
    public void m1()
    {
        Left.super.m1();
    }
}

class Test
{
    public static void main(String[] args)
    {
        Middle m=new Middle();
        m.m1();
    }
}

ex:6

interface Right
{
    default void m1()
    {
        System.out.println("Right-M1-Method");
    }
}
```



```
}

interface Left

{

    default void m1()

    {

        System.out.println("Left-M1-Method");

    }

}

class Middle implements Right,Left

{

    public void m1()

    {

        Right.super.m1();

        Left.super.m1();

    }

}

class Test

{

    public static void main(String[] args)

    {

        Middle m=new Middle();

        m.m1();

    }

}
```



## Static methods in interface

- It is introduced in java 8 version.
- To declare static method we need to use static keyword.
- It is a non-abstract method.
- We can't override static method.

ex:

**interface A**

```
{  
    static void m1()  
    {  
        System.out.println("M1-Method");  
    }  
}
```

**class Test**

```
{  
    public static void main(String[] args)  
    {  
        A.m1();  
    }  
}
```

