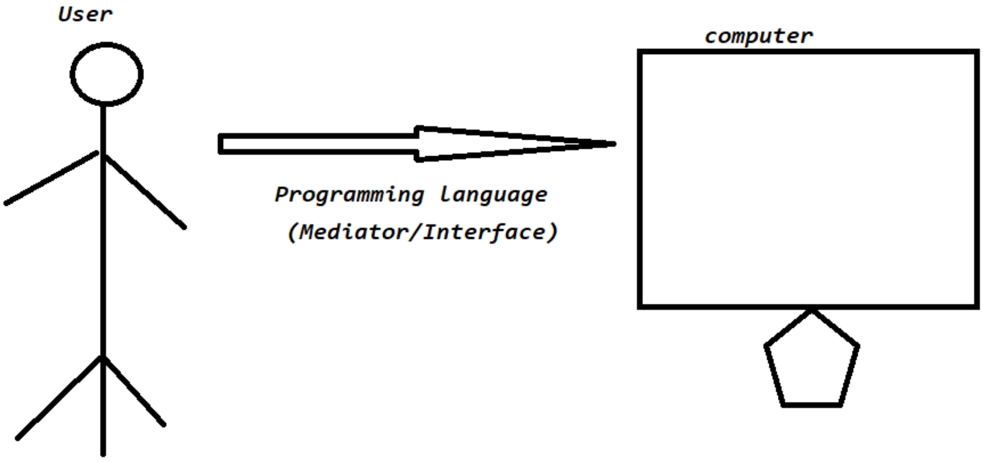
CORE JAVA

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

Diagram:



Programming language acts like a mediator or interface between user and computer.
A programming language is 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.
Computer may understands many languages , But to understand machine language it does not required any translator.

Advantages:

> A program writtens in machine language consumes less memory.
> It does not required any translator.
> It is more efficient when compare to other languages.

Disadvantages:

>It is a burdun on a programmer to remember donzen's of binary code.
>If anywhere error is raised 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(JUMP,AVG,SUM).
It is also known as symbolic language.
Assembly language can't understand by a computer directly.We required translator.

We have three types of translators.

1)Assembler
2)Compiler
3)Interpreter

1)Assembler

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

Merits:

> If anywhere error is raised then locating and handling that error becomes 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,C++,C#,Java,.Net,Python 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.

Disadvantages:

> A program writtens in high level language consumes huge amount of memory.
> It require translators.
> It is not efficient when compare to low level language.

Interview Questions
======================

1)What is programming language ?
2)What is low level language?
3)What is high level language?
4)How many translators are there ?
5)What is debugging?

Q)Differences between C and C++ ?

C	**C++**
========	======
It was developed by Dennis Ritchie.	It was developed by Bjarne Stroustrup.
It is a procedure oriented programming language.	It is a object oriented programming language.
Top-Down approach is used to compile the program.	Bottom-Up approach is used to compile the program.
variables can be declare only in declaration block.	variables can be declare in declaration block and execution block.
malloc(),calloc(),realloc() and free() function is used to allocate the memory and deallocate the memory.	new and delete operator is used to allocate and deallocate the memory.
It does not support class and object.	It supports class and object.
It does not support inheritance.	It supports inheritence.
It does not support polymorphism.	It supports polymorphism.
To save c program we need to use ".C" extension.	To save C++ program we need to use ".CPP" extension.

Q)Differences between C++ and Java?

C++	**Java**
=======	==========
It was developed by Bjarne Stroutstrup.	It was developed by James Gosling.
It is a partial object oriented programming language.	It is a purely object oriented programming language.
It is platform dependent.	It is a platform independent.
It supports multiple inheritence.	It does not support multiple inheritence.
It supports goto statement.	It does not support goto statement.
It supports pointers.	It does not support pointers.
It supports operator overloading.	It does not support operator overloading.
It supports constructor and destructor.	It supports only constructor.
It supports three access specifiers i.e public, private and protected.	It supports access modifiers i.e default,public,private & protected.
It supports preprocessor(#).	It does not support preprocessor.
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.
We can save C++ program by using ".CPP" extension.	We can save Java program by using ".java" extension.

Comments in Java

====================

Comments are created for documentation purpose.
Comments are used to improve readability of our code.
It is highly recommanded to use comments in our regular programming.
Whenever compiler reads comment, it will ignore the statement.
We have two types of comments in java.

1)Single Line Comment

It is used to comment a single line.

syntax:

```
// comment here
```

2)Multiple Line Comment

It is used to comment multiple lines.

syntax:

```
/*
        -
        - comment here
        -
*/
```

Q)Differences between Python and Java ?

Python ==========	**Java** =========
It is a product of Microsoft.	It is a product of Oracle Corporation(Sun Micro System).
It was developed by Guido Van Rossum.	It was developed by James Gosling.
It is a scripting language.	It is a object oriented programming language.
It is a Interpreted language.	It is a compiled language.
It is a dynamically typed language.	It is statically typed language.
It is easy then java.	It is easy.
Performance is low.	Performance is high.
It supports PVM(Python virtual Machine).	It supports JVM (Java Virtual Machine).

Q)What is Java?
Java is a object oriented ,platform independent, case sensitive, strongly typed checking,robust,highly secured,multi-threaded ,high level, open source programming language.

Modules in Java

====================

We have three modules in java.

```
                    Java
  |---------------------------|----------------------------------|
J2SE/JSE                J2EE/JEE                          J2ME/JME
```

(Java Standard Edition)	**(Java Enterprises Edition)**	**(Java Mobile Edition)**
Standalone Application	Distributed Application	Mobile Application
Desktop Application	Enterprises Application	
Two-Tier Application	N-Tier Application	
	Applet	

Standalone Application

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

ex:

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

Desktop Application

It is a software application or a program which is developed for desktops.

ex:

recycle bin
vlc media player
control panel and etc.

Two-Tier Application

Having more then one tier is called two-tier application.

Diagram:

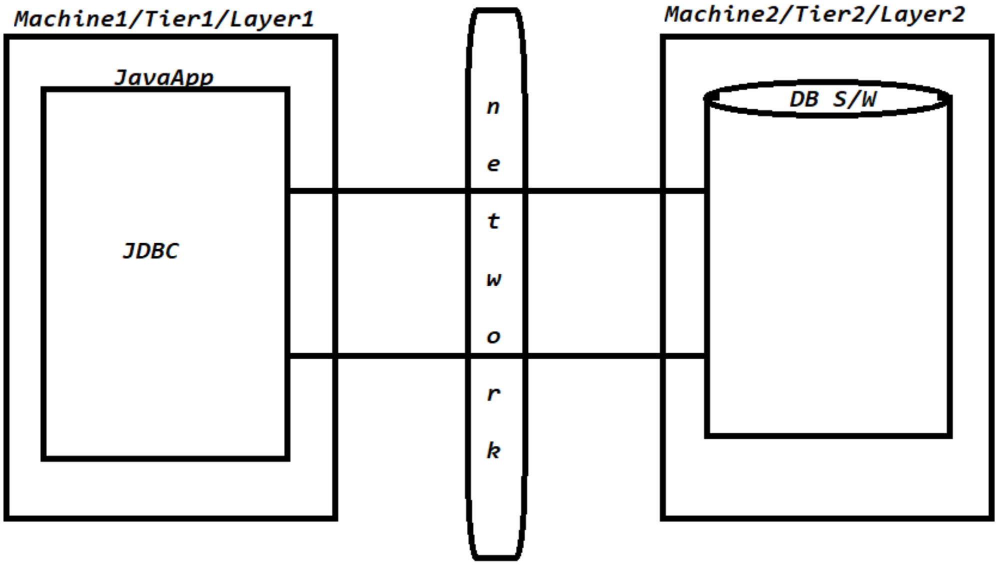
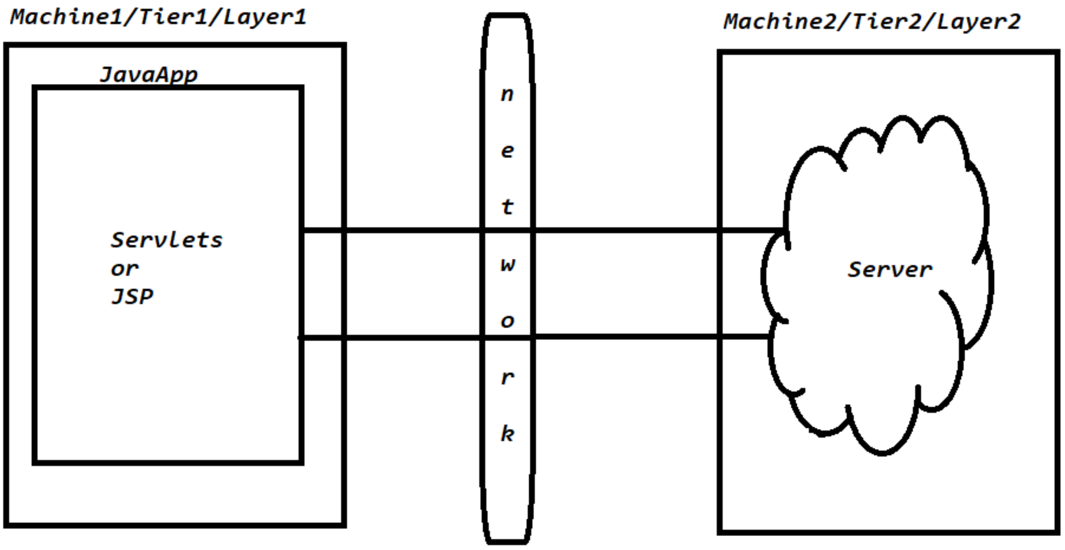


Diagram:



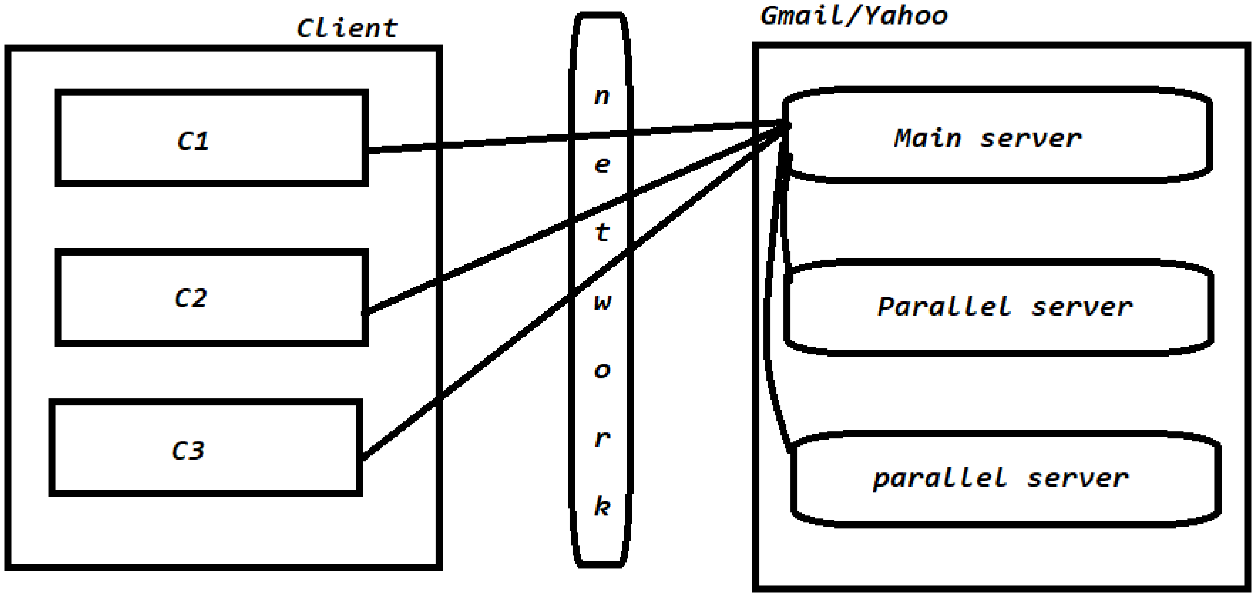
JEE Module

===========

1)Distributed Application

In client-Server application , if multiple request goes to main server then main server will distribute that request to it's parallel server to reduce the burdun of main server is called distributed application.

Diagram: java2.1



2)Enterprises Application

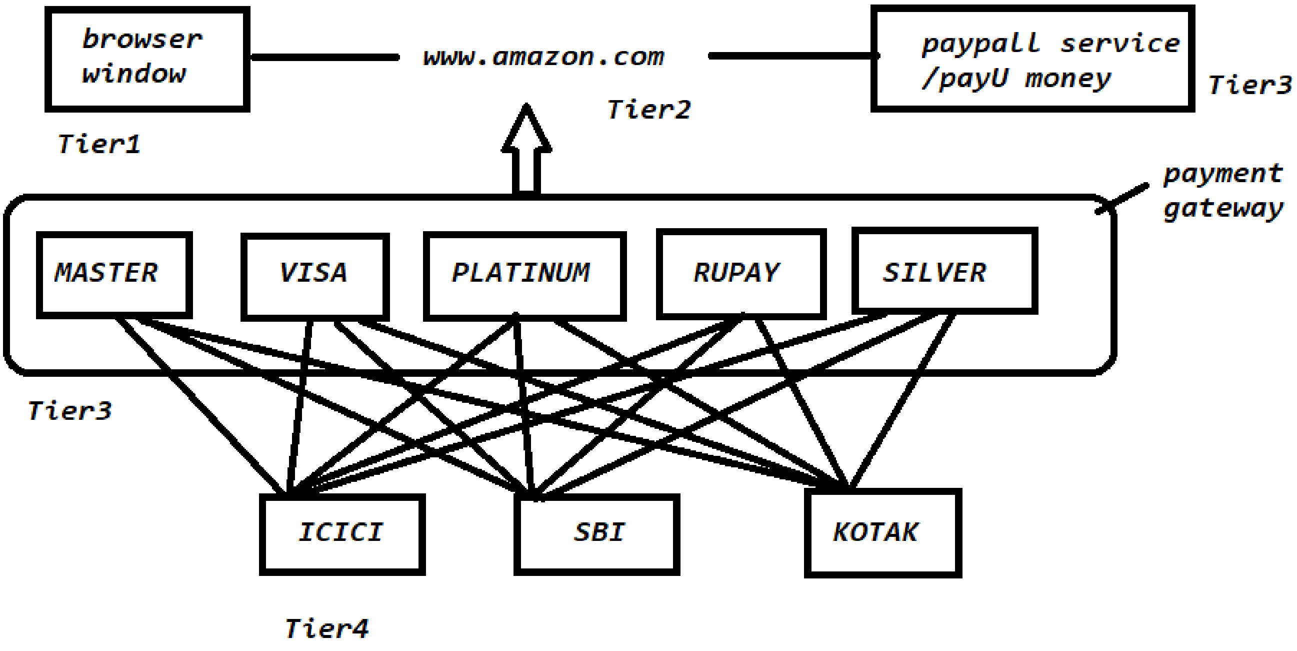
An application which deals with large business complex logic with the help of middleware services is called enterprises application.

Here middleware services means addition services like authentication, autherization,security,firewall,malwareproduction,transaction and etc.

3)N-Tier Application

Having more then two-tier is called N-tier application.

Diagram: java2.2



4)Applets

Applet is a compiled java class which is used to send over the network as web pages.
If we need a web page with performance then we need to use HTML.
If we need a web page with security then we need to use Applet.

JME
======
Mobile Application

It is a software application or a program which is developed for wireless network devices like phone,cell,tab,cellular and etc.Rather then laptop's and PC's.
ex:

GooglePay
PhonePay
zomato
watsapp
facebook
PayTM and etc.

Note:

JME module is outdate in java.
Now Andriod and IOS are used to developed mobile applications.

History of Java
=================
In 1990, Sun micro system has taken one project to developed 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 it was renamed to green project.
James Gosling, Mike Sheradin and Patrick Naughton were there to develop that project.They met in a place called Aspan/Colarado to start the work with graphic system. James Gosling decided to use C and C++ languages to develop this project.But problem what they faced is C and C++ languages are system dependent.James Gosling decided to create our own programming language which is system independent.
In 1991,he has developed a programming language called an OAK. OAK means strength , itself is a coffee seed name and it is a national tree for many contries like Germany,France,USA and etc.
In the year of 1995, they have renamed OAK to JAVA.JAVA is a island of an Indonasia where first coffee of seed was produced.During the development of project there were consuming lot of coffee's that's why simple of JAVA is cup of coffee with saucer.

Escape Sequences / Escape Characters
======================================
Whenever we want to design our output in neat and clean manner then we need to use escape sequences or escape characters.
Every escape sequence starts with back slash(\) folled by a single character.
The list of escape characters present in java are
1) \n (new line)
2) \t (horizontal tab)
3) \b (back space)
4) \r (carriage return)
5) \f (form feeding)
6) \\ (back slash)
7) \" (double quote)
8) \' (single quote)
and etc.

1) \n (new line)

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

o/p:

```
IHUB
TALENT
```

2) \t (horizontal tab)

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

o/p:

```
IHUB				Talent
```

3) \b (back space)

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

o/p:

```
IHUTalent
```

ex:

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

o/p:

```
ITalent
```

4) \r (carriage return)

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

o/p: Talent

<u>or</u>

```
class Mastan
{
                    public static void main(String[] args)
                    {
                        System.out.println("Talent\rIHUB");
                    }
}
```

o/p: IHUBnt

6)\\ (back slash)

====================

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

o/p: IHUB\Talent

7)\" (double quote)

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

o/p: IHUB"Talent

8)\' (single quote)

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

o/p:IHUB'Talent

Interview questions:

----

1) class Test

```
{
                    public static void main(String[] args)
                    {
   System.out.println("This is \"IHUB Talent\" for \'Freshers\' those who are looking for---\t IT JOBS");
                    }
}
```

o/p:

```
This is "IHUB Talent" for 'Freshers' those who are looking for---   IT Jobs
```

2)

```
class Test
{
                              public static void main(String[] args)
                              {
                                  System.out.print("\njk");
                                  System.out.print("\bti");
                                  System.out.print("\rha");
                              }
}
```

o/p: hai

Q)What are the features of Java?

Some of the important features of java are

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

Q)What is Native Method in java?

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

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

JDK:

JDK stands for Java Development Kit.

It is a installable software used for java applications.It contains Java Runtime Environment(JRE), an interpreter/loader (java),
a compiler (javac), an archiever(jar), a document generator (javadoc) and
other tools needed for java application development.

JRE:

JRE stands for Java Runtime Environment.

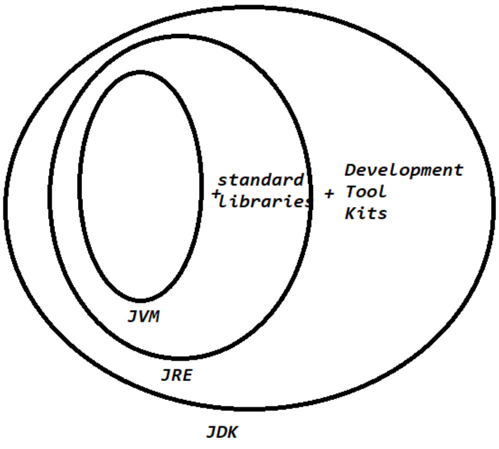
It provides very good environment to run java applications only.

JVM:

JVM stands for Java Virtual Machine.

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

Diagram: java4.1



Naming Conventions in java
==============================
In java , upper case letters 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.

ex:
classes
interfaces
variables
methods
keywords
packages and
constants

classes

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

predefine classes	**user define classes**
System	Test
String	Example
File	TestApp
FileWriter	ModalApp
and etc.	

interfaces

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

predefine interfaces	**user define interfaces**
Runnable	ITest
Cloneable	ITestApp
Serializable	IModalApp
Iterator	IExample
ListIterator	
Enumeration and etc.	

variables

In java, every variable must and should start with lower case letter and if it contains multiple words then each inner word must starts with initcap.
ex:

predefine variables	**userdefine varibles**
out	i
in	a
length	empId
and etc	studName

methods

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

ex:

predefine methods	**userdefine methods**
------------------	--------------------
hashCode()	methodOne()
toString()	getBusinessDetails()
getClass()	calculateBillAmt()
getPriority()	generateHospitalBill()
setName()	
and etc.	

keywords

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

ex:

predefine keywords

----

if ,else,switch,do , while, public,private, protected and etc.

packages

In java, all packages we need to write under lowe case letters only.

ex:

predefine packages	**userdefine packages**
-----------------	-------------------
java.lang (default pkg)	srinivasu
java.io	aziz
java.util	sanjeev
java.util.stream	com.google.www
java.text	com.vishali.www
java.sql	
javax.servlet	
javax.jsp	

constants

In java , all constants we need to write under upper case letters only.

ex:

predefine constants	**userdefine constants**
------------------	---------------------
MAX_PRIORITY	int N=10;
MIN_PRIORITY	LIMIT
NORM_PRIORITY	TOTAL
MAX_VALUE	
MIN_VALUE	
and etc.	

programs
===========

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

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

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

```
class Test
{
                                   public static void main(String[] args)
                                   {
                                        int a=10,b=20;
                                        System.out.println("sum of two numbers is ="+(a+b));
                                   }
}
```

Q)Write a java program to perform square of a number?

```
class Test
{
                                   public static void main(String[] args)
                                   {
                                        int n=5;
                                        int square=n*n;
                                        System.out.println("square of a number is ="+square);
                                   }
}
```

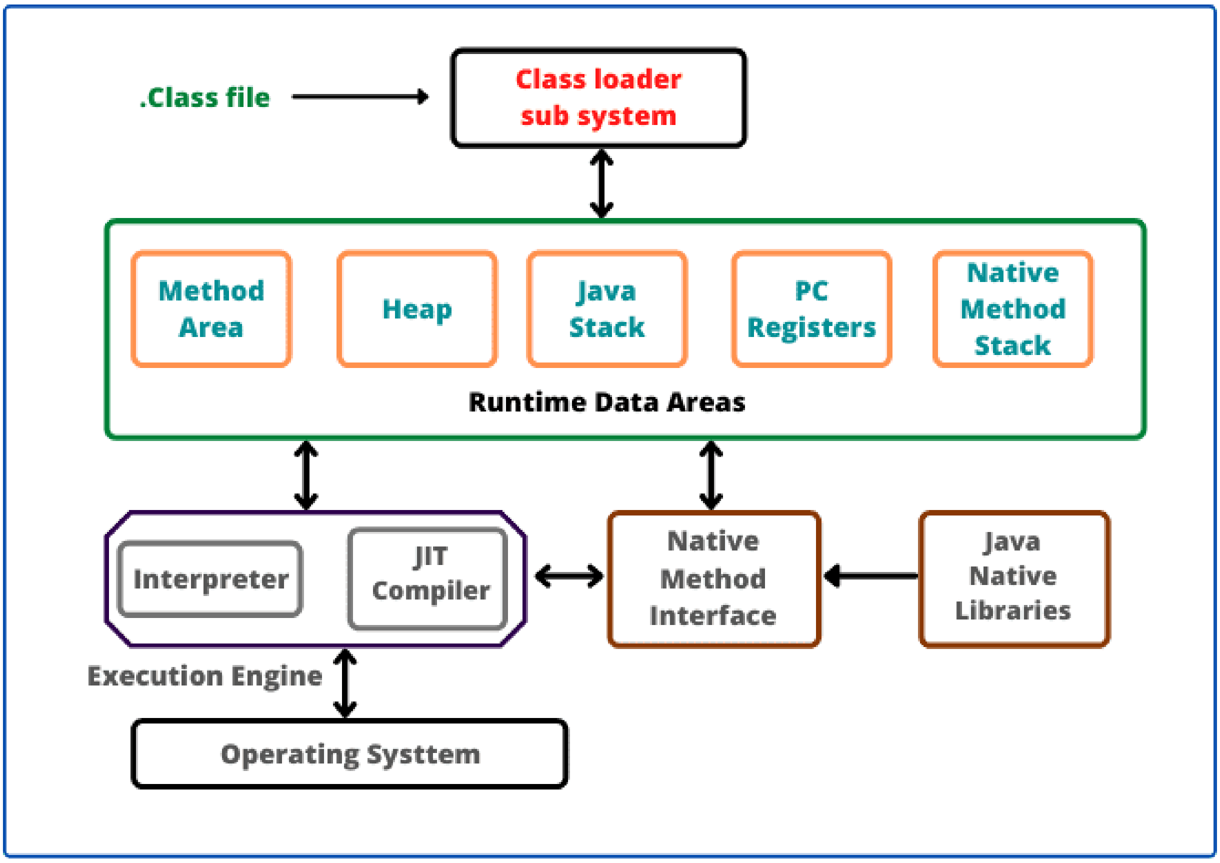
Q)Write a java program to perform cube of a number?

```
 class Test
{
                                   public static void main(String[] args)
                                   {
                                        int n=5;
                                        int cube=n*n*n;
                                        System.out.println("Cube of a number is ="+cube);
                                   }
}
```

Internal Architecture of JVM

================================

Diagram: java6.1



Java application contains java code instructions(source code).Once if we compile java program , compiler converts java code instructions to byte code instructions in .class file.
JVM will call one module i.e classloader or sub system to load all the byte code instructions from .class file.The work of a 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, it will allocate the memory.

Java contains 5 types of memories.
1)Method Area :

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

2)Heap:

Our object creations will store in Heap area.

Note:
Whenever JVM loads byte code from .class file ,it automatically creates method area and heap area.

3)Java Stack

To execute java methods we required some memory that memory will allocate in Java Stack.

4)PC Register

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

5)Native Method Stack

All native methods will store in native method stack.But to execute native methods we required a program called Native method interface.

Execution engine contains interpreter and JIT compiler.

Whenever JVM loads byte code instructions from .class file.It simultenously uses interpreter and JIT compiler.

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

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

Interview Questions
=====================
Q)In which year java was developed?

In 1995.

Q)Who is the creator of Java?

James Gosling

Q)Java is a product of which company?

Oracle corporation

Q)Java orginally known as ____?

OAK

Q)Which is a default package in java?

java.lang package

Q)How many classes are there in java?

java 1.7v --> 4024 classes
java 1.8v --> 4240 classes
java 1.9v --> 6005 classes
java 1.10v --> 6003 classes

Q)How many classloaders are there in java?

We have three classloaders in java.

1)Bootstrap classloader

It is parent of extension classloader.
It loads rt.jar file which contains all the classes of java.lang package.
ex:
C:\Program Files\Java\jdk1.8.0_181\jre\lib

2)Extension classloader

It is child of bootstrap classloader and it is parent of system classloader.
It is used to load all the jar files from extension folder(ext).
ex:
C:\Program Files\Java\jre1.8.0_181\lib\ext

3)System/Application classloader

It is a child of extension classloader
It is used to load .class file from CLASSPATH.

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?
JIT stands for Just In Time.
It is a part of a JVM which is used to increase the execution speed of our program.

Q)What is Garbage Collector?

Garbage collector is used to destroy unused and useless objects from java.
There are two ways to call garbage collector.

1)System.gc();
2)Runtime.getRuntime().gc();

Identifier

=============

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

ex:

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

Here Test,main,String,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:

```
      int $=10; //valid
      int $alary=20; //valid
      int #=30; //invalid
      int sal@ry=40; //invalid
```

Rule3:

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

ex:

```
      int  a1234; //valid
      int  1abcd; //invalid
      int _abcd; //valid
```

Rule4:

There is no length limit for an identifier. But is not recommanded to use more then 15 characters.

Rule5:

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

ex:

```
      int  if;
      int  else;
```

Rule6:

Identifier can be alpha numeric character also.

ex:

```
      int  emp_No1;
      int  emp$sal2;
      int  emp#Name; //invalid
```

Rule7:

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

ex:

```
int  String=10;  //valid
int  Runnable=20;  //valid
```

Reserved words

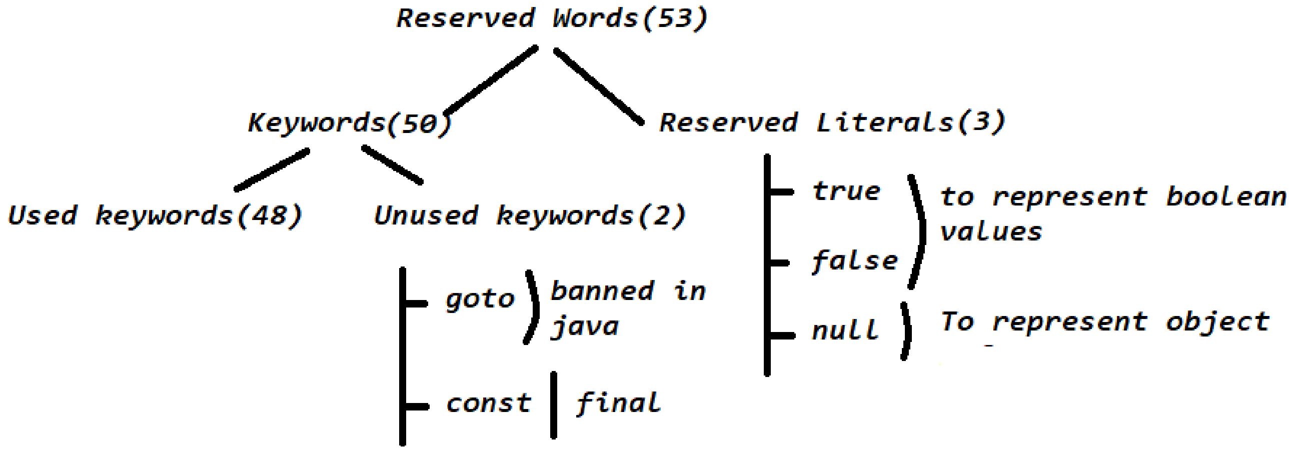
=================

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 and all reserved words we need to declare under lower case letters only.

Reserved words are divided into two types.

Diagram: java7.1



Used keywords with respect to class

----

package
import
enum
class
extends
implements
interface

Used keywords with respect to object

----

new
instanceof
this
super

Used keywords with respect to datatype

----

byte
short
int
long
float
double
boolean
char

Used keywords with respect to modifiers
----
default
public
private
protected
final
abstract
static
synchronized
strictfp
transient
volatile

Used keywords with respect to flow control
---
if
else
switch
do
while
for
break
case
continue

Used keywords with respect to return type
----
void

Used keywords with respect to exception handling
---
try
catch
finally
throw
throws
assert

Interview Questions
==========================
Q)Is java support Access specifiers ?

No, Java does not support Access specifiers.

Java supports Access modifiers i.e default , public, private and protected.

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

default class

----

we can access that class with in the same package.

ex:

```
class Test
{
      -
      -//code to be execute
      -
}
```

public class

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

ex:

```
public class Test
{
      -
      -//code to be execute
      -
}
```

Q)What is operating System ?

Collection/Set of software is called operating system.
It is a mediator or interface between software components and hardware component.

Q)What is software?

Collection/Set of applications is called software.

Q)What is Application?

Collection/Set of programs is called application.

Q)What is program?

Collection/Set of instructions is called program.

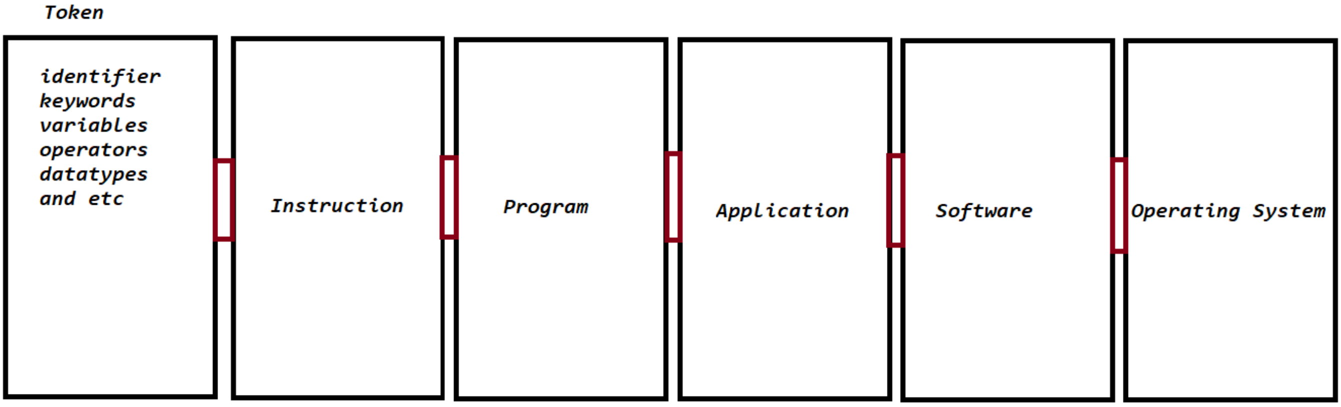
Q)What is instruction ?

Collection/Set of tokens is called instruction.

Q)What is Token ?

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

Diagram: java7.2

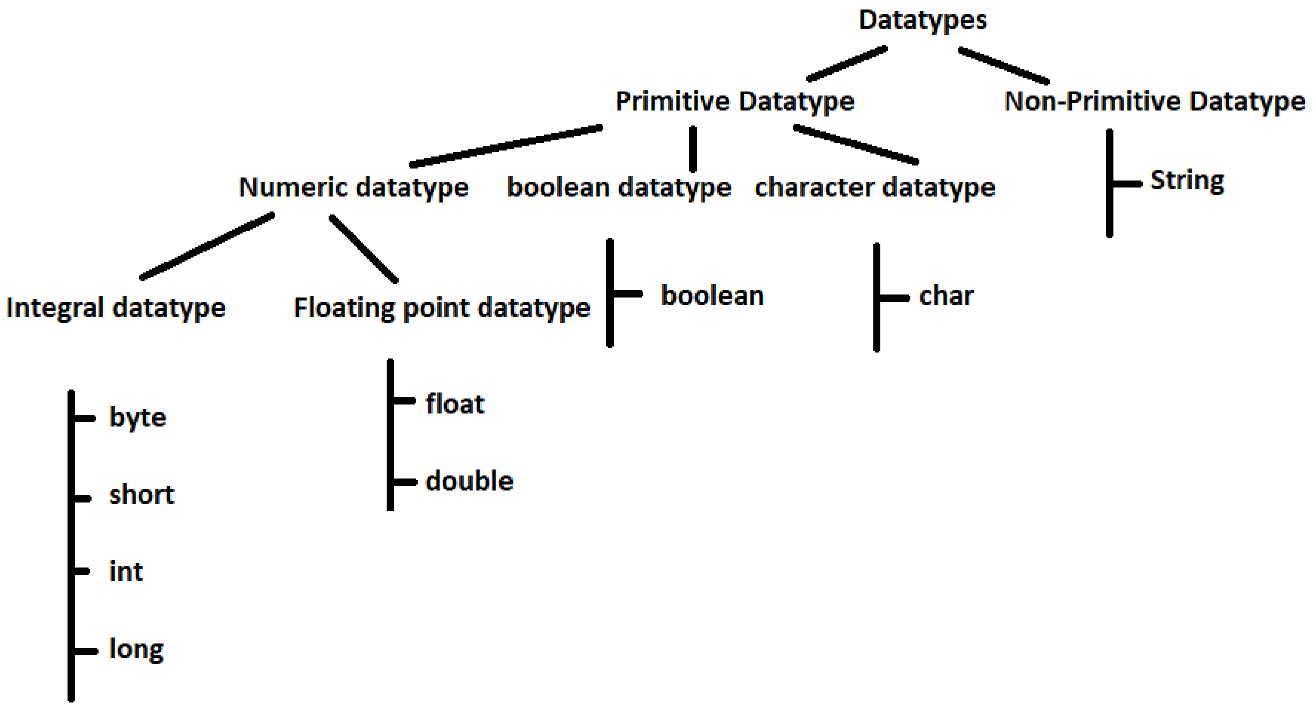


Datatypes
===============
Datatype describes what type of value we want to store inside a variable.

Datatypes also tell how much memory has to be created for a variable.

In java datatypes are divided into two types.

Diagram: java7.1



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.
size: 2 bytes (16 bits)
Range: -32768 to 32767 (-2^15 to 2^15-1)
ex:

```
        1) byte b=40;
           short s=b;
           System.out.println(s);//40

        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 (-2^31 to 2^31-1)
ex:

```
        1) int i='a';
           System.out.println(i);// 97 is a universal unicode value.

        2) int i="true";
           System.out.printn(i);//C.T.E

        3) int i=true;
           System.out.println(i);//C.T.E
```

Note:

Java does not have ASCII code (American Standard Code for Information Interchange).
Java support Universal Unicode value.
ex:

```
        a = 97
        A = 65
```

long
========
If integer 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= 10.56;
           System.out.println(l);//C.T.E

        2) long l='d';
           System.out.println(l);//100

        3) long l=true;
           System.out.println(l);//C.T.E

        4) long l="hi";
           System.out.println(l);//C.T.E
```

float
=======

If we need 4 to 6 decimal point of accuracy then we need to use float datatype.

Size: 4 bytes (32 bits).

Range: -3.4e38 to 3.4e38

To represent float value we need to suffix with 'f' or 'F'.
ex:

```
        float f=10.56f;
```

double
==========

If we need 14 to 16 decimal point of accuracy then we need to use double datatype.

Size: 8 bytes (64 bits).

Range: -1.7e308 to 1.7e308.

To represent double value we need to suffix with 'd' or 'D'.
ex:

```
        double d=10.56d;
```

Note
-----

integral datatypes --> float/double //valid
float/double ---> integral datatypes //invalid

ex:1

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

ex:2

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

ex:3

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

ex:4

```
class Test
{
            public static void main(String[] args)
            {
                        double d=10.5d;
                        int i=d;
                        System.out.println(i);//C.t.E
            }
}
```

boolean
========

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

Size: (Not Applicable)

Range: (Not Applicable)

ex:

```
            1) boolean b="false";
              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 under single quotation.

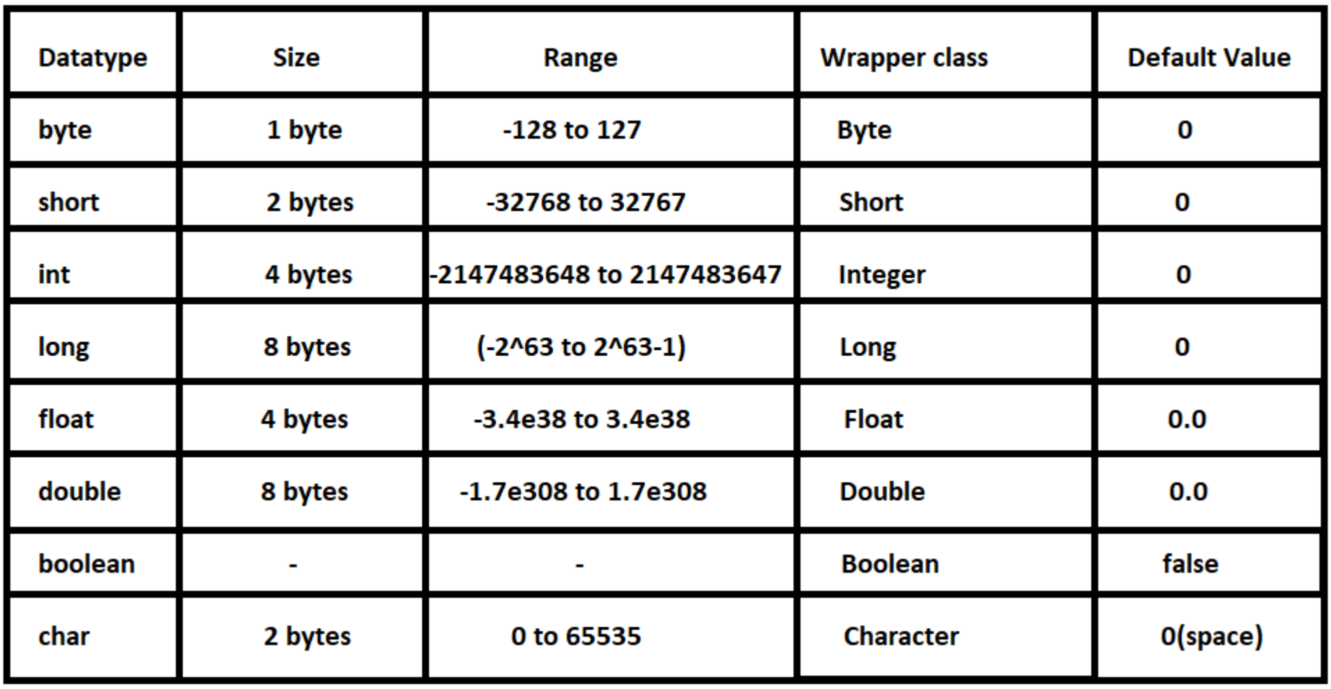
Size: 2 bytes (16 bits)

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=10.56;
              System.out.println(ch);//C.T.E

            4) char ch="a";
              System.out.println(ch);//C.T.E
```

Diagram: java7.2



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 65535	Character	0(space)

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

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

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

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

Types of variables in java

============================

A name which is given to a memory location is called variable.
Purpose of variable is used to store/hold the data.
We have two types of variables in java.

1)Primitive variables

Primitive variables are used to represent primitive values.

2)Reference variables

Reference variables are used to represent objects.

ex:

```
        Student s=new Student();
          |
        reference variable
```

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

1)Instance variables / Non-static variables

2)Static variables / global variables

3)Local variables / temperory variables / Automatic variables

1)Instance variables

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

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

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

To access instance variables 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
{
	//instance variable
	int i=10;

	public static void main(String[] args)
	{
		//call or invoke
		Test t=new Test();
		t.m1();

	}

	//non-static method
	public void m1()
	{
		System.out.println(i);//10
	}
}
```

Note:

If we won't initialized any value to instance variables then JVM will initialized default values.

ex:

```
class Test
{
	//instance variable
	boolean b;

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

2)Static variables

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

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

All static variables will store in method area.

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

A static variable we can access directly from instance area as well as from static area.

Static variables can access by using object reference and class name.

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 variable
	static int i=10;

	//static method
	public static void m1()
	{
		System.out.println(i);//10
	}

	public static void main(String[] args)
	{
		//call or invoke
		m1();

		Test t=new Test();
		t.m1();

		Test.m1();

	}
}
```

Note:

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

ex:

```
class Test
{
	//static variable
	static String s;

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

3)Local variables

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

A 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.

All local variables will store in JavaStack.

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 initialized any value to local variable then JVM will not initialized default values.

ex:2

```
class Test
{

	public static void main(String[] args)
	{
		//local variable
		int i;
		System.out.println(i);//C.T.E variable i might not have been initialized
	}
}
```

A local variable will accept only one modifier i.e final.

ex:
```
class Test
{

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

Assignment
===============

Q)Write a java program to perform sum of two numbers in static method?

Q)Write a java program to perform sum of two numbers in non-static method?

Q)Write a java program to perform sum 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();

		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 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();

		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 Test
{
	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 find out cube 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();

		//logic
		int cube = n*n*n;
		System.out.println("Cube of a given  number is ="+cube);
	}
}
```

Q)Write a java program to find out swapping 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();

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

	System.out.println("Before Swapping a="+a+" and b="+b);
		//logic
		int temp=a;
		a=b;
		b=temp;

	System.out.println("After swapping a="+a+" and b="+b);
	}
}
```

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

```
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();

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

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

		System.out.println("After swapping a="+a+" and b="+b);
	}
}
```

approach2

```
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();

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

		//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 find out area of a circle?

```
import java.util.Scanner;
class Test
{
	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 find out perimeter of a circle?

```
import java.util.Scanner;
class Test
{
	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);
	}
}
```

var-arg method

==================

Untill 1.4v, it is not possible to declare a method with variable number of arguments.

But from 1.5v onwards it is possible to declare a method with variable number of arguments.

We can declare var-arg method as follow.

ex:

```
                 var-arg parameter
                 --------
        methodOne(int... i);
                |
                  ellipse
```

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

ex:

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

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

ex:1

```
class Test
{
        public static void main(String[] args)
        {
                methodOne();
        }
        public static void methodOne()
        {
                System.out.println("0-arg method");
        }
}
```

ex:2

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

ex:3

```
class Test
{
        public static void main(String[] args)
        {
                methodOne(10,20);
        }
        public static void methodOne(int i,int j)
        {
                System.out.println("two-arg method");
        }
}
```

ex:4

```
class Test
{
        public static void main(String[] args)
        {
                methodOne();
                methodOne(10);
                methodOne(10,20);
                methodOne(10,20,30,40,50);
        }
        public static void methodOne(int... i)
        {
                System.out.println("var-arg method");
        }
}
```

case1:

We can mix var-arg parameter with general parameters.

ex:

```
methodOne(int a,int b,int... i);
```

case2:

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

ex:

```
methodOne(int a,int b,int... i);

methodOne(int... a,int b,int c);// invalid
```

case3:

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

ex:

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

main method

=============

Our program contains main method or not. Either it is properly declared or not.It is not a responsibility of a compiler to check.It is a responsibility of a JVM to look for main method always at runtime.

If JVM won't find main method then it will through 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 we will get 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 won't return anything to JVM.
main: It is an identifier.
String[] args: It is command line argument.

We can perform following changes inside main method.

1)Order of modifiers are not important , incase of public static we can place 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 replace args with any java valid identifier.

ex:

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

4)We can replace String[] with var-arg parameter.

ex:

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

5)Main method will accept following modifiers i.e synchronized,strictfp and final.

ex:

```
public static synchronized strictfp final void  main(String[]   args)
```

command line argument
===================

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

The main objective of command line argument is to change the behaviour of main method.

Note:

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

ex:

```
javac    Test.java

java     Test  101   raja   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
```

Q)Write a java program to accept one name and display it?

```
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
```

System.out.println()

======================

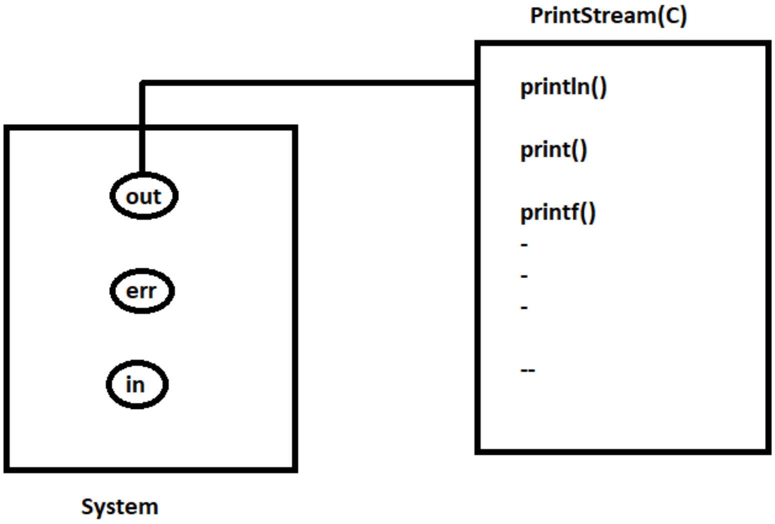
It is output statement in java.

It is used to display userdefine statements and data.

syntax:

```
          static variable
            |
            System.out.println("");
            |             |
            predefine    predefine method
            final class
```

Diagram: java9.1



How to display the data in java

=================================

1)

```
            System.out.println("Hello 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=1,j=2,k=3;
            System.out.println(i+" "+j+" "+k);
```

Q)Difference between System.out.println() and System.err.println()?

System.out.println():

It is mostly used to display results on the console.

ex:

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

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

System.err.println():

It is used to display the output on console and it will direct to physical file.

It is mostly used to output error texts.
ex:

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

                System.err.println("Welcome ");
        }
}
```

o/p:

```
javac  Test.java
java   Test  2>abc.txt
```

Interview Questions
====================
Q)Difference between default class and public class?
default class

A default class can access within the package.
ex:

```
class Test
{
        -//variables
        -//methods
}
```

public class

A public class can access within the package and outside of the package.
ex:

```
public class Test
{
        -//variables
        -//methods
}
```

Q)What is final class?

If we declare any class as final then creating child class is not possible.
ex:

```
        final class Father
        {

        }
   X   class Child extends Father
        {
        }
```

Q)What is singleton class?

A class which allows us to create only one object is called singleton class.

Q)What is abstract class?

If we declare any class as abstract then creating object for that class is not possible.
ex:

```
        abstract class Test
        {
                -
                -//code here
                -
        }
```

Q)What is literal?

A value which is not change during the program execution is called literal.
ex:

```
        int i = 10;
     |  |    |___value /literal
        |  |_________variable/identifier
        |_____________datatype/keyword
```

Editplus Editor
===============
download link : https://www.editplus.com/download.html

Fully Qualified Name
=======================
Whenever we are implementing fully qualified name we should not use
import statement.
Fully qualified name means a class along with package name and It will increase readability of our code.

ex:

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

Import statements
===================
Whenever we use import statement we should not use fully qualified name.Using short name also we can achieve.

It is used improve readability of our code.

We have three types of import statements in java.

1)Explicit class import

2)Implicit class import

3)Static import

1)Explicit class import

This type of import statement is highly recommanded to use because it will increase readability of our code.

ex:1

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

A java.time package is introduced in java 1.8v.

ex:2

```
import java.time.LocalDateTime;
class Test
{
	public static void main(String[] args)
	{
		//current Date and Time
		LocalDateTime ldt=LocalDateTime.now();
		System.out.println(ldt);
	}
}
```

ex:3

```
import java.time.LocalDateTime;
import java.time.LocalDate;
import java.time.LocalTime;
class Test
{
	public static void main(String[] args)
	{
		//current Date and Time
		LocalDateTime ldt=LocalDateTime.now();
		System.out.println(ldt);

		LocalDate ld=LocalDate.now();
		System.out.println(ld);

		LocalTime lt=LocalTime.now();
		System.out.printn(lt);
	}
}
```

2)Implicit class import

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

ex:

```
import java.time.*;
class Test
{
        public static void main(String[] args)
        {
                //current Date and Time
                LocalDateTime ldt=LocalDateTime.now();
                System.out.println(ldt);

                LocalDate ld=LocalDate.now();
                System.out.println(ld);

                LocalTime lt=LocalTime.now();
                System.out.println(lt);
        }
}
```

3)Static import

Using static import we can access static member easily/directly.

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

ex:1

```
import static java.lang.System.*;
class Test
{
        public static void main(String[] args)
        {
                out.println("stmt1");
                out.println("stmt2");
                out.println("stmt3");
        }
}
```

ex:2

```
import static java.lang.System.*;
class Test
{
        public static void main(String[] args)
        {
                out.println("stmt1");
                exit(0);
                out.println("stmt3");
        }
}
```

Type casting in java

=====================

Process of converting from one datatype to another datatype is called typecasting.

There are two ways to perform typecasting in java.

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
```

ex:1

```
class Test
{
	public static void main(String[] args)
	{
		byte b=100;
		int i=b;
		System.out.println(i);//100
	}
}
```

ex:2

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

ex:3

```
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 to 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
```

ex:1

```
class Test
{
	public static void main(String[] args)
	{
		float f=10.56f;
		int i=(int)f;
		System.out.println(i);//10
	}
}
```

ex:2

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

ex:3

```
class Test
{
	public static void main(String[] args)
	{
		int i=130;
		byte b=(byte)i;
		System.out.println(b);//-126
	}
}
```

Java Source File Structure
===========================

case1:
A java program can have multiple classes.

case2:
If a java program contains multiple class then we need to save
our java program name with that class which contains main method.

ex:

```
class A
{
	-
	-
}
class B
{
	-
	-
	-
}
class C
{
	public static void main(String[] args)
	{
		-
		-
	}
}
```

case3:
If java program contains multiple class with main method
then we can save that 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");
		}
}
```

If we compile above program , three .class files will be generated i.e A.class ,B.class and C.class.

ex:

```
		javac   IHub.java
```

But we can execute each class seperately.

ex:

```
		cmd>java   A
		cmd>java   B
		cmd>java   C
```

case4:

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

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");
		}
}

cmd> javac  A.java
cmd> java  A
cmd> java  B
cmd> javac C
```

Types of blocks in java
==========================
A block in Java is a set of code enclosed within curly braces { } within any class, method, or constructor. It begins with an opening brace ({) and ends with an closing braces (}).

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 intialize instance variables.
Instance block must and should declare immediately after the class but not inside methods and constructors.
Instance block will executed at the time of object creation.Hence it is also known as lazy loading.
We can declare instance block as follow.
syntax:

```
        //instance block
        {
                -
                -//code to be execute
                -
        }
```

ex:1

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

o/p:

```
Main method
```

ex:2

```
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
```

ex:3

```
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
```

ex:4

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

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

2)Static block

A static block is used to initialize static variables.

A static block must and should declare immediately after the class by using static keyword but not inside methods and constructors.

A static block executed at the time of classloading.Hence it is also known as early loading.

We can declare static block as follow.

syntax:

```
	//static block
	static
	{
		-
		-//code to be execute
		-
	}
```

ex:1

```
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
```

ex:2

```
class Test
{
	//static variable
	static int i;
	//static block
	static
	{
		i=100;
	}
	public static void main(String[] args)
	{
		System.out.println(i);//100
	}
}
```

ex:3

```
class Test
{
	//instance block
	{
		System.out.println("Instance block");
	}
	//static block
	static
	{
		System.out.println("Static 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 local variables.

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

A local block will execute just like a normal statement.

We can declare local block as follow.

syntax:

```
	//local block
	{
		-
		-//code to be execute
		-
	}
```

ex:1

```
class Test
{

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

		//local block
		{
			System.out.println("stmt2");
		}

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

ex:2

```
class Test
{

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

		//local block
		{
			i=200;
		}

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

Interview Question
================
Q)Can we execute java program without main method?

Yes ,untill 1.6v it is possible to execute java program without main method
by using static block.But from 1.7v onwards it is not possible to execute java program without main method.

ex:

```
class Test
{
	//static block
	{
		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, logical operation, conditional operation, bitwise operation, assignment operation and etc.

We have following eight types of operators in java.

1)Logical Operators

2)Bitwise Operators

3)Shift Operators

4)Conditional or Ternary Operators

5)Unary Operators

6)Arithmetic Operators

7)Assignment Operators

8)Relational Operators

ex:

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

o/p:
C.T.E : variable i might not have been initialized

ex:

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

o/p:

Reinitialization is possible in java.

ex:

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

Note:

If we declare any variable as final then reinitialization is not possible.

ex:

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

Note:

We can't initialized more then one value at a time.

ex:

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

ex:

```
public class Test
{
		public static void main(String[] args)
		{
				boolean b=5>9;
				System.out.println(b);// false
		}
}
```

ex:

```
public class Test
{
		public static void main(String[] args)
		{
				boolean b=50>9;
				System.out.println(b);// true
		}
}
```

ex:

```
public class Test
{
		//instance variables
		 int i=10,j=20;

		public static void main(String[] args)
		{
				//local variables
				int i=100,j=200;

				System.out.println(i+" "+j);//100 200

		}
}
```

1)Logical AND Operator (&&)

Truth table

```
T		T		= T
T		F		= F
F		T		= F
F		F		= F
```

ex:

```
public class Test
{
		public static void main(String[] args)
		{
				boolean b= (5>2) && (4<2);
				System.out.println(b);//false
		}
}
```

ex:
--

```
public class Test
{
        public static void main(String[] args)
        {
                boolean b= (5>2) && (4<20);
                System.out.println(b);//true
        }
}
```

Logical OR operator (||)

Truth table

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

ex:

```
public class Test
{
        public static void main(String[] args)
        {
                boolean b= (5>2) || (4<2);
                System.out.println(b);//true
        }
}
```

ex:

```
public class Test
{
        public static void main(String[] args)
        {
                boolean b= (5>20) || (4<2);
                System.out.println(b);//false
        }
}
```

Logical NOT operator (!)
----

ex:

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

ex:

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

ex:

```
public class Test
{
	public static void main(String[] args)
	{
		boolean b= !((6>2) || (9<2) && (10 <= 19));
		System.out.println(b);//false
	}
}
```

Q)How to convert decimal to binary?

10 - decimal number

1010 - binary number

```
	2|10
	  ----- 0
	2|5
	  ----- 1
	2|2            ^
	  ----- 0      |
	    1          |
	---------------|
	1010
```

2 - decimal number
0010 - binary number

```
	2|2
	  --- 0    ^
	  1        |
	-----------|
	0010
```

Q)How to convert binary to decimal?

binary -- 1010

decimal - 10

```
	1010
	      <----
	0*1  + 1*2  + 0*4  + 1*8
	0 + 2 + 0 + 8 = 10
```

Bitwise Operators
====================

Bitwise AND operator (&)

Bitwise AND operator deals with binary number.

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
			      <--- right to left
	0*1+1*2+0*4+1*8
	0+2+0+8
	10
*/
```

ex:

```
class Test
{
	public static void main(String[] args)
	{
		int a=3,b=2;
		int c= a & b;
		System.out.println(c);//2
	}
}
/*
	3 - 0011
	2 - 0010
	----------
	& - 0010
			    <---
  0*1 + 1*2 +  0*4 + 0*8
  0+2+0+0 = 2
*/
```

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=5 ,b=10;
                int c= a | b;
                System.out.println(c);//15
        }
}
/*
        5 - 0101
        10- 1010
        ------------
        | - 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=5,b=10;
                int c= a ^ b;
                System.out.println(c);//15
        }
}
```

```
/*
            5 - 0101
            10- 1010
            ------------
            ^ - 1111
                                    <----
   1*1  + 1*2  + 1*4  + 1*8
            1+2+4+8 = 15
*/
```

Bitwise NOT operator (~)
----

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= ~21;
                        System.out.println(i);//-22
            }
}
```

ex:

```
class Test
{
            public static void main(String[] args)
            {
                        int i= ~-9;
                        System.out.println(i);//8
            }
}
```

Shift operators
==================

Right Shift operator (>>)
----

10 >> 1 = 10 / 2

10 >> 2 = 10 / 4

10 >> 3 = 10 / 8

10 >> 4 = 10 / 16 = 0

10 >> 5 = 10 / 32 = 0

...
...
...

ex:

```
class Test
{
	public static void main(String[] args)
	{
		int i = 20 >> 5;
		System.out.println(i);// 20 / 32 = 0
	}
}
```

ex:

```
class Test
{
	public static void main(String[] args)
	{
		int i = 100 >> 3;
		System.out.println(i);// 100 / 8 = 12
	}
}
```

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 = 30 << 4;
		System.out.println(i);// 30 * 16 = 480
	}
}
```

ex:

```
class Test
{
	public static void main(String[] args)
	{
		int i = 100 << 2;
		System.out.println(i);// 100 * 4 = 400
	}
}
```

Arithmetic Operators

======================

% - modulus

/ - division

* - multiplication

+ - addition

- - subtraction

ex:

```
class Test
{
	public static void main(String[] args)
	{
		int i = 5*6%3+7/2*5+6+7-12;
		System.out.println(i);
	}
}

/*
 (5*6%3)+(7/2*5)+(6)+(7)-12
  (5*0)+(7/2*5)+(6)+(7)-12
  (5*0)+(3*5)+(6)+(7)-12
  0+15+6+7-12
  28-12
  16
*/
```

Assignment operators
======================

ex:

```
class Test
{
	public static void main(String[] args)
	{
		int i=10;
		i+=2;//i = i + 2
		System.out.println(i);//12
	}
}
```

ex:

```
class Test
{
	public static void main(String[] args)
	{
		int i=10;
		i-=2;//i = i - 2
		System.out.println(i);//8
	}
}
```

ex:

```
class Test
{
	public static void main(String[] args)
	{
		int i=10;
		i*=2;//i = i * 2
		System.out.println(i);//20
	}
}
```

ex:

```
class Test
{
	public static void main(String[] args)
	{
		int i=10;
		i/=2;//i = i / 2
		System.out.println(i);//5
	}
}
```

ex:

```
class Test
{
	public static void main(String[] args)
	{
		int i=10;
		i%=2;//i = i % 2
		System.out.println(i);//0
	}
}
```

Conditional Operator or Ternary Operator (?:)

===

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>20)?true:false;
                        System.out.println(b);//false
            }
}
```

Relational Operator

=====================

ex:

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

ex:

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

ex:

```
class Test
{
            public static void main(String[] args)
            {
                        System.out.println(10 > 5);//true
                        System.out.println(10 < 2);//false
                        System.out.println(10 >= 10);//true
                        System.out.println(10 <= 10);//true
            }
}
```

Unary Operators
==================

Increment and Decrement operators (++/--)
--

We have two types of increment operators.

```
1)Post increment
	ex:
		i++;

2)pre increment
	ex:
		++i;
```

We have two types of decrement operators.

```
1)Post decrement
	ex:
		i--;

2)Pre decrement
	ex:
		--i;
```

ex:

Note :
	post - first take then change

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

ex:

```
class Test
{
	public static void main(String[] args)
	{
		int i=10;
		int j=i-- + i-- + i--;
		System.out.println(i+" "+j);//7  27
	}
}
```

ex:

```
class Test
{
	public static void main(String[] args)
	{
		int i=10;
		System.out.println(i++ - i--);//-1

	}
}
```

Note:

Pre- firt change 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;
		System.out.println(i+" "+j);//12  23

	}
}
```

ex:

```
class Test
{
	public static void main(String[] args)
	{
		int i=10;
		int j=--i + --i;
		System.out.println(i+" "+j);//8 17

	}
}
```

ex:

```
class Test
{
	public static void main(String[] args)
	{
		int i=10;
		System.out.println(i++ + ++i);//10+12=22

	}
}
```

Interview Programs
====================

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

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();

		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 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();

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

	}
}
```

Q)Write a java program to perform square 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 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 Test
{
	public static void main(String[] args)
	{
		Scanner sc=new Scanner(System.in);

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

		int cube=n*n*n;

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

	}
}
```

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();//5

		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();//5

		System.out.println("Enter the Third Number :");
		int c=sc.nextInt();//15

		int max=(a>b)?(a>c?a:c):(b>c?b:c);

		System.out.println("Greatest of three numbers is ="+max);

	}
}
```

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

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

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

		float area=3.14f*r*r;
		System.out.println("Area of a circle is ="+area);
	}
}
```

Q)Write a java program to find out perimeter of a circle?

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

		float perimeter=2*3.14f*r;
		System.out.println("Area of a circle is ="+perimeter);
	}
}
```

Q)Write a java program to perform swapping 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();

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

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

		int temp=a;
		a=b;
		b=temp;

		System.out.println("After swapping a="+a+"and b="+b);
	}
}
```

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

```
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();

		System.out.println("Before swapping a="+a+"and b="+b);
		a=a+b;
		b=a-b;
		a=a-b;
		System.out.println("After swapping a="+a+"and b="+b);
	}
}
```

approach2

```
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();

		System.out.println("Before swapping a="+a+"and b="+b);
		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% percent of tax from the given salary?

ex:

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

		float tax=(float)sal*10/100;
		System.out.println("10 percent of tax is ="+tax);
	}
}
```

Q)Write a java program to find out greatest of two numbers using ternary operator or conditional 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();//20

		System.out.println("Enter the Second Number :");
		int b=sc.nextInt();//15

		int max=(a>b)?a:b;
		System.out.println("Greatest of two numbers is ="+max);//20

	}
}
```

Q)Write a java program to find out greatest of three numbers using ternary operator or conditional 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();//8

		System.out.println("Enter the Second Number :");
		int b=sc.nextInt();//10

		System.out.println("Enter the Third Number :");
		int c=sc.nextInt();//4

		int max=(a>b)?(a>c?a:c):(b>c?b:c);
		System.out.println("Greatest of three numbers is ="+max);//10

	}
}
```

Control Statements
=====================

Control statement enables the programmer to control flow of our program.

Control statement allows us 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 make the decision 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:1

```
class Test
{
	public static void main(String[] args)
	{
		System.out.println("stmt1");
		if(4>2)
		{
			System.out.println("stmt2");
		}
		System.out.println("stmt3");
	}
}
```

o/p:

```
stmt1
stmt2
stmt3
```

ex:2

```
class Test
{
	public static void main(String[] args)
	{
		System.out.println("stmt1");
		if(4>20)
		{
			System.out.println("stmt2");
		}
		System.out.println("stmt3");
	}
}
```

o/p:

```
stmt1
stmt3
```

ex:3

```
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:4

```
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:

```
stmt1
stmt2
stmt3
```

Q)Write a java program to find out greatest of two numbers using if stmt?

```
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();

		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 using if stmt?

```
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();

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

		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:1

```
class Test
{
	public static void main(String[] args)
	{
		System.out.println("stmt1");
		if(true)
		{
			System.out.println("stmt2");
		}
		else
		{
			System.out.println("stmt3");
		}
		System.out.println("stmt4");
	}
}
```

o/p:

```
stmt1
stmt2
stmt4
```

ex:2

```
class Test
{
	public static void main(String[] args)
	{
		System.out.println("stmt1");
		if(false)
			System.out.println("stmt2");
		else
			System.out.println("stmt3");
		System.out.println("stmt4");
	}
}
```

o/p:

```
stmt1
stmt3
stmt4
```

Q)Write a java program to find out 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 n=sc.nextInt();//25

		if(n>=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 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();//25
		System.out.println("Enter the Second Number :");
		int b=sc.nextInt();//5

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

Q)Write a java program to find out given number is positive or negative?

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();//25

		if(n>0)
			System.out.println("It is positive number");
		else
			System.out.println("It is negative number");
	}
}
```

Q)Write a java program to find out 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();//10

		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();//10

		if(n%2==1 || 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 n=sc.nextInt();//2022

		if(n%4==0)
			System.out.println("It is a leap year");
		else
			System.out.println("It is not a leap year");
	}
}
```

Assingment

=============

Learn all java interview programs

iii)if else if ladder

========================

It will 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:1

```
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 n=sc.nextInt();

		if(n==100)
			System.out.println("It is police number");
		else if(n==103)
			System.out.println("It is Enquiry number");
		else if(n==108)
			System.out.println("It is Emergency number");
		else
			System.out.println("Invalid option ");
	}
}
```

Q)Write a java program to find out given alphabet is uppercase letter, lower case letter , digit or 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 upper case letter");
		else  if(ch>='a' && ch<='z')
			System.out.println("It is lower case letter");
		else if(ch>='0' && ch<='9')
			System.out.println("It is Digit");
		else
			System.out.println("It is special symbol");


	}
}
```

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

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);

		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");
		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 accept six marks of a student then find out total, average and grade?

i)if average is greater then equals to 70 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=89,m2=79,m3=43,m4=55,m5=39,m6=48,total=0;
		float avg;

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

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

		if(avg>=70 && avg<=100)
			System.out.println("Grade : A grade");
		else if(avg>=50 && avg<=69)
			System.out.println("Grade : B grade");
		else if(avg>=35 && avg<=49)
			System.out.println("Grade : C grade");
		else
			System.out.println("Grade : Failed ");
	}
}
```

iv) nested if stmt

=====================

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

syntax:

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

ex:1

```
class Test
{
	public static void main(String[] args)
	{
		System.out.println("stmt1");
		if(true)
		{
			System.out.println("stmt2");
			if(5>3)
			{
				System.out.println("stmt3");
			}
			System.out.println("stmt4");
		}
		System.out.println("stmt5");
	}
}
```

o/p:

```
stmt1
stmt2
stmt3
stmt4
stmt5
```

ex:2

```
class Test
{
	public static void main(String[] args)
	{
		System.out.println("stmt1");
		if(false)
		{
			System.out.println("stmt2");
			if(5>3)
			{
				System.out.println("stmt3");
			}
			System.out.println("stmt4");
		}
		System.out.println("stmt5");
	}
}
```

o/p:

```
stmt1
stmt5
```

ex:3

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

Q)Write a java program to find out given number is +ve or -ve using nested if stmt?

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();

		if(n!=0)
		{
			if(n>0)
				System.out.println("It is +ve number");
			if(n<0)
				System.out.println("It is -ve number");
		}
	}
}
```

2)Selection statement
======================

switch case

It is used to 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 false.
        }
```

Declaration of break statement is optional in switch case.

If we won't declare break statement then from where our condition is satisfied from there all cases will be executed.

The allowed datatype for a switch case are byte,short,int,char ,wrapper class ,enum and String.

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 option :");
                int n=sc.nextInt();//10

                switch(n)
                {
                        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:2

```
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 n=sc.nextInt();//103

                switch(n)
                {
                        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");
                }
        }
}
```

Q)Write a java program to find out given alphabet is a vowel or consonant?

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 consonant");
                }
        }
}
```

ex:3

```
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();

		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("Invalid string");
		}
	}
}
```

3)Iteration statement

========================

Iteration statement is used to execute the code repeatedly.

Iteration statement is possible by using loops.

We have four types of loops.

1)do while loop

2)while loop

3)for loop

4)for each loop

1)do while loop

It will execute the source code untill 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:1

```
class Test
{
	public static void main(String[] args)
	{
		int i=1;
		do
		{
			System.out.print(i);//infinite 1
		}
		while (i<=10);
	}
}
```

Q)write a java program to display 10 natural numbers?
natural numbers : 1 2 3 4 5 6 7 8 9 10

ex:

```
class Test
{
	public static void main(String[] args)
	{
		int i=1;
		do
		{
			System.out.print(i+" ");//1 2 3 4 5 6 7 8 9 10
			i++;
		}while (i<=10);
	}
}
```

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

```
class Test
{
	public static void main(String[] args)
	{
		int i=10;
		do
		{
			System.out.print(i+" ");//10 9 8 7 6 5 4 4 3 2 1
			i--;
		}while (i>=1);
	}
}
```

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 perform sum of 10 natural numbers?

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?

factorial number : n=5 --> 5*4*3*2*1 = 120

```
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 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();//

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

2)while loop

==================

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

```
syntax:
	while(condition)
	{
		-
		-//code to be execute
		-
	}
```

ex:

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

o/p: nothing

ex:

```
class Test
{
	public static void main(String[] args)
	{
		int i=1;
		while(i<=10)
		{
			System.out.println(i); //infinite 1
		}
	}
}
```

Q)Write a java program to display 10 natural numbers using while loop?

```
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 perform sum of 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 display even and odd numbers from 1 to 10?

```
import java.util.Scanner;
class Test
{
	public static void main(String[] args)
	{
		int i=1;
		while(i<=10)
		{
			if(i%2==0)
				System.out.println("even");
			else
				System.out.println("odd");

			i++;
		}

	}
}
```

Q)Write a java program to find out sum of digits of a given number?

input : 123

output : 6(1+2+3)

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,sum=0;

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

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

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

input : 121
output : It is a palindrome number

input : 123
output : It is not a 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,sum=0;

                while(n>0)
                {
                        rem=n%10;
                        sum=sum*10+rem;
                        n=n/10;
                }
                if(temp==sum)
                        System.out.println("It is 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) = (1+8+1)=(10)
output : It is not an armstrong number

input : 153 (1*1*1+5*5*5+3*3*3) = (1+125+27) = (153)
output : It is an 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 an armstrong number");
		else
			System.out.println("It is not an armstrong number");
	}
}
```

Q)Write a java program to perform sum of two numbers without using arithmetic operator i.e '+'?

ex:

```
import java.util.*;
class Test
{
	public static void main(String[] args)
	{
		Scanner sc=new Scanner(System.in);
		System.out.println("Enter the First Number :");
		int x=sc.nextInt();//3
		System.out.println("Enter the Second Number :");
		int y=sc.nextInt();//5

		while(y!=0)
		{
			int carry=x&y;
			x=x^y;
			y=carry<<1;
		}
		System.out.println("Sum of two numbers is ="+x);
	}
}
```

iii)for loop

=============

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

syntax:

```
	for(initialization;condition;incrementation/decrementation)
	{
		-
		-//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.

ex:

```
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)
	{
		int i=1;
		for(;i<=10;)
		{
			System.out.print(i+" ");//1 2 3 4 5 6 7 8 9 10
			i++;
		}
	}
}
```

ex:

```
class Test
{
	public static void main(String[] args)
	{
		for(;;)
		{
			System.out.print("hello ");//infinite hello
		}
	}
}
```

ex:

```
class Test
{
	public static void main(String[] args)
	{
		for(int i=10;i>=1;i--)
		{
			System.out.print(i+" ");//10 9 8 7 6 5 4 3 2 1
		}
	}
}
```

Q)write a java program to display 100 natural numbers?

```
class Test
{
	public static void main(String[] args)
	{
		for(int i=1;i<=100;i++)
		{
			System.out.print(i+" ");
		}
	}
}
```

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

```
class Test
{
	public static void main(String[] args)
	{
		int sum=0;
		for(int i=1;i<=10;i++)
		{
			sum=sum+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 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 perform sum of even and odd numbers from 1 to 10?

2+4+6+8+10 = 30

1+3+5+7+9 = 25

ex:

```
class Test
{
	public static void main(String[] args)
	{
		int even=0,odd=0;
		for(int i=1;i<=10;i++)
		{
				if(i%2==0)
					even+=i;
				else
					odd+=i;
		}
		System.out.println("Sum of even numbers is ="+even);
		System.out.println("Sum of odd numbers is ="+odd);
	}
}
```

Q)Write a java program to perform number of even and odd numbers from 1 to 10?

even numbers : 5

odd numbers : 5

ex:

```
class Test
{
	public static void main(String[] args)
	{
		int even=0,odd=0;
		for(int i=1;i<=10;i++)
		{
				if(i%2==0)
					even++;
				else
					odd++;
		}
		System.out.println("Number of even numbers is ="+even);
		System.out.println("Number of odd numbers is ="+odd);
	}
}
```

Q)Write a java program to perform sum of alternate even numbers from 1 to 20?

2 + 6 + 10 + 14 + 18 = 50

```
class Test
{
	public static void main(String[] args)
	{
		int sum=0;
		for(int i=1;i<=20;i++)
		{
				if(i%2==0)
				{
					sum+=i;
					i=i+2;
				}
		}
	System.out.println("Sum of alternate even numbers is :"+sum);
	}
}
```

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

prime numbers : 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97.

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(); // 10

		int flag=0;
		for(int i=2;i<=n/2;i++)
		{
			if(n%i==0)
			{
				flag=1;
				break;
			}
		}

		if(flag==0)
			System.out.println("It is prime number ");
		else
			System.out.println("It is not a prime number");

	}
}
```

Q)Write a java program to find out fibonaci series of a given number?

fibonaci series : 0 1 1 2 3 5 8

```
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+" ");
		for(int i=1;i<=n;i++)
		{
				c=a+b;
				System.out.print(c+" ");
				a=b;
				b=c;
		}
	}
}
```

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

input
base : 5
power : 3

output : 125

ex:

```
import java.util.Scanner;
class Test
{
	public static void main(String[] args)
	{
		int result=1;

		Scanner sc=new Scanner(System.in);
		System.out.println("Enter the Base Number :");
		int base=sc.nextInt();//5

		System.out.println("Enter the Power Number:");
		int power=sc.nextInt();//3

		for(int i=1;i<=power;i++)
		{
			result=base*result;
		}
		System.out.println("Power of a Number is ="+result);
	}
}
```

LOOP Patterns
===============

1)

```
1 1 1 1
2 2 2 2
3 3 3 3
4 4 4 4
```

ex:

```
import java.util.Scanner;
class Test
{
	public static void main(String[] args)
	{
		//rows
		for(int i=1;i<=4;i++)
		{
			//cols
			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
```

```
import java.util.Scanner;
class Test
{
	public static void main(String[] args)
	{
		//rows
		for(int i=1;i<=4;i++)
		{
			//cols
			for(int j=1;j<=4;j++)
			{
				System.out.print(j+" ");
			}
			//new line
			System.out.println("");
		}
	}
}
```

3)

```
A A A A
B B B B
C C C C
D D D D
```

```
import java.util.Scanner;
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("");
		}
	}
}
```

ex:4

```
A B C D
A B C D
A B C D
A B C D
```

```
import java.util.Scanner;
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(j+" ");
			}
			//new line
			System.out.println("");
		}
	}
}
```

ex:5

```
* * * *
* * * *
* * * *
* * * *
```

ex:

```
import java.util.Scanner;
class Test
{
	public static void main(String[] args)
	{
		//rows
		for(int i=1;i<=4;i++)
		{
			//cols
			for(int j=1;j<=4;j++)
			{
				System.out.print("* ");
			}
			//new line
			System.out.println("");
		}
	}
}
```

ex:6

```
4 4 4 4
3 3 3 3
2 2 2 2
1 1 1 1
```

```
import java.util.Scanner;
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("");
        }
    }
}
```

ex:7

```
D D D D
C C C C
B B B B
A A A A
```

ex:

```
import java.util.Scanner;
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("");
        }
    }
}
```

ex:8

```
* * * *
*     *
*     *
* * * *
```

ex:

```
class Test
{
	public static void main(String[] args)
	{
		for(int i=1;i<=4;i++)
		{
			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("");
		}
	}
}
```

ex:9

```
* - - -
- * - -
- - * -
- - - *
```

```
class Test
{
	public static void main(String[] args)
	{
		for(int i=1;i<=4;i++)
		{
			for(int j=1;j<=4;j++)
			{
				if(i==j)
					System.out.print("* ");
				else
					System.out.print("- ");
			}
			//new line
			System.out.println("");
		}
	}
}
```

ex:10:

```
* - - - *
- * - * -
- - * - -
- * - * -
* - - - *
```

```
class Test
{
	public static void main(String[] args)
	{
		for(int i=1;i<=5;i++)
		{
			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("");
		}
	}
}
```

Left Side Elements
===================
1)

```
1
2 2
3 3 3
4 4 4 4
```

```
class Test
{
	public static void main(String[] args)
	{
		for(int i=1;i<=4;i++)
		{
			for(int j=1;j<=i;j++)
			{
				System.out.print(i+" ");
			}
			//new line
			System.out.println("");
		}
	}
}
```

2)

```
1
1 2
1 2 3
1 2 3 4
```

```
class Test
{
	public static void main(String[] args)
	{
		for(int i=1;i<=4;i++)
		{
			for(int j=1;j<=i;j++)
			{
				System.out.print(j+" ");
			}
			//new line
			System.out.println("");
		}
	}
}
```

3)

```
*
* *
* * *
* * * *
```

```
class Test
{
	public static void main(String[] args)
	{
		for(int i=1;i<=4;i++)
		{
			for(int j=1;j<=i;j++)
			{
				System.out.print("* ");
			}
			//new line
			System.out.println("");
		}
	}
}
```

4)

```
4 4 4 4
3 3 3
2 2
1
```

```
class Test
{
	public static void main(String[] args)
	{
		for(int i=4;i>=1;i--)
		{
			for(int j=1;j<=i;j++)
			{
				System.out.print(i+" ");
			}
			//new line
			System.out.println("");
		}
	}
}
```

5)

```
* * * *
* * *
* *
*
```

```
class Test
{
	public static void main(String[] args)
	{
		for(int i=4;i>=1;i--)
		{
			for(int j=1;j<=i;j++)
			{
				System.out.print("* ");
			}
			//new line
			System.out.println("");
		}
	}
}
```

6)

```
A
B B
C C C
D D D D
```

```
class Test
{
	public static void main(String[] args)
	{
		for(char i='A';i<='D';i++)
		{
			for(char j='A';j<=i;j++)
			{
				System.out.print(i+" ");
			}
			//new line
			System.out.println("");
		}
	}
}
```

7)

```
D D D D
C C C
B B
A
```

```
class Test
{
	public static void main(String[] args)
	{
		for(char i='D';i>='A';i--)
		{
			for(char j='A';j<=i;j++)
			{
				System.out.print(i+" ");
			}
			//new line
			System.out.println("");
		}
	}
}
```

8)

```
1
2 3
4 5 6
7 8 9 0
```

ex:

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

        for(int i=1;i<=4;i++)
        {
            for(int j=1;j<=i;j++)
            {
                if(k<=9)
                    System.out.print(k++ +" ");
                else
                    System.out.print("0 ");
            }
            //new line
            System.out.println("");
        }
    }
}
```

10)

```
*
* *
* * *
* * * *
* * *
* *
*
```

```
class Test
{
    public static void main(String[] args)
    {
        for(int i=1;i<=4;i++)
        {
            for(int j=1;j<=i;j++)
            {
                    System.out.print("* ");
            }
            //new line
            System.out.println("");
        }
        for(int i=3;i>=1;i--)
        {
            for(int j=1;j<=i;j++)
            {
                System.out.print("* ");
            }
            //new line
            System.out.println("");
        }
    }
}
```

Right Side Elements

====================

1)

```
   1
  2 2
 3 3 3
4 4 4 4
```

ex:

```
class Test
{
        public static void main(String[] args)
        {
                for(int i=1;i<=4;i++)
                {
                        //space
                        for(int j=4;j>i;j--)
                        {
                                System.out.print("  ");
                        }
                        //right side 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
```

```
class Test
{
        public static void main(String[] args)
        {
                for(int i=4;i>=1;i--)
                {
                        //space
                        for(int j=4;j>i;j--)
                        {
                                System.out.print("  ");
                        }
                        //right side elements
                        for(int j=1;j<=i;j++)
                        {
                                System.out.print(i+" ");
                        }
                        //new line
                        System.out.println("");
                }
        }
}
```

3)

```
     A
   B B
  C C C
D D D D
```

```
class Test
{
	public static void main(String[] args)
	{
		for(char i='A';i<='D';i++)
		{
			//space
			for(char j='D';j>i;j--)
			{
				System.out.print("  ");
			}
			//right elements
			for(char j='A';j<=i;j++)
			{
				System.out.print(i+" ");
			}
			//new line
			System.out.println("");
		}
	}
}
```

4)

```
D D D D
  C C C
   B B
     A
```

ex

```
class Test
{
	public static void main(String[] args)
	{
		for(char i='D';i>='A';i--)
		{
			//space
			for(char j='D';j>i;j--)
			{
				System.out.print("  ");
			}
			//right elements
			for(char j='A';j<=i;j++)
			{
				System.out.print(i+" ");
			}
			//new line
			System.out.println("");
		}
	}
}
```

5)

```
   *
  * *
 * * *
* * * *
 * * *
  * *
   *
```

ex:

```
class Test
{
	public static void main(String[] args)
	{
		for(int i=1;i<=4;i++)
		{
			//space
			for(int j=4;j>i;j--)
			{
				System.out.print("  ");
			}
			//right elements
			for(int j=1;j<=i;j++)
			{
				System.out.print("* ");
			}
			//new line
			System.out.println("");
		}
		for(int i=3;i>=1;i--)
		{
			//space
			for(int j=4;j>i;j--)
			{
				System.out.print("  ");
			}
			//right 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
```

ex:

```
class Test
{
	public static void main(String[] args)
	{
		for(int i=1;i<=4;i++)
		{
			//space
			for(int j=4;j>i;j--)
			{
				System.out.print("  ");
			}
			//left side elements
			for(int j=1;j<=i;j++)
			{
				System.out.print(j+" ");
			}
			//right side elements
			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
```

ex:

```
class Test
{
	public static void main(String[] args)
	{
		for(int i=4;i>=1;i--)
		{
			//space
			for(int j=4;j>i;j--)
			{
				System.out.print("  ");
			}
			//left side elements
			for(int j=1;j<=i;j++)
			{
				System.out.print(j+" ");
			}
			//right side elements
			for(int j=i-1;j>=1;j--)
			{
				System.out.print(j+" ");
			}
			//new line
			System.out.println("");
		}

	}
}
```

3)

```
      A
    A B A
  A B C B A
A B C D C B A
```

ex:

```
class Test
{
	public static void main(String[] args)
	{
		for(char i='A';i<='D';i++)
		{
			//space
			for(char j='D';j>i;j--)
			{
				System.out.print("  ");
			}
			//left side elements
			for(char j='A';j<=i;j++)
			{
				System.out.print(j+" ");
			}
			//right side elements
			for(char j=(char)(i-1);j>='A';j--)
			{
				System.out.print(j+" ");
			}
			//new line
			System.out.println("");
		}

	}
}
```

4)

```
A B C D C B A
  A B C B A
    A B A
      A
```

ex:

```
class Test
{
	public static void main(String[] args)
	{
		for(char i='D';i>='A';i--)
		{
			//space
			for(char j='D';j>i;j--)
			{
				System.out.print("  ");
			}
			//left side elements
			for(char j='A';j<=i;j++)
			{
				System.out.print(j+" ");
			}
			//right side elements
			for(char j=(char)(i-1);j>='A';j--)
			{
				System.out.print(j+" ");
			}
			//new line
			System.out.println("");
		}

	}
}
```

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.

1)Break Statement
2)Continue Statement

1)Break Statement

It is used to break the execution of loops and switch case.

For conditional statements we can use if condition.

syntax:

```
	break statement;
```

ex:1

```
class Test
{
	public static void main(String[] args)
	{
		System.out.println("stmt1");
		break;// break outside switch or loop
		System.out.println("stmt2");
	}
}
```

ex:2

```
class Test
{
	public static void main(String[] args)
	{
		System.out.println("stmt1");
		if(true)
		{
			break;//break outside switch or loop
		}
		System.out.println("stmt2");
	}
}
```

ex:3

```
class Test
{
	public static void main(String[] args)
	{
		for(int i=1;i<=10;i++)
		{
			if(i==5)
			{
				break;
			}
			System.out.print(i+" ");//1 2 3 4
		}
	}
}
```

2)Continue Statement

Continue statement is used to continue the execution of loops.

For conditional statements we can use if condition.

syntax:

```
	continue statement;
```

ex:1

```
class Test
{
	public static void main(String[] args)
	{
		System.out.println("stmt1");
		continue;//continue outside of loop
		System.out.println("stmt2");
	}
}
```

ex:2

```
class Test
{
	public static void main(String[] args)
	{
		System.out.println("stmt1");
		if(true)
		{
				continue;//continue outside of loop
		}
		System.out.println("stmt2");
	}
}
```

ex:3

```
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
		}
	}
}
```

Assignment

==============

Q)Write a java program to display prime numbers from 1 to 100?

ARRAYS

Arrays
==============
Array is a collection of homogeneous data elements.
The main advantages of array are.

1)We can represent multiple elements using single variable name.
ex:

```
        int[]  arr={10,20,30};
```

2)Performance point of view array is recommanded to use.

The main disadvantages of array are.

1)Arrays are fixed in size.Once if we create an array there is no chance of increasing or decreasing the size of an array.

2)To use array concept ,in advanced we should know what is the size of an array which is always not possible.

Arrays are categorized into three types.

1)Single Dimensional Array

2)Two Dimensional Array / Double Dimensional Array

3)Three Dimensional Array / Multi-Dimensional Array

Array Declaration
====================
At the time of array declaration ,we should not specify array size.

Single Dimensional Array

```
int[]  arr;
int   arr[];
int   []arr;
```

Two Dimensional Array

```
int[][]  arr;
int  [][]arr;
int  arr[][];
int[]  []arr;
int[]  arr[];
int   []arr[];
```

Three Dimensional Array

```
int[][][]  arr;
int  [][][]arr;
int  arr[][][];
int[][]  []arr;
int[][]  arr[];
int[]   [][]arr;
int[]   arr[][];
int[]   []arr[];
int   [][]arr[];
int   []arr[][];
```

Array Creation

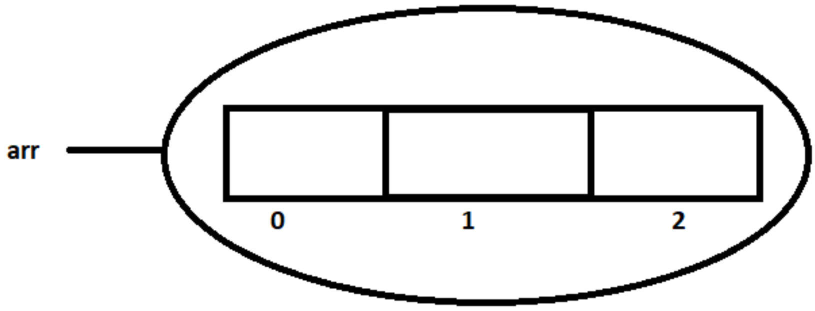
==================

In java, every array consider as an object.Hence we will use new operator to create an array.

ex:

```
int[]  arr=new int[3];
```

Diagram: java22.1



Rules to construct an array

Rule1:

At the time of array creation compulsary we need to specify array size.

ex:

```
int[] arr=new int[3]; //valid

int[] arr=new int[]; //invalid array dimension 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]; // R.E NegativeArraySizeException
```

Rule4:

The allowed datatype for an array size is byte,short,int and char. If we take other datatypes 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[10.5f];
```

Rule5:

The maximum length we can take for an array size is maximum length of an integer.

ex:

```
int[]  arr=new int[2147483647];
```

Array Initialization

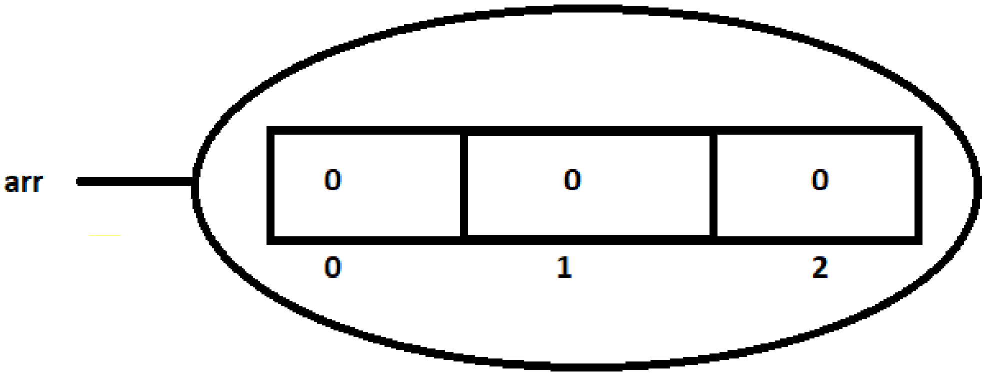
=====================

Once if we create an array , every array elements will initialized with default values.

ex:

```
int[] arr=new int[3];
```

Diagram: java22.2

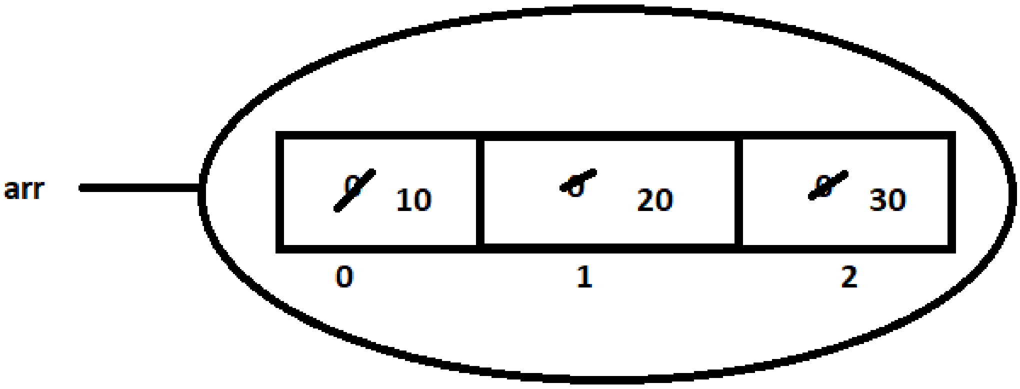


If we are not satisified with default values then we can change with customized values.

ex:

```
int[] arr=new int[3];
arr[0]=10;
arr[1]=20;
arr[2]=30;
arr[3]=40;//R.E ArrayIndexOutOfBoundsException
```

Diagram: java22.3



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};

                                ==> String[] sarr={"hi","hello","bye"};

                                ==> char[] carr={'a','b','c'};
```

Q)What is the difference between length and length() method?

length

A length 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[3];

                        System.out.println(arr);//[I@hexadecimalno

                        System.out.println(arr.length);
            }
}
```

length()

It is a final method which is applicable only for String objects.

It will return number of characters present in String.

ex:

```
class Test
{
            public static void main(String[] args)
            {
                        String str="ihub";
                        System.out.println(str.length());//4
                        System.out.println(str.length);//C.T.E cannot find symbol

            }
}
```

Single Dimensional Array Programs

===================================

Q)Write a java program to accept some array elements and display them?

```
import java.util.*;
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];

		//insert elements
		for(int i=0;i<arr.length;i++)
		{
			System.out.println("Enter the Element :");
			arr[i]=sc.nextInt();
		}

		//display elements
		for(int i=0;i<arr.length;i++)
		{
			System.out.print(arr[i]+" ");
		}
	}
}
```

approach2

```
import java.util.*;
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];

		//insert elements
		for(int i=0;i<arr.length;i++)
		{
			System.out.println("Enter the Element :");
			arr[i]=sc.nextInt();
		}

		//display elements
		//for each loop
		for(int i:arr)
		{
			System.out.print(i+" ");
		}
	}
}
```

Q)Write a java program to perform sum of array elements?

array : 2 6 8 1 3 5

sum of array elements : 25

ex:

```
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();

		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();
		}

		//sum of array elements
		int sum=0;
		for(int i=0;i<arr.length;i++)
		{
				sum=sum+arr[i];
		}

		System.out.println("Sum of array elements is ="+sum);
	}
}
```

Approach2

```
import java.util.Scanner;
class Test
{
	public static void main(String[] args)
	{
		int[] arr={2,6,8,1,3,5};

		//sum of array elements
		int sum=0;
		for(int i=0;i<arr.length;i++)
		{
				sum=sum+arr[i];
		}

		System.out.println("Sum of array elements is ="+sum);
	}
}
```

Q)Write a java program to find out number of even and odd elements from given array?

array : 2 5 8 7 9 1

even elements : 2
odd elements : 4

ex:

```
import java.util.Scanner;
class Test
{
	public static void main(String[] args)
	{
		int[] arr={2,5,8,7,9,1};

		//logic
		int even=0,odd=0;
		for(int i=0;i<arr.length;i++)
		{
				if(arr[i]%2==0)
					even++;
				else
					odd++;
		}
		System.out.println("No of even elements is ="+even);//2
		System.out.println("No of odd elements is ="+odd);//4
	}
}
```

Q)Write a java program to display highest element from given array?

array : 5 8 2 3 1 7

highest element : 8

ex:

```
import java.util.Scanner;
class Test
{
	public static void main(String[] args)
	{
		int[] arr={5,8,2,3,1,7};

		int big=arr[0];

		//logic
		for(int i=0;i<arr.length;i++)
		{
			if(arr[i]>big)
			{
				big=arr[i];
			}
		}
		System.out.println("Highest element in a given array is ="+big);
	}
}
```

Q)Write a java program to display smallest/least/lowest element from given array?

array : 5 8 2 3 1 7

least element : 1

ex:

```
import java.util.Scanner;
class Test
{
	public static void main(String[] args)
	{
		int[] arr={5,8,2,3,1,7};

		int small=arr[0];

		//logic
		for(int i=0;i<arr.length;i++)
		{
			if(arr[i]<small)
			{
				small=arr[i];
			}
		}
		System.out.println("Least element in a given array is ="+small);
	}
}
```

Q)Write a java program to display array elements in ascending order?

array : 4 7 2 9 1 5

output : 1 2 4 5 7 9

ex:

```
import java.util.Scanner;
class Test
{
	public static void main(String[] args)
	{
		int[] arr={4,7,2,9,1,5};

		//ascending logic
		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
		for(int i=0;i<arr.length;i++)
		{
			System.out.print(arr[i]+" ");
		}
	}
}
```

Q)Write a java program to display array elements in descending order?

array : 4 7 2 9 1 5
output : 9 7 5 4 2 1

```
import java.util.Scanner;
class Test
{
    public static void main(String[] args)
    {
        int[] arr={4,7,2,9,1,5};

        //descending logic
        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
        for(int i=0;i<arr.length;i++)
        {
            System.out.print(arr[i]+" ");
        }
    }
}
```

Q)Write a java program display duplicate elements from given array ?

array : 2 5 6 2 1 9 5 1 6 7 4
duplicate elements : 2 5 6 1

ex:

```
import java.util.Scanner;
class Test
{
    public static void main(String[] args)
    {
        int[] arr={2,5,6,2,1,9,5,1,6,7,4};

        //duplicate logic
        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 elements from given array?
array : 2 5 6 2 1 9 5 1 6 7 4
unique elements : 9 7 4

ex:

```
import java.util.Scanner;
class Test
{
        public static void main(String[] args)
        {
                int[] arr={2,5,6,2,1,9,5,1,6,7,4};

                //duplicate logic
                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 maximum repeating number in a given array?

```
public class Test
{
        public static void main(String[] args)
        {
                int[] arr={1,2,3,4,5,6,7,8,9,3,2,4,2};
                int maxCount=0;
                int element=0;
                for(int i=0;i<arr.length;i++)
                {
                        int count=1;
                        for(int j=i+1;j<arr.length;j++)
                        {
                                if(arr[i]==arr[j])
                                {
                                        count++;
                                }
                        }
                        if(maxCount<count)
                        {
                                maxCount=count;
                                element=arr[i];
                        }
                }
                System.out.println(element +" occurs "+maxCount+" times");
        }
}
```

Q)Write a java program to accept one element then find out number of occurance?

array : 4 6 2 1 3 1 5 1 2 9

element : 1

occurance : 3 times

ex:

```
import java.util.Scanner;
class Test
{
        public static void main(String[] args)
        {
                int[] arr={4,6,2,1,3,1,5,1,2,9};

                Scanner sc=new Scanner(System.in);
                System.out.println("Enter the element :");
                int ele=sc.nextInt();//1

                //logic
                int cnt=0;
                for(int i=0;i<arr.length;i++)
                {
                        if(arr[i]==ele)
                        {
                                cnt++;
                        }
                }
                System.out.println("Number of occurance is ="+cnt);
        }
}
```

Q)Write a java program to accept one element and delete the element from given array?

array : 4 6 2 1 3 1 5 1 2 9

element : 1

output : 4 6 2 3 5 2 9

ex:

```
import java.util.Scanner;
class Test
{
        public static void main(String[] args)
        {
                int[] arr={4,6,2,1,3,1,5,1,2,9};

                Scanner sc=new Scanner(System.in);
                System.out.println("Enter the element :");
                int ele=sc.nextInt();//1

                //logic
                for(int i=0;i<arr.length;i++)
                {
                        if(arr[i]!=ele)
                        {
                                System.out.print(arr[i]+" ");
                        }
                }
        }
}
```

Q)Write a java program to segregate 0's and 1's in given array?

input : 0,1,0,1,0,0,1,0,1,1,0

output : 0 0 0 0 0 0 1 1 1 1 1

```
class Test
{
    public static void main(String[] args)
    {
        int[] arr={0,1,0,1,0,0,1,0,1,1,0};
        int j=0;
        //inserting 0 in array
        for(int i=0;i<arr.length;i++)
        {
            //if value is equals to index 0
            if(arr[i]==0)
            {
                arr[j++]=arr[i];
            }
        }
        //now move 1 in array
        while(j<arr.length)
        {
                arr[j++]=1;
        }
        //display
        for(int k=0;k<arr.length;k++)
        {
                System.out.print(arr[k]+" ");
        }
    }
}
```

Q)Write a java program to find out three largest elements from given array?

```
class Test
{
    public static void main(String[] args)
    {
        int[] arr={9,4,5,7,8,2,6};
        int first=0,second=0,third=0;
        for(int i=0;i<arr.length;i++)
        {
            if(arr[i]>first)
            {
                third=second;
                second=first;
                first=arr[i];
            }
            else if(arr[i]>second)
            {
                third=second;
                second=arr[i];
            }
            else if(arr[i]>third)
            {
                third=arr[i];
            }
        }
        System.out.println("First Element :"+first);
        System.out.println("Second Element :"+second);
        System.out.println("Third Element :"+third);
```

```
	}
}
```

Assignment

============

Q)Write a java program to display missing elements from given array?

array : 4 2 1 8 9 7 6

missing elements : 3 5 10

Two Dimensional Array

========================

Two dimensional array is a combinatino of rows and columns.
The main objective of two dimensional array is memory utilization.
We can declare two domensional array as follow.

syntax:

```
                                                     optional
                                                         |
          datatype  variable_name[rows][cols]=initialization;
```

ex:

```
          int   arr[3][3];
```

Two dimensional array is used to develop gaming applications, business oriented applications, matrix type of applications and etc.

Q)write a java program to accept some elements and display 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[3][3];

		//inserting element
		for(int i=0;i<rows;i++)
		{
			for(int j=0;j<cols;j++)
			{
				System.out.println("Enter the Elements :");
				arr[i][j]=sc.nextInt();
			}
		}
		//display
		for(int i=0;i<rows;i++)
		{
			for(int j=0;j<cols;j++)
			{
				System.out.print(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[3][3];

                //inserting element
                for(int i=0;i<rows;i++)
                {
                        for(int j=0;j<cols;j++)
                        {
                                System.out.println("Enter the Elements :");
                                arr[i][j]=sc.nextInt();
                        }
                }

                //display
                int sum=0;
                for(int i=0;i<rows;i++)
                {
                        for(int j=0;j<cols;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[3][3];

		//inserting element
		for(int i=0;i<rows;i++)
		{
			for(int j=0;j<cols;j++)
			{
				System.out.println("Enter the Elements :");
				arr[i][j]=sc.nextInt();
			}
		}

		//display
		int sum=0;
		for(int i=0;i<rows;i++)
		{
			for(int j=0;j<cols;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 sum of 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[3][3];

		//inserting element
		for(int i=0;i<rows;i++)
		{
			for(int j=0;j<cols;j++)
			{
				System.out.println("Enter the Elements :");
				arr[i][j]=sc.nextInt();
			}
		}

		//display
		int sum=0;
		for(int i=0;i<rows;i++)
		{
			for(int j=0;j<cols;j++)
			{
				if(i>j)
				{
					sum=sum+arr[i][j];
				}
			}
		}

		System.out.println("Sum of  lower triangle elements is ="+sum);
	}
}
```

Multi-Dimensional Array

=========================

Array which contains more then two dimensions is called three/multi dimensional array.

ex:

```
	int  arr[3][3][3];

	Here it will accept 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);

		for(int i=0;i<3;i++)
		{
			for(int j=0;j<3;j++)
			{
				for(int k=0;k<3;k++)
				{
					System.out.println("Enter the element :");
					arr[i][j][k]=sc.nextInt();
				}
			}
		}
		//display
		for(int i=0;i<3;i++)
		{
			for(int j=0;j<3;j++)
			{
				for(int k=0;k<3;k++)
				{
					System.out.print(arr[i][j][k]+" ");
				}
			}
			//new line
			System.out.println("");
		}

	}

}
```

OOPS

OOPS
=====
OOPS stands for Object Oriented Programming System/Structure.

object oriented technology

A technology which provide very good environment to represent the data in the form of object is called object oriented technology.

A technology is said to be object oriented if it supports following features.
ex:

class
object
Inheritance
Abstraction
Encapsulation and
polymorphism.

class
=======
A class is a collection of data members and behaviours in a single unit.

Here data members means variables, fields and properties.

Here behaviour means methods, actions and characteristics.

In general, A class is a collection of variables and methods.

A class is also known as blueprint 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

object
==========
It is a instance of a class.
Here instance means allocating memory for our data members.

It is a outcome of a blueprint.

Memory space will be created when we create an object.

It is possible to create more then one object in a single class.

We can create object as follow.

syntax:

```
        class_name   reference_variable=new constructor();
```

ex:

```
        Test t=new Test();
```

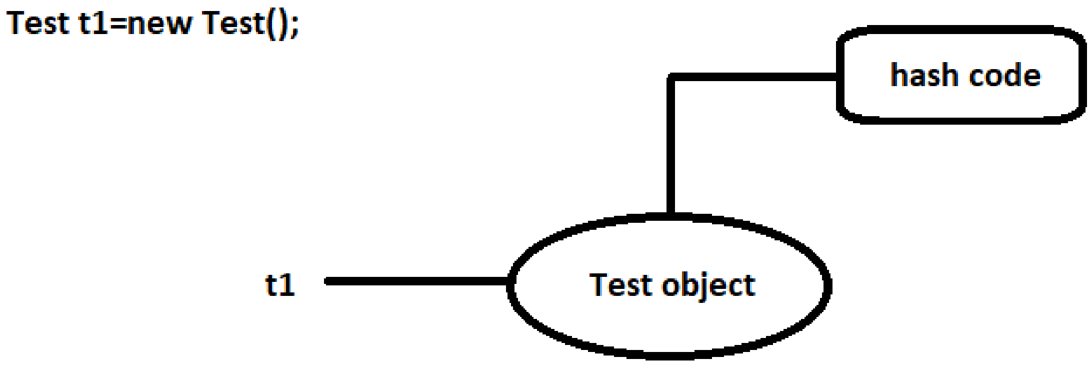
ex:

```
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);//Test@Hexadecimalno
                System.out.println(t2.toString());
                System.out.println(t3.toString());
        }
}
```

hashCode()
============
It is a method of Object class.

It will display unique identification number which is created for every object by the JVM.

Diagram: java25.1



toString()
============

It is a method of Object class.

Whenever we are trying to display object reference.Directly or indirectly toString() method will be executed.

Object class
================

Object class present in java.lang package.

It is a parent class for every java program.

Object class contains following methods which are applicable for child classes.

ex:

```
	hashCode()
	toString()
	equals()
	getClass()
	wait()
	notify()
	notifyAll()
	finalize()
	clone()
	and etc.
```

To know methods of a class we need to execute below command in command prompt.

ex:

```
	cmd> javap   java.lang.Object
```

We can check methods of a class or interface in a documentation.

ex:

https://docs.oracle.com/javase/8/docs/api/

Data Hiding
================

Our data should not go out directly.

It means outside person must not access our data directly.

Using "private" modifier we can achieve data hiding concept.

The main objective of data hiding is to provide security.

ex:

```
	class Account
	{
		private double balance;
		-
		-
	}
```

Abstraction

==================

Hiding internal implementation and highlighting the set of services is called Abstraction.

Using abstract classes and interfaces we can implement abstraction.

The best example of abstraction is GUI(Graphical User Interface) ATM machine.Where bank people hide internal implementation and highlights set of services like banking,withdrawl,mini statement and etc.

The main advantages of abstraction are.

1)It gives security because it will hide internal implementation 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

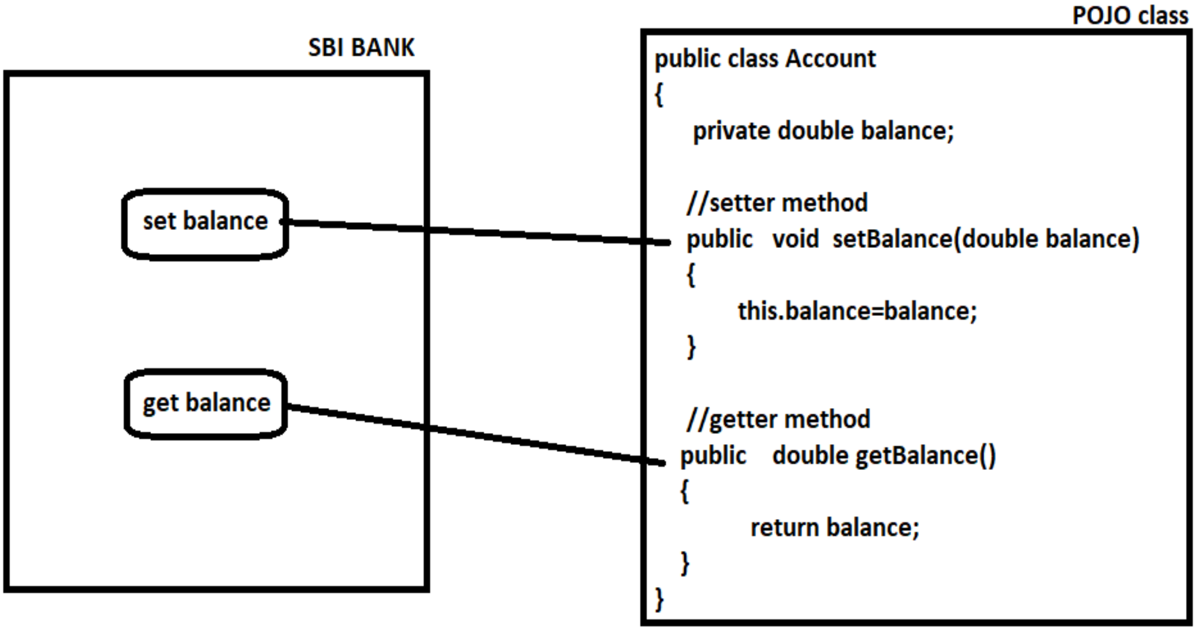
==================

The process of encapsulting or grouping variables and it's assoicate 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: java26.1



```
public class Account
{
    private double balance;

   //setter method
   public   void  setBalance(double balance)
   {
        this.balance=balance;
   }

   //getter method
  public    double getBalance()
  {
         return balance;
  }
}
```

The main advantages of encapsulation are.

1)It gives security.

2)Enhancement becomes more easy.

3)It provides flexibility to the enduser to use the system.

4)It improves maintainability of an application.

The main disadvantage of encpasulation is ,It will increase length of our code and slow down the execution process.

Approach1

============

```
class Student
{
	private int studId;
	private String studName;
	private double studFee;

	//setters and getters
	public void setStudId(int studId)
	{
		this.studId=studId;
	}
	public int getStudId()
	{
		return studId;
	}

	public void setStudName(String studName)
	{
		this.studName=studName;
	}
	public String getStudName()
	{
		return studName;
	}

	public void setStudFee(double studFee)
	{
		this.studFee=studFee;
	}
	public double getStudFee()
	{
		return studFee;
	}
	public static void main(String[] args)
	{
		Student s=new Student();
		s.setStudId(101);
		s.setStudName("Jassica");
		s.setStudFee(1000d);

		System.out.println("Student Id :"+s.getStudId());
		System.out.println("Student Name :"+s.getStudName());
		System.out.println("Student Fee :"+s.getStudFee());
	}
}
```

Approach2

============

```
class Student
{
	private int studId;
	private String studName;
	private double studFee;

	//setters and getters
	public void setStudId(int studId)
	{
		this.studId=studId;
	}
	public int getStudId()
	{
		return studId;
	}

	public void setStudName(String studName)
	{
		this.studName=studName;
	}
	public String getStudName()
	{
		return studName;
	}

	public void setStudFee(double studFee)
	{
		this.studFee=studFee;
	}
	public double getStudFee()
	{
		return studFee;
	}

}
class Test
{
	public static void main(String[] args)
	{
		Student s=new Student();
		s.setStudId(101);
		s.setStudName("Jassica");
		s.setStudFee(1000d);

		System.out.println("Student Id :"+s.getStudId());
		System.out.println("Student Name :"+s.getStudName());
		System.out.println("Student Fee :"+s.getStudFee());
	}
}
```

Interview questions
==================
Q) Is java purely object oriented or not?

ans) No, Java is not a purely object oriented programming language because it does not support many OOPS concepts like multiple inheritence,operator overloading and more ever we depends upon primitive datatypes which are non-objects.

Q)Difference between POJO class and Java Bean class?

POJO

POJO stands for Plain Old Java Object.

A class is said to be pojo class, If it supports following 2 properties.

1) All variables must be private.

2) All variables must contain setter and getter methods.

Java Bean class

A class is said to be java bean class,If it supports following 4 properties.

1) A class should be public.

2) A class should contain atleast zero argument constructor.

3) All variables must be private.

4) All variables must contain setter and getter methods.

Note:

Every Bean class is a pojo class.
But every pojo class is not a bean class.

Is-A relationship
================
Is-A relationship is also known as Inheritance.

Using "extends" keyword we can implement Is-A relationship.

The main objective of Is-A relationship is to provide 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(); X
	}
}
```

conclusion
----

1)Whatever our parent contains properties by default it goes to child.But whever our child contains properties will not goes back to parent.

2)A parent reference can hold child object.But a child reference can't hold parent object.

Types of 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 / super / base class)
        |
        |
        |
        |
        B (child / sub /  derived class)
```

ex:

```
class A
{
        public void m1()
        {
                System.out.println("A-M1 Method");
        }
}
class B extends A
{
        public void m2()
        {
                System.out.println("B-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 multi-level inheritance.

ex:

```
        A
        |
        |
        B
        |
        |
        C
```

ex:

```
class A
{
        public void m1()
        {
                System.out.println("A-M1 Method");
        }
}
class B extends A
{
        public void m2()
        {
                System.out.println("B-M2 Method");
        }
}
class C extends B
{
        public void m3()
        {
                System.out.println("C-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 multiple base classes is called multiple inheritance.

ex:

```
                A       B       C
                |       |       |
                -----------------
                        |
                        D
```

In java, we can't extends more then one class simulteneously because java does not support multiple inheritance.

ex:

```
        class A
        {
        }
        class B
        {
        }
        class C extends A,B  X
        {}
```

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             Object
        {                   |
                            |
                            |
        }                   A
```

If our class extends with some other class then our class is a indirect child class of Object class.

```
ex:                         Diag:
        class A                     Object
        {                           |
        }                           |
        class B extends A           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 may 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("A-M1 Method");
	}
}
class B extends A
{
	public void m2()
	{
		System.out.println("B-M2 Method");
	}
}
class C extends A
{
	public void m3()
	{
		System.out.println("C-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 two or more inheritance.
Java does not support hybrid inhertance.

ex:

```
                    A
                    |
      |------------------------------|
      |                              |
      B                              C
      |                              |
      |------------------------------|
                    |
                    D
```

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 funcationality
		-
	}
	class Car
	{
		Engine e=new Engine();
		-
		-
	}
```

ex:

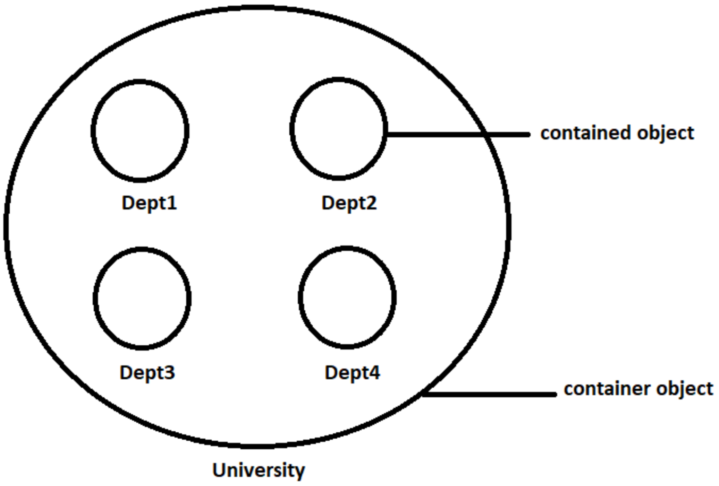
```
class Course
{
	public String courseName()
	{
		return "FSD Course";
	}
	public double courseFee()
	{
		return 25000d;
	}
	public String courseTrainer()
	{
		return "Niyaz Sir";
	}
}
```

```
class Usha
{
            public  void getCourseDetails()
            {
                        Course c=new Course();
                        System.out.println("Course Name :"+c.courseName());
                        System.out.println("Course Fee :"+c.courseFee());
                        System.out.println("Course Trainer :"+c.courseTrainer());
            }
}
class Student
{
            public static void main(String[] args)
            {
                        Usha u=new Usha();
                        u.getCourseDetails();
            }
}
```

Composition
================

Without existing container object there is no chance of having contained object then the relationship between container and contained object is called composition which is strongly association.

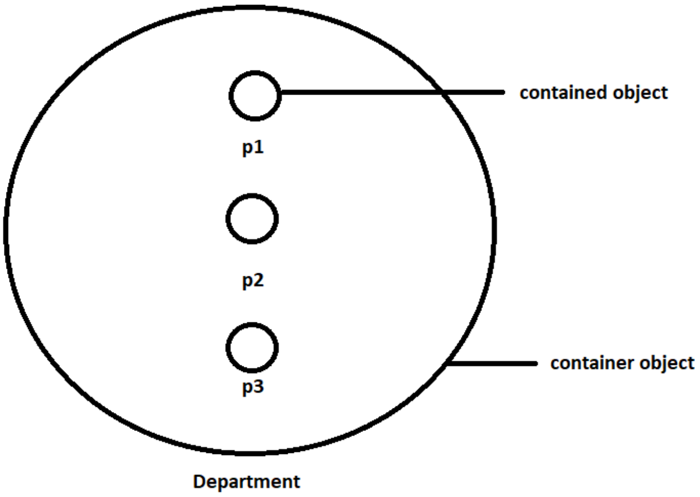
Diagram: java28.1



Aggregation
===============

Without existing container object there may chance of having contained object then the relationship between container and contained object is called aggregation which is loosely association.

Diagram: java28.2



Method Overloading

======================

Having same names with different parameters in a single class is called method overloading.

All the methods present in a class are called overloaded methods.

Method overloading reduces complexity of the programming.

ex:

```
class A
{
	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 arguments in two different classes is called method overriding.

Methods which are present in parent class are called overridden methods.

Methods which are present in 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
	public void marry()
	{
		System.out.println("Trisha/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(); //Trisha/Rashmika

		Parent p1=new Child();
		p1.property();//cash+gold+land
		p1.marry();// Trisha/Rashmika
	}
}
```

If we declare any method as final then overriding is not possible.

ex:

```
class Parent
{
	public final void property()
	{
		System.out.println("cash+gold+land");
	}
	//overridden method
	public final void marry()
	{
		System.out.println("Subhalakshmi");
	}
}
```

```
class Child extends Parent
{
        //overriding
        public void marry()
        {
                System.out.println("Trisha/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(); //Trisha/Rashmika

                Parent p1=new Child();
                p1.property();//cash+gold+land
                p1.marry();// Trisha/Rashmika
        }
}
```

o/p: C.T .E child cannot override parent method

Method Hiding
==================

Method hiding is exactly same as method overriding with following differences.

Method Overriding	Method Hiding
All the methods present in method overriding must be non-static.	All the methods present in method hiding must be static.
Method resolutation will taken care by JVM based on runtime object.	Method resolution will taken care by compiler based on reference type.
It is also known as runtime polymorphism or dynamic polymorphism or late bind.	It is also known as compile time polymorphism, static 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("Trisha/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(); //Trisha/Rashmika

		Parent p1=new Child();
		p1.property();//cash+gold+land
		p1.marry();// Subhalakshmi
	}
}
```

Q)Can we overload main method in java?
ans)
Yes , we can overload main method in java. But JVM always 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[] args)
		{
				System.out.println("int-arg method");
		}
}
```

Polymorphism
=================

Polymorphism has taken from Greek word.

A poly means many and morphism means forms.

The ability to represent in different forms is called polymorphism.

The main objective of polymorphism is used to provide flexibilty.

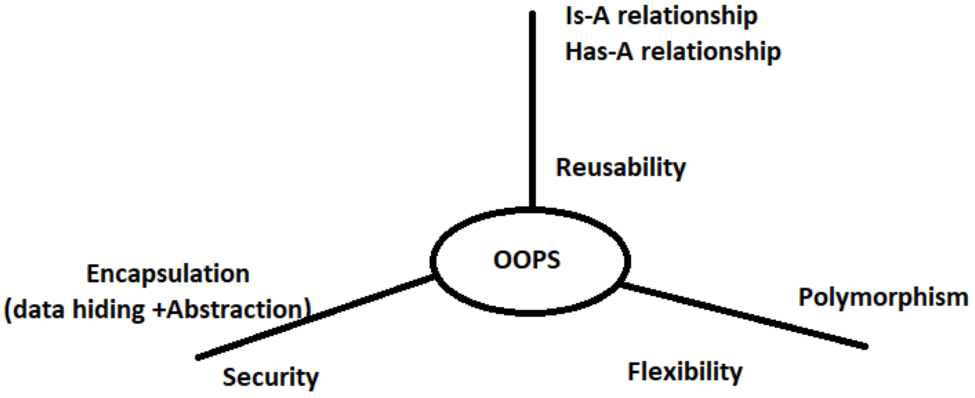
We have two types of polymorhism.

1)Compile time polymorphism / Static polymorphism / early binding
----
A polymorphism which exhibits at compile time is called compile time polymorphism.
ex:
Method Overloading
Method Hiding

2)Runtime polymorphism / Dynamic polymorphism / late binding

A polymorphism which exhibits at runtime is called runtime polymorphism.
ex:
Method overriding

<u>Summary Diagram</u>: java29.1



Constructors

==============

A constructor is a special method which is used to initialized an object.

Having same name as class name is called constructor.

Constructor does not allow any return type.If we take any returntype then we won't get any compile time error or runtime error.

A constructor will execute when we create an object.

A constructor will accept following modifiers.

ex:

```
		default
		public
		private
		protected
```

A constructor must and should declare immediately after the class.

In java , constructors are divided into two types.

1)Userdefine constructor

2)Default constructor

1)Userdefine constructor

A constructor which is created by the user based on the application requirement is called userdefine constructor.

It is categories into two types.

i)Zero Argument Constructor

ii)Parameterized Constructor

i)Zero Argument Constructor

Suppose if we are not passing any argument to userdefine constructor then that constructor is called zero-argument constructor.

ex:

```
class Test
{
		//constructor
		Test()
		{
				System.out.println("0-arg const");
		}
		public static void main(String[] args)
		{
				System.out.println("Main Method");
		}

}
```

o/p:

```
Main Method
```

ex:2

```
class Test
{
	//constructor
	Test()
	{
		System.out.println("0-arg const");
	}
	public static void main(String[] args)
	{
		System.out.println("Main Method");
		Test t=new Test();
	}

}
```

o/p:

```
Main Method
0-arg const
```

ex:3

```
class Test
{
	//constructor
	Test()
	{
		System.out.println("0-arg const");
	}
	public static void main(String[] args)
	{
		Test t1=new Test();
		System.out.println("Main Method");
		Test t2=new Test();
	}

}
```

o/p:

```
0-arg const
Main Method
0-arg const
```

ii) parameterized constructor

Suppose if we are passing atleast one argument to userdefine constructor then that constructor is called parameterized constructor.

ex:

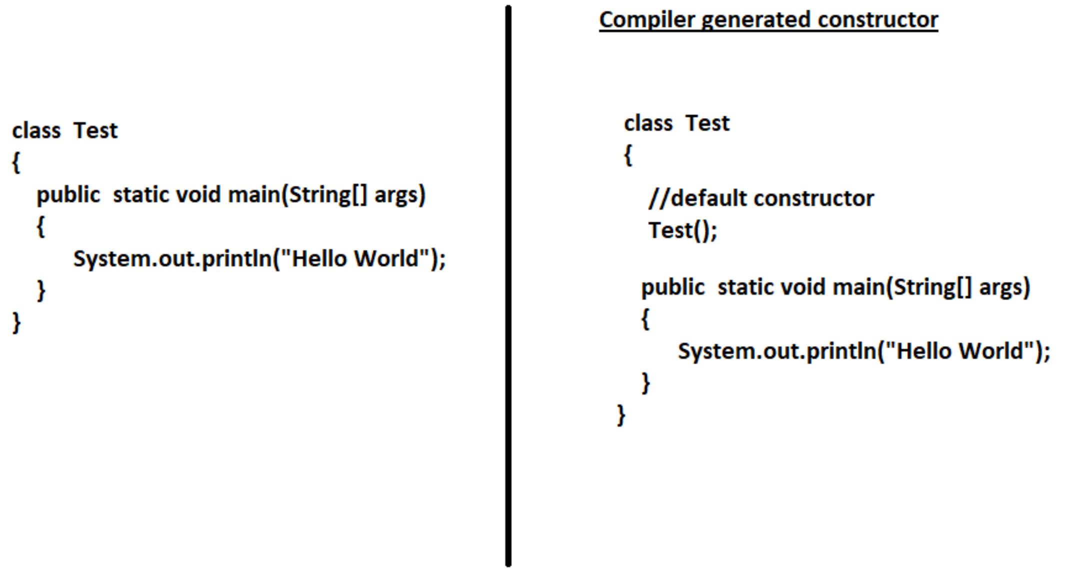
```
class Employee
{
	//instance variables
	private int empId;
	private String empName;
	private double empSal;
	//parameterized constructor
	public Employee(int empId,String empName,double empSal)
	{
		this.empId=empId;
		this.empName=empName;
		this.empSal=empSal;
	}
	public void getEmployeeDetails()
	{
		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(101,"Luci",1000d);
		e.getEmployeeDetails();
	}
}
```

2)Default constructor

========================

It is a compiler generated constructor for every java program where we are not defining atleast zero-argument constructor.

Diagram : java29.2



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

Compiler generated constructor

```
class  Test
{
   //default constructor
   Test();

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

If we want to see default constructor we need to use below command.

ex:

```
	cmd> javap  -c  Test
```

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(double d)
	{
		System.out.println("double-arg const");
	}
}
class Test
{
	public static void main(String[] args)
	{
		A a1=new A();
		A a2=new A(10);
		A a3=new A(10.5d);
	}
}
```

this keyword
================

A "this" keyword is a java keyword which is used to refer current class object reference.

We can utilize this keyword in following ways.

i)We can refer current class variables.

ii)We can refer current class methods.

iii)We can refer current class constructors.

i)We can refer current class variables
----
syntax:

```
	this.variable_name
```

ex:

```
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);
	}
}
```

ii)We can refer current class methods

----

syntax:

```
	this.method_name
```

ex:

```
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();
	}
}
```

iii)We can refer current class constructors

---

syntax:

```
	this();
	or
	this(value);
```

ex:

```
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)
	{
		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.

i)To refer super class variables

ii)To refer super class methods

iii)To refer super class constructors

i)To refer super class variables

----

syntax:

```
	super.variable_name
```

ex:

```
class A
{
	int i=10;
	int j=20;
}
class B extends A
{
	int i=100;
	int j=200;
	B(int i,int j)
	{
		System.out.println(this.i+" "+this.j);//100 200
		System.out.println(super.i+" "+super.j);//10  20
		System.out.println(i+" "+j);//1000 2000
	}
}
class Test
{
	public static void main(String[] args)
	{
		B b=new B(1000,2000);
	}
}
```

ii)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();
	}
}
```

iii)To refer super class constructors

----

syntax:

```
	super();
	or
	super(value);
```

ex:

```
class A
{
	A()
	{
		System.out.println("A const");
	}
}
class B extends A
{
	B()
	{
		super();
		System.out.println("B const");
	}
}
class Test
{
	public static void main(String[] args)
	{
		B b=new B();

	}
}
```

Interfaces

===============

It is a collection of zero or more abstract methods.

Abstract method is a incomplete method which ends with semicolon and does not have any body.

ex:

```
	void  m1();
```

It is not possible to create object for interfaces.

To write implemenation of abstract methods of an interface we will use implementation class.

It is possible to create object for implementation class because it contains method with body.

By default every abstract method is public and abstract.

ex:

```
	public abstract void m1();
```

Interface contains only constants i.e public static final.

syntax:

```
	optional
	|
	modifier interface interface_name
	{
		-
		-//constants
		-//abstract methods
		-
	}
```

ex:1

```
interface A
{
	public abstract void m1();
}
class B implements A
{
	public  void m1()
	{
		System.out.println("M1 Method");
	}
}
class Test
{
	public static void main(String[] args)
	{
		B b=new B();
		b.m1();

		//or
		A a=new B();
		a.m1();
	}
}
```

ex:2

```
interface A
{
	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("A-m1 method");
			}
		};
		a.m1();
	}
}
```

Note:

If our interface contains 4 methods then we need to override all methods otherwise we will get compile time error.

ex:

```
interface A
{
	public abstract void m1();
	public void m2();
	abstract void m3();
	void m4();
}
class B implements A
{
	public void m1()
	{
		System.out.println("M1 Method");
	}
	public void m2()
	{
		System.out.println("M2 Method");
	}
	public void m3()
	{
		System.out.println("M3 Method");
	}
	public void m4()
	{
		System.out.println("M4 Method");
	}
}

class Test
{
	public static void main(String[] args)
	{
		A a=new B();
		a.m1();
		a.m2();
		a.m3();
		a.m4();
	}
}
```

In java , a class can't extends more then one class simultenously.
But interface can extends more then one interface.

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)
	{
		D d=new D();
		d.m1();
		d.m2();
		d.m3();
	}
}
```

A class can implements more then one interface simultenously.

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();
	}
}
```

Note:

According to java 1.8 version, interface is a collection of abstract methods, default methods and static methods.

Abstract classes
==================

Abstract class is a collection of zero or more abstract method and zero or more concrete methods.

A "abstract" keyword is applicable for methods and classes but not for variables.

It is not possible to create object for abstract class.

To implement abstract methods 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;

	public abstract void getRate();

	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.
interface is a collection of abstract methods,default methods and static methods.	Abstract class is a collection of abstract methods and concrete methods.
We can't declare blocks.	We can declare blocks.
We can't declare constructor.	We can declare constructor.
It supports multiple inheritance.	It does not support multiple inheritance.
It contains only constants.	It contains instance variables.
If we know specifications then we need to use interface.	If we know partial implementation then we need to use abstract class.

Inner classes

==================

Sometimes we will declare a class inside another class such concept is called inner class.

syntax:

```
        class Outer_class
        {
                class Inner_class
                {
                        -
                        -
                        -
                }
        }
```

Inner classes introduced as a part of event handling to remove GUI bugs.

But due to powerful features and benefits of inner classes.Programmers started to use in regular programming code.

Inner class does not accept static members i.e static variables, static methods and static blocks.

Accessing inner class data from static area of outer class

===

approach1:

```
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();

	}
}
```

Note:

If we compile above program two .class files will be generated i.e Outer.class and Outer$Inner.class.

approach2:

```
class Outer
{
	class Inner
	{
		public void m1()
		{
			System.out.println("M1 Method");
		}
	}

	public static void main(String[] args)
	{
		new Outer().new Inner().m1();

	}
}
```

Note:

If we compile above program two .class files will be generated 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();
	}
}
```

Note:

If we compile above program two .class files will be generated i.e
Outer.class and Outer$Inner.class.

Enum
========

Enum is a group of named constants.

Enum is a special class which was introduced in 1.5 version.

Using enum we can create our own datatype called enumerated datatype.

To declare enum we will use "enum" keyword.

When compare to old language enum, java enum is more powerful.

syntax:

```
	enum  enum_name
	{
		value1,value2,...,valueN
	}
```

ex:

```
	enum  Months
	{
		JAN,FEB,MAR
	}
```

Internal implementation of enum

==================================

Every enum is a class concept and it extends with java.lang.Enum class.

Every enum constant is a reference variable of type enum.

```
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.DEC;
                    System.out.println(m);//C.T.E cannot find symbol
          }
}
```

enum vs switch case

=====================

From 1.5 version onwards switch case allowed enum.

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;
		}
	}
}
```

enum vs inheritance

===================

By seeing below programs we can conclude that inheritance concept is not applicable for enum.

1)

```
class A
{
}
enum B extends A
{
}
```

o/p: invalid

2)

```
enum A
{
}
enum B extends A
{
}
```

o/p: invalid

java.lang.Enum
=================

Power to enum will be inherited from java.lang.Enum class.

It contains following two methods.

1)values()

It is a static method which is used to read all the constants of enum.

2)ordinal()

It will 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:

Months.java

```
enum Months
{
	JAN,FEB,MAR;

	static int i=10;

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

Wrapper classes
=================

The main objective of wrapper classes are.

1)To wrap primitive type to wrapper object and vice versa.

2)To define serveral 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 following two constructors. One will take primitive as an argument and another will take 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
Character	char

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);
	}
}
```

ex:2

```
class Test
{
	public static void main(String[] args)
	{
		Boolean b1=new Boolean(true);
		Boolean b2=new Boolean("true");

		System.out.println(b1+" "+b2);
	}
}
```

ex:3

```
class Test
{
	public static void main(String[] args)
	{
		Character c=new Character('a');
		System.out.println(c);
	}
}
```

Utility Methods
=================

1)valueOf() method

It is similar to constructor.
It is used to convert primitive type 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() method

It is used to convert wrapper object to primitive type.

ex:

```
class Test
{
	public static void main(String[] args)
	{
		Integer i1=new Integer(10);
		byte b=i1.byteValue();
		System.out.println(b);//10

		Integer i2=new Integer("20");
		short s=i2.shortValue();
		System.out.println(s);//20
	}
}
```

3)parseXxx() method

It is used to convert String type to primitive type.

ex:

```
class Test
{
	public static void main(String[] args)
	{
		String str="10";
		int i1=Integer.parseInt(str);
		System.out.println(i1);

		long l1=Long.parseLong(str);
		System.out.println(l1);
	}
}
```

4)toString() method

It is used to convert wrapper object to String type.

ex:

```
class Test
{
	public static void main(String[] args)
	{
		Integer i1=new Integer(10);
		String s=i1.toString();
		System.out.println(s);
	}
}
```

Interview Questions

===================

Q)Given an array of size n and an integer X.Find if there's a triplet in the array which sum up to the given integer X.

Input :

```
		int[] arr={1,4,45,6,10,8};
		n=6;
		int x=13;
```

output :		1 4 8

ex:

```
class Test
{
	public static void main(String[] args)
	{
		int[] arr={1,4,45,6,10,8};
		int x=13,n=6;

		for(int i=0;i<n;i++)
		{
			for(int j=i+1;j<n;j++)
			{
				for(int k=j+1;k<n;k++)
				{
					if(arr[i]+arr[j]+arr[k]== x)
					{
			System.out.println(arr[i]+" "+arr[j]+" "+arr[k]);
					}
				}
			}
		}
	}
}
```

Q)

Count triplets with sum smaller than a given value

Given an array of distinct integers and a sum value. Find count of triplets with sum smaller than given sum value.

Input :

```
	arr[] = {-2, 0, 1, 3},
	sum = 2
```

ex:

```
class Test
{
	public static void main(String[] args)
	{
		int[] arr={-2, 0, 1, 3};
		int sum=2,n=4,cnt=0;

		for(int i=0;i<n;i++)
		{
			for(int j=i+1;j<n;j++)
			{
				for(int k=j+1;k<n;k++)
				{
					if(arr[i]+arr[j]+arr[k]< sum)
					{
						cnt++;
					}
				}
			}
		}
		System.out.println(cnt);
	}
}
```

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, for every change a new object will be created such type of object is called immutable object.

ex:

String and Wrapper classes

2)Mutable object

After object creation if we perform any changes,all the changes will be done to a single object such type of object is called mutable object.

ex:

StringBuffer and StringBuilder

String
==========

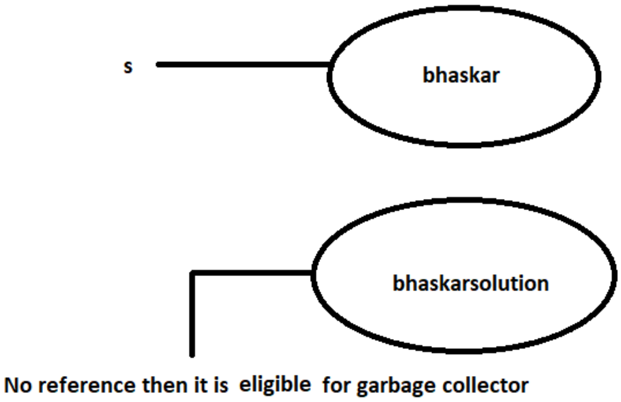
case1:

Once if we create a String object we can't perform any changes.If we perform any changes for every change a new object will be created such behaviour is called immutability of an object.

ex:

```
		String s=new String("bhaskar");
		s.concat("solution");
		System.out.println(s);// bhaskar
```

Diagram: java33.1



case2:

Difference between == and .equals() method?

==

It is a equality/comparision operator which returns boolean values.
It is used for address comparision or reference 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);
		}
}
```

.equals()

It is a method present in String class and it returns boolean values.
It is used for content comparision.
It is case sensitive.

ex:

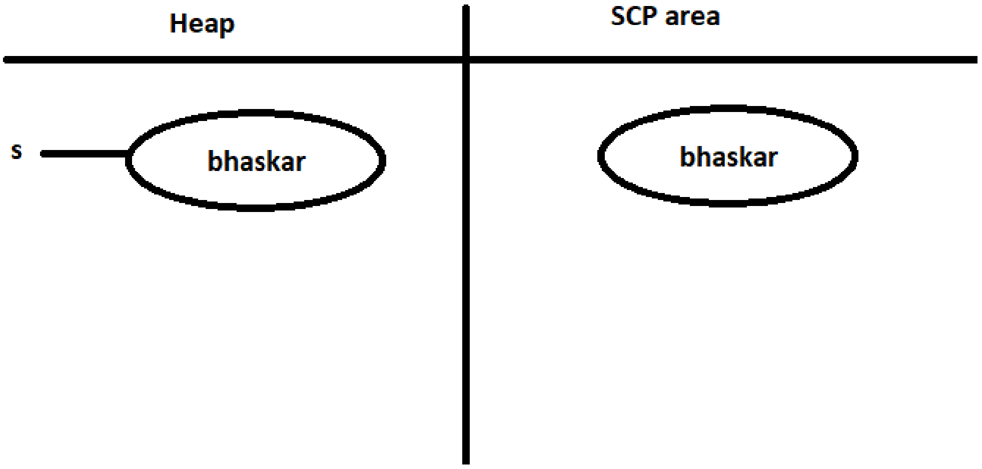
```
class Test
{
		public static void main(String[] args)
		{
				String s1=new String("bhaskar");
				String s2=new String("bhaskar");
				System.out.println(s1.equals(s2));//true
				System.out.println(s1.equals("BHASKAR"));//false
		}
}
```

case 3:

Once if we create a 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 only.
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 JVM 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.

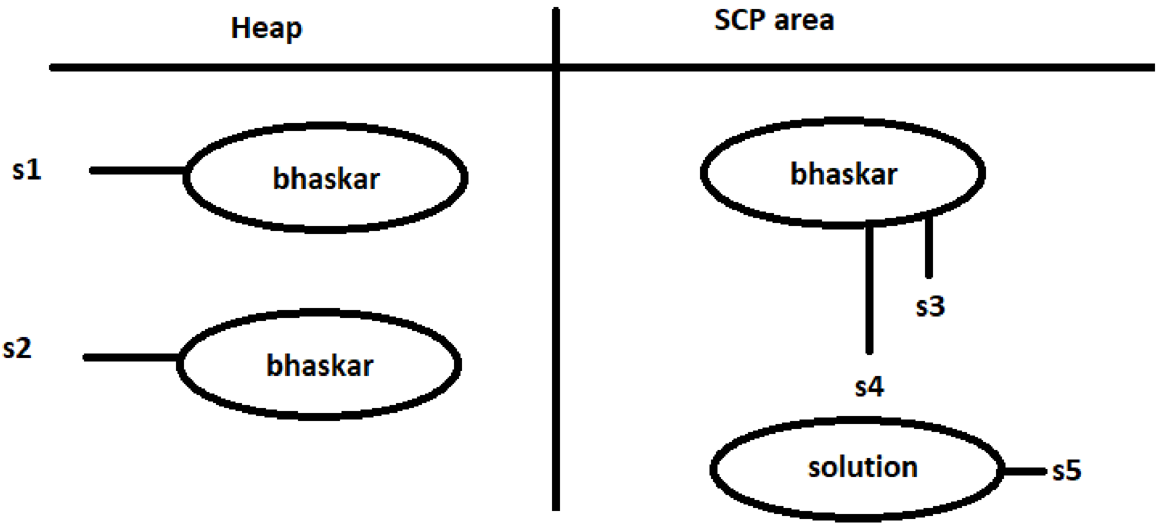
Even though SCP object do not have any reference.Garbage collector can't access them.

When JVM shutdowns , SCP objects will be destroyed automatically.

ex:

```
        String s1=new String("bhaskar");
        String s2=new String("bhaskar");
        String s3="bhaskar";
        String s4="bhaskar";
        String s5="solution";
```

Diagram: java33.3



Interning of String Object

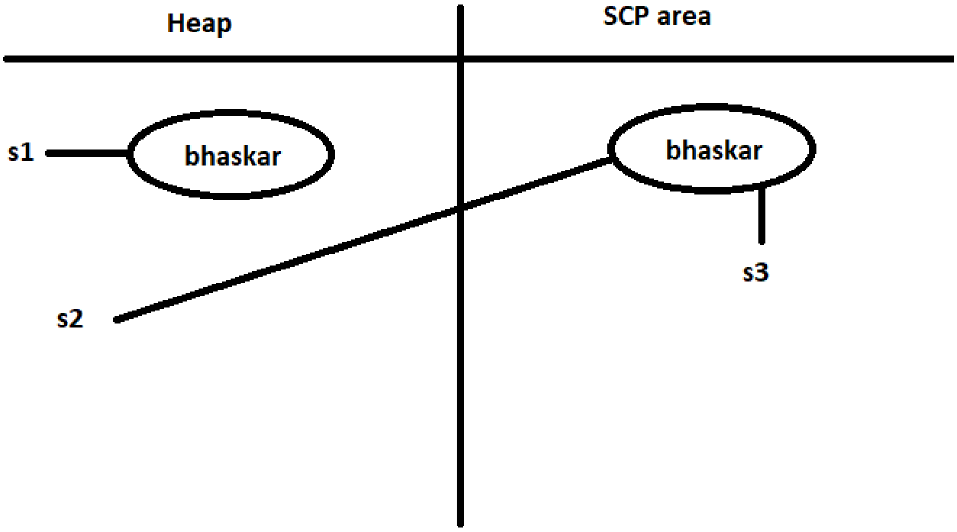
==============================

With the help of heap object reference if we need corresponding SCP object reference then we need to use intern() method.

ex:

```
	String s1=new String("bhaskar");
	String s2=s1.intern();
	String s3="bhaskar";
	System.out.println(s2==s3);//true
```

<u>Diagram</u>: java33.4



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.indexOf('a'));//2

		System.out.println(str.lastIndexOf('a'));//5

		System.out.println(str.concat("solution"));//bhaskarsolution

		System.out.println(str.replaceAll(str,"jojo"));//jojo

		System.out.println(str.contains("kar"));//true
	}
}
```

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 name=sc.next();
		System.out.println("Welcome ="+name);
	}
}
```

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 name=sc.nextLine();
		System.out.println("Welcome ="+name);
	}
}
```

Q)Write a java program to display length of the string?

```
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();
		System.out.println("Length of the string is ="+str.length());
	}
}
```

Q)Write a java program to compare two strings?

```
import java.util.Scanner;
class Test
{
	public static void main(String[] args)
	{
		Scanner sc=new Scanner(System.in);
		System.out.println("Enter the String1 :");
		String str1=sc.next();
		System.out.println("Enter the String2 :");
		String str2=sc.next();
		if(str1.equals(str2))
			System.out.println("Both are equals");
		else
			System.out.println("Both are not equals");
	}
}
```

Q)Write a java program to display reverse of a string?

input: This Is Java Class
output: ssalC avaJ sI sihT

ex:

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

		System.out.println("Enter the String1 :");
		String str=sc.nextLine();

		//convert string to char array
		char[] carr=str.toCharArray();

		//reading character in reverse
		for(int i=carr.length-1;i>=0;i--)
		{
			System.out.print(carr[i]);
		}

	}
}
```

Q)Write a java program to display reverse of a sentence?

input : This is Java Class
output : class Java Is This

ex:

```
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();

		String[] sarr=str.split(" ");

		//read in  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?

input : This Is Java Class
output: sihT sI avaJ ssalC

ex:

```
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();

                String[] sarr=str.split(" ");

                //for each loop
                for(String s1: sarr)
                {
                        //convert string to char array
                        char[] carr=s1.toCharArray();

                        //read in reverse
                        for(int i=carr.length-1;i>=0;i--)
                        {
                                System.out.print(carr[i]);
                        }
                        //space
                        System.out.print(" ");
                }
        }
}
```

StringBuffer
================

If our content change frequently then it is not recommanded to use String object.

To overcome this limitation sun micro system introduced StringBuffer concept.

In StringBuffer concept all the required changes will be done in a same or single object.

constructor
----

1)StringBuffer sb=new StringBuffer()

It will create empty StringBuffer object with default initial capacity of 16.
If we reach to maximum capacity then new capacity will be created with below formulea.
ex:

```
        new capacity = current_capacity+1*2;
```

ex:

```
class Test
{
	public static void main(String[] args)
	{
		StringBuffer sb=new StringBuffer();
		System.out.println(sb.capacity());//16

		sb.append("abcdefghijklmnop");
		System.out.println(sb.capacity());//16

		sb.append("qr");
		System.out.println(sb.capacity());//16+1*2= 34
	}
}
```

2)StringBuffer sb=new StringBuffer(initialcapacity)

It will create StringBuffer object with default 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 which is equivalent to String.
Here capacity will be created with below formulea.

ex:

```
	capacity = s.length()+16
```

ex:

```
class Test
{
	public static void main(String[] args)
	{
		StringBuffer sb=new StringBuffer("bhaskar");
		System.out.println(sb.capacity());//7+16=23
	}
}
```

Q)Write a java program to display reverse of a String?

```
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();
		StringBuffer sb=new StringBuffer(str);
		System.out.println(sb.reverse());
	}
}
```

Q) Write a java program to display number of vowels present in a given string?

input : hello
output : e o

ex:

```
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();

		char[] carr=str.toCharArray();
		for(char c:carr)
		{
			if(c=='a' || c=='e' || c=='i' || c =='o' || c=='u')
				System.out.print(c +" ");
		}
	}
}
```

Q) Write a java program to remove duplicate characters from String?
input : Google
output : Gogle

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

		StringBuffer sb=new StringBuffer();
		str.chars().distinct().forEach(c->sb.append((char)c));
		System.out.println(sb);
	}
}
```

Q) Write a java program to display duplicate characters from String?

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

		String characters="";
		String duplicates="";
```

```
        //reading one by one character from string
        for(int i=0;i<str.length();i++)
        {
            //converting each character to String
            String current=Character.toString(str.charAt(i));

            //checking String is available or not.
            if(characters.contains(current))
            {
                //checking string is not present in duplicates variable
                if(!duplicates.contains(current))
                {
                    //add the string
                    duplicates+=current;
                }
            }
            characters+=current;
        }
        System.out.println(duplicates);
    }
}
```

Q) Write a java program to display unique characters from String?

ex:

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

        String characters="";
        String duplicates="";

        //reading one by one character from string
        for(int i=0;i<str.length();i++)
        {
            //converting each character to String
            String current=Character.toString(str.charAt(i));

            //checking String is available or not.
            if(characters.contains(current))
            {
                //checking string is not present in duplicates variable
                if(!duplicates.contains(current))
                {
                    //add the string
                    duplicates+=current;
                }
                continue;
            }
            characters+=current;
        }
        System.out.println(characters);
    }
}
```

StringBuilder
===============
StringBuilder is exactly same as StringBuffer with following differences.

StringBuffer --------------	**StringBuilder** ------------------
All methods present in StringBuffer are synchronized.	No method present in StringBuilder is synchronized.
At a time only one thread is allowed to access that object.Hence we achieve thread safety.	Multiple threads are allowed to access that object.Hence we can't achieve thread safety.
Waiting time of a thread will increases effectively performance is low.	There is no waiting time effectively performance is high.
It is introduced in 1.0v.	It is introduced in 1.5v.

Note:

If our content not change frequently then we need to use String.

If our content change frequently where thread safety is required then we need to use StringBuffer.

If our content change frequently where thread safety is not required then we need to use StringBuilder.

StringTokenizer
==================
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 StringTokenize 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:2

```
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:3

```
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:4

```
import java.util.StringTokenizer;
class Test
{
	public static void main(String[] args)
	{
		StringTokenizer st=new StringTokenizer("This Is Java Class"," ");
		while(st.hasMoreElements())
		{
				String s=(String)st.nextElement();
				System.out.println(s);
		}
	}
}
```

ex:5

```
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

Exception Handling
=====================

Q)Difference between Exception and Error?

Exception

Exception is a problem for which we can provide solution programmatically.
Exception will occur due syntax errors.
ex:

  ArithmeticException
  FileNotFoundException
  IllegalArgumentException

Error

Error is a problem for which we can't provide solution programmatically.
Error will occur due to lack of system resources.
ex:

  OutOfMemoryError
  StackOverFlowError
  LinkageError

As a part of 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

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 any 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 we must and should handle that exception otherwise our program will terminates abnormally.

Exception will display name of the exception, description of the exception and line number of the 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)Predefine exceptions
2)User define exceptions

1)Predefine exceptions

Built in exceptions are called predefine exceptions.

Predefine exceptions are divided into two types.

i)Checked exceptions

Exceptions which are checked by the compiler at the time of compilation are called checked exceptions.

ex:

FileNotFoundException
InterruptedException
EOFException

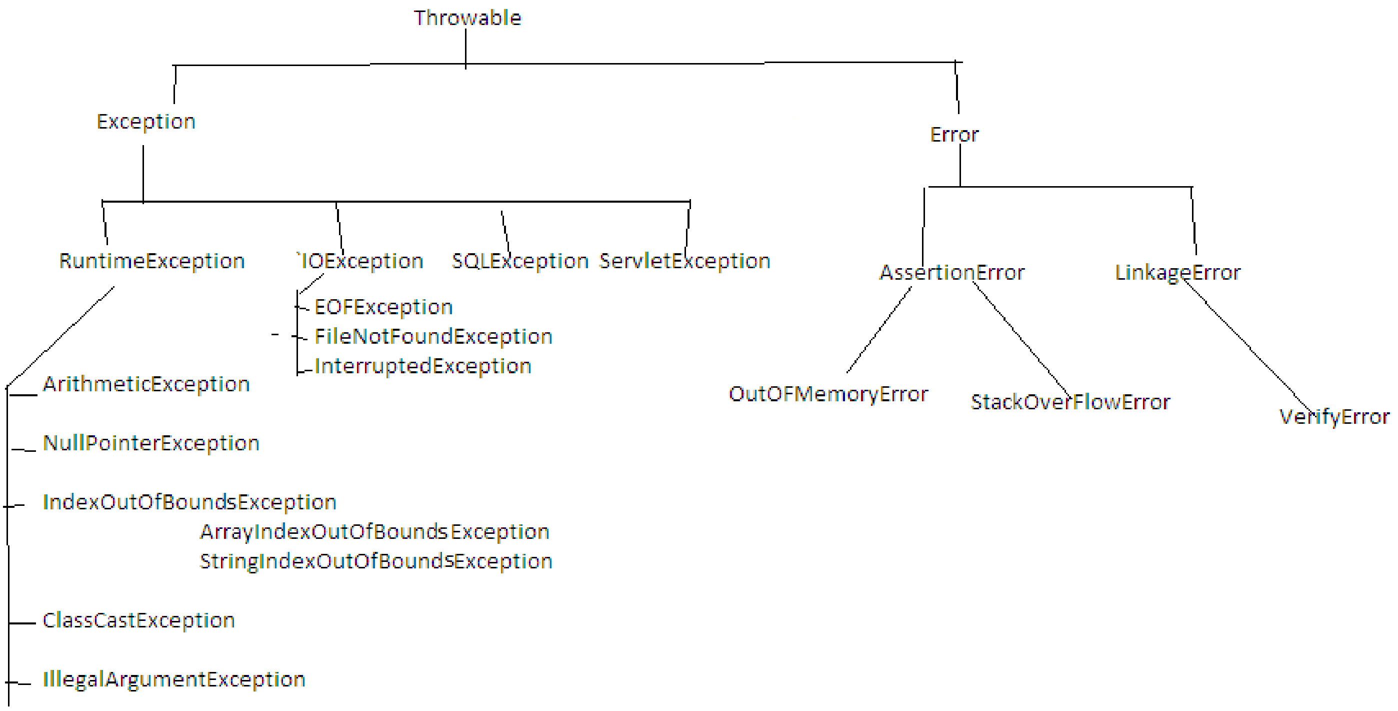
ii)Unchecked exceptions

Exceptions which are checked by the jvm at the time of runtime are called unchecked exceptions.

ex:

ArithmeticException
IllegalArgumentException
ClassCastException

Diagram: java35.1



Checked exceptions again divided into two types.

a)Fully checked exceptions

A class is said to be fully checked exception if and only if it sub classes must be checked exceptions.

b)partially checked exceptions

A class is said to be partially checked exception if and only if atleast one class must be unchecked exception.

If any checked exception raised in our program we must and should handle the exception by using try and catch block.

try block
=============
It is a block which contains Risky Code.

It always associate with catch block.

If any exception raised in try block then try block won't be executed.

A try block is used to throw the exception to catch block.

If exception comes in the middle of the try block then rest of the code won't be executed.

catch block
=============
It is a block which contains Error Handling code.

It always associate with try block.

It is used to catch the exception which is throws by try block.

A catch block will take exception name as parameter and that name must match with exception class name .

If no exception raised in try block then catch block won't be executed.

syntax:

```
 try
 {
             -
             - //risky code
             -
 }
 catch(ArithmeticException ae)
 {
             -
             -
             -
 }
```

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(Exception e)
		{
			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
```

A try with multiple catch blocks

===================================

It is possible to declare a try with multiple catch blocks.

If we declare a try block with multiple catch blocks then order of catch blocks are very important.It should be child to parent but not from parent to child.

ex:

```
class Test
{
        public static void main(String[] args)
        {
                try
                {
                        System.out.println(10/0);
                }
                catch(ArithmeticException ae)
                {
                        System.out.println("From AE");
                }
                catch(RuntimeException re)
                {
                        System.out.println("From RE");
                }
                catch(Exception e)
                {
                        System.out.println("From E");
                }
        }
}
```

Methods to display exception details

======================================

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.

ex:

```
class Test
{
        public static void main(String[] args)
        {
                try
                {
                        System.out.println(10/0);
                }
                catch(ArithmeticException ae)
                {
                        ae.printStackTrace();

                        System.out.println("======================");

                        System.out.println(ae.toString());

                        System.out.println("======================");

                        System.out.println(ae.getMessage());
                }

        }
}
```

finally block
=================

It is a block which contains cleanup code.

finally block always associate with try and catch block.

It is never recommanded to maintain cleanup code in try block because if any exception comes in try block won't be executed.

It is never recommanded to maintain cleanup code in catch block becausae if no exception raised 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. Handle or not.such block is called finally block.

syntax:

```
try
{
        -
        -//Risky Code
        -
}
catch(Exception e)
{
        -
        -//Error Handling Code
        -
}
finally
{
        -
        -//cleanup code
        -
}
```

ex:1

```
class Test
{
	public static void main(String[] args)
	{
		try
		{
			System.out.println("Try-Block");
		}
		catch(Exception e)
		{
			e.printStackTrace();
		}
		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(Exception e)
		{
			e.printStackTrace();
		}
		finally
		{
			System.out.println("Finally-Block");
		}

	}
}
```

o/p:

```
java.lang.ArithmeticException: / by zero
	at Test.main(Test.java:7)
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");
                        }

            }
}
```

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 the 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. Either handle or not.

finalized

It is a method called by garbage collector just before destroying an object for cleanup activity.

throw statement
=================

Sometimes we will create exception object explicitly and handover to JVM manually by using throw statement.

ex:

```
            throw  new ArithmeticException("Dont divide with zero");
```

ex:

```
class Test
{
            public static void main(String[] args)
            {
                        throw new ArithmeticException("dont divide by zer0000");

            }
}
```

throws statement

=================

If 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.

ex1:

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

		try
		{
			Thread.sleep(4000);
			System.out.println("Welcome to Java");
		}
		catch (InterruptedException ie)
		{
			ie.printStackTrace();
		}
	}
}
```

ex:2

```
class Test
{
	public static void main(String[] args)throws InterruptedException
	{
			Thread.sleep(4000);
			System.out.println("Welcome to Java");
	}
}
```

2)Userdefine Exception

========================

Exceptions which are created by the user based on the application requirement are called userdefine exceptions or custom exceptions.

ex:

StudentsNotPracticingException
ACWorkingException
ClassOnlineException
FundNotFoundException
TooYoungException
TooOldException
and etc.

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)
        {
                String sage=args[0];
                //converting string to int
                int age=Integer.parseInt(sage);

                if(age<18)
                        throw new TooYoungException("not eligible to vote");
                else
                        throw new TooOldException("eligible to vote");
        }
}
```

o/p:

```
javac   Test.java
java    Test   5
java    Test   24
```

JAVA IO PACKAGE

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);
```

If a file is not available then FileWriter object will create a new file.
ex:

```
		FileWriter fw=new FileWriter("aaa.txt");
		or
		File f=new File("aaa.txt");
		FileWriter fw=new FileWriter(f);
```

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 FileWriter object.

ex:1

```
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 ");
	}	}
```

ex:2

```
import java.io.*;
class Test
{
	public static void main(String[] args)
	{
		try
		{
			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 ");
		}
		catch (IOException ioe)
		{
			ioe.printStackTrace();
		}
	}
}
```

FileReader
=============

FileReader is used to read character oriented data from a file.

constructor
----

1)FileReader fr=new FileReader(String s);
2)FileReader fr=new FileReader(File f);

We can create object as follow.

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 the 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];

		//loading data from file to char array
		fr.read(carr);

		//for each loop
		for(char c:carr)
		{
			System.out.print(c);
		}
		fr.close();
	}
}
```

Usage of FileWriter and FileReader is not recommanded to use
==
While inserting the data by using FileWriter object we need to use insert line seperators(\n) which is very headache for the programmer.

While reading the data by using FileReader object we can 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.

constructors
----
1)BufferedWriter bw=new BufferedWriter(Writer w);

2)BufferedWriter bw=new BufferedWriter(Writer w,int buffersize);

BufferedWriter object does not communicate with file 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"));
```

If file is not available then it 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 BufferedWriter object.

6)newLine()

It will insert a 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.flush();
		bw.close();

		System.out.println("Please check the location");
	}
}
```

BufferedReader
===============

It is enhanced reader to read character oriented data from a file.

constructors

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 the 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 or FileReader is we can read the data 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 the file.

3)close()

It is used to close BufferedReader object.

4)readLine()

It will 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 writer 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 object will interact with files directly or it will take the support of some writer objects also.

ex:

```
	PrintWriter pw=new PrintWriter("ccc.txt");
	or
	File f=new File("ccc.txt");
	PrintWriter pw=new PrintWriter(f);
	or
	FileWriter fw=new FileWriter("ccc.txt");
	PrintWriter pw=new PrintWriter(fw);
```

The main advantage of PrintWriter over FileWriter and BufferedWriter is
we can insert any type of data.

Specially when we have primitive values then PrintWriter is best choice.

methods

```
write(int ch)
write(char[] ch)
write(String s)
flush()
close()

println(int i);
println(char ch);
println(String s);
println(boolean b);
println(double d);

print(int i);
print(char ch);
print(String s);
print(boolean b);
print(double d);
```

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.println('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
======================================

We have following ways to provde input values through keyboard.

1) Command line argument

2) BufferedReader class

3) Console class

4) Scanner class

1) Command line argument
----
ex:

```
class Test
{
	public static void main(String[] args)
	{
		String name=args[0];
		System.out.println("Welcome ="+name);
	}
}
```

```
javac  Test.java
java   Test   JamesGosling
```

2) BufferedReader class

BufferedReader class present in java.io package.

BufferedReader class will take the support of InputStreamReader object as a parameter which is embedded with System.in.

We can read inputs by using 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 is present in java.io package.

We can create Console class object by using console() method of System class.

ex:

```
    Console c=System.console();
```

We can read input values by using 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 in java.util package.

WE can create Scanner object as follow.

ex:

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

We can read input values by using nextXxx() methods.

Here nextXxx() method means nextInt(),nextFloat(),nextDouble() 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.It means we can provide guarantee that what type of elements are present in array.

If requirement is there to store String values we need to use String[] array.

ex:

```
	String[] sarr=new String[100];
	sarr[0]="hi";
	sarr[1]="bye";
	sarr[2]=10;   // C.T.E
```

At the time of retrieving 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 s= sarr[0];
```

Collection is not a typesafe.It means we can't give guarantee that what type of elements are present in Collection.

If requirement is there to store String values then it is never recommanded to use ArrayList.Here we won't get any compile time error or runtime error but some times our program will get failure.

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 s=(String)al.get(0);
```

To overcome these limitations Sun micro system introduced Generics concept in 1.5 version. The main objective of Generics are

1) To make Collection as typesafe.

2) To avoid typecasting problem.

Q)Differences between Array and Collection?

Array	**Collection**
----------	----------
It is a collection of homogeneous data elements.	It is a collection of homogeneous and hetrogeneous data elements.
Arrays are fixed in size.	Collections are growable in nature.
Perfomance point of view array is recommanded to use.	Performance point of view Collection is not recommanded to use.
Memory point of view array is not recommanded to use.	Memory point of view Collection is recommanded to use.
Arrays are not implemented based on data structure concept so we can't expect any readymade methods.For every logic we need to write the code explicitly.	Collections are implemented based on data structure concept so we can expect readymade methods.
Arrays can hold primitive types and object.	Collection can hold only objects.

COLLECTIONS

java.util package
==================

Collection
=============

If we want to represent group of objects in a single entity then we need to use Collection.

Collection Framework
===================

It defines several classes to represent group of objects in a single entity.

Collection
==============

Collection is an interface present in java.util package.

Collection 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 some common methods which are applicable for entire Collection objects.

Collection interface contains following methods.

ex:

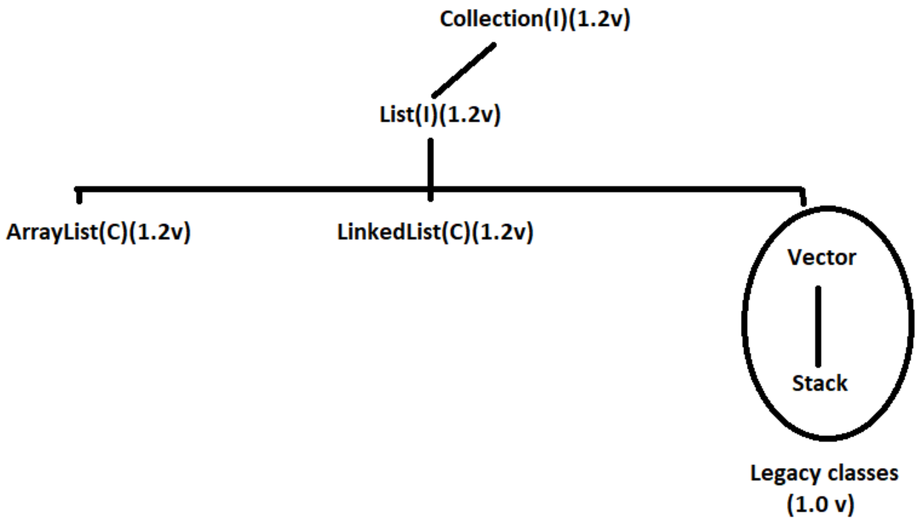
```
        public abstract int size();
        public abstract boolean isEmpty();
        public abstract boolean add(Object o);
        public abstract boolean addAll(Collection c);
        public abstract boolean remove(Object o);
        public abstract boolean removeAll(Collection c);
        public abstract boolean contains(Object o);
        public abstract boolean containsAll(Collection c);
        public abstract boolean retainsAll(Collection c);
        public abstract void clear();
        public abstract Iterator  iterator();
        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: java38.1



ArrayList
=========

The underlying data structure is resizable array or growable array.

Duplicates are allowed.

Insertion order is preserved.

Hetrogenous objects are allowed.

Null insertions is possible.

It implements Serializable,Cloneable and RandomAccess interface.

If our frequent operation is select/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(10);
		//System.out.println(al);//hetrogenous not allowed
		al.add(null);
		System.out.println(al);//[one,two,three,one,10,null]
	}
}
```

ex:3

```
--------
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(1,"raja");
                System.out.println(al);//[one, raja, two, three]
                al.remove("raja");
                System.out.println(al);//[one, two, three]
                System.out.println(al.contains("one"));//true
                al.clear();
                System.out.println(al);//[]
        }
}
```

ex:4

```
------
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++)
                {
                        System.out.println(al.get(i));
                }
        }
}
```

ex:5

```
------
import java.util.*;
class Test
{
        public static void main(String[] args)
        {
                ArrayList<String> al=new ArrayList<String>();
                al.add(new String("one"));
                al.add(new String("two"));
                al.add(new String("three"));
                System.out.println(al);
        }
}
```

ex:6

```
import java.util.*;
class Test
{
	public static void main(String[] args)
	{
		List<String> list=new ArrayList<String>();

		list.add("one");
		list.add("two");
		list.add("three");
		System.out.println(list);//[one, two, three]

	}
}
```

LinkedList

===============

The underlying data structure is doubly LinkedList.

Insertion order is preserved.

Duplicate objects are allowed.

Hetrogeneous objects are allowed.

Null insertion is possible.

It implements Deque,Serializable and Cloneable.

If our frequent operation is insert or delete in the middle then LinkedList is a best choice.

LinkedList class contains following methods.

ex:

```
	addFirst();
	addLast();
	removeFirst();
	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");
		System.out.println(ll);//[one, two, three, one]
		ll.add(null);
		System.out.println(ll);//[one, two, three, one, null]
	}
}
```

ex:2

```
import java.util.*;
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("jojo");
                ll.addLast("gogo");
                System.out.println(ll);//[jojo, one, two, three, gogo]

                System.out.println(ll.getFirst());//jojo
                System.out.println(ll.getLast());//gogo

                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]
        }
}
```

Vector

=========

The underlying data structure is resizable array or growable array.

Duplicates are allowed.

Insertion order is preserved.

Hetrogenous objects are allowed.

Null insertions is possible.

It implement Serializable , Cloneable and RandomAccess interface.

Note:

All the methods present in Vector are synchronized.
At a time only one thread is allowed to access.Hence we can achieve Thread safety.

Vector class contains following methods.

ex:

```
    addElement();
    removeElementAt();
    removeAllElements();
    capacity();
    firstElement();
    lastElement();
    elements()
    and etc.
```

ex:1

```
import java.util.*;
class Test
{
    public static void main(String[] args)
    {
        Vector<Integer> v=new Vector<Integer>();

        System.out.println(v.capacity());//

        for(int i=1;i<=10;i++)
        {
            v.addElement(i);
        }
        System.out.println(v);//[1,2,3,4,5,6,7,8,9,10]

        v.removeElementAt(9);
        System.out.println(v);//[1,2,3,4,5,6,7,8,9]

        System.out.println(v.firstElement());//1
        System.out.println(v.lastElement());//9

        v.removeAllElements();
        System.out.println(v);//[]
    }
}
```

ex:2

```
import java.util.*;
class Test
{
		public static void main(String[] args)
		{
				Vector<Integer> v=new Vector<Integer>();

				System.out.println(v.capacity());//

				for(int i=1;i<=10;i++)
				{
						v.add(i);
				}
				System.out.println(v);//[1,2,3,4,5,6,7,8,9,10]

				v.remove(9);
				System.out.println(v);//[1,2,3,4,5,6,7,8,9]

				v.clear();
				System.out.println(v);//[]
		}
}
```

Stack

========

It is a child class of Vector class.

If we depend upon Last In First Out(LIFO) order then we need to use Stack.

constructor

```
		Stack  s=new Stack();
```

Methods

----

1)push(Object o)

It is used to add the object into a stack.

2)pop()

IT is used to remove the object from the stack.

3)peek()

It is used to read toppest element in a stack.

4)isEmpty()

It is used to check stack is empty or not.

5)search(Object o)

It will return offset values if element is present 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]
		s.pop();
		System.out.println(s);//[A,B]

		System.out.println(s.isEmpty());//false

		System.out.println(s.peek());//B

		System.out.println(s.search("A"));//2

		System.out.println(s.search("Z"));//-1
	}
}
```

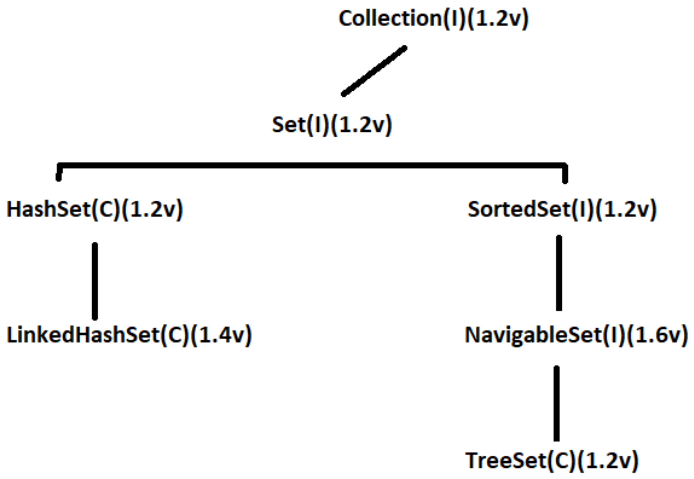
Set

=======

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: java39.1



HashSet

=========

The underlying data structure is hashtable.

Insertion order is not preserved because objects are arrange based on hashcode of an object.

Duplicates are not allowed.

Hetrogenous objects are allowed.

Null insertion is possible.

It implements Serializable and Cloneable interface.

ex:

```
import java.util.*;
class Test
{
        public static void main(String[] args)
        {
                HashSet<String> hs=new HashSet<String>();
                hs.add("one");
                hs.add("ten");
                hs.add("four");
                hs.add("six");
                System.out.println(hs);//[six,four,one,ten]
                hs.add("one");
                System.out.println(hs);//[six,four,one,ten]
                hs.add(null);
                System.out.println(hs);//[null,six,four,one,ten]
        }
}
```

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 Hastable.	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 1.4v.

ex:

```
import java.util.*;
class Test
{
        public static void main(String[] args)
        {
                LinkedHashSet<String> lhs=new LinkedHashSet<String>();
                lhs.add("one");
                lhs.add("ten");
                lhs.add("four");
                lhs.add("six");
                System.out.println(lhs);//[one,ten,four,six]
                lhs.add("one");
                System.out.println(lhs);//[one,ten,four,six]
                lhs.add(null);
                System.out.println(lhs);//[one,ten,four,six,null]
        }
}
```

TreeSet
==========
The underlying data structure is Balanced Tree.

Duplicate objects are not allowed.

Insertion order is not preserved because it is based on sorting order of an object.

Hetrogeneous objects are not allowed.If we try to insert hetrogenous objects then we will get ClassCastException.

Null insertion is possible only once.

For empty TreeSet if we insert null then we will get NullPointerException.

After insertion the elments ,if we are trying to insert null then we will
get NullPointerException.

It implements NavigableSet , Cloneable and Serializable interface.

ex:1

```
import java.util.*;
class Test
{
	public static void main(String[] args)
	{
		TreeSet<Integer> ts=new TreeSet<Integer>();
		ts.add(5);
		ts.add(10);
		ts.add(1);
		ts.add(3);
		ts.add(7);
		System.out.println(ts);
	}
}
```

ex:2

```
import java.util.*;
class Test
{
	public static void main(String[] args)
	{
		TreeSet<Integer> ts=new TreeSet<Integer>();
		ts.add(5);
		ts.add(10);
		ts.add(1);
		ts.add(3);
		ts.add(7);
		System.out.println(ts);//[1,3,5,7,10]
		ts.add(5);
		System.out.println(ts);//[1,3,5,7,10]
		ts.add(null);
		System.out.println(ts);//R.E NullPointerException
	}
}
```

ex:3

```
import java.util.*;
class Test
{
	public static void main(String[] args)
	{
		TreeSet ts=new TreeSet();
		ts.add(5);
		ts.add(10);
		ts.add(1);
		ts.add(3);
		ts.add(7);
		System.out.println(ts);//[1,3,5,7,10]
		ts.add("hi");
		System.out.println(ts);//R.E ClassCastException

	}
}
```

Q)What is the difference between Comparable and Comparator interface?

Comparable
===========

Comparable interface present in java.lang package.

Comparable interface contains following one method i.e compareTo() method.

ex:

```
	public  int  compareTo(Object o)
```

ex:

```
	obj1.compareTo(obj2)
```

It will return -ve if obj1 comes before obj2.
It will return +ve if obj1 comes after obj2.
IT will return 0 if both objects are equals.

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
	}
}
```

Note:

If we depend upon default natural sorting(ascending order) order then we need to use Comparable interface.

Comparator

================

Comparator interface present in java.util package.

Comparator interface contains following two methods.

1) public int compare(Object obj1,Object obj2)

    It will return +ve if obj1 comes before obj2.
    It will return -ve if obj1 comes after obj2.
    It will return 0 if both objects are equals.

2) public boolean equals(Object obj)

Whenever we are using Comparator interface we should write
implementation only for compare() method.

Implementation for equals() method is optional because equals() method is available by default by Object class throw inheritence.

Note:

If we depend upon customized sorting order then we need to use Comparator interface.

ex:1

```
import java.util.*;
class Test
{
        public static void main(String[] args)
        {
                TreeSet<Integer> ts=new TreeSet<Integer>(new MyComparator());
                ts.add(2);
                ts.add(10);
                ts.add(1);
                ts.add(5);
                System.out.println(ts);//[1,2,5,10]
        }
}
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:2

```
import java.util.*;
class Test
{
	public static void main(String[] args)
	{
		TreeSet<Integer> ts=new TreeSet<Integer>(new MyComparator());
		ts.add(2);
		ts.add(10);
		ts.add(1);
		ts.add(5);
		System.out.println(ts);//[1,2,5,10]
	}
}
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.

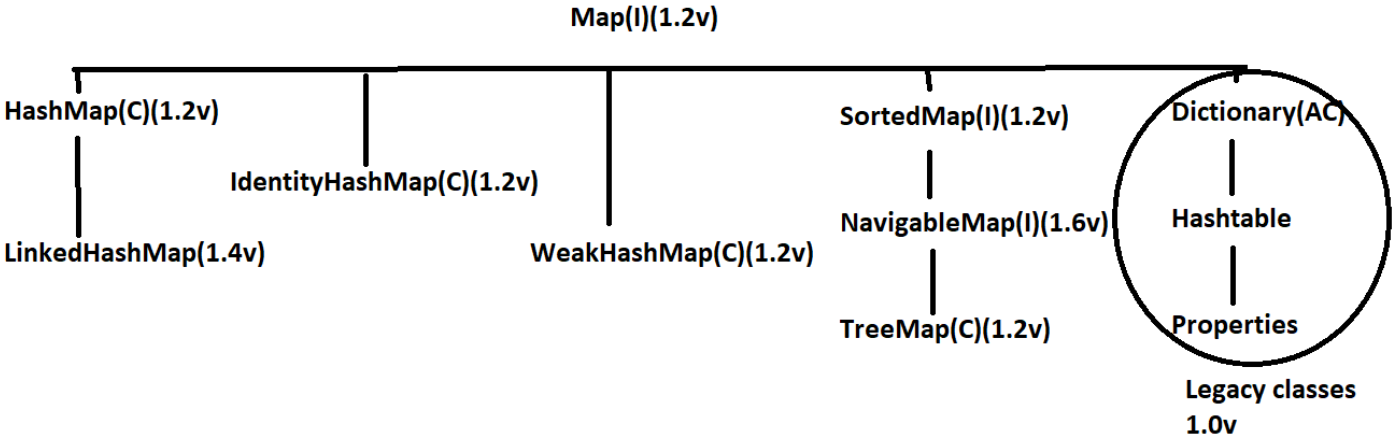
If we want to represent group of individual objects in a key-value pair then we need to use Map interface.

Both key and value are objects only.

Duplicate keys are not allowed but values can be duplicated.

Each key-value pair is called "one entry".

Diagram: java40.1



HashMap
===========
The underlying data structure is Hashtable.

Duplicate keys are not allowed but values can be duplicated.

Insertion order is not preserved and it is based on hash code of the keys.

Hetrogeneous objects are allowed for both keys and values.

Null insertion is allowed for key(only once) and for values (any number).

ex:1

```
import java.util.*;
public class Test {
	public static void main(String[] args)
	{
		HashMap hm=new HashMap();
		hm.put("one","raja");
		hm.put("two","ravi");
		hm.put("six","nany");

		System.out.println(hm);//{six=nany, one=raja, two=ravi}
		hm.put(10,100);
		System.out.println(hm);//{six=nany, one=raja, 10=100, two=ravi}

		hm.put(null,"gogo");
		System.out.println(hm);//{null=gogo, six=nany, one=raja, 10=100, two=ravi}

		hm.put("four","ramana");
		System.out.println(hm);//{null=gogo, six=nany, four=ramana, one=raja, 10=100, two=ravi}

		hm.put("one","rani");
		System.out.println(hm);//{null=gogo, six=nany, four=ramana, one=rani, 10=100, two=ravi}
	}	}
```
ex:2

```
import java.util.*;
public class Test {

	public static void main(String[] args)
	{
		HashMap<String,String> hm=new HashMap<String,String>();
		hm.put("one","raja");
		hm.put("two","ravi");
		hm.put("six","nany");

		System.out.println(hm);//{six=nany, one=raja, two=ravi}

		hm.put(null,"gogo");
		System.out.println(hm);//{null=gogo, six=nany, one=raja, 10=100, two=ravi}

		hm.put("four","ramana");
		System.out.println(hm);//{null=gogo, six=nany, four=ramana, one=raja, 10=100, two=ravi}

		hm.put("one","rani");
		System.out.println(hm);//{null=gogo, six=nany, four=ramana, one=rani, 10=100, two=ravi}
	}
}
```

ex:3

```
import java.util.*;
public class Test {

	public static void main(String[] args)
	{
		HashMap<String,String> hm=new HashMap<String,String>();
		hm.put("one","raja");
		hm.put("two","ravi");
		hm.put("six","nany");

		Set s=hm.keySet();
		System.out.println(s);//[six, one, two]

		Collection c=hm.values();
		System.out.println(c);//[nany, raja, ravi]

		Set s1=hm.entrySet();
		System.out.println(s1);//[six=nany, one=raja, two=ravi]
	}

}
```

LinkedHashMap
================

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 Hastable.	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<Integer,String> lhm=new LinkedHashMap<Integer,String>();
		lhm.put(1,"raja");
		lhm.put(4,"kelvin");
		lhm.put(7,"nelson");
		lhm.put(2,"jose");
		System.out.println(lhm);//{1=raja, 4=kelvin, 7=nelson, 2=jose}
	}
}
```

TreeMap
=========
The underlying data structure is RED BLACK TREE.

Duplicate keys are not allowed but values can be duplicated.

Insertion order is not preserved. All entries will store in the sorting order of keys.

If we depends upon natural sorting order then keys can be homogeneous and Comparable.

If we depends upon customized order then keys can be hetrogeneous and non-comparable.

For emptry TreeMap if we insert NULL as key then we will get NullPointerException.

After insertion elements if we are trying to insert NULL as key the we will get NullPointerException.

But there is no restrictions on NULL values.

ex:1

```
import java.util.*;
class Test
{
	public static void main(String[] args)
	{
		TreeMap<Integer,Integer> tm=new TreeMap<Integer,Integer>();
		tm.put(1,100);
		tm.put(4,400);
		tm.put(9,900);
		tm.put(7,700);
		tm.put(5,500);
		System.out.println(tm);//{1=100, 4=400, 5=500, 7=700, 9=900}
	}
}
```
ex:2

```
import java.util.*;
class Test
{
	public static void main(String[] args)
	{
		TreeMap<Integer,Integer> tm=new TreeMap<Integer,Integer>();
		tm.put(1,100);
		tm.put(4,400);
		tm.put(9,900);
		tm.put(7,700);
		System.out.println(tm);//{1=100, 4=400, 7=700, 9=900}
		tm.put(5,null);
		System.out.println(tm);//{1=100, 4=400, 5=null, 7=700, 9=900}
		tm.put(null,600);
		System.out.println(tm);//R.E NullPointerException
	}
}
```

Hashtable

==========

The underlying data structure is Hastable.

Insertion order is not preserved.

Hetrogenous keys and values are allowed.

Duplicate keys are not allowed but values can be duplicated.

Key and value can't be null otherwiser we will get NullPointerException.

ex:1

```
import java.util.*;
class Test
{
    public static void main(String[] args)
    {
        Hashtable<Integer,String> ht=new Hashtable<Integer,String>();
        ht.put(1,"100");
        ht.put(9,"900");
        ht.put(5,"500");
        ht.put(7,"700");
        System.out.println(ht);//{9=900, 7=700, 5=500, 1=100}
    }
}
```

ex:2

```
import java.util.*;
class Test
{
    public static void main(String[] args)
    {
        Hashtable<Integer,String> ht=new Hashtable<Integer,String>();
        ht.put(1,"100");
        ht.put(9,"900");
        ht.put(5,"500");
        ht.put(7,"700");
        System.out.println(ht);//{9=900, 7=700, 5=500, 1=100}

        //ht.put(null,"400");
        //System.out.println(ht);//R.E NullPointerException

        ht.put(4,null);
        System.out.println(ht);//R.E NullPointerException
    }
}
```

Types of Cursors in java
===========================
Cursors are used to retrieve the objects one by one from Collection.

We have three types of cursors in java.

1)Enumeration

2)Iterator

3)ListIterator

1)Enumeration
==============
Enumeration interface present in java.util package.

It is used to read objects one by one from Legacy Collection objects.

Enumeration object can be created by using elements() method.

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

Using Enumeration we can read objects one by one from legacy Collection objects.Hence it is not a universal cursor.

Using Enumeration interface we can perform read operation but not delete/remove operation.

To overcome this limitation sun micro system introduced Iterator interface.

2)Iterator
==========

It is used to retrieve the objects one by one from any Collection object.Hence it is known as universal cursor.

Using Iterator interface we can perform read and remove operation.

We can create Iterator object by using iterator() method.

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);
                }
        }
}
```

limitations with Iterator

Using Enumeration and Iterator we can read objects only in forward direction but not in backward direction.Hence they are not bi-directional cursors.

Using Iterator 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.

It is used to read objects one by one from List Collection objects.

Using ListIterator we can read objects in forward direction and backward direction.Hence it is a bi-directional cursor.

Using ListIterator we can perform read, remove ,adding and replacement of new objects.

We can create ListIterator interface by using listIterator() method.

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 void remove();
        public void add(Object o);
        public void set(Object o);
        public void nextIndex();
        public void previousIndex();
```

ex:1

```
import java.util.*;
class Test
{
        public static void main(String[] args)
        {
                ArrayList al=new ArrayList();
                al.add("venki");
                al.add("nag");
                al.add("chiru");
                al.add("bala");
                System.out.println(al);//[venki,nag,chiru,bala]

                ListIterator litr=al.listIterator();
                while(litr.hasNext())
                {
                        String s=(String)litr.next();
                        System.out.println(s);
                }
        }
}
```

ex:2

```
import java.util.*;
class Test
{
	public static void main(String[] args)
	{
		ArrayList al=new ArrayList();
		al.add("venki");
		al.add("nag");
		al.add("chiru");
		al.add("bala");
		System.out.println(al);//[venki,nag,chiru,bala]

		ListIterator litr=al.listIterator();
		while(litr.hasNext())
		{
			String s=(String)litr.next();
			if(s.equals("nag"))
			{
				litr.remove();
			}
		}
		System.out.println(al);//[venki, chiru, bala]
	}
}
```

ex:3

```
import java.util.*;
class Test
{
	public static void main(String[] args)
	{
		ArrayList al=new ArrayList();
		al.add("venki");
		al.add("nag");
		al.add("chiru");
		al.add("bala");
		System.out.println(al);//[venki,nag,chiru,bala]

		ListIterator litr=al.listIterator();
		while(litr.hasNext())
		{
			String s=(String)litr.next();
			if(s.equals("nag"))
			{
				litr.add("Chetu");
			}
		}
		System.out.println(al);//[venki, nag, Chetu, chiru, bala]
	}
}
```

ex:4

```
import java.util.*;
class Test
{
	public static void main(String[] args)
	{
		ArrayList al=new ArrayList();
		al.add("venki");
		al.add("nag");
		al.add("chiru");
		al.add("bala");
		System.out.println(al);//[venki,nag,chiru,bala]

		ListIterator litr=al.listIterator();
		while(litr.hasNext())
		{
			String s=(String)litr.next();
			if(s.equals("nag"))
			{
				litr.set("Chetu");
			}
		}
		System.out.println(al);//[venki, Chetu, chiru, bala]
	}
}
```

Interview Questions
====================

1)Arrays vs Collection?

2)ArrayList vs Vector?

3)ArrayList vs LinkedList?

4)List vs Set?

5)HashSet vs LinkedHashSet?

6)HashSet vs TreeSet?

7)HashMap vs LinkedHashMap?

8)HashMap vs TreeMap?

9)TreeMap vs Hashtable?

10)Enumeration vs Iterator vs ListIterator?

11)Comparable vs Comparator?

12)Type of Datastructure in java?

MULTI-THREADING

Multi-Threading
===================
Q)What is the difference between Thread and Process?

Thread

It is a light weight sub-process.

We can run multiple thread concurently.

One thread can communicate with another thread.

In java, all threads will store in Java Stack memory.

ex:

class is one thread
method is one thread
request is one thread

Process

Process is a collection of threads.

We can run multiple process concurently.

One process can't communicate with another process.

ex:

typing a java program in a editor is a process
listening mp3 song is a process
downloading a file from internet is a process.

Multi-tasking
==============
Executing several task simultenously such concept is called multi-tasking.

We have two types of multi-tasking.

1)Thread based multi-tasking

2)Process based multi-tasking

1)Thread based multi-tasking

Executing several task simultenously where each task is a same part of a program such type of multi-tasking is called thread based multi-tasking.

2)Process based multi-tasking

Executing several task simultenously where each task is a independent process such type of multi-tasking is called process based multi-tasking.

Multi-Threading

==================

Executing several threads simultenously such concept is called multi-threading.

In multithreading only 10% of work should be done by a programmer and 90% of work will be done by JAVA API.

The main important application area of multi-threading are

1)To implement Multi media graphics.

2)To develop animation

3)To develop video games.

How many ways we can start/create/instantiate a thread

===

Using two ways we can 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)
                    {
                    //instantiate a thread
                    MyThread t=new MyThread();

                    //start a thread
                    t.start();

                    for(int i=1;i<=5;i++)
                    {
                           System.out.println("Parent-Thread");
                    }
                    }
}
```

case 1: Thread Schedular

==========================

If multiple threads are waiting for execution which thread will be executed will decided by Thread schedular.

What algorithm ,mechanism and behaviours used by Thread schedular is depends upon JVM vendor.

Hence we can't expect any execution order and exact output in Multi-threading.

case 2: 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)
                        {
                        //instantiate a thread
                        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 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)
                        {
                        //instantiate a thread
                        MyThread t=new MyThread();
                        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 Thread class run() method will execute just like normal method.

Thread class run() method is a empty implementation so we can't expect any output.

ex:

```
class MyThread extends Thread
{

}
class Test
{
				public static void main(String[] args)
				{
				//instantiate a thread
				MyThread t=new MyThread();

				//start a 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() always execute run() method with no arguments only.

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)
				{
				//instantiate a thread
				MyThread t=new MyThread();

				//start a 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.No new Thread will be created but
start() method will execute just like normal method.

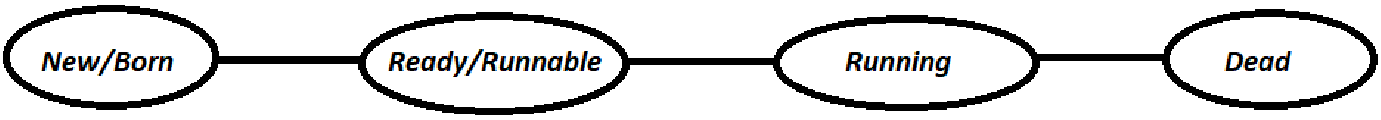
ex:

```
class MyThread extends Thread
{
                        public void start()
                        {
                        System.out.println("start method");
                        }

}
class Test
{
                        public static void main(String[] args)
                        {
                        //instantiate a thread
                        MyThread t=new MyThread();

                        t.start();

                        for(int i=1;i<=5;i++)
                        {
                                System.out.println("Parent-Thread");
                        }
                        }
}
```

case 6: Life cycle of a thread

==================================

Diagram: java24.2



Once if we create a Thread object then our thread will be in New/Born state.

If we call t.start() method then our thread goes to Ready/Runnable state.

IF Thread Schedular allocates CPU then our thread enters to Running state.

Once the run() method execution is completed our thread 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();
                     Thread t=new Thread(r);//r is a targatable interface
                     t.start();
                     for(int i=1;i<=5;i++)
                     {
                            System.out.println("Parent-Thread");
                     }
                    }
}
```

case study

===========

```
MyRunnable r=new MyRunnable();
Thread t1=new Thread(r);
Thread t2=new Thread();
```

t1.start()

If we call t1.start() method then a new thread will be created which is responsible to execute MyRunnable class run() method automatically.

ex:

```
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();
                     Thread t1=new Thread(r);//r is a targatable interface
                     t1.start();
                     for(int i=1;i<=5;i++)
                     {
                            System.out.println("Parent-Thread");
                     }
                    }
}
```

t1.run()

If we call t1.run() method then no new thread will be created but MyRunnable class run() method will execute just like a normal method.

ex:

```
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();
                             Thread t1=new Thread(r);//r is a targatable interface
                             t1.run();

                             for(int i=1;i<=5;i++)
                             {
                                       System.out.println("Parent-Thread");
                             }
                            }
}
```

t2.start()

If we call t2.start() method then a new thread will be created which is responsible to execute Thread class run() method automatically.

ex:

```
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();
                             Thread t2=new Thread();
                             t2.start();

                             for(int i=1;i<=5;i++)
                             {
                                       System.out.println("Parent-Thread");
                             }
                            }
}
```

t2.run()

If we call t2.run() method then no new thread will be created but Thread class run() method will execute just like normal method.

ex:

```
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();
                              Thread t2=new Thread();
                              t2.run();

                              for(int i=1;i<=5;i++)
                              {
                                        System.out.println("Parent-Thread");
                              }
                              }
}
```

r.start()

If we call r.start() method then we will get C.T.E cannot find symbol

ex:

```
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();
                              r.start();

                              for(int i=1;i<=5;i++)
                              {
                                        System.out.println("Parent-Thread");
                              }
                              }
}
```

r.run()

If we call r.run() method then no new thread will be created but MyRunnable class run() method will execute just like normal method.

ex:

```
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();
                        r.run();

                        for(int i=1;i<=5;i++)
                        {
                                System.out.println("Parent-Thread");
                        }
                        }
}
```

How to prevent a Thread from execution
======================================
There are three ways to prevent(stop) a thread from execution.

1)yield()

2)join()

3)sleep()

1)yield()

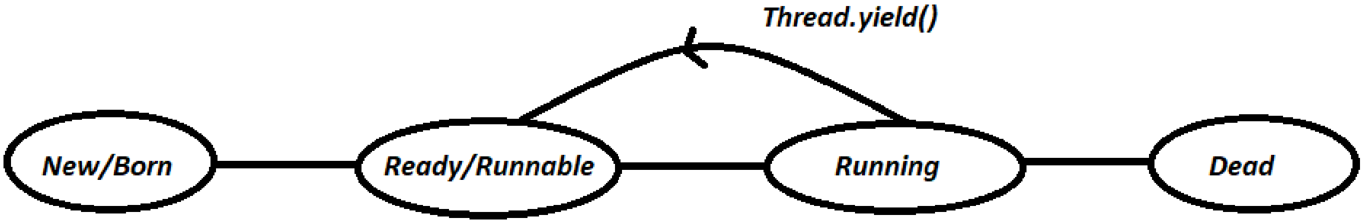
It will pause the current execution thread and gives the change to other threads which are having same priorities.

If multiple threads having same priority then we can't exect any execution order.

A thread which is yielded when it will get a chance for execution is depends upon mercy of thread schedular.
ex:

```
                        public static native void yield()
```

Diagram: java41.1



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()
===========

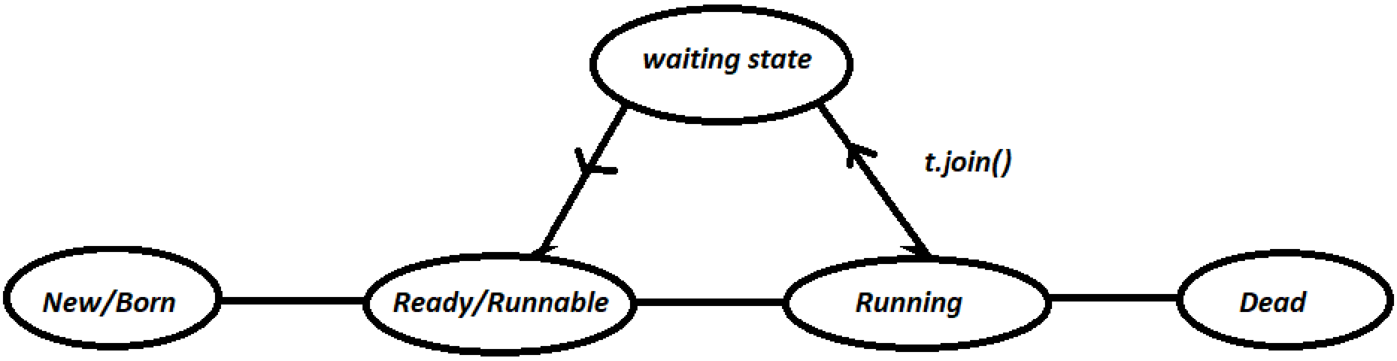
If a thread wants to wait untill the completion of some other thread then we need to use join() method.

A join() method throws one checked exception called InterruptedException so we must and should handle that exception by using try and catch block or 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: java41.2



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)throws InterruptedException
                              {
                               MyThread t=new  MyThread();
                               t.start();
                               t.join();
                              for(int i=1;i<=5;i++)
                              {
                                        System.out.println("Parent-Thread");
                              }
                              }
}
```

3)sleep()

===========

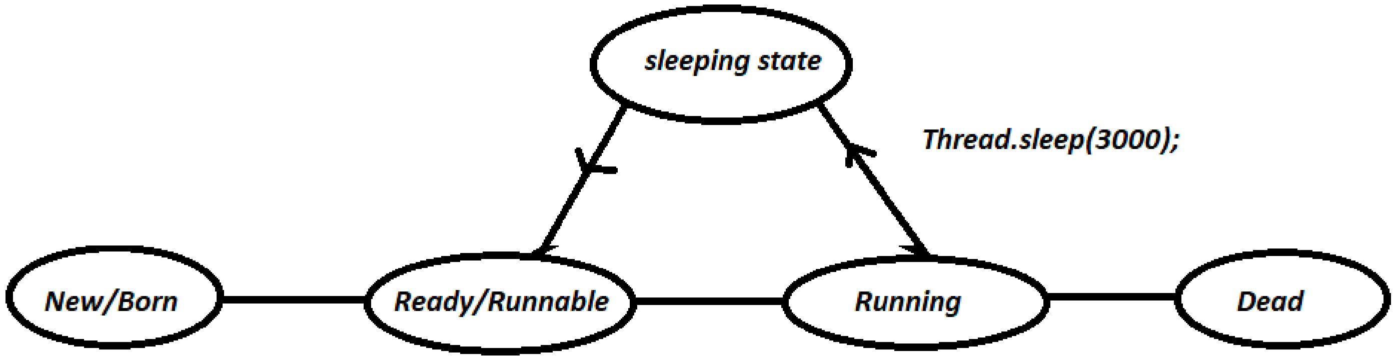
If a thread don't want to perform any operation on a perticular amount of time then we need to use sleep() method.

A sleep() method throws one checked exception called InterruptedException so we must and should handle that execution 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: java41.3



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");
                    }
                    }
}
```

Setting and Getting name of a thread

=====================================

In java ever thread has a name, explicitly provided by the programmer or automatically generated by JVM.

We having following methods to set or 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");
                     t.setName("Child-Thread");
                     System.out.println(Thread.currentThread().getName());//Parent-Thread
                     System.out.println(t.getName());//Child-Thread
                    }
}
```

Thread Priority

=====================

In java, every thread has a name.Explicitly provided by the programmer or automatically generated by JVM.

The valid range of thread priority is 1 to 10.Where as 1 is a least priority and 10 as highest priority.

Thread class defines following standard constants as thread priorities.

ex:

```
                    MAX_PRIORITY  --> 10
                    NORM_PRIORITY           --> 5
                    MIN_PRIORITY   --> 1
```

We have don't such constants like LOW_PRIORITY and HIGH_PRIORITY.

Thread schedular uses thread priority while allocating to CPU.

A thread which is having highest priority will executed first.

If multiple threads having same priority then we can't expect execution order.

If we take more then 10 priority then we will get RuntimeException called IllegalArgumentException.

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);
                      t.setPriority(4);

                     System.out.println(Thread.currentThread().getPriority());//9
                      System.out.println(t.getPriority());//4

                     t.setPriority(11);//R.E IllegalArgumentException
                    }
}
```

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 recommanded 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 concurently.But other threads are allowed to execute non-synchronized method concurently.

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();
                                }
}
```

Inter-Thread Communication

==========================

Two threads can communicate with one another by using wait(),notify() and notifyAll() method.

The Thread which is expecting updations it has to wait() method and the thread which is performing updations it has to call notify() method.

wait(),notify() and notifyAll() method present in Object class but not in Thread class.

To call wait(),notify() and notifyAll() method our current thread must be in a synchronized area otherwise we will get IllegalMonitorStateException.

Once a thread calls wait() method on a given object ,1st it will release the lock of that object immediately and entered into waiting state.

Once a thread calls notify() and notifyAll() method on a given object.It will release the lock of that object but not immediately.

Except wait(),notify() and notifyAll() method ,there is no such concept where lock release can happen.

ex:

```
class MyThread extends Thread
{
                int total=0;
                public void run()
                {

                 synchronized(this)
                 {
                           System.out.println("Child Thread started calculation");
                           for(int i=1;i<=10;i++)
                           {
                                total=total+i;
                           }
                           System.out.println("Child thread giving notification");
                           this.notify();
                 }
                }
}
class Test
{
                public static void main(String[] args)throws InterruptedException
                {
                 MyThread t=new MyThread();
                 t.start();
                 synchronized(t)
                 {
                      System.out.println("Main Thread waiting for updating");
                      t.wait();
                      System.out.println("Main -Thread got notification ");
                      System.out.println(t.total);
                 }
                }
}
```

DeadLock in java
===============

DeadLock will occur in a suitation when one thread is waiting to access object lock which is acquired by another thread and that thread is waiting to access object lock which is acquired by first thread.

it is programming situation where two or more threads blocked forever

Here both the threads are waiting release the thread but no body will release such situation is called DeadLock.

ex:

```
class Test
{
		public static void main(String[] args)
		{
		final String res1="hi";
		final String res2="bye";
		Thread t1=new Thread()
		{
			public void run()
			{
				synchronized(res1)
				{
					System.out.println("Thread1: Locking Resource 1");
					synchronized(res2)
					{
						System.out.println("Thread1: Locking Resource2");
					}
				}
			}
		};
		Thread t2=new Thread()
		{
			public void run()
			{
				synchronized(res2)
				{
					System.out.println("Thread2: Locking Resource 2");
					synchronized(res1)
					{
						System.out.println("Thread1: Locking Resource 1");
					}
				}
			}
		};
		t1.start();
		t2.start();
		}
}
```

Drawbacks of multithreading
=====================

1)DeadLock

2)Thread Starvation

JAVA 1.8 FEATURES

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();
			}
}
```

Marker Interface

================

An interface does not have any methods but using those interface we will get some ability such type of interface is called marker interface.

ex:

Serializable
Cloneable
Remote and etc.

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));
                }
}
```

Java default methods in interface

==========================

Java provide sa facility to create a default methods inside the interface.

Methods which are defined inside the interface and tagged with default keyword are known as default methods.

Default methods are non-abstract methods.

We can override default methods.

syntax

```
                Modifier interface  interface_name
                {
                //abstract methods
                //default methods
                }
```

ex:1

```
interface A
{
                public abstract void m1();

                default void m2()
                {
                 System.out.println("from default method");
                }
}
class B implements A
{
                public void m1()
                {
                 System.out.println("from abstract method");
                }
}
class Test
{
                public static void main(String[] args)
                {
                 B b=new B();
                 b.m1();
                 b.m2();
                }
}
```

ex:2

```
interface A
{
                public abstract void m1();

                default void m2()
                {
                 System.out.println("from default method");
                }
}
class B implements A
{
                public void m1()
                {
                 System.out.println("from abstract method");
                }
                public void m2()
                {
                 System.out.println("Override default method");
                }
}
class Test
{
                public static void main(String[] args)
                {
                 B b=new B();
                 b.m1();
                 b.m2();
                }
}
```

As we know , java does not support mutiple inheritence .
But we can solve this problem by using default methods of an interface.

ex:1

```
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:2

```
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: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()
                    {
                    Left.super.m1();
                    }
}
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();
                         Left.super.m1();
                        }
}
class Test
{
                        public static void main(String[] args)
                        {
                         Middle m=new Middle();
                         m.m1();
                        }
}
```

static methods in interface
=======================

Static methods in interface are those methods which are defined in the interface with static keyword.

Unlike other methods in interface, these static methods contains the complete definition of the function and since the definition is complete and method is static ,there fore these methods cannot be overridden or change in class.

syntax

```
                        modifier interface interface_name
                        {
                         //default methods
                         //abstract methods
                         //static methods
                        }
```

ex:1

```
interface A
{
                        public static void m1()
                        {
                         System.out.println("From static method");
                        }
}
class Test
{
                        public static void main(String[] args)
                        {
                         A.m1();
                        }
};
```

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]

                               //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);
                l.add(65);
                l.add(15);
                System.out.println(l);//[45,89,76,25,65,15]

                //How many students are failed
                long failed=l.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> 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]

                              //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);
                    }
}
```

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);
                     }

                    }
}
```