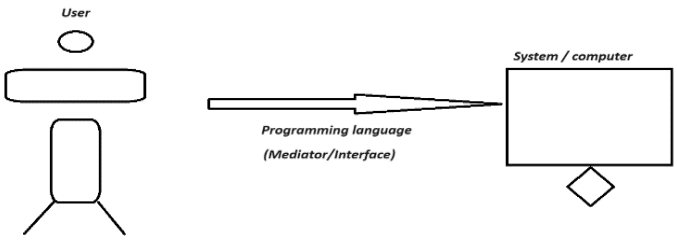
CORE-JAVA

Programming language

Diagram: introduction1.1



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

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

Java

>Object oriented programming language.
>Platform independent programming language.
>Case sensitive programming language.
>Strongly typed checking language
>High level programming language.
>Open-source programming language.

1995 --> James Gosling --> Sun Micro System (Oracle Corporation)
JDK software

C

>Procedure oriented programming language.
>Platform dependent programming language.
>Case sensitive programming language.
>Loosely typed checking language.
>Middle level language (LOW + HIGH)

Interview Questions

Q) What is Java?

It is a object oriented, platform independent, case sensitive, strongly typed checking, high level, open source programming language developed by James Gosling in the year of 1995.

Q) What is the difference between Python and Java?

Python	Java
>It is developed by Guido Gosling.	>It is developed by James van Rossum
>It is a product of Microsoft. Corporation.	>It is a product of Oracle
>It is a scripting language.	>It is a object oriented programming Language.
>It is a interpreted language.	>It is a compiled language.
>It contains PVM.	>It contains JVM.
>It is a dynamically typed language	>It is a statically typed language.
>Performance is low.	>Performance is high.
>There is less security.	>It is highly secured.

Note:

.java file ----------> .class file ----------> JVM
(Byte Code)

.py file ----------> PVM

Note:

.java file ----------> .class file ----------> JRE

.py file ----------→computer/machine

ex1:

```
python
------
class Test:
```

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

ex:2

```
      python
      ------
      i = 10;

      java
      ----
      int i = 10;
```

ex:3

```
      python
      ------
      for i in range(1,11):
            print(i);

      java
      ----
      for(int i=1;i<=10;i++)
            System.out.println(i);
```

Project
========
A project is a collection of modules.

We have following list of modules.

ex:
- report generation module
- admin module
- login module
- registration module
- payment module
- and etc.

Every project contains two domains.

1) Technical Domain

Technical domain describes using which technology we developed our project.

ex:
Java

2) Functional Domain

Functional domain describes state of a project.
ex:
Healthcare domain
Banking domain
Insurance domain
ERP domain
and etc.

***Escape Characters or Escape Sequences**

==========

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

Escape character starts with back slash (\) followed by a character.
ex:
\n

Mostly escape characters are placed in the output statement in java.
ex:
System.out.println("\n");

We have following list of escape characters in java.

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

o/p:
IHUB
TALENT

2) \t (horizontal tab)

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

o/p:

```
      IHUB  TALENT
```

3) \b (back space)

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

o/p:

```
      IHUBTALEN
```

ex:

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

o/p:

```
      ITALENT
```

4) \r (carriage return)

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

o/p:

```
      TALENT
```

ex:

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

```
            System.out.println("TALENT\rIHUB");
      }
}
o/p:
      IHUBNT
```

```
6) \\ (back slash)
------------------
class Harsha
{
      public static void main(String[] args){
            System.out.println("IHUB\\TALENT");
      }
}


o/p:
      IHUB\TALENT
```

```
C program
=========
Q) Write a c program to print %d ?

void main()
{
      clrscr();

      printf("%d"); // 0

      getch();
}

ex:
---
void main()
{
      clrscr();

      printf("%%d"); // %d

      getch();
}
```

```
7) \" (double quote)
--------------------
class Balaji
{
      public static void main(String[] args)
      {
            System.out.println("I Love \"Java\" programming");
      }
```

```
}

o/p:
      I Love "Java" programming


8) \' (single quote)
---------------------
class Priya
{
      public static void main(String[] args)
      {
            System.out.println("I Love 'java' programming");
            System.out.println("I Love \'java\' programming");
      }
}

o/p:
      I Love 'java' programming
      I Love 'java' programming
```

***Programming language**

==========

Diagram: java1.1

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

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

We have two types of programming language.

1) Low Level Language

2) High Level Language

1) Low Level Language

A language which 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.

A computer may understands many languages. But to understand machine language it does not required any translator.

Advantages:

1) A program writtens in machine language consumes less memory.

2) It does not required any translator.

3) It is more efficient when compare to other languages.

Disadvantages:

1) It is a burdun on a programmer to remember dozen's of binary code.

2) If anywhere error raised in our program then locating and handling that error becomes difficult.

3) Modification 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.

Assembly language is also known as symbolic logic.

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 converts assemblic code to machine code.

Merits:

1) If anywhere error raised in our program then locating and handling that error becomes easy.

2) Modifications can be done easily.

Demerits:
1) It is a mind trick to remember all symbolic code.
2) It requires translator.

3) It is less efficient when compare to machine language.

Q)What is Debugging?

Bugs are also known as Errors.

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

2) High Level Language

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

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

ex:
Java, .Net, Python, Perl and etc.

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

compiler

It will compile and execute our program at a time.

interpreter

It will execute our program line by line procedure.

Advantages:

1) It is easy to learn and easy to use because it is similar to english language.

2) Debugging can be done easily.

3) Modifications can done easily.

Disadvantages:

1) A program writtens in high level language consumes huge amount of memory.

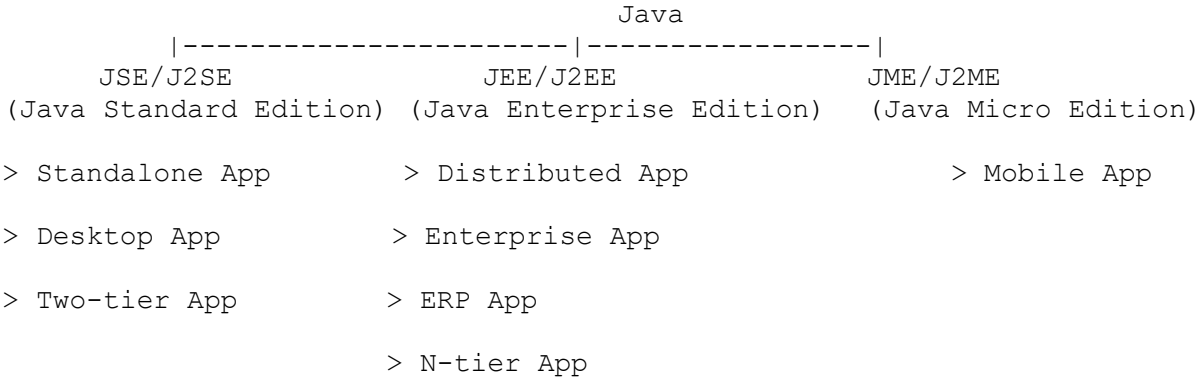
2) It requires translator

3) It is not efficient when compare to low level language.

***Modules in Java**

==========

We have three modules in java.



> Standalone App

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

ex:

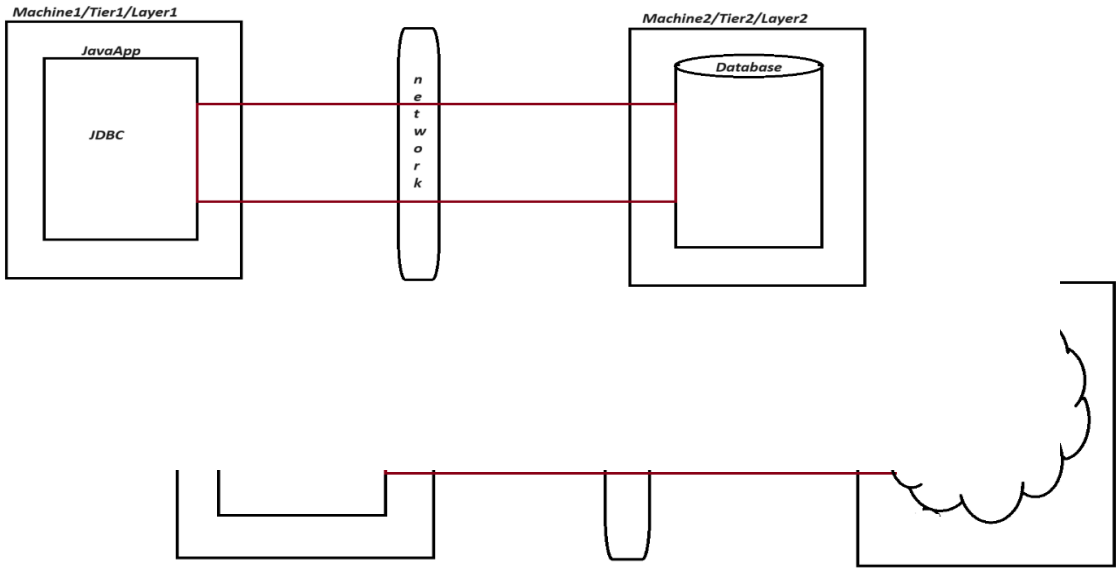
```
class  Test
{
      public  static  void  main(String[] args)
      {
            -
            - // any logic here
            -
      }
}
```

> Desktop App

It is a software application which is used to perform perticular task.
ex:
Control Panel
Recycle bin
VLC Media Player
and etc

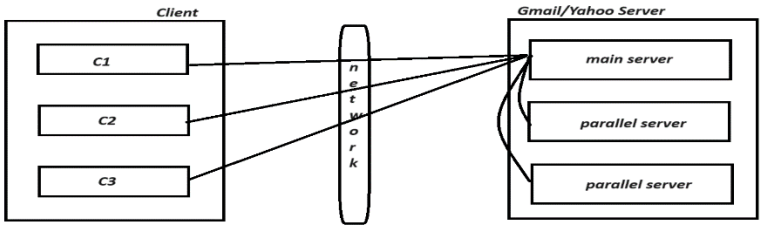
> Two-tier App

Having more then one tier is called two tier application.
Diagram: java2.1



> Distributed App

Diagram: java2.3



In client-server architecture ,if multiple request goes to main serve then main server will distribute those request to parallel server to reduce the burdun on main server such type of applications is called distributed applications.

> Enterprise App

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

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

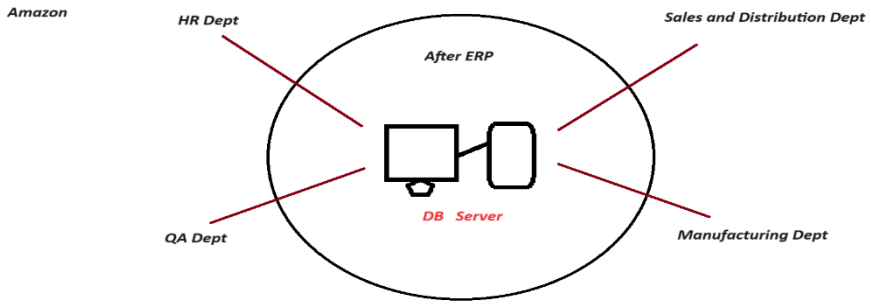
ex:
Facebook
Online Shopping website

> ERP App

ERP stands for Enterprise Resource Planning.

It is used to maintain the data in enterprise.

Diagram: java2.4



> N-tier App

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

>Mobile App

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

ex:
- PubG
- PhonePay
- Gpay
- templeRun
- and etc.

Q) What is Java ?

Java is a versatile language which is used to develop verious type of applications like distributed applications, enterprise applications, ERP applications, N-tier applications and etc.

Q) What is the difference between C++ and Java?

C++	Java
>It is developed by Bjarne Stroustrup.	>It is developed by James Gosling.
It is a partial object oriented programm--Ing language.	>It is a purely object oriented programming language.
>It is a platform dependent.	>It is platform independent.
>Memory allocation and deallocation will taken care by a programmer.	>Memory allocation and deallocation will taken care by JVM.
>It supports multiple inheritance.	>It does not support multiple inheritance.
>It supports pointers.	>It does not support pointers.
>It supports operator overloading.	>It does not support operator overloading.
>It supports three access specifiers i.e, public, private and protected.	>It supports four access modifiers i.e default Public ,private and protected.
>It supports three types of loops i.e loop ,do while loop, while loop and for loop.	>It supports four types of loops i.e do while while loop, for loop and for each loop.
>It supports preprocessor directory(#).	>It does not support preprocessor directory(#).
>To save c++ program we will use .cpp extension.	>To save java program we will use .java Extension.

Q) What is the difference between .Net and Java ?

.Net	Java
>It is a product of Microsoft.	>It is a product of Oracle Corporation.
>It is platform dependent.	>It is platform independent.
>There is less security.	It is highly secured.

>It contains small set of frameworks. >It contains large set of frameworks.

>It is best suitable for medium scale projects. >It is best suitable for large scaleprojects.

Types of IT companies
=====================
We have three types of IT companies.

1) Product Based companies
   ex:
      Oracle Corporation , Microsoft , IBM and etc.

2) Service Based companies
   ex:
      Cognizent , TCS , Capgemini and etc.

3) Cloud Based companies
ex:
      Amazon , Google, Microsoft and etc.

*SDLC

=======
SDLC stands for Software Development Life Cycle.

It is a process adopt by IT Industries to develop accurate and quality of softwares.

We have six phases in SDLC.

1) Feasibility study

2) Analysis

3) Designing

4) Coding

5) Testing

6) Delivery and Maintainence

1) Feasibility study

Feasibility study completely depends upon TELOS formulea.
ex:
   T - Technical feasibility
   E - Economical feasibility
   L - Legal feasibility
   O - Operational feasibility
   S - Scheduled feasibility

All the above information they will keep in a document called BDD.

BDD stands for Business Design Document.

2) Analysis

In analysis phase, system analyst or product owner will involved.

They will seperate system requirement and software requirements.

Later they will keep all the above information in a document called SRS.

SRS stands for Software Requirements Specification.

3) Designing

We have two types of designing.

i) High Level designing

A manager is responsible to perform high level designing.
In high level designing, we will design main/major modules.

ii) Low Level designing

A team lead/project lead is responsible to perform low level designing.

In low level designing , we will design child/sub modules.

All the above information they will keep in a document called PDD/TDD.

PDD stands for Product Design Document.
TDD stands for Technical Design Document.

4) Coding

In coding phase, developers will involved.

Developers are responsible to generate the build.

Developers even responsible to perform unit testing.

5) Testing

In testing phase, Testing team or QA team will involved.

They will use one software component called STLC to test the build.

STLC stands for Software Testing Life Cycle.

6) Delivery and Maintainence

Before delivery , we will perform UAT testing.

UAT stands for User Acceptance Testing.

UAT testing is divided into two types.

i) Alpha testing

ii) Beta testing

*Comments in java

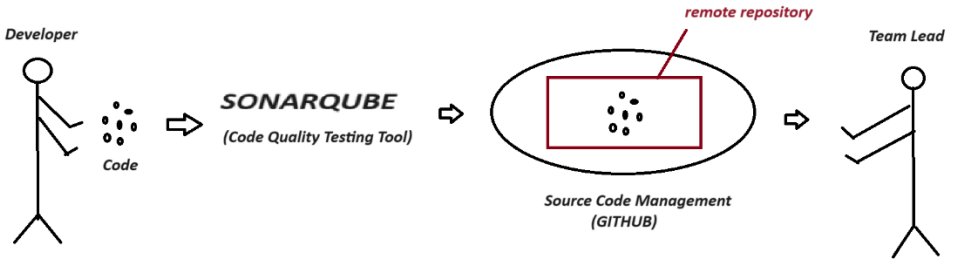
==========

>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.

Diagram: java4.1



Comments will not display in output because they will not compiled by the compiler.

In java, we have two types of comments.

1) Single line comment

It is used to comment a single line.
ex:
// comment here

2) Multiple line comment

It is used to comment multiple lines.
ex:
/*

```
                -
                - comment here
                -
          */

ex:
---
//class declaration
class  Example
{

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

            //variable declaration
            int i=10;


            //output stmt
            System.out.println(i);
      }
}
```

***Naming Conventions in Java**

==========

In java, uppercase letters will consider as different and lowercase letters will consider as different. Hence 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
- constants

classes

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

ex:

Predefined classes	Userdefined classes
System	Test
File	TestApp
BufferedWriter	DemoExample
StringBuffer	EmployeeDetails

and etc.	and etc.

interfaces

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

ex:

predefined interfaces	userdefined interfaces
Runnable	ITest
Serializable	IDemoApp
ListIterator	IExampleDemo
Enumeration	IEmployeeDetails
and etc.	and etc.

variables

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

ex:

predefined variables	userdefined variables
length	empId
out	studName
err	deptNo
in	countTokens
and etc.	and etc.

Methods

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

ex:

predefined methods	userdefined methods
getPriority()	getDetails()
setName()	setInfo()
getClass()	getEmployeeDetails()
hashCode()	calculateBillAmt()
toString()	discount()
and etc.	and etc.

keywords

In java, all keywords we need to declare under lowercase letters only.

ex:

predefined keywords
---
if, else, for, switch, while, public, static, void and etc.

packages

In java, all packages we need to declare under lowercase letters only.

ex:

predefined packages	user-defined packages
java.lang(default pkg)	com.ihub.www
java.io	com.google.www
java.util	com.java.www
java.util.stream	and etc.
java.sql	
java.time	
java.text	
and etc.	

constants

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

ex:

predefined constants	userdefined constants
MAX_PRIORITY	int LIMIT=10;
MIN_PRIORITY	
NORM_PRIORITY	
MAX_VALUE	
MIN_VALUE	
and etc.	

Interview Questions

Q) Which package is a default package in java?

java.lang package

Q) How many reserved words are present in java?

53

Q) Java is platform dependent or independent?

platform independent.

Q) JVM is platform dependent or independent?

JVM is platform dependent.

Q) How many classes are there in java?

According to Java 8 we have 4240 classes.

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

JDK

JDK stands for Java Development Kit.

It is a installable software which consist of Java Runtime Environment (JRE), Java Virtual Machine (JVM), compiler (javac) , interpreter (java), an archiever (.jar) , document generator (javadoc) and other tools needed for java application development.

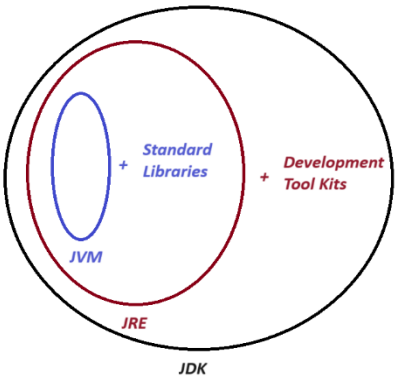
JRE

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: java4.2



5) package : com.qualitythought.www

*History of Java

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

James Gosling, Mike Sheradin and Patrick Naughton were there to develop the project and they have 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 the project. But the problem what they have faced is C and C++ languages are system dependent. Then James Gosling decided why don't we created our own programming language which is system independent.

In 1991, They have develop a programming language called 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.

Later in 1995, they have renamed OAK to Java. Java is a island of an Indonasia where first coffee of seed was produced and during the development of project, they were consuming lot of coffee's. Hence a symbol of java is a cup of coffee with saucer.

*Identifiers

A name in java is called identifier.

It can be class name, variable name, method name or label name.

ex:

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

            System.out.println(x);
      }
}
```

Here Test, main , args , x , String, System are identifiers.

Rules to declare an identifiers

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  emp$alary;
int  emp_id;
int  emp#Name; //invalid
```

Rule3:

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

ex:

```
int emp123;
int empSal;
int $alary;
int 1abcd; // invalid
```

Rule4:

We can't take reserved words as an identifiers.

ex:

```
int  if;  //invalid
int  else; //invalid
int   for; //invalid
```

Rule5:

Every identifier is a case sensitive.

ex:

```
int  number;
int  NUMBER;
int  NuMbEr;
```

Rule6:

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

Reserved Words
==========

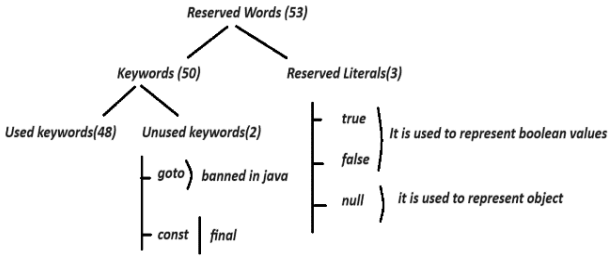
There are some identifiers which are reserved to associate some functionality or meaning
such type of identifiers are called reserved words.

Java supports 53 reserved words.

All reserved words we need to declare under lowercase letters only.

In java , reserved words are divided into two types.

Diagram: java5.1



~~enum~~
extends
implements
package
import

Used keywords with respect to object
---
new
instanceof
this
super

Used keywords with respect to datatypes
---
byte
short
int
long
float
double
boolean
char

Used keywords with respect to modifiers
---
default

public
private
protected
abstract
final
static
strictfp
synchronized
transient
volatile
native

Used keywords with respect to returntype
---
void

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

for
case

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

Interview Questions

Q) Who is the creator of Java?

James Gosling

Q) In which year java was developed?

In 1995.

Q) Java originally known as ___ ?

OAK

Q) What are the features of Java?

We have following important features in Java.

1) Simple

2) Object oriented

3) Platform independent

4) Architecture Neutral

5) Multithreaded

6) Robust

7) Portable

8) Highly secured

9) Dynamic

10) Distributed

and etc.

***INSTALLATION OF JAVA**

==========

Version : Java 8

Software : JDK 11

Creator : James Gosling

Vendor : Oracle Corporation

Open source : Open source

Website : www.oracle.com/in/java

Download link :

https://drive.google.com/file/d/1GtRLHXK4y3s97BH2UcYiJPNBaROR1DBV/view?usp=drive_link

Steps to setup environmental variables

==========

step1:

Make sure JDK 11 installed successfully.

step2:

Copy "lib" directory from java-home folder.

ex:

```
            C:\Program Files\Java\jdk-11\lib

step3:
------
      Paste "lib" directory in environmental variables.
      ex:
            Right click to My PC --> properties --> Advanced System
settings -->
            Environmental variables --->

            user variables  --> click to new button
            variable name :  CLASSPATH
            variable value : C:\Program Files\Java\jdk-11\lib; ---> ok.

            system variables --> click to new button
            variable name : path
            variable value : C:\Program Files\Java\jdk-11\bin; -->   ok
      --> ok --> ok.


step4:
-----
      Test the environment setup done perfectly or not.
      ex:
            cmd> javap
            cmd> java -version


Steps to develop first Java application
=======================================
step1:
------
      Make sure JDK 11 installed successfully.


step2:
-----
      Make sure environmental setup done perfectly.

step3:
-----
      Create a "javaprog" folder inside 'E' drive.

step4:
-----
      Open the notepad and develop simple Hello World program.
      ex:
            class Test
            {
                  public static void main(String[] args)
                  {
                        System.out.println("Hello World");
                  }
            }

step5:
------
```

Save above program by using same name as class name inside "javaprog" location.

step6:

Open the command prompt from javaprog location.

step7:

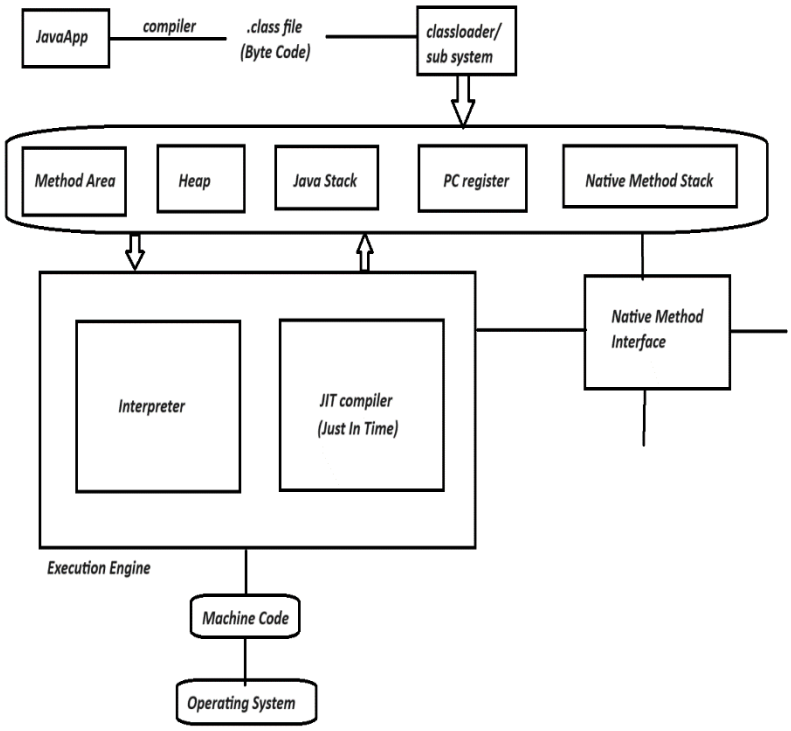
Compile java program by using below command.
ex:
```
javac Test.java
      |
      file name
```

step8:

Run java program by using below command.
ex:
```
java Test
      |
      class name
```

*Internal Architecture of JVM

Diagram: java6.1



Java program contains java code instructions. Once if we compiled , our java code instructions convert to byte code instructions in .class file.

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

We have five 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.

3) Java Stack

[whenever we declare variables inside a method they will strore in java Stack]
Java methods stored in method area. But to execute those methods we required some memory and that memory will be allocated in Java Stack.

4) PC Register

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

5) Native Method Stack

Java methods execute in method area.
Similarly native methods execute in native method stack.
But we can't execute native methods directly. We required a program called Native method interface.

Execution engine

Execution engine contains interpreter and JIT compiler.

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

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

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

Atlast ,JVM converts our byte code converts to machine code and that code execute in OS.

Interview Questions

==========

Q) What is Native Method in java?

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

Q) What is JIT compiler?

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

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)How many classloaders are there in java?

We have three predefined classloaders in java.

1) Bootstrap classloader

2) Extension classloader

3) System/Application classloader

*Datatypes

==========

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

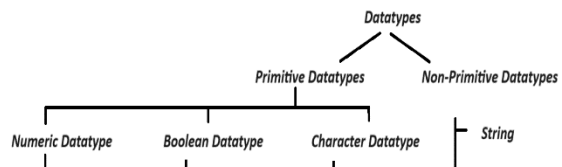
Datatype also tells how much memory has to be created for a variable.

In java, datatypes are divided into two types.

1) Primitive Datatypes

2) Non-Primitive Datatypes

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

      2) short s="hi";
         System.out.println(s); // C.T.E
```

```
      3) short s=true;
         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) byte b=10;
         short s=b;
         int i=s;
         System.out.println(i); // 10

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

      3) int i=10.5;
         System.out.println(i); // C.T.E

      4) int i='a';
         System.out.println(i); // 97

Note:
-----
In java for every character we have universal unicode value.

ex:
      A -----> 65
      a -----> 97

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

Size : 8 bytes (64 bits)

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

ex:
      1) long l="a";
           System.out.println(l); //C.T.E

      2) long l='A';
         System.out.println(l); //65

      3) long l=10.5;
         System.out.println(l); //C.T.E

float                              double
----------                         --------
If we need 4 to 6 decimal point of accuracy   If we need 14 to 16
decimal point of accuracy
```

then we need to use float.

Size: 4 bytes (32 bits)

Range: -3.4e38 to 3.4e38

To declare a float value we need to we need to suffix
suffix with 'f' or 'F'.
ex:
```
10.56f
```

then we need to use double.

Size: 8 bytes (64 bits)

Range: -1.7e308 to 1.7e308

To declare a double value with 'd' or 'D'.
ex:
```
10.56d
```

ex:

```
1) float f=10.5f;
   System.out.println(f); // 10.5

2) float f=10;
   System.out.println(f); // 10.0

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

4) float f="hi";
   System.out.println(f); // C.T.E

5) float f=true;
   System.out.println(f); // C.T.E
```

ex:

```
1) double d=10.5d;
   System.out.println(d); // 10.5

2) double d=10;
   System.out.println(d); // 10.0

3) double d='a';
     System.out.println(d); // 97.0

4) double d="hi";
   System.out.println(d); // C.T.E

5) double d=true;
   System.out.println(d); // C.T.E
```

boolean

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

Size : (Not Applicable) (1-bit)

Range : (Not Applicable)

ex:
```
1) boolean b="true";
```

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

   2) boolean b=TRUE;
      System.out.println(b); // C.T.E

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

char

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

Size : 2 bytes (16 bits)

Range : 0 to 65535

ex:

```
   1) char ch='a';
      System.out.println(ch); // a

   2) char ch=97;
      System.out.println(ch); // a

   3) char ch='ab';
      System.out.println(ch); // C.T.E
```

Diagram: java7.2

Datatypes	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	0L
float	4 bytes	-3.4e38 to 3.4e38	Float	0.0f
double	8 bytes	-1.7e308 to 1.7e308	Double	0.0d
boolean	-	-	Boolean	false
char	2 bytes	0 to 65535	Character	0(space)

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

INTERVIEW QUESTIONS:

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

Range : -2147483648 to 2147483647

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

Q) Is java , purely object oriented or not?

No, java will not consider as 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.

*Types of variables

==========

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

Purpose of variable is used to store the data.

In java, we have two types of variables.

1) Primitive variables

It is used to represent primitive values.

2) Reference variables

It is used to represent object reference.

ex:

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

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

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

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

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

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

ex:1

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

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

ex:2

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

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

Note:

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

ex:3

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

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

ex:4

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

	public static void main(String[] args)
	{
		Test t1=new Test();
		Test t2=new Test();

		System.out.println(t1.i); //10
		System.out.println(t2.i); //10

		t1.i=100;

		System.out.println(t1.i); //100
		System.out.println(t2.i); //10
	}
}
```

ex:5

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

}
```

2) Static variables

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

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

Static variable will store in method area.

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

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

Static variable we can access by using object reference and class name.

ex:1

```
class Test
{
	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
	}
}
```

Note:

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

ex:2

```
class Test
{
	static String s=”ram”;

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

		Test t=new Test();
		System.out.println(t.s); //null

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

```
}
```

ex:3

```
class Test
{
	static int i=10;

	public static void main(String[] args)
	{
		Test t1=new Test();
		Test t2=new Test();

		System.out.println(t1.i); // 10
		System.out.println(t2.i); // 10

		t1.i=100;

		System.out.println(t1.i); // 100
		System.out.println(t2.i); // 100
	}
}
```

ex:4

```
class Test
{
	public static void main(String[] args)
	{
		m1();

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

		Test.m1();
	}
	public static void m1()
	{
		System.out.println("static-method");
	}
}
```

3) Local variables

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

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

Local variable will store in java stack.

ex:

```
class Test
```

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

Note:

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

ex:2

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

A local variable will accept only one modifier is final.

ex:3

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

Assignment
==========
Q) Declare a class?

```
	class Test
	{

	}
```

Q) Declare a instance variable?

```
	int i=10;
```

Q) Declare a static variable?

```
	static int i=10;
```

Q) Declare a main method?

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

}
```

Q) Declare a non-static method?

```
public void methodOne()
{
}
```

Q) Declare a static method?

```
public static void methodOne()
{

}
```

***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 liability of a JVM to look for main method.

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

JVM always look for main method with following signature.

ex:

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

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

Q) Explain main method in java?

public

JVM wants to call main method from anywhere.

static

JVM wants to call main method without using object reference.

void

Main method does not return anything to JVM.

main

      It is an identifier given to main method.

String[] args

      It is a command line argument.

We can perform following changes in main method.

1) Order of modifiers is not important in case of public static we can declare static public also.
   ex:
      static public void main(String[] args)

2) We can declare 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 String[] with var-arg parameter.
   ex:
      public static void main(String... args)

4) We can change args with any java valid identifier.
   ex:
      public static void main(String[] ihub)

5) Main method will accept following modifiers.
   ex:
      synchronized, strictfp and final.

***Command line argument**
==========
Arguments which are passing through command prompt such type of arguments are called command line arguments.

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

ex:
      javac Test.java

      java Test 101   Alan 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]);
	}
}
```

System.out.println()
==========
It is a output statement in java.

Whenever we want to display any userdefined statements or data then we need to use output stmt.

syntax:

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

Diagram: java9.1

ex:

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

Various ways to display the data
---

1)

```
	System.out.println("Hello World");
```

2)

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

2)

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

3)

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

*Fully Qualified Name

Fully qualified name means we will declare a class or interface along with package.

ex:

```
java.lang.System(C)
java.lang.Runnable(I)
```

Fully qualified name is used to improve readability of our code.

ex:

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

*Import Statements

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

Using short name also we can achieve.

In java, we have three types of import statements.

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 improve readability of our code.

ex:

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

ex:

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

ex:

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

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

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)
      {
            LocalDate date=LocalDate.now();
            System.out.println(date);

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

3) Static import

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

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

ex:

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

ex:

```
import static java.lang.System.*;
class Test
{
	public static void main(String[] args)
	{
		out.println("stmt1");
		exit(0);
		out.println("stmt2");
	}
}
```

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

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

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

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

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

		//logic
		int c = a + b;

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

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

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

```
class Example2
{
	public static void main(String[] args)
	{
		Scanner sc=new Scanner(System.in);

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

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

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

Assignment
==========

Q) Write a java program to perform area of a rectangle?

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

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

```
input:
	5

output:
	25

ex:

import java.util.Scanner;
class Example3
{
	public static void main(String[] args)
	{
		Scanner sc=new Scanner(System.in);

		System.out.println("Enter the number :");
		int n=sc.nextInt(); //5

		//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?

```
input:
	5
```

output:
	125

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

		System.out.println("Enter the number :");
		int n=sc.nextInt(); //5

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

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

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

```
import java.util.Scanner;
class Example5
{
	public static void main(String[] args)
	{
		Scanner sc=new Scanner(System.in);

		System.out.println("Enter the Radius :");
		int r=sc.nextInt();//5

		float area=3.14f*r*r;

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

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

```
import java.util.Scanner;
class Example6
{
	public static void main(String[] args)
	{
		Scanner sc=new Scanner(System.in);

		System.out.println("Enter the Radius :");
		int r=sc.nextInt();//5

		//logic
		float perimeter=2*3.14f*r;
```

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

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

```
input:
      a = 10  and  b = 20

output:
      a = 20  and  b = 10
```

```
import java.util.Scanner;
class Example7
{
      public static void main(String[] args)
      {
            Scanner sc=new Scanner(System.in);

            System.out.println("Enter the first number :");
            int a=sc.nextInt(); //10

            System.out.println("Enter the second number :");
            int b=sc.nextInt(); //20

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

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

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

      }
}
```

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

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

            System.out.println("Enter the first number :");
            int a=sc.nextInt(); //10

            System.out.println("Enter the second number :");
            int b=sc.nextInt(); //20

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

            //swapping logic
```

```
            a = a + b;
            b = a - b;
            a = a - b;

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

      }
}
```

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

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

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

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

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

      }
}
```

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

```
import java.util.Scanner;
class Example10
{
      public static void main(String[] args)
      {
            Scanner sc=new Scanner(System.in);

            System.out.println("Enter the CGPA :");
            float cgpa=sc.nextFloat();

            float percentage=cgpa*9.5f;

            System.out.println("CGPA to percentage is ="+percentage);

      }
}
```

*Typescasting in java

==========

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

In java, typescasting can be performed in two ways.

1) Implicit typecasting

2) Explicit typecasting

1) Implicit typecasting (Upcasting)

If we want to store small value in 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=10;
		int i=b;
		System.out.println(i);//10
	}
}
```

ex:2

```
class Test
{
	public static void main(String[] args)
	{
		char ch='a';

		double d=ch;

		System.out.println(d); // 97.0
	}
}
```

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 (downcasting)

If we want to store bigger value in 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)
      {

            double d=10.5d;

            int i=(int)d;

            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
      }
}
```

Assignment
==========
Q) Write a java program to accept six marks of a student then find out total and average?

*Types of blocks in Java

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

In java, we have three types of blocks.

1) Instance block

2) Static block

3) Local block

1) Instance block

It is used to initialize the values to instance variables.

Instance block must and should declare immediately after the class but not inside methods and constructors.

Instance block will execute when we create an object.

We can declare instance block as follow.

syntax:

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

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:

```
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();
	}
}
```

ex:

```
class Test
{
```

```
	//instance variable
	int i;

	//instance block
	{
		i=100;
	}

	public static void main(String[] args)
	{
		Test t=new Test();
		System.out.println(t.i);//100
	}
}
```

2) Static block

A static block is used to initialize the values to static variables.

A static block must and should declare immediately after the class using static keyword.

A static block will execute at the time of classloading.

We can declare static block as follow.

syntax:

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

ex:

```
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:

```
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

ex:
----
class Test
{
	//static variable
	static int i;

	//static block
	static
	{
		i=200;
	}

	public static void main(String[] args)
	{
		System.out.println(i);//200
	}
}
```

3) Local block

A local block is used to initialize the local variables.

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

A local block will execute just like normal statement.

We can declare a local block as follow.

syntax:

```
	//local block
	{
		-
```

```
            - //set of statements
            -
      }
```

ex:

```
class Test
{

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

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

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

ex:

```
class Test
{

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

            //local block
            {
                  i=300;
            }

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

Interview Question
==========

Q) Can we execute java program without main method?

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

ex:

```
class Test
{
```

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

Q) What is Literal ?

A value which is assigned to a variable is called literal.

ex:

```
      int   i  =  10;
       |    |         |___ value of a variable / Literal
       |    |_________ variable name / identifier
       |______________ Datatype / keyword
```

***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, bitwise operation and etc.

We have following list of operators in java.

1) Assignment operators

2) Ternary / Conditional operators

3) Logical operators

4) Bitwise operators

5) Arithmetic operators

6) Relational operators

7) Shift operators

8) Unary operators

1) Assignment operators
----
ex:

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

            i=20;

            i=30;

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

Note:

      Re-initialization is possible in java.

ex:

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

            i=20;

            i=30;

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

Note:

      We can't change or modify final variable.

ex:

```
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 assign multiple values.

ex:

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

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

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

Note:

Here priority goes to local variables.

ex:

```
class Test
{
	//static variable
	static int i=100;

	//instance variable
	int i=200;

	public static void main(String[] args)
	{
		Test t=new Test();
		System.out.println(t.i); // C.T.E
	}
}
```
Note:
-----
variable i is already defined in class Test

ex:

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

		i+=5;  // i = i + 5

		System.out.println(i); //15
	}
}
```

ex:

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

		i-=5;  // i = i - 5

		System.out.println(i); //5
	}
}
```

ex:

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

		i*=5;  // i = i * 5

		System.out.println(i); //50
	}
}
```

ex:

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

		i/=5;  // i = i / 5

		System.out.println(i); //2
	}
}
```

ex:

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

		i/=50;  // i = i / 50

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

ex:

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

		i %= 5;

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

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

		i %= 15;

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


2) Ternary operators
====================
syntax:
-------
	(condition)?value1:value2;

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

ex:
---
class Test
{
	public static void main(String[] args)
	{
		int i=(5>20)?1:0;
		System.out.println(i);//0
	}
}

ex:
---
```

```
class Test
{
	public static void main(String[] args)
	{
		String s=(true)?"hi":"bye";
		System.out.println(s);//hi
	}
}
```

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(); // 20

		//logic
		int max=(a>b)?a:b;

		System.out.println(max+" is greatest");
	}
}
```

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

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

		System.out.println("Enter the first number :");
		int a=sc.nextInt(); // 10

		System.out.println("Enter the second number :");
		int b=sc.nextInt(); // 20

		System.out.println("Enter the third number ");
		int c=sc.nextInt(); // 5

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

		System.out.println(max+" is greatest");
	}
}
```

3) Logical operators
====================

Logical AND operator (&&)

Logical AND operator deals with boolean values either true or false.

Truth table

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

ex:

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

ex:

```
class Test
{
	public static void main(String[] args)
	{
		boolean b = (5>2) && (10<20);

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

ex:

```
class Test
{
	public static void main(String[] args)
	{
		String s=((6>2) && (10<5))?"Welcome":"Thankyou";

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

Logical OR operator (||)

Logical OR operator deals with boolean values either true or false.

Truth table

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

ex:

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

ex:

```
class Test
{
      public static void main(String[] args)
      {
            boolean b = (5>2) || (60<10);

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

ex:

```
class Test
{
      public static void main(String[] args)
      {
            int i=(5>20 || false)?1:0;

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

Logical NOT operator (!)

Logical NOT operator deals with boolean values either true or false.

ex:

```
class Test
{
      public static void main(String[] args)
      {
            String str=(!(5>2))?"Hi":"Bye";

            System.out.println(str); // Bye
      }
```

```
}
```

ex:

```
class Test
{
      public static void main(String[] args)
      {
            String str=(!(5>20))?"Hi":"Bye";

            System.out.println(str); // Hi
      }
}
```

How to convert decimal to binary number

```
      Decimal number    -     10
      Binary number     -     1010

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

How to convert binary to decimal number

```
      binary number     -     1010
      decimal number    -     10

      1010
            <---

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

      0 + 2 + 0 + 8 = 10
```

4) Bitwise operators

Bitwise AND operator (&&)

```
--------------------------
Bitwise AND operator deals with binary numbers.

Truth table
----------
T     T     = T
T     F     = F
F     T     = F
F     F     = F

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

            int c = a & b;

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



/*
      10 - 1010
      15 - 1111
      ----------
      &  - 1010
                        <----

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

      0 + 2 + 0 + 8 = 10
*/

ex:
---
class Test
{
      public static void main(String[] args)
      {
            int a=2, b=3;

            int c = a & b;

            System.out.println(c); //2
      }
}
/*
      2 - 0010
      3 - 0011
      --------
      & - 0010
                        <----
```

```
      0*1  +  1*2 +  0*4  +  0*8

      0 + 2 + 0 + 0 = 2
*/
```

Bitwise OR operator (||)

Bitwise AND operator deals with binary numbers.

Truth table

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

ex:

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

            int c = a | b;

            System.out.println(c); //15
      }
}
/*
      10 - 1010
      15 - 1111
      ----------
      |  - 1111
                        <----

      1*1 + 1*2 + 1*4 + 1*8

      1 + 2 + 4 + 8 = 15
*/
```

Bitwise XOR operator (^)

Bitwise AND operator deals with binary numbers.

Truth table

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

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

```
            int a=10,b=15;

            int c = a ^ b;

            System.out.println(c); // 5
      }
}
/*
      10 - 1010
      15 - 1111
      ----------
      ^  - 0101
                        <----

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

      1 + 0 + 4 + 0 = 5
*/
```

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= ~56;

            System.out.println(i); //  -57
      }
}
```

ex:

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

            System.out.println(i); //  9
      }
}
```

5) Arithmetic Operators
==========
% - modules
/ - division
* - multiplication
+ - addition
- - subtraction

ex:

```
class Test
{
	public static void main(String[] args)
	{
		int i= 6+7%2+5*2+8/2+9/10+6*2+10%20-40;
		System.out.println(i); // 3
	}
}
/*
	6 + 7%2  +  5*2  +  8/2  +  9/10  +  6*2  +  10%20  -  40

	6 + 1  +  10   +  4  +  0  +  12  +  10  - 40

	43 - 40
	3  */
```
6) Relational operators
==========
```
class Test
{
	public static void main(String[] args)
	{
		System.out.println(10 > 20); //false

		System.out.println(10 >= 20 ); //false

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

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

ex:

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

		System.out.println(10 == 20); //false

		System.out.println(10 != 10); //false

		System.out.println(10 != 20); //true

	}
}
```

7) Shift operators
==========

Right shift operator (>>)
---
10 >> 1 = 10/2

10 >> 2 = 10/4 (10/2*2)

10 >> 3 = 10/8 (10/2*2*2)

10 >> 4 = 10/16 (10/2*2*2*2)

ex:

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

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

	}
}
```

ex:

```
class Test
{
	public static void main(String[] args)
	{
		int i= 100 >> 5; // 100 / 32

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

	}
}
```

Left shift operator (<<)
---
10 << 1 = 10*2

10 << 2 = 10*4 (10/2*2)

10 << 3 = 10*8 (10/2*2*2)

10 << 4 = 10*16 (10/2*2*2*2)

ex:

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

```
            int i = 5 << 3;  // 5 * 8

            System.out.println(i); //  40

      }
}
```

ex:

```
class Test
{
      public static void main(String[] args)
      {
            int i = 10 << 6;

            System.out.println(i); //  10 * 64  = 640

      }
}
```

8)Unary operators
==========

Increment/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;
```

Post increment / decrement
==========

Rule1: First Take

Rule2: Then Change

ex:

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

            i++;

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

      }
}
```

ex:

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

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

ex:

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

            int j=i++;

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

      }
}
```

ex:

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

            int j=i-- + i--; //10 + 9

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

      }
}
```

ex:

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

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

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

      }
}
```

Pre Increment/Decrement

Rule1: First Change

Rule2 : Than Take

ex:1

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

            ++i;

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

      }
}
```

ex:

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

            System.out.println(++i);//11

      }
}
```

ex:

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

		int j=++i;

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

	}
}
```

ex:

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

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

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

	}
}
```

ex:

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

		System.out.println(i++ + ++i);// 10  + 12 = 22

	}
}
```

ex:

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

		100++;

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

	}
}
```

ex:

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

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

	}
}
```

ex:

```
class Test
{
	public static void main(String[] args)
	{
		byte b=127;

		b++;

		System.out.println(b);//-128

	}
}
```

Question

Q) Write a java program to find 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(); //5

		int result=(int)Math.pow(n,2);

		System.out.println(result);
	}
}
```

Assignment

Q) Write a java program to convert USD to INR ?

Input:

	1

Output:

	83.52

Q) Write a java program to convert INR to USD?

Input:
	83.52

Output:
	1

Control Statements

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

Control statement allows us to make decisions, to jump from one section of code to another section and to execute the code repeatedly.

In java, we have four control statements.

1) Decision Making Statement

2) Select Statement

3) Iteration Statement

4) Jump Statement

1) Decision Making Statement

It is used to declare condition in our code.

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 is used to execute the code only if our condition is true.

syntax:

```
	if(condition)
	{
		-
		- //code to be execute if cond is true
		-
	}
```

ex:

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

            if(5>2)
            {
                  System.out.println("stmt2");
            }

            System.out.println("stmt3");
      }
}
```
o/p:
```
      stmt1
      stmt2
      stmt3
```

ex:

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

            if(!(5>2))
            {
                  System.out.println("stmt2");
            }

            System.out.println("stmt3");
      }
}
```
o/p:
```
      stmt1
      stmt3
```

ex:

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

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

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

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

            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?

```
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)
{
      -
      - //code to be execute if cond is true
      -
}
else
{
      -
      - //code to be execute if cond is false
      -
}
```

ex:

```
class  Test
{
      public static void main(String[] args)
      {
            System.out.println("stmt1");
            if(6!=10)
            {
                  System.out.println("stmt2");
            }
            else
            {
                  System.out.println("stmt3");
            }
            System.out.println("stmt4");
      }
}
```

o/p:

```
stmt1
stmt2
stmt4
```

ex:

```
class  Test
{
      public static void main(String[] args)
      {
            System.out.println("stmt1");
            if(6==10)
            {
                  System.out.println("stmt2");
            }
            else
            {
                  System.out.println("stmt3");
            }
            System.out.println("stmt4");
      }
```

```
}
o/p:
      stmt1
      stmt3
      stmt4
```

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

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

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

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

      }
}
```

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

```
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)
            {
                  System.out.println("It is not a positive or negative
number");
                  System.exit(0);
            }

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

      }
}
```

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

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

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

            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 check 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();

            if(n%2!=0)
                  System.out.println("It is odd number");
            else
                  System.out.println("It is not odd number");
      }
}
```

Q) Write a java program to check given year is a leap year or not?

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

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

            if(year%4==0 && year%100!=0 || year%400==0)
                  System.out.println("It is a leap year");
            else
                  System.out.println("It is not a leap year");
      }
}
```

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 cond is true
      }
      else
      {
            -//code to be execute if all condition are false.
      }

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

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

            if(option==100)
                  System.out.println("It is a police number");
            else if(option==103)
                  System.out.println("It is enquiry number");
            else if(option ==108)
                  System.out.println("It is emergency number");
            else
                  System.out.println("Invalid option");

      }
}
```

Q) Write a java program to check given alphabet is a uppercase letter, lowercase letter, digit or a special symbol?

```
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 uppercase letter");
            else if(ch>='a' && ch<='z')
                  System.out.println("It is lowercase letter");
            else if(ch>='0' && ch<='9')
                  System.out.println("It is a digit");
            else
                  System.out.println("It is a special symbol");


      }
}
```

Q) Write a java program to check given alphabet is a vowel 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 alphabet:");
            char ch=sc.next().charAt(0);

            if(ch=='a' || ch=='A')
                  System.out.println("It is a vowel");
            else if(ch=='e' || ch=='E')
                  System.out.println("It is a vowel");
            else if(ch=='i' || ch=='I')
                  System.out.println("It is a vowel");
            else if(ch=='o' || ch=='O')
                  System.out.println("It is a vowel");
            else if(ch=='u' || ch=='U')
                  System.out.println("It is a vowel");
            else
                  System.out.println("It is not a vowel");
      }
}
```

Assignment

==========

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.

iv) nested if stmt

==========

If declare if stmt inside another if stmt is called nested if stmt.

syntax:

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

ex:

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

                  if(true)
                  {
                        System.out.println("stmt3");
                  }

                  System.out.println("stmt4");
            }
            System.out.println("stmt5");
      }
}
```

o/p:

```
      stmt1
      stmt2
      stmt3
      stmt4
      stmt5
```

ex:

```
---
class Test
```

```
{
	public static void main(String[] args)
	{
		System.out.println("stmt1");
		if(5>20)
		{
			System.out.println("stmt2");

			if(true)
			{
				System.out.println("stmt3");
			}

			System.out.println("stmt4");
		}
		System.out.println("stmt5");
	}
}
o/p:
	stmt1
	stmt5

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

			if(false)
			{
				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 ?

```
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 positive number");
                        System.exit(0);
                  }

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

2) Selection statement

==========

switch

=======

It will execute the source code based on multiple conditions.

It is similar to if else if ladder.

syntax:

```
      switch(condition)
      {
            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.

      }
```

ex:

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

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

            switch(option)
            {
                  case 100: System.out.println("It is a police number");
                              break;
```

```
                  case 103: System.out.println("It is a enquirynumber");
                                    break;
                  case 108: System.out.println("It is a emergency
number");
                                    break;
                  default:  System.out.println("Invalid option");
            }


      }
}
```

Declaration of break stmt is optional. If won't declare break stmt then from where our condition is satisfied from there all cases will be executed. That state is called fall through state of switch case.

ex:

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

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

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


      }
}
```

The allowed datatype of switch case are byte , short, int, char and String.

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

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

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

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

            switch(ch)
            {
                  case 'a': System.out.println("It is a vowel"); break;
                  case 'e': System.out.println("It is a vowel"); break;
                  case 'i': System.out.println("It is a vowel"); break;
                  case 'o': System.out.println("It is a vowel"); break;
                  case 'u' : System.out.println("It is a vowel"); break;
                  default:  System.out.println("It is a consonent");
            }


      }
}

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

            System.out.println("Enter the 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("Coming Soon...");
            }


      }
}
```

3) Iteration Statement

==========

Iteration statement is used to execute the source code repeatedly.

Iterataion statement is possible by using LOOPs.

We have four types of loops.

i) do while loop

ii) while loop

iii) for loop

iv) for each loop

i) do while loop

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

syntax:

```
      do
      {
            -
            - //code to be execute
            -
      }while(condition);
```

ex:

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

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

ex:

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


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 descending order?

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

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

```
output:
	1+2+3+4+5+6+7+8+9+10 = 55

ex:
---
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);
	}
}
```

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

```
input:
      n=5

output:
      120 (5*4*3*2*1)


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

            int i=n ,fact=1;
            do
            {
                  fact = fact * i;
                  i--;
            }
            while (i>=1);

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

ii) 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=1;

            while(i<=10)
            {
                  System.out.print(i+" "); // infinite 1
            }
      }
}
```

ex:

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

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

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

```
class Test
{
      public static void main(String[] args)
      {
            int i=1;
            while(i<=100)
            {
                  System.out.print(i+" "); // 1 2 3 ... 100
                  i++;
            }
      }
}
```

Q) Write a java program to display 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);
      }
}
```

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(fact);
      }
}
```

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

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

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

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

  }

 }
}
```

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

```
input:
      123

output:
      6

ex:

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

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

            int rem,sum=0;
            while(n>0)
            {
                  rem=n%10;
                  sum=sum+rem;
                  n=n/10;
            }
            System.out.println(sum);
      }
}
```

Q) Write a java program to check given number is armstrong or not?

```
input:
      153

output:
      It is an amrstrong number (1*1*1+5*5*5+3*3*3)(1+125+27)(153)

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(); //153

            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 display reverse of a given number?

```
Input:
      123

Output:
      321
```

ex:

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

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

            int rem,rev=0;
            while(n>0)
            {
                  rem=n%10;
                  rev=rev*10+rem;
                  n=n/10;
            }
            System.out.println(rev);
      }
}
```

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

```
input:
      121

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

Orchasp company Interview Question

==========

Write a java program to calculate costs based on user input. The program
should prompt users to enter the total weight of items(in kilograms) and the shipping destination (domestic or international). For demostic orders, the program should charge Rs.500 for weights upto 5 kg and Rs.100 per additional kg. For international orders, it should charge Rs.1000 for weights upto 5 kg , Rs.200 per additional kg , and a Rs.500 surcharge for weights exceeding 10 kg. print calculated shipping cost.

Input:

```
      Enter total weight of items : 11
      Enter the shipping destination : domestic
```

Output:

```
      1100  (weight <=5 --> Rs. 500 + 600(6kgs) ) --> Domestic

      2700  (weight<=5 --> Rs. 1000 + 1200(6*200) +  500) -->
International
```

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

            System.out.println("Enter the Total Weight of Item :");
            int weight=sc.nextInt(); //11
```

```
            System.out.println("Enter the destination :");
            String destination=sc.next();//domestic

            if(destination.equals("domestic"))
            {
                  if(weight<=5)
                        System.out.println(500);
                  else if(weight>5)
                        System.out.println(500 + (weight-5) * 100);
            }
            else
            {
                  if(weight<=5)
                        System.out.println(1000);
                  else if(weight>5 && weight<=10)
                        System.out.println(1000 + (weight-5)*200);
                  else if(weight>10)
                        System.out.println(1000 + (weight-5)*200 + 500);
            }

      }
}
```

3) for loop

==========

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

syntax:

```
      for(initialization;condition;incrementation/decrementation)
      {
            -
            - //code to be execute
            -
      }
```

Note:

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







ex:
---
class Test
{
      public static void main(String[] args)
      {
            int sum=0;
            for(int i=1;i<=10;i++)
            {
                  sum+=i;
            }
            System.out.println(sum); //55
      }
}

ex:
---
class Test
{
      public static void main(String[] args)
      {
            for(int i=1;i<=10;i++)
            {
                  if(i%2==0)
                  {
                        System.out.print(i+" "); //2 4 6 8 10
                  }
            }
      }
}

ex:
```

```
---
class Test
{
	public static void main(String[] args)
	{
		int cnt=0;

		for(int i=1;i<=10;i++)
		{
			if(i%2==0)
			{
				cnt++;
			}
		}
		System.out.println(cnt); //5
	}
}

ex:
---
class Test
{
	public static void main(String[] args)
	{
		for(;;)
		{
			System.out.print("Hello ");
		}
	}
}
```

Q) Write a java program to check 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();

		boolean flag=true;

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

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

Assignment
==========

Q) Write a java program to find out factorial of a given number using for loop?

```
Input:
      n = 5

Output:
      120
```

Q) Write a java program to display reverse of a given number?

```
Input:
      123

Output:
      ThreeTwoOne
```

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

```
output:
      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:
---
class Test
{
      public static void main(String[] args)
      {
            for(int n=2;n<=100;n++)
            {
                  boolean flag=true;
                  for(int i=2;i<=n/2;i++)
                  {
                        if(n%i==0)
                        {
                              flag=false;
```

```
                        break;
                    }
                }
                if(flag==true)
                    System.out.print(n+" ");
            }
        }
}
```

Q) Write a java program to check given number is perfect or not?

```
Input:
      6

output:
      It is a perfect 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(); //6

            int sum=0;
            for(int i=1;i<n;i++)
            {
                  if(n%i==0)
                  {
                        sum+=i;
                  }
            }
            if(n==sum)
                  System.out.println("It is a perfect number");
            else
                  System.out.println("It is not a perfect number");
      }
}
```

Q) Write a java program to find out GCD(Greatest Common Divisor) of two numbers?

```
Input:
      12    18

Output:
      6

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

```
            int a=12,b=18,gcd=0;

            for(int i=1;i<=a && i<=b;i++)
            {
                  if((a%i==0) && (b%i==0))
                  {
                        gcd=i;
                  }
            }
            System.out.println("GCD of two numbers is ="+gcd);
      }
}
```

Q) Write a java program to display fibonacci series of a given number?

```
input:
      6

output:
      0 1 1 2 3 5 8

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

            int a=0,b=1,c;
            System.out.print(a+" "+b+" ");

            for(int i=2;i<=n;i++)
            {
                  c=a+b;
                  System.out.print(c+" ");
                  a=b;
                  b=c;
            }
      }
}
```

LOOP Patterns

==========

1)

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

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

```
            //rows
            for(int i=1;i<=4;i++)
            {
                  //cols
                  for(int j=1;j<=4;j++)
                  {
                        System.out.print(i+" ");
                  }
                  //new line
                  System.out.println();
            }
      }
}
```

2)

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

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

3)

*** * * ***
*** * * ***
*** * * ***
*** * * ***

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

```
                }
                //new line
                System.out.println();
            }
      }
}
```

4)

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

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

5)

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

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

6)

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

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

7)

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

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

8)

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

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

9)

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

```
class Test
{
	public static void main(String[] args)
	{
		//rows
		for(int i=1;i<=5;i++)
		{
			//cols
			for(int j=1;j<=5;j++)
			{
				if(i==j || i+j==6)
					System.out.print("* ");
				else
					System.out.print("- ");
			}
			//new line
			System.out.println();
		}
	}
}
```

Assignment
==========
Write a java program to display below loop pattern?

ex:

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

Write a java program to display below loop pattern?

ex:

```
1  1  1
1  0  1
1  1  1
```

Left Side Loop Patterns
==========
1)

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

ex:

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

2)

```
4 4 4 4
3 3 3
```

2 2
1

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

3)

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

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

```
                        System.out.print("* ");
                  }
                  //new line
                  System.out.println();
            }
      }
}
```

4)

1
2 3
4 5 6
7 8 9 0

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

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

5)

2
4 6
8 10 12
14 16 18 20

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

```
            int num=2;

            //rows
            for(int i=1;i<=4;i++)
            {
                  //cols
                  for(int j=1;j<=i;j++)
                  {
                        System.out.print(num+" ");
                        num+=2;
                  }
                  //new line
                  System.out.println();
            }
      }
}
```

5)

1
3 5
7 9 11
13 15 17 19

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

            //rows
            for(int i=1;i<=4;i++)
            {
                  //cols
                  for(int j=1;j<=i;j++)
                  {
                        System.out.print(num+" ");
                        num+=2;
                  }
                  //new line
                  System.out.println();
            }
      }
}
```

6)

2
3 5
7 11 13
17 19 23 29

```
class Test
{
      public static void main(String[] args)
      {
            int num=2;
```

```
            //rows
            for(int i=1;i<=4;i++)
            {
                  //cols
                  for(int j=1;j<=i;j++)
                  {
                        while(true)
                        {
                              boolean flag=true;
                              for(int k=2;k<=num/2;k++)
                              {
                                    if(num%k==0)
                                    {
                                          flag=false;
                                          break;
                                    }
                              }
                              if(flag==true)
                              {
                                    System.out.print(num+" ");
                                    num++;
                                    break;
                              }
                              num++;
                        }
                  }
                  //new line
                  System.out.println();
            }
      }
}
```

7)

1
2 1
1 2 3
4 3 2 1

```
class Test
{
      public static void main(String[] args)
      {
            //rows
            for(int i=1;i<=4;i++)
            {
                  //odd records
                  if(i%2!=0)
                  {
                        for(int j=1;j<=i;j++)
                        {
                              System.out.print(j+" ");
                        }
                        //new line
                        System.out.println();
                  }
                  else
                  {
```

```
                        for(int j=i;j>=1;j--)
                        {
                              System.out.print(j+" ");
                        }
                        //new line
                        System.out.println();
                  }

            }

      }
}
```

Right side loop patterns

```
1)

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

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

                  //right elements
                  for(int j=1;j<=i;j++)
                  {
                        System.out.print(i+" ");
                  }
                  //new line
                  System.out.println();
            }

      }
}
```

```
2)

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

ex:

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

```
	{
		//rows
		for(int i=4;i>=1;i--)
		{
			//space
			for(int j=4;j>i;j--)
			{
				System.out.print("  ");
			}

			//right elements
			for(int j=1;j<=i;j++)
			{
				System.out.print(i+" ");
			}
			//new line
			System.out.println();
		}

	}
}
```

3)

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

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

			//element
			for(int j=1;j<=i;j++)
			{
				System.out.print("* ");
			}
			//new line
			System.out.println();
		}
```

```
            //descending
            //rows
            for(int i=3;i>=1;i--)
            {
                  //space
                  for(int j=4;j>i;j--)
                  {
                        System.out.print("  ");
                  }

                  //element
                  for(int j=1;j<=i;j++)
                  {
                        System.out.print("* ");
                  }
                  //new line
                  System.out.println();
            }

      }
}
```

Pyramid patterns

==========

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)
      {
            //rows
            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)
      {
            //rows
            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();
            }
      }
}
```

Interview Questions

==========

Q) Write a java program to display below loop pattern?

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

ex:

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

```
            //rows
            for(int i=1;i<=5;i++)
            {
                  //cols
                  for(int j=1;j<=5;j++)
                  {
                        if(i==3 || j==3)
                              System.out.print("* ");
                        else
                              System.out.print("  ");
                  }
                  //new line
                  System.out.println();
            }
      }
}
```

Q) Write a java program to display below loop pattern?

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

ex:

```
class Test
{
      public static void main(String[] args)
      {
            int rows=4;

            //rows
            for(int i=1;i<=rows;i++)
            {
                  //left side
                  for(int j=1;j<=i;j++)
                  {
                        System.out.print(j+" ");
                  }
                  //space
                  for(int j=1;j<=(rows-i)*2;j++)
                  {
                        System.out.print("  ");
                  }
                  //right side
                  for(int j=i;j>=1;j--)
                  {
                        System.out.print(j+" ");
                  }

                  //new line
                  System.out.println();
            }
      }
}
```

Assignment

==========

Q) Write a java program to display below loop pattern?

```
    1
   2 2
  3 3 3
 4 4 4 4
5 5 5 5 5
```

Q)Write a java program to display pascal triangle?

```
    1
   1 1
  1 2 1
 1 3 3 1
1 4 6 4 1
```

ex:

```
class Test
{
	public static void main(String[] args)
	{
		//rows
		for(int i=0;i<5;i++)
		{
			//spaces
			for(int j=1;j<5-i;j++)
			{
				System.out.print(" ");
			}

			int number=1;
			for(int k=0;k<=i;k++)
			{
				System.out.print(number+" ");
				number = number * (i-k)/(k+1);
			}

			//new line
			System.out.println();
		}
	}
}
```

Q) Write a java program to display pyramid loop pattern?

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

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

```
	{
		//rows
		for(int i=0;i<5;i++)
		{
			//spaces
			for(int j=1;j<5-i;j++)
			{
				System.out.print(" ");
			}

			for(int k=0;k<=i;k++)
			{
				System.out.print("* ");
			}

			//new line
			System.out.println();
		}
	}
}
```

4) Jump Statements

==========

Jump statements are used to jump from one section of code to another section.

We have two types of jump statements.

i) break stmt

ii) continue stmt

i) break stmt

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

For conditional statements we can use if condition.

syntax:

```
		break;
```

ex:

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

		break;
```

```
            System.out.println("stmt2");
      }
}
o/p:
      C.T.E : break outside switch or loop


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

            if(true)
            {
                  break;
            }
            System.out.println("stmt2");
      }
}

o/p:
      C.T.E : break outside switch or loop

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

ii) continue stmt
------------------
It is used to continue the execution of loops.

For conditional statement we can use if condition.

syntax:
      continue;

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

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

            continue;

            System.out.println("stmt2");
      }
}
o/p:
      C.T.E : continue outside of loop


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

            if(true)
            {
                  continue;
            }
            System.out.println("stmt2");
      }
}


o/p:
      C.T.E : continue outside of loop

ex:
---
class Test
{
      public static void main(String[] args)
      {
            for(int i=1;i<=10;i++)
            {
                  if(i==5)
                  {
                        continue;
                  }

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

Various ways to declare methods in java

==========

There are four ways to declare methods in java.

1) No returntype with No argument method

2) No returntype with Argument method

3) With returntype with No argument method

4) With returntype with Argument method

1) No returntype with No argument method

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

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

```
import java.util.Scanner;
class Test
{
	public static void main(String[] args)
	{
		//caller method
		sum();
	}
	//callie method
	public static void sum()
	{
		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();//20

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

Q) Write a java program to find out factorial of a given number using no return type with no argument method?

```
import java.util.Scanner;
class Test
{
	public static void main(String[] args)
	{
		//caller method
		factorial();
	}
	//callie method
	public static void factorial()
	{
		Scanner sc=new Scanner(System.in);
		System.out.println("Enter the number :");
		int n=sc.nextInt(); //5

		int fact=1;
		for(int i=n;i>=1;i--)
```

```
            {
                  fact*=i;
            }

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

2) No returntype with Argument method

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

Number of arguments are depends upon number of inputs.

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

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

            //caller method
            sum(a,b);
      }
      //callie method
      public static void sum(int a,int b)
      {
            int c=a+b;
            System.out.println("sum of two numbers is ="+c);
      }
}
```

Q) Write a java program to display reverse of a given number using no returntype with argument method?

```
import java.util.Scanner;
class Test
{
      public static void main(String[] args)
      {
            Scanner sc=new Scanner(System.in);
            System.out.println("Enter the number :");
            int n=sc.nextInt(); //123

            //caller method
            reverse(n);
```

```
	}
	//callie method
	public static void reverse(int n)
	{
		while(n>0)
		{
			switch(n%10)
			{
			  case 0: System.out.print("Zero");
				  break;
			  case 1: System.out.print("One");
				  break;
			  case 2: System.out.print("Two");
				  break;
			  case 3: System.out.print("Three");
				  break;
			  case 4: System.out.print("Four");
				  break;
			  case 5: System.out.print("Five");
				  break;
			  case 6: System.out.print("Six");
				  break;
			  case 7: System.out.print("Seven");
				  break;
			  case 8: System.out.print("Eight");
				  break;
			  case 9: System.out.print("Nine");
			  break;
			}
			n=n/10;
		}
	}
}
```

Assignment

==========

Q) Write a java program to check given number is palindrome or not using

i) No Returntype with no argument

ii) No Returntype with no argument

3) With returntype with No argument method

Here return type is depends upon output datatype.

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

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

```
	{
		//caller method
		int k=sum();
		System.out.println("sum of two numbers is ="+k);
	}
	//callie method
	public static int sum()
	{
		Scanner sc=new Scanner(System.in);

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

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

		//logic
		int c=a+b;

		return c;
	}
}
```

Q) Write a java program to find out area of a circle using with returntype with no argument method?

```
import java.util.Scanner;
class Test
{
	public static void main(String[] args)
	{
		//caller method
		float k=circle();
		System.out.println("Area of a circle is ="+k);
	}
	//callie method
	public static  float  circle()
	{
		Scanner sc=new Scanner(System.in);
		System.out.println("Enter the radius :");
		int r=sc.nextInt();

		//logic
		float area=3.14f*r*r;

		return area;
	}
}
```

4) With returntype with Argument method
==========

Q) Write a java program to perform sum of two numbers using with returntype with argument method?

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

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

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

		//caller method
		System.out.println("sum of two numbers is ="+sum(a,b));
	}
	//callie method
	public static int sum(int a,int b)
	{
		int c=a+b;

		return c;
	}
}
```

Q) Write a java program to find out given number is even or odd using with returntype with argument method?

approach1

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

		//caller method
		System.out.println(find(n));
	}
	//callie method
	public static String find(int n)
	{
		if(n%2==0)
			return "It is even number";
		else
			return "It is odd number";
	}
}
```

approach2

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

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

		//caller method
		int k=find(n);

		if(k==1)
			System.out.println("It is even number");
		else
			System.out.println("It is odd number");

	}
	//callie method
	public static int find(int n)
	{
		if(n%2==0)
			return 1;
		else
			return 0;
	}
}
```

approach3

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

		//caller method
		boolean b=find(n);

		if(b)
			System.out.println("It is even number");
		else
			System.out.println("It is odd number");

	}
	//callie method
	public static boolean find(int n)
	{
		if(n%2==0)
			return true;
		else
			return false;
	}
}
```

Assignment

```
============
```

Q) Write a java program to check given number is Armstrong or not?

Recursion

==========

A method which call itself for many number of times is called recursion.

It is similar to loopings.

Whenever we use recursion, we should not use loops.

Q) Write a java program to display 10 natural numbers without using loops?

```
import java.util.Scanner;
class Test
{
	public static void main(String[] args)
	{
		//caller method
		display(1);
	}
	//callie method
	public static void display(int i)
	{
		if(i<=10)
		{
			System.out.print(i+" ");
			display(i+1);
		}
	}
}
```

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

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

		//caller method
		System.out.println(factorial(n));
	}
	//callie method
	public static int factorial(int n)
	{
		if(n<0)
			return -1;
		if(n==0)
```

```
                return 1;

            return n*factorial(n-1);
      }
}
```

Q) Write a java program to find out Nth-element of fibonacci series ?

Fibonacci sequence : 0 1 1 2 3 5 8 . . . .

Input:
4

Output:
2

```
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(); // 4

            //caller method
            System.out.println(fib(n));
      }
      //Callie method
      public static int fib(int n)
      {
            if(n==0 || n==1)
                  return 0;
            if(n==2)
                  return 1;

            return fib(n-1)+fib(n-2);
      }
}
```

*Arrays

=========
Arrays 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 arrays are recommended 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.

In java, Arrays are divided into three types.

1) Single Dimensional Array

2) Double Dimensional / Two Dimensional Array

3) Multi Dimensional / Three Dimensional Array

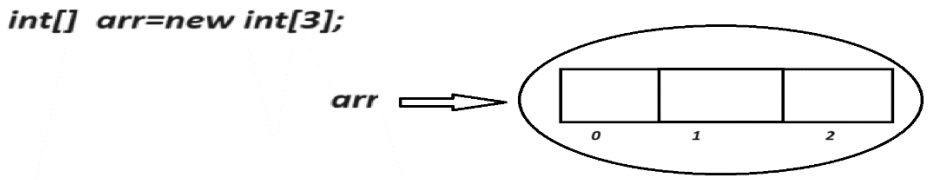
Array Declaration
==========
At the time of array declaration we should not specify array size.

Array

Single Dimensional Array	Double Dimensional Array	Multi Dimensional Array
int[] arr;	int[][] arr;	int[][][] arr;
int []arr;	int [][]arr;	int [][][]arr;
int arr[];	int arr[][];	int arr[][][];
	int[] []arr;	int[][] []arr;
	int[] arr[];	int[][] arr[];
	int []arr[];	int[] [][]arr;
		int[] arr[][];
		int[] []arr[];
		int [][]arr[];
		int []arr[][];

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

Diagram: java22.1



Rules to constructor an array

Rule1:

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

ex:

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

int[] arr=new int[]; // C.T.E Array Dimension 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 numbers as an array size otherwise we will get runtime exception called NegativeArraySizeException.

ex:

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

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]; //C.T.E
```

Rule5:

The maximum length we can take for array size is maximum length of int datatype.

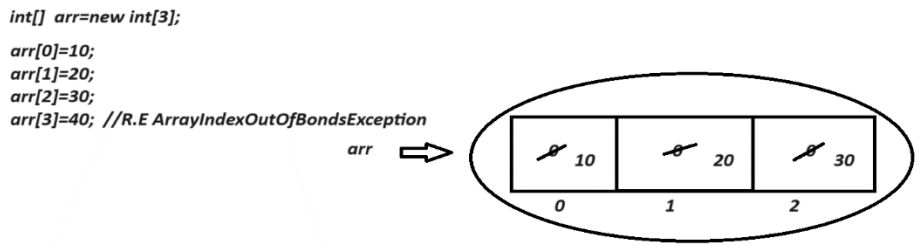
ex:

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

Array Initialization
====================
Once if we create an array , every array element initialized with default values.

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

Diagram: java22.2




```
arr[1]=20;
arr[2]=30;  ==> int[] arr={10,20,30};

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

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

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

length

It is a final variable which is applicable for arrays.

It will return size of an array.

ex:

```
class Test
{
      public static void main(String[] args)
      {
            int[] arr=new int[5];
            System.out.println(arr.length); //5
```

```
	}
}
```

length() method

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

It will return number of characters present in string.

ex:

```
class Test
{
	public static void main(String[] args)
	{
		String str="bhaskar";
		System.out.println(str.length()); //7
	}
}
```

Single Dimensional Array programs
==========

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

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

		System.out.println("Enter the array size :");
		int size=sc.nextInt(); // 5

		int[] arr=new int[size];

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

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

```
}
```

2) Write a java program to display array elements?

```
input:
      4 8 1 3 9 7


class Test
{
      public static void main(String[] args)
      {
            int[] arr={4,8,1,3,9,7};

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

```
for each loop
==============
For each loop is used to iterate the elements from arrays.

ex:

class Test
{
      public static void main(String[] args)
      {
            int[] arr={4,8,1,3,9,7};

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

3) Write a java program to display array elements in reverse order?

```
input:
      4 8 1 3 9 7

output:
      7 9 3 1 8 4

ex:

class Test
```

```
{
	public static void main(String[] args)
	{
		int[] arr={4,8,1,3,9,7};

		for(int i=arr.length-1;i>=0;i--)
		{
			System.out.print(arr[i]+" ");
		}
	}

}
```

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

```
input:
	4 8 1 3 9 7

output:
	32

ex:

class Test
{
	public static void main(String[] args)
	{
		int[] arr={4,8,1,3,9,7};

		int sum=0;
		for(int i:arr)
		{
			sum+=i;
		}

		System.out.println(sum);
	}

}
```

5) Write a java program to display even elements from given array?

```
input:
	4 8 1 3 9 7

output:
	4 8

ex:

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

```
            int[] arr={4,8,1,3,9,7};


            for(int i:arr)
            {
                  if(i%2==0)
                  {
                        System.out.print(i+" ");
                  }
            }
      }

}
```

6) Write a java program to display number of odd elements?

```
input:
      4 8 1 3 9 7
```

```
class Test
{
      public static void main(String[] args)
      {
            int[] arr={4,8,1,3,9,7};

            int cnt=0;

            for(int i:arr)
            {
                  if(i%2!=0)
                  {
                        cnt++;
                  }
            }

            System.out.println(cnt);
      }

}
```

7) Write a java program to display prime elements from given array?

```
input:
      4 2 9 5 7 12 13 20

output:
      2 5 7 13

ex:
---
class Test
{
```

```
	public static void main(String[] args)
	{
		int[] arr={4,2,9,5,7,12,13,20};

		for(int n:arr)
		{
			boolean flag=true;
			for(int i=2;i<=n/2;i++)
			{
				if(n%i==0)
				{
					flag=false;
					break;
				}
			}
			if(flag==true)
				System.out.print(n+" ");
		}
	}

}
```

Assignment

==========

8) Write a java program to display array elements in sorting order?

input:

```
	5 8 1 3 9 6
```

output:

```
	1 3 5 6 8 9
```

8) Write a java program to display array elements in sorting order?

input:

```
	9 1 5 2 7 6 4
```

output:

```
	1 2 4 5 6 7 9
```

ex:

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

		Arrays.sort(arr);
```

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

9) Write a java program to display array elements in sorting order without using sort() method?

```
input:
      9 1 5 2 7 6 4

output:
      1 2 4 5 6 7 9

ex:
---
class Test
{
      public static void main(String[] args)
      {
            int[] arr={9,1,5,2,7,6,4};

            //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:arr)
            {
                  System.out.print(i+" ");
            }
      }
}
```

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

```
input:
      9 1 5 2 7 6 4

output:
      9 7 6 5 4 2 1

ex:
```

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

		Arrays.sort(arr);//1 2 4 5 6 7 9

		//reading reverse
		for(int i=arr.length-1;i>=0;i--)
		{
			System.out.print(arr[i]+" ");
		}
	}
}
```

11) Write a java program to display array elements in descending order without using sort() method?

```
input:
	9 1 5 2 7 6 4

output:
	9 7 6 5 4 2 1

ex:
---

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

		//decending 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:arr)
		{
			System.out.print(i+" ");
		}

	}
```

```
}
```

12) Write a java program to find out highest element from given array?

```
input:
      1 5 2 7 9 6 4

output:
      9

ex:
----

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

            int big=arr[0];

            for(int i:arr)
            {
                  If(i>big)
                  {
                        big=i;
                  }
            }
            System.out.println(big);
      }
}
```

13) Write a java program to find out least element from given array?

```
input:
      5 2 7 1 9 6 4

output:
      1
+-
ex:
---

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

            int small=arr[0];

            for(int i:arr)
            {
                  if(i<small)
                  {
                        small=i;
```

```
                }
            }
            System.out.println(small);
      }
}
```

14) Write a java program to display three highest elements from given array?

```
input:
      4 8 2 9 6 1 7 5

output:
      9 8 7

ex:



class Test
{
      public static void main(String[] args)
      {
            int[] arr={4,8,2,9,6,1,7,5};

            int firstElement=Integer.MIN_VALUE;
            int secondElement=Integer.MIN_VALUE;
            int thirdElement=Integer.MIN_VALUE;

            for(int i:arr)
            {
                  if(i>firstElement)
                  {
                        thirdElement=secondElement;
                        secondElement=firstElement;
                        firstElement=i;
                  }
                  else if(i>secondElement)
                  {
                        thirdElement=secondElement;
                        secondElement=i;
                  }
                  else if(i>thirdElement)
                  {
                        thirdElement=i;
                  }
            }
            System.out.println(firstElement+" "+secondElement+"
"+thirdElement);
      }
}
```

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

```
input:
      5 7 1 3 9 4 1 3 7 6

output:
      7 1 3

ex:


class Test
{
      public static void main(String[] args)
      {
            int[] arr={5,7,1,3,9,4,1,3,7,6};

            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]+" ");
                        }
                  }
            }
      }
}
```

16) Write a java program to display unique elements from given array?

```
input:
      5 7 1 3 9 4 1 3 7 6

output:
      5 9 4 6

ex:

class Test
{
      public static void main(String[] args)
      {
            int[] arr={5,7,1,3,9,4,1,3,7,6};

            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]+" ");
            }
      }
```

```
}
```

17) Write a java program to display most repeating element from given array?

```
input:
      5 7 1 3 9 4 1 3 7 6 1 8 1

output:
      1 repeating for 4 times


ex:
---

class Test
{
      public static void main(String[] args)
      {a
            int[] arr={5,7,1,3,9,4,1,3,7,6,1,8,1};

            int maxCount=0;
            int element=0;

            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>maxCount)
                  {
                        maxCount=cnt;
                        element=arr[i];
                  }
            }

            System.out.println(element+" is repeating for "+maxCount+"
times" );
      }
}


Assignment
==========
```

18) Write a java program to display second highest element from given array without using sort() method?

```
input:
      5 8 1 9 6 2 7

output:
      8
```

19) Write a java program to segregate array elements?

```
input:
      0 1 1 0 0 1 0 1 0 1

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

ex:

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

            int[] newArr=new int[arr.length];

            //for each loop
            int j=0;
            for(int i:arr)
            {
                  if(i==0)
                  {
                        newArr[j++]=i;
                  }
            }

            //inserting 1
            while(j<arr.length)
            {
                  newArr[j++]=1;
            }

            //display new array
            for(int i:newArr)
            {
                  System.out.print(i+" ");
            }

      }
}
```

20) Write a java program to find out leader elements from given array?

```
input:
      5 2 34 7 16 3 9

output:
      9 16 34

ex:
---
class Test
{
```

```
	public static void main(String[] args)
	{
		int[] arr={5,2,34,7,16,3,9};

		int max=arr[arr.length-1];

		System.out.print(max+" ");

		for(int i=arr.length-2;i>=0;i--)
		{
			if(arr[i]>max)
			{
				max=arr[i];
				System.out.print(max+" ");
			}
		}


	}
}
```

21) Write a java program to display missing element from given array?

```
input:
	7 3 1 5 6 2

output:
	4


ex:
---
class Test
{
	public static void main(String[] args)
	{
		int[] arr={7,3,1,5,6,2};

		int sum_of_ele=arr.length+1;

		int sum=(sum_of_ele * (sum_of_ele + 1))/2;

		for(int i:arr)
		{
			sum-=i; //sum = sum - i;
		}

		System.out.println(sum);
	}
}
```

22) Write a java program to delete first occurance of a given element?

```
input:
```

```
      arr = 8 3 2 5 6 2 9 1 2

      element  = 2

output:
      8 3 5 6 2 9 1 2

ex:
---
class Test
{
      public static void main(String[] args)
      {
            int[] arr={8,3,2,5,6,2,9,1,2};
            int element=2;

            int[] newArr=new int[arr.length-1];

            int cnt=0,j=0;
            for(int i=0;i<arr.length;i++)
            {
                  if(arr[i]==element && cnt==0)
                  {
                        cnt=1;
                        continue;
                   }
                  newArr[j++]=arr[i];
            }

            //display the data
            for(int i:newArr)
            {
                  System.out.print(i+" ");
            }

      }
}
```

23) Write a java program to insert given element in a given index of array?

```
input:
      arr = 7 1 9 4 7 2

      ele = 10

      index = 3

output:
      7 1 9 10 4 7 2

ex:
---
import java.util.Arrays;
class Test
{
      public static void main(String[] args)
```

```
	{
		int[] arr={7,1,9,4,7,2};
		int ele=10;
		int index=3;

		arr=Arrays.copyOf(arr,arr.length+1);

		for(int i=arr.length-1;i>=index;i--)
		{
			arr[i]=arr[i-1];
		}

		arr[index]=ele;

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

24) Write a java program to merge two arrays and display them in sorting order?

```
input:
	5 1 4 3 2
	9 6 8 7 10

output:
	1 2 3 4 5 6 7 8 9 10

ex:

import java.util.Arrays;
class Test
{
	public static void main(String[] args)
	{
		int[] arr1={5,1,4,3,2};
		int[] arr2={9,6,8,7,10};

		int size1=arr1.length;
		int size2=arr2.length;

		arr1=Arrays.copyOf(arr1,size1+size2);

		int j=0;
		for(int i=size1;i<arr1.length;i++)
		{
			arr1[i]=arr2[j++];
		}

		Arrays.sort(arr1);

		//display elements
```

```
            for(int i:arr1)
            {
                  System.out.print(i+" ");
            }
      }
}
```

Q)

Write a java program to identify and print all elements in an array that are greater than both their immediate predecessors and successors, considering the first and last elements as having only one neighbor?

```
Input:
      1 3 20 4 75 0 90
Output:
      20 75 90
```

ex:

```
---
class Test
{
      public static void main(String[] args)
      {
            int[] arr={1,3,20,4,75,0,90};


            //first element
            if(arr[0]>arr[1])
            {
                  System.out.print(arr[0]+" ");
            }

            //Middle elements
            for(int i=1;i<arr.length-1;i++)
            {
                  if(arr[i]>arr[i-1] && arr[i]>arr[i+1])
                  {
                        System.out.print(arr[i]+" ");
                  }
            }


            //last element
            if(arr[arr.length-1]>arr[arr.length-2])
            {
                  System.out.println(arr[arr.length-1]);
            }
      }
}
```

Q)

Write a java program to determine the smallest number of coins needed to total
86 rupees. Use the denominations provided in the array {1,2,5,10}?

```
Output:
      1 coin(s) of 1 rupee(s)
      1 coin(s) of 5 rupee(s)
      8 coin(s) of 10 rupee(s)

ex:
---
class Test
{
      public static void main(String[] args)
      {
            int[] denominations={1,2,5,10};
            int amount=86;

            //caller method
            int[] coins=findMinimumCoins(denominations,amount);

            for(int i=0;i<coins.length;i++)
            {
                  if(coins[i]>0)
                  {
                        System.out.println(coins[i]+" coin(s) of
"+denominations[i]+" rupee(s)");
                  }
            }
      }
      //callie method
      public static int[] findMinimumCoins(int[] denominations,int
amount)
      {
            int[] coinsCount=new int[denominations.length];

            for(int i=denominations.length-1;i>=0;i--)
            {
                  coinsCount[i]= amount / denominations[i];

                  amount = amount % denominations[i];
            }

            return coinsCount;
      }
}
```

Double Dimensional Array
==========
Double dimensional array is implemented based on array of arrays approach but not matrix form.

Double dimensional array is a combination of rows and columns.

The main objective of double dimensional array is a memory utilization.

We can use double dimensional array is used to develop business oriented applications, gaming applications, matrix type of applications and etc.

We can declare double dimensional array as follow.

syntax:

```
datatype[][] variable_name=new datatype[rows][columns];
```

ex:

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

Here we can store 9(3*3) elements in array.

Q) Write a java program to display array elements in matrix form?

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

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

		System.out.println("Enter the columns :");
		int cols=sc.nextInt();

		int[][] arr=new int[rows][cols];

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

		//display elements
		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 find out square of a matrix?

```
input:
      1 2 3
      4 5 6
      7 8 9

output:
      1  4  9
      16 25 36
      49 64 81

ex:
----
class Test
{
      public static void main(String[] args)
      {
            int[][] arr={
                                    {1,2,3},
                                    {4,5,6},
                                    {7,8,9}
                              };

            int rows=arr.length;
            int cols=arr[0].length;

            //display square of a matrix
            for(int i=0;i<rows;i++)
            {
                  for(int j=0;j<cols;j++)
                  {
                        System.out.print(arr[i][j]*arr[i][j]+" ");
                  }
                  //new
                  System.out.println();
            }
      }
}
```

Q) Write a java program to find sum of diagonal elements?

```
input:
      1 2 3
      4 5 6
      7 8 9

class Test
{
      public static void main(String[] args)
      {
            int[][] arr={
                                    {1,2,3},
                                    {4,5,6},
                                    {7,8,9}
                              };

            int rows=arr.length;
            int cols=arr[0].length;
```

```
            //sum of diagonal elements
            int sum=0;
            for(int i=0;i<rows;i++)
            {
                  for(int j=0;j<cols;j++)
                  {
                        if(i==j)
                        {
                              sum+=arr[i][j];
                        }
                  }
            }
            System.out.println("sum of diagonal elements is ="+sum);
      }
}
```

Q) Write a java program to find sum of upper triangle elements?

```
input:
      1 2 3
      4 5 6
      7 8 9

ex:

class Test
{
      public static void main(String[] args)
      {
            int[][] arr={
                                    {1,2,3},
                                    {4,5,6},
                                    {7,8,9}
                              };

            int rows=arr.length;
            int cols=arr[0].length;

            //sum of diagonal elements
            int sum=0;
            for(int i=0;i<rows;i++)
            {
                  for(int j=0;j<cols;j++)
                  {
                        if(i<j)
                        {
                              sum+=arr[i][j];
                        }
                  }
            }
            System.out.println("sum of upper triangle elements is
="+sum);
      }
}
```

Q) **Write a java program to find sum of lower triangle elements?**

```
input:
      1 2 3
      4 5 6
      7 8 9

ex:

class Test
{
      public static void main(String[] args)
      {
            int[][] arr={
                                    {1,2,3},
                                    {4,5,6},
                                    {7,8,9}
                              };

            int rows=arr.length;
            int cols=arr[0].length;

            //sum of diagonal elements
            int sum=0;
            for(int i=0;i<rows;i++)
            {
                  for(int j=0;j<cols;j++)
                  {
                        if(i>j)
                        {
                              sum+=arr[i][j];
                        }
                  }
            }
            System.out.println("sum of lower triangle elements is
="+sum);
      }
}
```

Anonymous Array

==========

Sometimes we will declare an array without name such type of nameless array is called anonymous array.

The main objective of anonymous array is a just for instance use.

We can declare anonymous array as follow.

```
ex:
      new int[]{10,20,30};
      new int[][]{{10,20,30},{40,50,60}};

ex:
---
class Test
{
      public static void main(String[] args)
      {
            //caller method
```

```
            sum(new int[]{10,20,30});
      }
      //callie method
      public static void sum(int[] arr)
      {
            int sum=0;
            for(int i:arr)
            {
                  sum+=i;
            }
            System.out.println(sum);
      }
}
```

ex:

```
class Test
{
      public static void main(String[] args)
      {
            //caller method
            System.out.println(sum(new int[]{10,20,30}));
      }
      //callie method
      public static int sum(int[] arr)
      {
            int sum=0;
            for(int i:arr)
            {
                  sum+=i;
            }
            return sum;
      }
}
```

Interview Program
==========

Q) Write a java program to display largest prime number in the list?

Input:
      2 3 4 5 7 9 11 12
Output:
      11

```
class Test
{
      public static void main(String[] args)
      {
            int[] arr={2,3,4,5,7,9,11,12};

            int maxElement=Integer.MIN_VALUE;

            for(int n:arr)
            {
                  boolean flag=true;
                  for(int i=2;i<=n/2;i++)
                  {
```

```
                        if(n%i==0)
                        {
                              flag=false;
                              break;
                        }
                  }
                  if(flag==true)
                  {
                        if(n>maxElement)
                        {
                              maxElement=n;
                        }
                  }
            }

            System.out.println(maxElement);
      }

}
```

Q) Write a java program to display array elements in spiral form?

```
input:
      1 2 3
      4 5 6
      7 8 9

output:
      1 2 3 6 9 8 7 4 5

ex:
---
class Test
{
      public static void main(String[] args)
      {
            int[][] matrix={
                                    {1,2,3},
                                    {4,5,6},
                                    {7,8,9}
                              };
            int rows=matrix.length;
            int cols=matrix[0].length;

            int left=0;
            int right=rows-1;
            int top=0;
            int bottom=cols-1;

            while(true)
            {
                  if(left>right)
```

```
                {
                      break;
                }
                for(int i=left;i<=right;i++)
                {
                      System.out.print(matrix[top][i]+" ");
                }
                top++;

                if(top>bottom)
                {
                      break;
                }
                for(int i=top;i<=bottom;i++)
                {
                      System.out.print(matrix[i][right]+" ");
                }
                right--;


                if(left>right)
                {
                      break;
                }
                for(int i=right;i>=left;i--)
                {
                      System.out.print(matrix[bottom][i]+" ");
                }
                bottom--;

                if(top>bottom)
                {
                      break;
                }
                for(int i=bottom;i>=top;i--)
                {
                      System.out.print(matrix[i][left]+" ");
                }
                left++;
          }


     }
}
```

OOPS

======

OOPS stands for Object Oriented Programming System / Structure.

OOPS allows us to deal with real world entities using programming language.

A language is said to be object oriented if it supports following features.

ex:

- class
- object

Abstraction
Encapsulation
Inheritance
and
Polymorphism

class

======

A class is a blue print of an object.

A class is a collection of objects.

We can declare a class as follow.

syntax:

```
optional
   |
 Modifier  class  class_name <extends> Parent_classname
                        <implements> interface_name
{
      -
      -
      -
}
```

A class will accept following modifiers.

ex:

default
public
final
abstract

Realtime example

To constructor a building we required a design. That design is known as blue print/class.

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

default class	public class
If we declare a class without modifier is called default class.	If we declare a class with public modifier is called public class.

ex:
class Test
{
 	
}
	ex:
	public class Test
	{
	
	}

If we declare any class as default then we can access that class within the package.	If we declare any class as public then we can access that class within the package

and outside the package.

Q) What is the difference between final class and abstract class?

final class	abstract class
If we declare a class with final modifier is called final class.	If we declare a class with abstract is called abstract class.

ex:
final class Test
{
 	
}
	ex:
	abstract class Test
	{
	
	}

Child creation is not possible.	Child creation is possible.
We can create object.	We can't create object.

object

It is a outcome of a blue print.

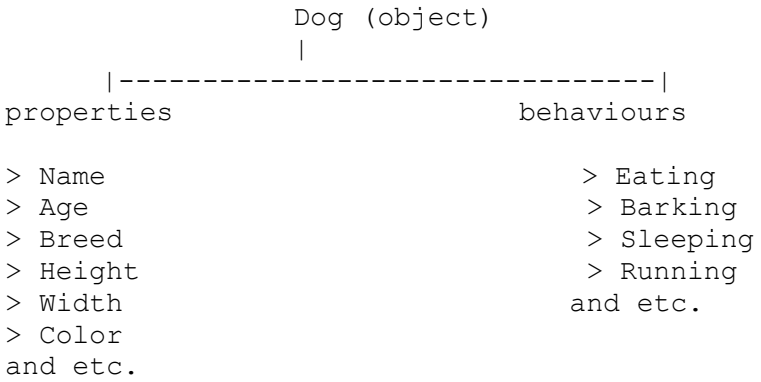
It is a instance of a class.

Here instance means allocating memory for our data members.

It is a physical entity.

It is a collection of properties and behaviours.

Realtime example



Memory space will be allocated when we create an object.

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

ex:

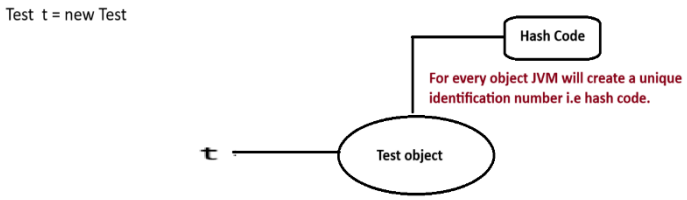
```
class Test
{
	public static void main(String[] args)
	{
		Test t1=new Test();
		Test t2=new Test();
		Test t3=new Test();

		System.out.println(t1.hashCode());
		System.out.println(t2.hashCode());
		System.out.println(t3.hashCode());

		System.out.println(t1); //Test@Hexadecimal
		System.out.println(t2.toString());
		System.out.println(t3.toString());
	}
}
```

Q) What is hash code in java?

For every object, JVM will create a unique identification number i.e hash code.

In order to read the hash code of an object we need to use hashCode() method of Object class.

Diagram: java27.1



Q) What is toString() method?

It is a method present in Object class.

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

Q) What is Object class?

Object class present in java.lang package.

It is a parent class for every java class.

What ever the properties are present in Object class that can be inherit by child classes.

Object class contains following methods.

ex:

```
cmd> javap   java.lang.Object

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

Q) What is the difference between a class and object ?

class	object
To declare a class we will use class keyword.	To declare an object we will use new .
It is a blueprint of an object.	It is a outcome of a blueprint.
It is a collection of objects.	It is a collection of properties & behaviourse.
It is a logical entity.	It is a physical entity.
It is declared once.	It is declared more then once.
It does not allocate the memory.	It allocates the memory.
We can't manipulate.	We can manipulate.

Data Hiding

==========

It is a technique of hiding object data from outsiders.

Using private modifier we can achieve data hiding.

The main objective of data hiding is to provide security.

ex:

```
class Account
{
      private double balance=10000d;
```

```
}
class Test
{
      public static void main(String[] args)
      {
            Account account=new Account();
            System.out.println(account.balance); // 10000
      }
}
```

Our internal data should not go out directly. It means outside person must not access our data directly.

Abstraction

==========

The process of hiding internal implementation and highlighting the set of services is called abstraction.

Using abstract classes and interfaces we can implements abstraction. The best example of abstraction is GUI ATM machine.

Where bank people will hide internal implementation and highlights the set of services like Banking, Withdrawl, MiniStatement and etc.

The main advantages of abstraction are

1) It gives security because it will hide internal implementation.

2) Enhancement becomes more easy because without effecting enduser they can perform many changes
   in our internal system.

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

4) It improves maintainability of an application.

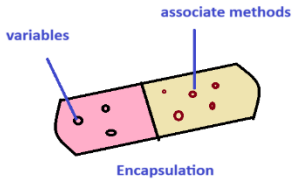
Ecapsulation

==========

The process of encapsulating or grouping variables and it's associate methods in a single entity is called encapsulation.

Encapsulation is used to protect the data.

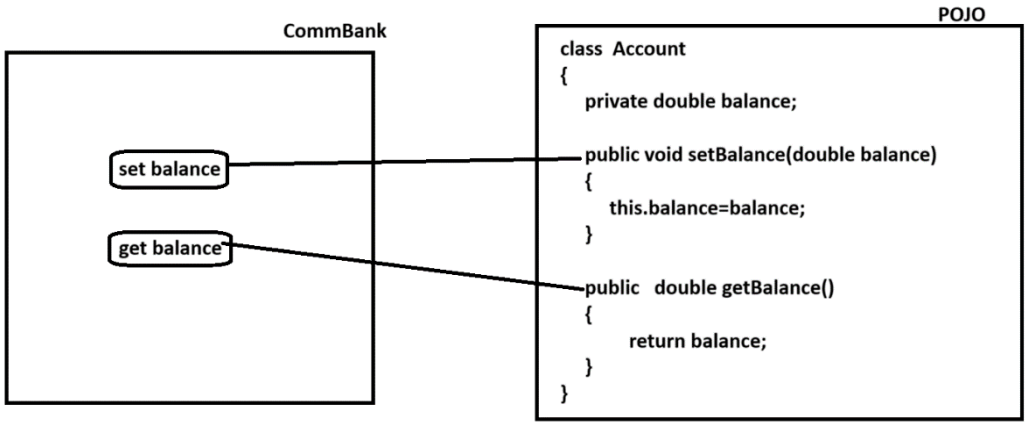
Diagram: java27.2



A class is said to be encapsulated class if it supports data hiding and abstraction.

In encapsulation for every variable we need to declare setter and getter methods.

Diagram: java27.3



2) Enhancement becomes more easy.

3) It provides flexibility to the end-user to use the system.

4) It improves maintainability of an application.

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

ex:

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

      //setter methods
      public void setStudId(int studId)
      {0
            this.studId=studId;
      }
      public void setStudName(String studName)
      {
            this.studName=studName;
      }
      public void setStudFee(double studFee)
      {
            this.studFee=studFee;
      }
```

```
	//getter methods
	public int getStudId()
	{
		return studId;
	}
	public String getStudName()
	{
		return studName;
	}
	public double getStudFee()
	{
		return studFee;
	}
}
class Test
{
	public static void main(String[] args)
	{
		Student s=new Student();
		s.setStudId(101);
		s.setStudName("Alan");
		s.setStudFee(1000d);

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

Q) What is tightly encapsulated?

A class is said to be tightly encapsulated if all the variables of that class must declare as private.

Here we don't need to check these variables contain setter and getter methods.

ex:

```
	class A
	{
		private int i;
	}
```

It is tightly encapsulated class.

ex:

```
	class A
	{
		int i;
		private int j;
	}
```

It is not tightly encapsulated class.

Is-A relationship
==========

Is-A relationship is also known as inheritance.

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

The main objective of Is-A relationship is to achieve reusability.

ex:

```
class Vehicle
{
      public void engine()
      {
            System.out.println("Engine Method");
      }
}
class Car  extends  Vehicle
{
      public void company()
      {
            System.out.println("Company Method");
      }
}
class Test
{
      public static void main(String[] args)
      {
            Vehicle v=new Vehicle();
            v.engine();

            Car c=new Car();
            c.engine();
            c.company();

            Vehicle v1=new Car();
            v1.engine();


            //Car c1=new Vehicle(); // invalid
      }
}
```

conclusion

Whatever our parent contains property it comes to child. But whatever child contains property never
goes back to parent.

A parent reference can hold child object but child reference can't hold parent object.

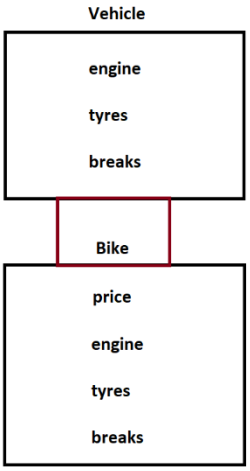
Inheritance
==========
Inheritance is a mechanism where we will derived a class in the presence of existing class.

Inheritance is a mechanism where one class will inherit the properties of another class.

Using "extends" keyword we can implements inheritance.

The main objective of inheritance is to achieve reusability.

Diagram: java28.1



2) Multi Level inheritance

3) Multiple inheritance

4) Hierarchical inheritance

5) Hybrid inheritance

1) Single Level inheritance

If we derived a class in the presence of one base class is called single level inheritance.

Diagram:

```
A (Parent/Base/Super class)
|
|
|
B (Child/Derived/Sub class)
```

ex:

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

```
	public void m2()
	{
		System.out.println("M2-Method");
	}
}
class Test
{
	public static void main(String[] args)
	{
		A a=new A();
		a.m1();

		B b=new B();
		b.m1();
		b.m2();
	}
}
```

```
ex:
---
class A
{
	int i=10;
}
class B extends A
{
	int j=20;
}
class Test
{
	public static void main(String[] args)
	{
		B b=new B();

		int c=b.i+b.j;
	    System.out.println(c); //30
	  }
}
```

2) Multi Level inheritance

If we derived a class in the presence of one base class and that class is derived from another base
class is called multi level inheritance.

Diagram:

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

ex:

```
class A
{
      public void m1()
      {
            System.out.println("M1-Method");
      }
}
class B extends A
{
      public void m2()
      {
            System.out.println("M2-Method");
      }
}
class C extends B
{
      public void m3()
      {
            System.out.println("M3-Method");
      }
}
class Test
{
      public static void main(String[] args)
      {
            A a=new A();
            a.m1();

            B b=new B();
            b.m1();
            b.m2();

            C c=new C();
            c.m1();
            c.m2();
            c.m3();
      }
}
```

3) Multiple inheritance

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

ex:

```
      class A
      {
      }
      class B
      {
      }
      class C extends A,B  --> invalid
      {
      }
```

But interface can extends more then one interface so we can achieve multiple inheritance concept

through interfaces.

```
ex: interface A
      {
      }
      interface B
      {
      }
      interface C extends A,B
      {
      }
```

If our class does not extends any other class then it is a direct child class of Object class.

```
ex:                          Diag:
      class A                      Object
      {                            |
                                   |
      }                            A
```

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

```
ex:                          Diag:
      class A                      Object
      {                            |
      }                            |
      class B extends A            A
      {                            |
      }                            |
                                   B
```

Java does not support cyclic inheritance.

```
ex:
      class A extends B
      {
      }
      class B extends A
       {
      }
```

Q) Why java does not support multiple inheritance?

There is a chance of raising ambiguity problem that's why java does not support multiple inheritance.

```
ex:
            P1.m1()                              P2.m1()
            |-------------------------------------------|
                              |
                              c.m1();
```

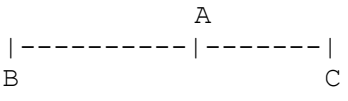
4) Hierarchical inheritance

If we derived multiple classes in the presence of one base class is called hierarchical inheritance.

ex:

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

Diagram:



ex:

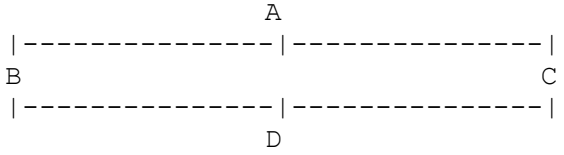
```
---
class A
{
	public void m1()
	{
		System.out.println("M1-Method");
	}
}
class B extends A
{
	public void m2()
	{
		System.out.println("M2-Method");
	}
}
class C extends A
{
	public void m3()
	{
		System.out.println("M3-Method");
	}
}
class Test
{
	public static void main(String[] args)
	{
		A a=new A();
		a.m1();

		B b=new B();
		b.m1();
		b.m2();

		C c=new C();
		c.m1();
		c.m3();
	}
}
```

5) Hybrid inheritance

Hybrid inheritance is a combination of more then one inheritance.

Java does not support hybrid inheritance.

Diagram:



Has-A relationship
==========
Has-A relationship is also known as composition and aggregation.

There is no specific keyword to implements Has-A relationship but mostly we will use new operator.

The main objective of Has-A relationship is to provide reusability.

Has-A relationship increase dependency between two components.

ex:

```
class  Course
{
      -
      - //course logic
      -
}
class Job
{
      Course c=new Course();
      -
      -
}
```

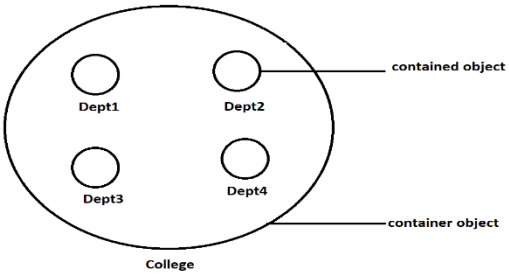
ex:

```
class Ihub
{
      public String courseName()
      {
            return "Full Stack Java";
      }
      public double courseFee()
      {
            return 30000d;
      }
      public String trainerName()
      {
            return "Niyaz Sir";
      }
}
class Usha
{
```

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

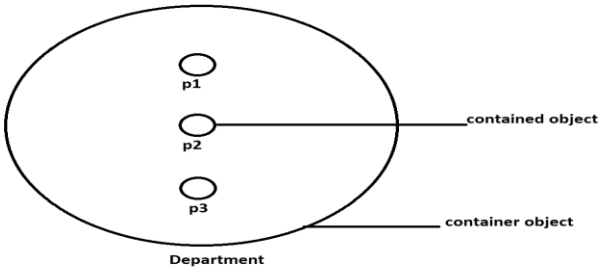
Composition
==========
Without existing container object there is no chance of having contained object then the relationship
between container and contained object is called composition which is strongly association.

Diagram: java28.2



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

Diagram: java28.3



Method overloading

==========

Having same method name with different parameters/signatures in a single class is called method
overloading.

Methods which are present in a class are called overloaded methods.

Method overloading will reduce complexity of the programming.

ex:

--

```
class MeeSeva
{
	//overloaded methods
	public void search(int voterId)
	{
		System.out.println("Details Found via voterId");
	}
	public void search(String houseNo)
	{
		System.out.println("Details Found via houseNo");
	}
	public void search(long aadharNo)
	{
		System.out.println("Details Found via aadharNo");
	}
}
class Customer
{
	public static void main(String[] args)
	{
		MeeSeva ms=new MeeSeva();
		ms.search(101);
		ms.search("1-6-4/1/A");
		ms.search(1024L);
	}
}
```

Q) Can we overload main method in java?

Yes, we can overload main method in java but JVM always execute main method with String[]
argument.

ex:

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

Method overriding

==========

Having same method name with same parameters in a two different class 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("Anushka");
	}
}
class Child extends Parent
{
	//overriding method
	public void marry()
	{
		System.out.println("Rashmika");
	}
}
class Test
{
	public static void main(String[] args)
	{
		Parent p=new Parent();
		p.property(); // cash+gold+land
		p.marry(); // Anushka

		Child c=new Child();
		c.property(); // cash+gold+land
		c.marry(); // Rashmika

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

```
            p1.marry(); // Rashmika

      }
}
```

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

ex:

```
class Parent
{
      public void property()
      {
            System.out.println("cash+gold+land");
      }
      //overridden method
      public final void marry()
      {
            System.out.println("Anushka");
      }
}
class Child extends Parent
{
      //overriding method
      public void marry()
      {
            System.out.println("Rashmika");
      }
}
class Test
{
      public static void main(String[] args)
      {
            Parent p=new Parent();
            p.property(); // cash+gold+land
            p.marry(); // Anushka

            Child c=new Child();
            c.property(); // cash+gold+land
            c.marry(); // Rashmika

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

      }
}
```

o/p:

```
      C.T.E : Overridden method is final
```

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

ex:

```
class Parent
{
```

```
	private void property()
	{
		System.out.println("House-Not For Sale");
	}
}
class Child extends Parent
{
	public void property()
	{
		System.out.println("House- For Sale");
	}
}
class Test
{
	public static void main(String[] args)
	{
		Parent p=new Parent();
		p.property();

		Child c=new Child();
		c.property();

	}
}
```

o/p:

C.T.E : property() has private access in Parent

Q) What is the difference method method overloading and method overriding?

Method overloading	**Method overriding**
Having same method name with different two different classes is called method overloading.	Having same method name with same parameters in method overriding.
Method resolution will taken care by a compiler based on reference type.	Method resolution will taken care by JVM based on runtime object.
Private and final methods can be overloaded.	Private and final methods can't be overridden.
It is also known as compile time polymorphism.	It is also known as runtime polymorphism.

Method Hiding

==========

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

Method overriding	**Method hiding**
Methods present in method overriding	Methods present in method hiding must be

must be non-static.	static.
Method resolution will taken care by a JVM based on runtime object.	Method resolution will taken care by a compiler based on reference type.
It is also known as runtime polymorhism, dynamic polymorphism or late binding.	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");
	}
	//overridden method
	public static void marry()
	{
		System.out.println("Anushka");
	}
}
class Child extends Parent
{
	//overriding method
	public static void marry()
	{
		System.out.println("Rashmika");
	}
}
class Test
{
	public static void main(String[] args)
	{
		Parent p=new Parent();
		p.property(); // cash+gold+land
		p.marry(); // Anushka

		Child c=new Child();
		c.property(); // cash+gold+land
		c.marry(); // Rashmika

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

	}
}
```

Q) Can we override main method in java?

No, we can't override main method in java because it is static.

Polymorphism

==========

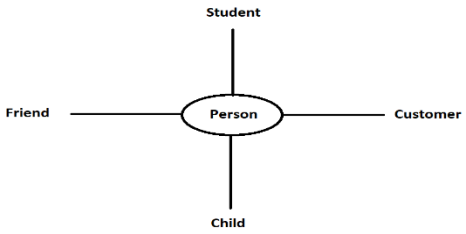
Polymorphism has taken from Greek.

Here poly means many and morphism means forms.

The ability to represent in different forms is called polymorphism.

The main objective of polymorphism is to provide flexibility.

Diagram: java29.1



1) Compile time polymorphism

A polymorphism which exhibits at compile time is called compile time polymorphism.

ex:

Method overloading
Method hiding

2) Runtime polymorphism

A polymorphism which exhibits at run time is called run time polymorphism.

ex:

Method overriding

Q) What is the difference between Abstraction and Encapsulation ?

Abstraction	Encapsulation
Hiding internal implementation and highlighting the set of services is called abstraction.	The process of encapsulating variables and it's associate methods in a single entity is called

encapsulation.

Using abstract classes and interfaces we can implements abstraction.	Using access modifiers we can implements Encapsulation.
It is used to hide the data.	It is used to protect the data.
It solves the issue at design level.	It solves the issue at implementation level.

Q) What is the difference between POJO class and Java Bean class?

POJO	Java Bean
It can't be serialized.	It can be serialized.
Fields can have any visibility.	Fields can have only private visibility.
There may or may not have 0-arg constructor.	It must have 0-argument constructor.
It does not extend any other class.	It can extends.
It does not implement any other interface.	It can implements.
It does not use any outside annotation.	It uses outside annotation.

ex:

POJO

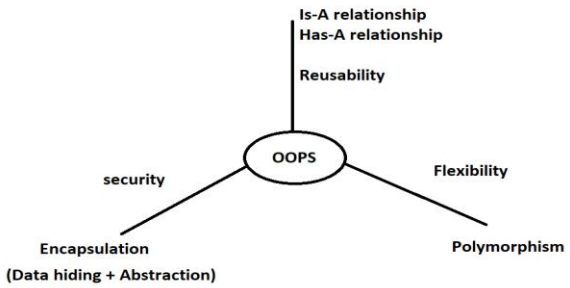
```
class Student
{
      private int studId;

      public void setStudId(int studId)
      {
            this.studId=studId;
      }
      public int getStudId()
      {
            return studId;
      }
}
```

Java Bean

```
class Student implements java.io.Serializable
{
      private int studId;

      Student()
```

```
	{

	}

	public void setStudId(int studId)
	{
		this.studId=studId;
	}
	public int getStudId()
	{
		return studId;
	}
}
```

Diagram: java29.2



constructors

==========

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

Having same name as class name is called constructor.

Constructor will call when we create an object.

A constructor will accept following modifiers.

ex:

- default
- public
- private
- protected

A constructor does not allow any returntype.

In java , constructors are divided into two types.

1) Userdefined constructor

2) Default constructor

1) Userdefined constructor

If a constructor is created by the user based on the application requirement is
called user defined constructor.

It is classified into two types.

i) Zero Argument constructor

ii) Parameterized constructor

i) Zero Argument constructor

Suppose if we are not passing any argument to userdefined constructor then that constructor
is called 0-arg constructor.

ex:

```
class Test
{
	Test()
	{
		System.out.println("0-arg const");
	}

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

ex:

```
class Test
{
	public 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:
---
class Test
{
      private 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

ex:
---
class Test
{
      protected 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 userdefined
constructor then that constructor
is called parameterized constructor.

ex:

class Employee
{
```

```
	//current class variables
	private int empId;
	private String empName;
	private double empSal;

	public Employee(int empId,String empName,double empSal)
	{
		this.empId=empId;
		this.empName=empName;
		this.empSal=empSal;
	}
	public void getEmployeeDetails()
	{
		System.out.println("Employee Id :"+empId);
		System.out.println("Employee Name :"+empName);
		System.out.println("Employee Salary :"+empSal);
	}
}
class Test
{
	public static void main(String[] args)
	{
		Employee e=new Employee(101,"Alan Morries",1000d);
		e.getEmployeeDetails();
	}
}
```

2) Default constructor

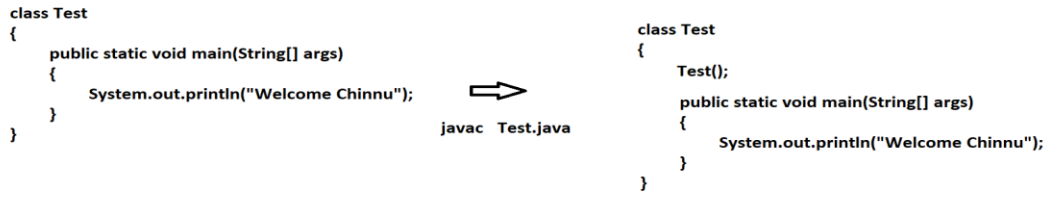
It is a compiler generated constructor for every java program where we are not defining
atleast zero argument constructor.

To see the default constructor we need to run below command.

ex:

```
	cmd> javac  Test.java
	cmd> javap  -c  Test
```

Diagram: java30.1



Q) What is 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.

1) To refer current class variables

2) To refer current class methods

3) To refer current class constructors

1) To refer current class variables

```
class A
{
      int i=10;
      int j=20;
      A(int i,int j)
```

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

2) To refer current class methods

```
---------------------------------
class A
{
	public void m1()
	{
		System.out.println("M1-Method");
		this.m2();
	}
	public void m2()
	{
		System.out.println("M2-Method");
 }
}
class Test
{
	public static void main(String[] args)
	{
		A a=new A();
		a.m1();
	}
}
```

3) To refer current class constructors

```
-----------------------------------
class A
{
	A()
	{
		System.out.println("0-arg const");
	}
	A(int i)
	{
		this();
		System.out.println("int-arg const");
	}
	A(double d)
	{
		this(10);
		System.out.println("double-arg const");
	}
}
class Test
{
```

```
	public static void main(String[] args)
	{
		A a=new A(10.5d);
	}
}
```

super keyword

==========

A super keyword is a java keyword which is used to refer super class object reference.

We can utilize super keyword in following ways.

1) To refer super class variables

2) To refer super class methods

3) To refer super class constructors

1) To refer super class variables

```
class A
{
	int i=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);
	}
}
```

2) To refer super class methods

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

```
	public void m2()
	{
		super.m1();
		System.out.println("M2-Method");
	}
}
class Test
{
	public static void main(String[] args)
	{
		B b=new B();
		b.m2();
	}
}
```

3) To refer super class constructors

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

}
```

Has-A relationship example

```
class Customer
{
	private int custId;
	private String custName;
	private Address address;
	public Customer(int custId,String custName,Address address)
	{
		this.custId=custId;
		this.custName=custName;
		this.address=address;
	}
	public int getCustId()
	{
		return custId;
```

```
	}
	public String getCustName()
	{
		return custName;
	}
	public Address getAddress()
	{
		return address;
	}
}
class Address
{
	private String houseNo;
	private String locality;
	private String city;
	public Address(String houseNo,String locality,String city)
	{
		this.houseNo=houseNo;
		this.locality=locality;
		this.city=city;
	}
	public String getHouseNo()
	{
		return houseNo;
	}
	public String getLocality()
	{
		return locality;
	}
	public String getCity()
	{
		return city;
	}

	public String toString()
	{
		return houseNo+" "+locality+" "+city;
	}
}
class Test
{
	public static void main(String[] args)
	{
		Address add=new Address("1-465/1/A","Ameerpet","Hyderabad");

		Customer cust=new Customer(101,"Alan",add);

		System.out.println(cust.getCustId());
		System.out.println(cust.getCustName());
		System.out.println(cust.getAddress());

	}
}
```

Interface

==========

Interface is a collection of zero or more abstract methods.

Abstract methods are incomplete methods because they ends with semicolon and does not have any body.

ex:

```
void m1();
```

By default every abstract method is a public and abstract.

ex:

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

It is not possible to create object for interfaces.

To write the implementation 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.

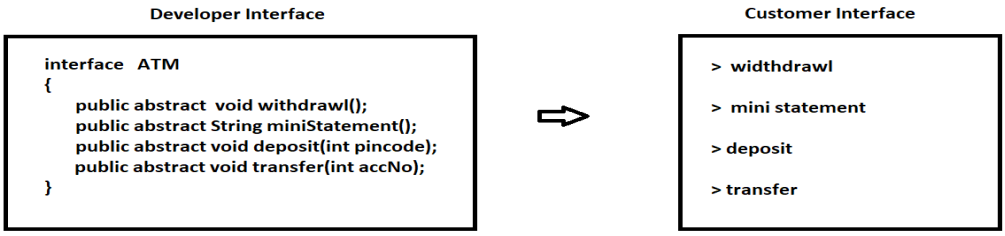
Interface contains constants i.e public static final.

syntax:

```
interface  <interface_name>
{
      - //abstract methods
      - //constants
}
```

If we know service requirement specification then we need to use interface.

Diagram: java31.1



ex:

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

ex:

```
interface A
{
	public abstract void m1();
}

class Test
{
	public static void main(String[] args)
	{
		A a=new A()
		{
			public void m1()
			{
				System.out.println("From M1 Method");
			}
		};
		a.m1();
	}
}
```

If interface contains four methods then we need to override all four methods.

ex:

```
interface A
{
	public abstract void see();
	public void show();
	abstract void view();
	void display();
}
class B implements A
{
	public void see()
	{
		System.out.println("See Method");
	}
	public void show()
	{
		System.out.println("Show Method");
	}
```

```
	public void view()
	{
		System.out.println("View Method");
	}
	public void display()
	{
		System.out.println("Display Method");
	}
}
class Test
{
	public static void main(String[] args)
	{
		A a =new B();

		a.show();
		a.see();
		a.view();
		a.display();
	}
}
```

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

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)
	{
```

```
            C c=new D();
            c.m1();
            c.m2();
            c.m3();
      }
}
```

A class can implements more then one interface.

ex:

```
interface Father
{
      float HT=6.2f;
      void height();
}
interface Mother
{
      float HT=5.8f;
      void height();
}
class Child implements Father,Mother
{
      public void height()
      {
            float height=(Father.HT+Mother.HT)/2;
            System.out.println("Child Height :"+height);
      }
}
class Test
{
      public static void main(String[] args)
      {
            Child c=new Child();
            c.height();
      }
}
```

Note:

From Java 8 version, Interface is a collection of abstract methods, default methods and static methods.

Q) What is marker interface?

Interface which does not contains any methods or constants is called marker interface

In general , Empty interface is called marker interface.

By using marker interface we will get some ability to do.

We have following list of marker interfaces.

ex:
- Serializable
- Cloneable

```
      Remote
      and etc.

ex:
---
class Item implements java.io.Serializable
{
      private int itemId;
      private String itemName;
      private double itemPrice;

      Item(int itemId,String itemName,doule itemPrice)
      {
            this.itemId=itemId;
            this.itemName=itemName;
            this.itemPrice=itemPrice;
      }
      public int getItemId()
      {
            return itemId;
      }
      public String getItemName()
      {
            return itemName;
      }
      public double getItemPrice()
      {
            return itemPrice;
      }
}
```

Abstract class

==========

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

A abstract keyword is applicable for methods and classes but not for variables.

It is not possible to create object for abstract classes.

To write the implementation of 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
      {
            -
            - // abstract methods
            - // concrete methods
            - // instance variables
      }
```

If we know partial implementation then we need to use abstract class.

ex:

```
abstract class Plan
{
	//instance variable
	protected double rate;

	//abstract method
	public abstract void getRate();

	//concrete method
	public void calculateBillAmt(int units)
	{
		System.out.println("Total Units :"+units);
		System.out.println("Total Bill :"+rate*units);
	}
}
class DomesticPlan extends Plan
{
	public void getRate()
	{
		rate=2.5d;
	}
}
class CommercialPlan extends Plan
{
	public void getRate()
	{
		rate=5.0d;
	}
}
class Test
{
	public static void main(String[] args)
	{
		DomesticPlan dp=new DomesticPlan();
		dp.getRate();
		dp.calculateBillAmt(250);

		CommercialPlan cp=new CommercialPlan();
		cp.getRate();
		cp.calculateBillAmt(250);
	}
}
```

Abstraction example
==========

```
abstract class Shape
{
	public abstract void draw();
}
class Rectangle extends Shape
{
```

```
	public void draw()
	{
		System.out.println("I draw rectangle");
	}
}
class Test
{
	public static void main(String[] args)
	{
		Rectangle rect=new Rectangle();
		rect.draw();
	}
}
```

Q) What is the difference between interface and abstract class?

interface	abstract class
To declare interface we will use interface keyword.	To declare abstract class we will use abstract keyword.
It is a collection of abstract methods, default methods and static methods.	It is a collection of abstract methods and concrete methods.
Multiple inheritence is possible.	Multiple inheritance is not possible.
It contains constants.	It contains instance variables.
It does not allow constructors.	It allows constructors.
It does not allow blocks.	It allows blocks.
To write the implementation of abstract methods we will use implementation class.	To write the implementation of abstract methods we will use sub class.
If we know only specification then we need to use interface.	If we know partial implementation then we need to use abstract class.

API
=====
API stands for Application Programming Interface.

It is a base for the programmer to develop software applications.

It is a collection of packages.

In java, We have three types of API's.

1) Predefined API

Built-In API is called predefined API.
ex:
https://docs.oracle.com/javase/8/docs/api/

2) User-defined API

API created by the user based on the application requirements.

3) Third party API

API which is given by third party vendor.
ex:
JAVAZOOM API
iText API and etc.

Package
===
A package is a collection of classes, interfaces , enums and annotations.

Here enum is a special class and annotation is a special interface.

In general, a package is a collection of classes and interfaces.

A package is also known as folder or a directory.

In java, we have two types of packages.

1) Predefined packages

2) Userdefined packages

1) Predefined packages

Built-In packages are called predefined packages.
ex:
java.lang
java.io
java.util
java.time
java.util.stream
java.text
java.sql
javax.servlet
and etc.

2) Userdefined packages

Packages which are created by the user based on the application requirements are called userdefined packages.

We can declare userdefined package as follow.

syntax:

```
      package   package_name;
```

It is always recommended to declare a package name in the reverse order of url.

ex:

```
      package  com.google.www;
```

ex:

```
---
package com.ihub.www;
import java.util.Calendar;
class Test
{
      public static void main(String[] args)
      {
                  Calendar c=Calendar.getInstance();
                  int h=c.get(Calendar.HOUR_OF_DAY);

                  if(h<12)
                        System.out.println("Good Morning");
                  else if(h<16)
                        System.out.println("Good Afternoon");
                  else if(h<20)
                        System.out.println("Good Evening");
                  else
                        System.out.println("Good Night");
      }
}
```

We can compile above program by using below command.

ex:

```
            current directory
                     |
      javac   -d     .    Test.java
               |
         destination folder
```

We can run above program by using below command.

ex:

```
      java    com.ihub.www.Test
                  |     |
            packagename  classname
```

Singleton class

==========

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

It is a design pattern that ensures that a class can only have one object.

If we call any method by using class name and that method returns same class object is called singleton class.

ex:

```
Calendar c=Calendar.getInstance();
LocalDate d=LocalDate.now();
LocalTime t=LocalTime.now();
```

To create a singleton class, we required private constructor and static method.

ex:

```
---
class Singleton
{
      static Singleton  singleton=null;

      //private constructor
      private Singleton()
      {
      }

      //static method
      public static Singleton getInstance()
      {
            if(singleton==null)
            {
                  singleton=new Singleton();
            }

            return singleton;
      }
}
class Test
{
      public static void main(String[] args)
      {
            Singleton s1=Singleton.getInstance();
            System.out.println(s1.hashCode());

            Singleton s2=Singleton.getInstance();
            System.out.println(s2.hashCode());
      }
}
```

Inner classes

==========

Sometimes we will declare a class inside another class such concept is called inner class.

ex:

```
      class Outer
      {
            class Inner
            {
                  -
                  - //code to be execute
                  -
```

```
            }
      }
```

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
inner classes in our regular programming.

Accessing inner class data from static area of outer class

```
class Outer
{
      class Inner
      {
            //non-static method
            public void m1()
            {
                  System.out.println("Inner-M1 Method");
            }
      }

      public static void main(String[] args)
      {
            Outer.Inner i=new Outer().new Inner();
            i.m1();
      }

}
```

Note:

If we compile above program we will get two .class files i.e Outer.class and Outer$Inner.class.

ex:

```
class Outer
{
      class Inner
      {
            //non-static method
            public void m1()
            {
                  System.out.println("Inner-M1 Method");
            }
      }

      public static void main(String[] args)
      {
            new Outer().new Inner().m1();
      }

}
```

Inner class does not allow static declaration.

ex:

```
class Outer
{
	class Inner
	{
		//static method
		public static void m1()
		{
			System.out.println("Inner-M1 Method");
		}
	}

	public static void main(String[] args)
	{
		new Outer().new Inner().m1();
	}
}
```
o/p:
	C.T.E : Illegal static declaration in inner class

Accessing inner class data from non-static area of outer class
---
```
class Outer
{
	class Inner
	{
		//non-static method
		public void m1()
		{
			System.out.println("Inner-M1 Method");
		}
	}

	public void m2()
	{
		Inner i=new Inner();
		i.m1();
	}

	public static void main(String[] args)
	{
		Outer o=new Outer();
		o.m2();
	}

}
```

Enum
======
Enum concept introduced in 1.5v.

Enum is a group of named constants.

Using enum we can create our own datatype called enumerated datatype.

When compare to old language enum, java enum is more powerful.

We can declare enum as follow.

syntax:

```
enum   enumtype_name
{
      value1,value2,.....,valueN
}
```

ex:

```
enum Months
{
      JAN,FEB,MAR
}
```

Internal implementation of enum

Every enum internally consider as class concept and extended with java.lang.Enum class.

Every enum constant is reference variable of enum type.

ex:

```
enum  Months                  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.MAR;

            switch(m)
            {
                  case JAN: System.out.println("January"); break;
                  case FEB: System.out.println("February"); break;
                  case MAR: System.out.println("March"); break;
            }
      }
}
```

java.lang.Enum

The power to enum will be inherited from java.lang.Enum class.

It contains following two methods.

1) values()

It will return group of constants from 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 Week
{
      MON,TUE,WED,THU,FRI,SAT,SUN;

      Week()
      {
            System.out.println("constructor");
      }
}
class Test
{
      public static void main(String[] args)
      {
            Week w=Week.TUE;
      }
}
```

ex:

```
enum Drinks
{
      COLA,CAMPA,PEPSI;

      static int i=100;

      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 several utility methods.

ex:

primitive type	wrapper class
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
boolean	Boolean
char	Character

constructor

There are two ways to create object for wrapper classes. One will take corresponding primitive as an argument and another takes corresponding String as an argument.

ex:

Wrapper class	constructor
Byte	byte or String
Short	short or String
Integer	int or String
Long	long or String
Float	float or String
Double	double or String
Boolean	boolean or String
Character	char

ex:

```
class Test
{
	public static void main(String[] args)
	{
		Integer i1=new Integer(10);
		System.out.println(i1);

		Integer i2=new Integer("20");
		System.out.println(i2);
	}
}
```

ex:

```
class Test
{
	public static void main(String[] args)
	{
		Boolean b1=new Boolean(true);
		System.out.println(b1);

		Boolean b2=new Boolean("false");
		System.out.println(b2);
	}
}
```

ex:

```
class Test
{
```

```
	public static void main(String[] args)
	{
		Character ch=new Character('a');
		System.out.println(ch); //a
	}
}
```

Utility methods
==========

1) parseXxx()

It is used to convert string type to primitive type.

ex:
--
```
class Test
{
	public static void main(String[] args)
	{
		String str="23";

		int i=Integer.parseInt(str);
		System.out.println(i); //23

		long l=Long.parseLong(str);
		System.out.println(l); //23

		float f=Float.parseFloat(str);
		System.out.println(f); //23.0

		double d=Double.parseDouble(str);
		System.out.println(d); //23.0
	}
}
```

2) toString()

It is used to convert wrapper object to String type.

ex:

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

		String str= i.toString();

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

ex:

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

		String str= i.toString();

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

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

```
input:
	1010
	0101
output:
	1111
```

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 binary : ");
		String binary1=sc.next(); //1010

		System.out.println("Enter the second binary :");
		String binary2=sc.next(); //0101

		//convert binary to decimal
		int a=Integer.parseInt(binary1,2);
		int b=Integer.parseInt(binary2,2);

		int c=a+b;

		//convert decimal to binary
		String result=Integer.toBinaryString(c);
		System.out.println(result);
	}
}
```

ex:

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

```
	{
		char ch='a';

		String s=Character.toString(ch);

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

Types of objects in java
==========
We have two types of objects in java.

1) Immutable object

2) Mutable object

1) Immutable object

After object creation if we perform any changes then for our 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 then all the required changes will be reflected to same object only such type of object is called mutable object.
ex:
	StringBuffer and StringBuilder

String
========
It is a collection of characters which is enclosed in a double quotation.

case1:

Once if we create a String object we can't perform any changes.If we perform any changes then for change a new object will be created such behaviour is called immutability of an object.

Diagram: java34.1

case2:

What is the difference between == and .equals() method?

==

It is a comparision operator which always return boolean value.

It is used for reference comparision or address comparision.

We can compare primitives and objects.

ex:

```
class  Test
{
      public static void main(String[] args)
      {
            String s1=new String("ihub");
            String s2=new String("ihub");
            System.out.println(s1==s2);//false
      }
}
```

.equals()

It is a method present in String class which always returns boolean value.

It is used for content comparision and it is case sensitive.

We can't compare primitives.

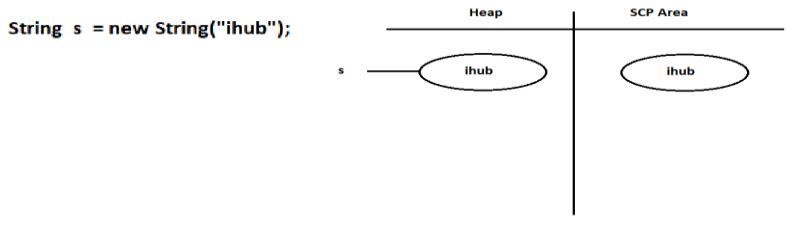
ex:

```
class  Test
{
      public static void main(String[] args)
      {
            String s1=new String("ihub");
            String s2=new String("ihub");
            System.out.println(s1.equals(s2));//true
      }
}
```

case3:

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.

Diagram: java34.2



Object creation in SCP area is always optional.

First JVM will check is there object is created with same content or not. If it is created then it simply refers to that object. If it is not created then JVM will create a new object. Hence there is no chance of having duplicate objects in SCP area.

Even though SCP area objects do not have any object reference , garbage collection can't access them.

Diagram: java34.3

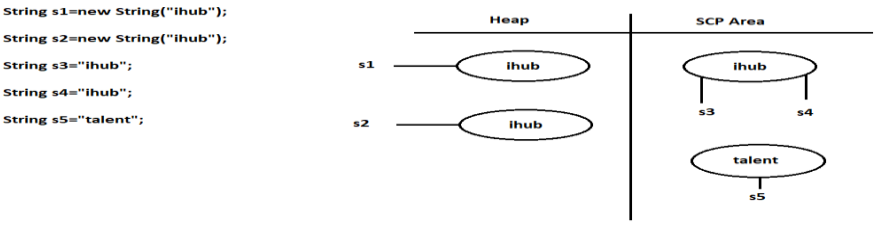
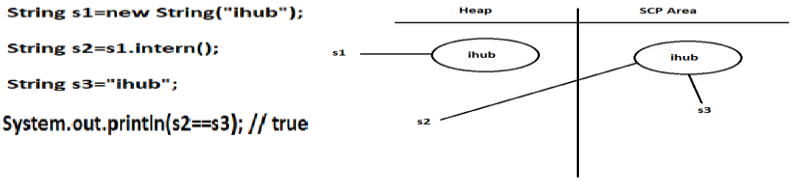


Diagram: java34.4



String important methods
==========

Q) Write a java program to display length of the string?

input:
hello

output:
5

ex:

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

```
	{
		String str="hello";

		System.out.println(str.length());//5
	}
}
```

Q) Write a java program to display the string character by character ?

```
input:
	ihub

output:
	i
	h
	u
	b

ex:

class  Test
{
	public static void main(String[] args)
	{
		String str="ihub";

		//reading characters
		for(int i=0;i<str.length();i++)
		{
			char ch=str.charAt(i);
			System.out.println(ch);
		}

	}
}
```

Q) Write a java program to convert lowercase string to uppercase ?

```
input:
	ihubtalent

output:
	IHUBTALENT

ex:

class  Test
{
	public static void main(String[] args)
	{
		String str="ihubtalent";

		str=str.toUpperCase();

		System.out.println(str);

	}
```

```
}
```

Q) Write a java program to convert uppercase string to lowercase ?

```
input:
      IHUBTALENT

output:
      ihubtalent

ex:
---
class  Test
{
      public static void main(String[] args)
      {
            String str="IHUBTALENT";

            str=str.toLowerCase();

            System.out.println(str);

      }
}
```

Q) Write a java program to concatinate two strings?

```
input:
      ihub
      talent

output:
      ihubtalent

ex:
---
class  Test
{
      public static void main(String[] args)
      {
            String str1="ihub";
            String str2="talent";
            String result=str1.concat(str2);
            System.out.println(result);

      }
}
```

Q) Write a java program to check given strings is equal or not?

```
input:
      ihubtalent
      qualitythought

output:
      Both are not equals
```

ex:

```
class  Test
{
      public static void main(String[] args)
      {
            String str1="ihubtalent";
            String str2="qualitythought";

            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 index of first occurance of a given character in a string?

input:
      str = bhaskar
      ch = 'a'

output:
      2

ex:

```
class  Test
{
      public static void main(String[] args)
      {
            String str="bhaskar";
            char ch='a';
            int index=str.indexOf(ch);
            System.out.println(index);
      }
}
```

Q) Write a java program to display last index of a given character in a string?

input:
      str = bhaskar
      ch = 'a'
output:
      5

ex:

```
class  Test
{
      public static void main(String[] args)
      {
            String str="bhaskar";
            char ch='a';
            int index=str.lastIndexOf(ch);
```

```
            System.out.println(index);
      }
}
```

Q) Write a java program to remove special characters from given string?

```
input:
      I_hu@bTa$len#t12

output:
      IhubTalent12

ex:
----
class  Test
{
      public static void main(String[] args)
      {
            String str="I_hu@bTa$len#t12";

            str = str.replaceAll("[^A-Za-z0-9]","");

            System.out.println(str);
      }
}
```

Q) Write a java program to display special characters from given string?

```
input:
      I_hu@bTa$len#t12

output:
      _@$#

ex:

class  Test
{
      public static void main(String[] args)
      {
            String str="I_hu@bTa$len#t12";

            str = str.replaceAll("[A-Za-z0-9]","");

            System.out.println(str);
      }
}
```

Q) Write a java program to concatinate two strings?

```
input:
      ihub23
      talent17

output:
      ihubtalent40
```

ex:

```
class  Test
{
      public static void main(String[] args)
      {
            String str1="ihub23";
            String str2="talent17";


            String word1=str1.replaceAll("[^A-Za-z]","");
            int num1=Integer.parseInt(str1.replaceAll("[^0-9]",""));

            String word2=str2.replaceAll("[^A-Za-z]","");
            int num2=Integer.parseInt(str2.replaceAll("[^0-9]",""));

            String word=word1+word2;
            int num=num1+num2;

            System.out.println(word+num);
      }
}
```

Q) Write a java program to return substring?

input:
      ihubtalent

output:
      talent

ex:

```
class Test
{
      public static void main(String[] args)
      {
            String str="ihubtalent";

            String newStr=str.substring(4);

            System.out.println(newStr);
      }
}
```

Q) Write a java program to return substring?

input:
      ihubtalent

output:
      hub

ex:

```
class Test
{
	public static void main(String[] args)
	{
		String str="ihubtalent";

		String newStr=str.substring(1,4);

		System.out.println(newStr);
	}
}
```

Q) Write a java program to insert given word in a given index of a string?

```
input:
	str = javaisindependent

	word = platform

	index = 6

output:
	javaisplatformindependent

ex:
---

class Test
{
	public static void main(String[] args)
	{
		String str ="javaisindependent";
		String word ="platform";
		int index = 6;


		String word1=str.substring(0,index);

		String word2=str.substring(index,str.length());

		System.out.println(word1+word+word2);
	}
}
```

Q) Write a java program to perform right rotation of a given string?

```
input:
	str = ihubtalent

	cnt = 2

output:
	ubtalentih

ex:
--
```

```
class Test
{
	public static void main(String[] args)
	{
		String str ="ihubtalent";
		int cnt=2;

		String word1=str.substring(cnt,str.length());
		String word2=str.substring(0,cnt);

		System.out.println(word1+word2);

	}
}
```

Q) Write a java program to display reverse of a given string?

```
input:
	hello

output:
	olleh

ex:

class Test
{
	public static void main(String[] args)
	{
		String str="hello";   // h  e  l  l  o


		String rev="";

		for(int i=str.length()-1;i>=0;i--)
		{
			rev+=str.charAt(i);
		}

		System.out.println(rev);
	}
}


approach2
----------
class Test
{
	public static void main(String[] args)
	{
		String str="hello";

		char[] carr=str.toCharArray(); // h  e  l  l  o

		String rev="";

		for(int i=carr.length-1;i>=0;i--)
		{
```

```
                rev+=carr[i];
            }

            System.out.println(rev);
      }
}
```

Q) Write a java program to check given string is palindrome or not?

```
input:
      racar

output:
      It is a palindrome string

ex:
--

class Test
{
      public static void main(String[] args)
      {
            String str="racar";

            char[] carr=str.toCharArray(); // r a  c  a r

            String rev="";

            for(int i=carr.length-1;i>=0;i--)
            {
                  rev+=carr[i];
            }

            if(str.equals(rev))
                  System.out.println("It is a palindrome string");
            else
                  System.out.println("It is not a palindrome string");
      }
}
```

Q) Write a java program to display reverse of a sentence?

```
input:
      This is java class

output:
      class java is This

ex:

class Test
{
      public static void main(String[] args)
      {
            String str="This is java class";

            String[] sarr=str.split(" "); // This  is   java   class
```

```
            String rev="";

            //reading reverse
            for(int i=sarr.length-1;i>=0;i--)
            {
                  rev+=sarr[i]+" ";
            }

            System.out.println(rev);
      }
}
```

Q) Write a java program to display reverse of a word in a given string?

```
input:
      This is java class

output:
      sihT si avaj ssalC
```

```
ex:
----
class Test
{
      public static void main(String[] args)
      {
            String str="This is java class";

            String[] sarr=str.split(" ");// This    is   java    class

            String rev="";

            //for each loop
            for(String s:sarr)
            {
                  char[] carr=s.toCharArray(); // T    h    i    s

                  //reading reverse
                  for(int i=carr.length-1;i>=0;i--)
                  {
                        rev+=carr[i];
                  }
                  //add space
                  rev+=" ";
            }

            System.out.println(rev);
      }
}
```

Q) Write a java program to display duplicate characters from given string?

```
input:
```

```
      google

output:
      og

ex:
--
class Test
{
      public static void main(String[] args)
      {
            String str="google";

            String duplicates="";
            String uniques="";

            for(int i=0;i<str.length();i++)
            {
                  String current=Character.toString(str.charAt(i));

                  if(uniques.contains(current))
                  {
                        if(!duplicates.contains(current))
                        {
                              duplicates+=current;
                              continue;
                        }
                  }
                  uniques+=current;
            }
            System.out.println(duplicates);
      }
}
```

Q) Write a java program to display unique characters from given string?

```
input:
      google

output:
      gole

ex:
---
class Test
{
      public static void main(String[] args)
      {
            String str="google";

            String duplicates="";
            String uniques="";

            for(int i=0;i<str.length();i++)
            {
                  String current=Character.toString(str.charAt(i));
```

```
                if(uniques.contains(current))
                {
                      if(!duplicates.contains(current))
                      {
                            duplicates+=current;
                            continue;
                      }
                }
                uniques+=current;
          }
          System.out.println(uniques);
     }
}
```

Q) Write a java program to display most repeating character in a given string?

```
input:
     ihubtalentinstitute

output:
     t is repeating for 5 times


ex:
---

class Test
{
     public static void main(String[] args)
     {
          String str="ihubtalentinstitute";

          int maxCount=0;
          char element=' ';

          for(int i=0;i<str.length();i++)
          {
                int cnt=0;

                for(int j=0;j<str.length();j++)
                {
                      if(str.charAt(i) == str.charAt(j))
                      {
                            cnt++;
                      }
                }

                if(maxCount<cnt)
                {
                      maxCount=cnt;
                      element=str.charAt(i);
                }
          }
          System.out.println(element+" is repeating for "+maxCount+"
times");

     }
```

```
}
```

Q) Write a java program to remove spaces from given string?

```
input:
      i hub tale nt

output:
      ihubtalent

ex:

class Test
{
      public static void main(String[] args)
      {
            String str="i hub tale nt";

            str=str.replaceAll("\\s","");

            System.out.println(str);
      }
}
```

Q) Write a java program to display the string in a given format?

```
input:
      str = This is java class

      delete = is

output:
      Th java class

ex:

class Test
{
      public static void main(String[] args)
      {
            String str ="This is java class";
            String delete ="is";

            str = str.replaceAll(delete,"");

            System.out.println(str);
      }
}
```

Q) Write a java program display given string is anagram or not?

```
input:
      silent
      listen

output:
      It is a anagram string
```

ex:

```
import java.util.Arrays;
class Test
{
	public static void main(String[] args)
	{
		String str1="silent";
		String str2="listen";

		char[] carr1=str1.toCharArray();
		char[] carr2=str2.toCharArray();

		Arrays.sort(carr1); // e i l n s t
		Arrays.sort(carr2); // e i l n s t

		boolean flag=true;
		for(int i=0;i<carr1.length && i<carr2.length;i++)
		{
			if(carr1[i]!=carr2[i])
			{
				flag=false;
				break;
			}
		}

		if(flag==true)
			System.out.println("It is a anagram string");
		else
			System.out.println("It is not a anagram string");
	}
}
```

Q) Write a java program to display the given string in a given format?

input:
	A1B2C3D4

output:
	ABBCCCDDDD

ex:

```
class Test
{
	public static void main(String[] args)
	{
		String str="A1B2C3D4";

		for(int i=0;i<str.length();i++)
		{
			if(Character.isAlphabetic(str.charAt(i)))
			{
				System.out.print(str.charAt(i));
			}
			else
```

```
                  {
                        int j=Character.getNumericValue(str.charAt(i));

                        for(int k=1;k<j;k++)
                        {

                              System.out.print(str.charAt(i-1));
                        }
                  }
            }
      }
}
```

Q) Write a java program to display the strings starting with uppercase letter?

```
input:
      This is java Class for Student

output:
      This Class Student

ex:
--
class Test
{
      public static void main(String[] args)
      {
            String str="This is java Class for Student";

            String[] sarr=str.split(" ");

            for(String s:sarr)
            {
                  if(s.charAt(0)>='A' && s.charAt(0)<='Z')
                  {
                        System.out.print(s+" ");
                  }
            }
      }
}
```

Q) Write a java program to display the strings starting with vowels?

```
input:
      Now is raining where my Umbrella

output:
      is umbrella

ex:


class Test
```

```
{
	public static void main(String[] args)
	{
		String str="Now is raining where my Umbrella";

		str=str.toLowerCase();

		String[] sarr=str.split(" ");

		for(String s:sarr)
		{
			char ch=s.charAt(0);

			if(ch=='a' || ch=='e' || ch=='i' || ch=='o'|| ch=='u')
			{
				System.out.print(s+" ");
			}
		}
	}
}
```

Q) Write a java program to display number of vowels present in a string?

```
input:
	umbrella

output:
	u e a

ex:

class Test
{
	public static void main(String[] args)
	{
		String str="umbrella";

		for(int i=0;i<str.length();i++)
		{
			char ch=str.charAt(i);

			if(ch=='a' || ch=='e' || ch=='i' || ch=='o'|| ch=='u')
			{
				System.out.print(str.charAt(i)+" ");
			}
		}

	}
}
```

Q) Write a java program to count number of vowels present in a string?

```
input:
	umbrella

output:
```

3

ex:

```
class Test
{
	public static void main(String[] args)
	{
		String str="umbrella";

		int cnt=0;

		for(int i=0;i<str.length();i++)
		{
			char ch=str.charAt(i);

			if(ch=='a' || ch=='e' || ch=='i' || ch=='o'|| ch=='u')
			{
				cnt++;
			}
		}
		System.out.println(cnt);
	}
}
```

Q) Write a java program to display the string in a given format?

input:
XYZ

output:
XY
XZ
YX
YZ
ZX
ZY

ex:

```
class Test
{
	public static void main(String[] args)
	{
		String str="XYZ";

		for(int i=0;i<str.length();i++)
		{
			for(int j=0;j<str.length();j++)
			{
				if(i!=j)
				{

System.out.println(str.charAt(i)+""+str.charAt(j));
				}

			}
		}
	}
```

```
}
```

Q) Write a java program to display permutation of a given string?

```
Input:
      ABC

output:
      ABC
      ACB
      BAC
      BCA
      CBA
      CAB

ex:

class Test
{
      public static void main(String[] args)
      {
            String str="ABC";
            //caller method
            permutation(str.toCharArray(),0);
      }
      //callie method
      public static void permutation(char[] carr,int fi)
      {
            if(fi==carr.length-1)
            {
                  System.out.println(carr);
                  return;
            }

            for(int i=fi;i<carr.length;i++)
            {
                  swap(carr,fi,i);
                  permutation(carr,fi+1);
                  swap(carr,fi,i);
            }
      }
      //callie method
      public static void swap(char[] carr,int fi,int i)
      {
            //swapping logic
            char temp=carr[fi];
            carr[fi]=carr[i];
            carr[i]=temp;
      }
}
```

Q) Write a java program to perform largest common subsequence in a given string?

```
Input:
      ABCAB
      AECB
```

Output:
	3

ex:

```
class Test
{
	public static void main(String[] args)
	{
		String str1="ABCAB";
		String str2="AECB";

		//caller method
		System.out.println(longestCommSubsequence(str1,str2));
	}
	//callie method
	public static int longestCommSubsequence(String s1,String s2)
	{
		return solve(s1,s2,0,0);
	}
	//callie method
	public static int solve(String s1,String s2,int i,int j)
	{
		if(i==s1.length())
			return 0;

		if(j==s2.length())
			return 0;

		int ans=0;
		if(s1.charAt(i)==s2.charAt(j))
		{
			ans=1+solve(s1,s2,i+1,j+1);
		}
		else
		{
			ans=Math.max(solve(s1,s2,i+1,j),solve(s1,s2,i,j+1));
		}

		return ans;
	}
}
```

StringBuffer
==========
If our content change frequently then it is never recommanded to go with String object because for every change a new object will be created.

To overcome this limitation Sun Micro System introduced StringBuffer object.

In Stringuffer all the required changes will be done in a single object only.

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.

syntax:

```
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("abcdefgjijklmnop");
		System.out.println(sb.capacity()); //16

		sb.append("qr");
		System.out.println(sb.capacity()); //16+1*2=34
	}
}
```

2) StringBuffer sb=new StringBuffer(int capacity);

It will create StringBuffer object with specified initial capacity.

ex:

```
class Test
{
	public static void main(String[] args)
	{
		StringBuffer sb=new StringBuffer(19);
		System.out.println(sb.capacity()); //19

	}
}
```

3) StringBuffer sb=new StringBuffer(String str);

It will create StringBuffer object 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("ihub");
		System.out.println(sb.capacity()); //4+16=20

	}
}
```

Q) Write a java program to display reverse of a string?

```
input:
	hello

output:
	olleh

ex:

class Test
{
	public static void main(String[] args)
	{
		String str="hello";

		StringBuffer sb=new StringBuffer(str);

		String rev=sb.reverse().toString();

		System.out.println(rev);

	}
}
```

Q) Write a java program to check given string is palindrome or not?

```
input:
	racar

output:
	It is a plaindrome string

eX:
---
class Test
{
	public static void main(String[] args)
	{
		String str="racar";

		StringBuffer sb=new StringBuffer(str);

		String rev=sb.reverse().toString();

		if(str.equals(rev))
			System.out.println("It is a palindrome string");
```

```
            else
                  System.out.println("It is not a palindrome string");

      }
}
```

Q) Write a java program to display the string in a given format?

```
input:
      ABBCCCDDDD

output:
      A1B2C3D4

ex:

class  Test
{
      public static void main(String[] args)
      {
            String str="ABBCCCDDDD";

            int count=1;

            StringBuffer sb=new StringBuffer();

            for(int i=0;i<str.length();i++)
            {
                  if(i<str.length()-1 && str.charAt(i)==str.charAt(i+1))
                  {
                        count++;
                  }
                  else
                  {
                        sb.append(str.charAt(i)).append(count);
                        count=1;
                  }
            }

            System.out.println(sb.toString());
      }
}
```

Q) Write a java program to multiply two arrays?

```
input:
      5 8 2
      2 6

output:
      15132 (582*26)

ex:
```

```
class  Test
{
	public static void main(String[] args)
	{
		int[] arr1={5,8,2};
		int[] arr2={2,6};

		//caller method
		int a=Integer.parseInt(arrayToString(arr1));
		int b=Integer.parseInt(arrayToString(arr2));
		System.out.println(a*b);

	}
	//callie method
	public static String arrayToString(int[] arr)
	{
		StringBuffer sb=new StringBuffer();

		for(int i:arr)
		{
			sb.append(i);
		}

		return sb.toString();
	}
}
```

Q) Write a java program to count number of 2's present in a given number?

```
input:
	22

output:
	6 (2,12,20,21,22)

ex:
----
class  Test
{
	public static void main(String[] args)
	{
		int num=22;

		StringBuffer sb=new StringBuffer();

		for(int i=1;i<=num;i++)
		{
			sb.append(i);
		}

		//counting number of 2's
		int count=0;
		for(int i=0;i<sb.length();i++)
		{
			int n=Character.getNumericValue(sb.charAt(i));
			if(n==2)
```

```
                {
                        count++;
                }
        }

        System.out.println(count);
    }

}
```

Q) Write a java program to encode a given string?

```
input:
    1106

output:
    AAJF

ex:

class  Test
{
    public static void main(String[] args)
    {
        String str="1106";

        //caller method
        System.out.println(encodeString(str));
    }
    //callie method
    public static  String  encodeString(String str)
    {
        StringBuffer sb=new StringBuffer();

        for(int i=0;i<str.length();i++)
        {
            int n=Character.getNumericValue(str.charAt(i));

            if(n>0)
            {
                sb.append((char)('A'+ n - 1));
            }
            else
            {
                int k=Integer.parseInt(str.substring(i-1,i+1));
                sb.append((char)('A'+ k - 1));
            }

        }

        return sb.toString();
    }

}
```

StringBuilder

==========

StringBuilder is exact same as StringBuffer with following differences.

StringBuffer	StringBuilder
Every method present in StringBuffer is synchronized.	No method present in StringBuilder is synchronized.
At a time only one thread is allowed to operator on StringBuffer object.Hence StringBuffer is thread safe.	Multiple threads are allowed to operator on StringBuilder object.Hence StringBuilder is not thread safe.
Waiting time of a thread will increase relatively performance is low.	There is no waiting time relatively 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 irrespective of regular expression.

We can create StringTokenizer object as follow.

syntax:

```
      StringTokenizer st=new StringTokenizer(String
str,RegularExpression  regEx);
```

StringTokenizer class contains following methods.

ex:
```
      public boolean hasMoreTokens()
      public String nextToken()
      public boolean hasMoreElements();
      public Object nextElement();
      public int countTokens()
```

ex:

```
import java.util.StringTokenizer;
class  Test
{
      public static void main(String[] args)
      {
            StringTokenizer st=new StringTokenizer("this is java
class");

            System.out.println(st.countTokens()); // 4
      }
}
```

Here default regular expression is space.

ex:

```
---
import java.util.StringTokenizer;
class  Test
{
      public static void main(String[] args)
      {
            StringTokenizer st=new StringTokenizer("this is java
class"," ");

            System.out.println(st.countTokens()); // 4
      }
}
```

ex:

```
---
import java.util.StringTokenizer;
class  Test
{
      public static void main(String[] args)
      {
            StringTokenizer st=new StringTokenizer("this is java
class"," ");

            while(st.hasMoreTokens())
            {
                  String s=st.nextToken();
                  System.out.println(s);
            }
      }
}
```

ex:

```
----
import java.util.StringTokenizer;
class  Test
{
      public static void main(String[] args)
      {
            StringTokenizer st=new StringTokenizer("this is java
class"," ");

            while(st.hasMoreElements())
            {
                  String s=(String)st.nextElement();
```

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

ex:
---
import java.util.StringTokenizer;
class  Test
{
      public static void main(String[] args)
      {
            StringTokenizer st=new StringTokenizer("9,99,999",",");

            while(st.hasMoreElements())
            {
                  String s=(String)st.nextElement();
                  System.out.println(s);
            }
      }
}
```

Assignment
==========

Q) Write a java program to display palindrome strings?

```
input:
      racar is madam for bit

output:
      racar  madam

ex:
---

class  Test
{
      public static void main(String[] args)
      {
            String str="racar is madam for bit";

            String[] sarr=str.split(" ");

            //for each
            for(String s:sarr)
            {
                  StringBuffer sb=new StringBuffer(s);
                  sb.reverse();

                  if(s.equals(sb.toString()))
                  {
                        System.out.print(s+" ");
                  }
            }
```

```
	}
}
```

Assignment
==========
Q) Write a java program to decode the string?

input:
	AAJF

output:
	1106

Exception Handling
==========

Q) What is the difference between Exception and Error?

Exception

	Exception is a problem for which we can provide solution programmatically.
	Exception will raise due to syntax errors.
	ex:
		FileNotFoundException
		ArithmeticException
		IllegalArgumentException

Error

	Error is a problem for which we can't provide solution programmatically.
	Error will raise due to lack of system resources.
	ex:
		LinkageError
		OutOfMemoryError
		StackOverFlowError

As a part of java application development it is a responsibility of a programmer to provide
smooth termination for every java program.

We have two types of terminations.

1) Smooth termination / Graceful termination

2) Abnormal termination

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.

ex:

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

1) Abnormal termination

During the program execution suppose if we are getting some interruptions 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 otherwiser our
program will terminates abnormally.

Here exception will display name of the exception , description of the exception and line number of
the exception.

Exception

==========
It is a unwanted, expected event which disturbs normal flow of our program.

Exceptions always raised at runtime so they are also known as runtime events.

The main objective of exception handling is to provide graceful termination.

In java, Exceptions are divided into two types.

1) Predefined exceptions

2) Userdefined exception

1) Predefined exceptions

Built-In exceptions are called predefined exceptions.

It is categories into two types.

i) Checked exceptions

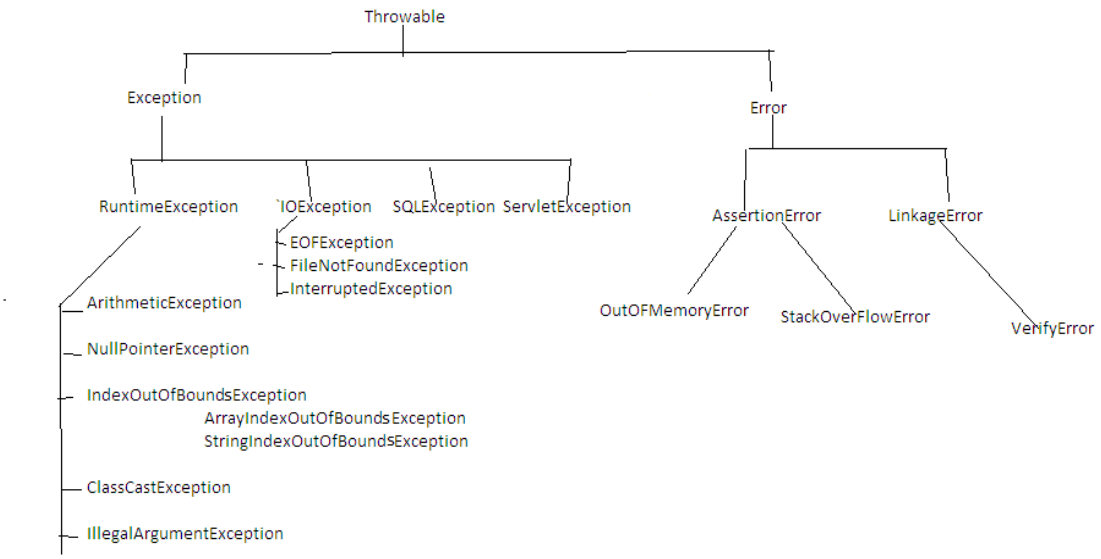
Exceptions which are checked by the compiler at the time of compilation are called
checked exceptions.
ex:
EOFException
FileNotFoundException
InterruptedException

ii) Unchecked exceptions

Exceptions which are checked by the JVM at the time of runtime are called unchecked exceptions.

ex:
ArithmeticException
ClassCastException
IllegalArgumentException

Diagram: java38.1



If any checked exception raised in our program we must and should handle that exception by using try and catch block.

try block

==========

It is a block which contains risky code.

A try block associate with catch block.

If any exception raise in try block then try block won't be executed.

A try block is used to throw the exception to catch block.

catch block

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

It is a block which contains error handling code.

A catch block always associate with try block.

A catch block is used to catch the exception which is thrown by try block.

If there is no exception in try block then catch block won't be executed.

A catch block will take exception name as a parameter and that name must match with exception class name.

syntax:

```
-------
      try
      {
            - // Risky Code
      }
      catch(ArithmeticException ie)
      {
            - // Error Handling Code
      }
```

ex:

```
---
class  Test
{
      public static void main(String[] args)
      {
            try
            {
                  System.out.println("try-block");
            }
            catch(Exception e)
            {
                  System.out.println("catch-block");
            }
      }
```

```
}
o/p:
      try-block

ex:2
-----
class  Test
{
      public static void main(String[] args)
      {
            try
            {
                  System.out.println(10/0);
            }
            catch(ArithmeticException ae)
            {
                  System.out.println("catch-block");
            }
      }
}
o/p:
      catch-block

ex:3
----
class  Test
{
      public static void main(String[] args)
      {
            try
            {
                  System.out.println("stmt1");
                  System.out.println(10/0);
                  System.out.println("stmt2");
            }
            catch(ArithmeticException ae)
            {
                  System.out.println("catch-block");
            }
      }
}
o/p:
      stmt1
      catch-block


ex:4
-----
class  Test
{
      public static void main(String[] args)
      {
            int i=1;

            try
            {
                  i++;
            }
```

```
            catch(Exception e)
            {
                  i++;
            }

            System.out.println(i); // 2
      }
}
```

A try with multiple catch block
==========
A try block can have multiple catch blocks.

If a try block contains multiple catch block then order of catch blocks are very important. It should be from 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");
            }
      }
}
```

Various ways to display exception details
==========
Throwable class defines following three ways to display exception details.

1) printStackTrace()

It is used to display name of the exception ,description of the exception and line number of the exception.

2) toString()

It is used to display name of the exception and description of the exception.

3) getMessage()

It is used to 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 never recommended to maintain cleanup code in try block because if any exception raise in try block then try block won't be executed.

It is never recommended to maintain cleanup code in catch block because if there is no exception in try block then catch block won't be executed.

But we need a place where we can maintain cleanup code and it should execute irrespective of
exception raise or not. Such block is called finally block.

syntax:

```
      try
      {
            - // Risky Code
      }
      catch(Exception e)
      {
            - // Error Handling Code
      }
      finally
      {
            - // Cleanup code
      }
```

ex:

```
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
```

ex:3

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

```
            try
            {
                  System.out.println("try-block");
                  System.out.println(10/0);
                  System.out.println("stmt2");
            }
            catch (Exception e)
            {
                  e.printStackTrace();
            }
            finally
            {
                  System.out.println("finally-block");
            }
      }
}
o/p:
      try-block
      java.lang.ArithmeticException: / by zero
        at Test.main(Test.java:8)
      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 finalize method?

final

It is a modifier which is applicable for variables ,methods and classes.
If we declare any variable as final then reinitialization of that variable is not possible.
If we declare any method as final then overriding of that method is not possible.
If we declare any class as final then creating child class is not possible.

finally

It is a block which contains cleanup code and it should execute irrespective of exception raised or not.

finalize

It is a method called by garbage collector just before destroying an object for cleanup activity.

throw Statement

Sometimes we will create exception objects explicitly and handover to JVM manually by using throw statement.

ex:

```
throw new ArithmeticException("Don't divide by zero");
```

ex:

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

Here exception object created and handover to JVM by main method.

ex:

```
class  Test
{
      public static void main(String[] args)
      {
            throw new ArithmeticException("don't divide by zero");
      }
}
```

Here exception object created explicitly and handover to JVM manually by using programmer via throw statement.

throws statement

If any checked exception raised in our program we must and should handle that exception by using try and catch block or by using throws statement.

ex:

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

            try
            {
                  Thread.sleep(3000);
```

```
                System.out.println("Welcome to Java");
            }
            catch (InterruptedException ie)
            {
                ie.printStackTrace();
            }
      }
}
```

ex:

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

                Thread.sleep(5000);
                System.out.println("Welcome to Java");

      }
}
```

2) Userdefined exceptions
==========

Exceptions which are created by the user based on the application requirements are called
customized exceptions or userdefined exceptions.

ex:

```
      NotInterestInJavaException
      NoPracticeException
      NoJobException
      EligileException
      NotEligibleException
```

ex:

```
import java.util.Scanner;
class EligibleException extends RuntimeException
{
      EligibleException(String s)
      {
            super(s);
      }
}
class NotEligibleException extends RuntimeException
{
      NotEligibleException(String s)
      {
            super(s);
      }
}
class Test
{
      public static void main(String[] args)
      {
            Scanner sc=new Scanner(System.in);
```

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

            if(age<18)
                  throw new NotEligibleException("U r not eligible to
vote");
            else
                  throw new EligibleException("U r eligile to vote");
      }
}
```

Q) Can we handle multiple exceptions in a single catch block?

Yes, it is possible to handle multiple exceptions in a single catch block.

ex:

```
class Test
{
      public static void main(String[] args)
      {
            try
            {
                  //System.out.println(10/0);

                  Thread.currentThread().setPriority(15);
            }
            catch (IllegalArgumentException | NullPointerException |
ArithmeticException  e)
            {
                  e.printStackTrace();
            }
      }
}
```

java.io package

==========

File
=======

```
      File f=new File("abc.txt");
```

File will check is there any abc.txt file already created or not.
If it is available it simply refers to that file.If it is not created then
it won't create any new file.

ex:

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

```
            File f=new File("abc.txt");
            System.out.println(f.exists());//false
      }
}
```

A File object can be used to create a physical file.

ex:

```
import java.io.*;
class Test
{
      public static void main(String[] args)throws IOException
      {
            File f=new File("abc.txt");
            System.out.println(f.exists());//false

            f.createNewFile();
            System.out.println(f.exists());//true

      }
}
```

A File object can be used to create a directory also.

ex:

```
import java.io.*;
class Test
{
      public static void main(String[] args)throws IOException
      {
            File f=new File("bhaskar123");
            System.out.println(f.exists());//false

            f.mkdir();
            System.out.println(f.exists());//true

      }
}
```

Q)Write a java program to Create a "cricket123" folder and inside that folder create "abc.txt" file?

```
import java.io.*;
class Test
{
      public static void main(String[] args)throws IOException
      {
            File f1=new File("cricket123");
            f1.mkdir();

            File f2=new File("cricket123","abc.txt");
            f2.createNewFile();

            System.out.println("Please check the location");

      }
```

```
}
```

FileWriter
==========
FileWriter is used to write character oriented data into a file.

constructor

```
FileWriter fw=new FileWriter(String s);
FileWriter fw=new FileWriter(File f);
```

ex:
```
      FileWriter fw=new FileWriter("aaa.txt");
      or

      File f=new File("aaa.txt");
      FileWriter fw=new FileWriter(f);
```

If file does not exist then FileWriter will create a physical file.

Methods
----

1)write(int ch)

      It will insert single character into a file.

2)write(char[] ch)

      It will insert array of characters into a file.

3)write(String s)

      It will insert String into a file.

4)flush()

      It gives guaranttee that last character of a file is also inserted.

5)close()

      It is used to close the FileWriter object.

ex:

```
import java.io.*;
class Test
{
      public static void main(String[] args)throws IOException
      {
            FileWriter fw=new FileWriter("aaa.txt");

            fw.write(98);// b
```

```
            fw.write("\n");

            char[] ch={'a','b','c'};
            fw.write(ch);
            fw.write("\n");

            fw.write("bhaskar\nsolution");
            fw.flush();
            fw.close();
            System.out.println("Please check the location");
      }
}
```

FileReader
==========
It is used to read character oriented data from a file.

constructor

```
FileReader fr=new FileReader(String s);
FileReader fr=new FileReader(File f);
```

ex:
```
      FileReader fr=new FileReader("aaa.txt");
      or
      File f=new File("aaa.txt");
      FileReader fr=new FileReader(f);
```

Methods
---
1)read()

      It will read next character from a file and return unicode value.
      If next character is not available then it will return -1.

2)read(char[] ch)

      It will read collection of characters from a file.

3)close()

      It is used to close FileReader object.

ex:1

```
import java.io.*;
class Test
{
      public static void main(String[] args)throws IOException
      {
            FileReader fr=new FileReader("aaa.txt");

            int i=fr.read();
            while(i!=-1)
            {
                  System.out.print((char)i);
                  i=fr.read();
```

```
            }
            fr.close();
      }
}
```

ex:2

```
import java.io.*;
class Test
{
      public static void main(String[] args)throws IOException
      {
            FileReader fr=new FileReader("aaa.txt");

            char[] carr=new char[255];

            //load the data from file to char array
            fr.read(carr);

            //reading the data from char array
            for(char c:carr)
            {
                  System.out.print(c);
            }

            fr.close();

      }
}
```

Usage of FileWriter and FileReader is not recommanded to use

While inserting the data by using FileWriter ,we need to insert line seperator(\n) which is very headache for the programmer.

While reading the data by using FileReader object ,we need to read character
by character which is not convenient to the programmer.

To overcome this limitation Sun micro system introduced BufferedWriter and BufferedReader.

BufferedWriter

It is used to insert character oriented data into a file.

constructor

```
BufferedWriter bw=new BufferedWriter(Writer w);
BufferedWriter bw=new BufferedWriter(Writer w,int buffersize);
```

BufferedWriter object does not communicate with files directly.
It will take the support of some writer objects.

ex:

```
      FileWriter fw=new FileWriter("bbb.txt");
```

```
      BufferedWriter bw=new BufferedWriter(fw);

      or

      BufferedWriter bw=new BufferedWriter(new FileWriter("bbb.txt"));
```

Methods
---

1)write(int ch)

      It will insert single character into a file.

2)write(char[] ch)

      It will insert array of characters into a file.

3)write(String s)

      It will insert String into a file.

4)flush()

      It gives guaranttee that last character of a file is also inserted.

5)close()

      It is used to close the BufferedWriter object.

6)newLine()

      It will insert new line into a file.

ex:

```
import java.io.*;
class Test
{
      public static void main(String[] args)throws IOException
      {

      BufferedWriter bw=new BufferedWriter(new FileWriter("bbb.txt"));
            bw.write(98);//b
            bw.newLine();

            char[] ch={'a','b','c'};
            bw.write(ch);
            bw.newLine();

            bw.write("bhaskar");
            bw.newLine();

            bw.flush();
            bw.close();
            System.out.println("Please check the location");
```

```
	}
}
```

BufferedReader

```
=================
```

It is enhanced reader to read character oriented data from a file.

```
constructor
------------
BufferedReader br=new BufferedReader(Reader r);
BufferedReader br=new BufferedReader(Reader r,int buffersize);
```

BufferedReader object can't communicate with files directly.IT will take
support of some reader objects.

```
ex:
	FileReader fr=new FileReader("bbb.txt");
	BufferedReader br=new BufferedReader(fr);

	or

	BufferedReader br=new BufferedReader(new FileReader("bbb.txt"));
```

The main advantage of BufferedReader over FileReader is we can read character line by line instead of character by character.

```
methods
---------
1)read()
--------
```

	It will read next character from a file and return unicode value.
	If next character is not available then it will return -1.

```
2)read(char[] ch)
-----------------
```

	It will read collection of characters from a file.

```
3)close()
---------
```

	It is used to close BufferedReader object.

```
4)nextLine()
------------
```

	It is used to read next line from the file.If next line is
	not available then it will return null.

ex:

```
import java.io.*;
class Test
{
	public static void main(String[] args)throws IOException
	{
		BufferedReader br=new BufferedReader(new
FileReader("bbb.txt"));
```

```
            String line=br.readLine();
            while(line!=null)
            {
                  System.out.println(line);
                  line=br.readLine();
            }

            br.close();

      }
}
```

PrintWriter
==========
It is enhanced write to write character oriented data into a file.

constructor

```
PrintWriter pw=new PrintWriter(String s);
PrintWriter pw=new PrintWriter(File f);
PrintWriter pw=new PrintWriter(Writer w);
```

PrintWriter can communicate with files directly and it will take the support of some writer objects.

ex:
```
      PrintWriter pw=new PrintWriter("ccc.txt");

      or

      PrintWriter pw=new PrintWriter(new File("ccc.txt"));

      or

      PrintWriter pw=new PrintWriter(new FileWriter("ccc.txt"));
```

The main advantage of PrintWriter over FileWriter and BufferedWriter is we can insert any type of data.

Assume if we want insert primitive values then PrintWriter is best choice.

methods

```
write(int ch)
write(char[] ch)
write(String s)
flush()
close()

writeln(int i)
writeln(float f)
writeln(double d)
writeln(String s)
writeln(char c)
```

writeln(boolean b)

write(int i)
write(float f)
write(double d)
write(String s)
write(char c)
write(boolean b)

ex:

```
import java.io.*;
class Test
{
      public static void main(String[] args)throws IOException
      {
            PrintWriter pw=new PrintWriter("ccc.txt");

            pw.write(100);// d
            pw.println(100);// 100
            pw.print('a');
            pw.println(true);
            pw.println("hi");
            pw.println(10.5d);

            pw.flush();
            pw.close();
            System.out.println("Please check the location");
      }
}
```

various ways to provide input values from the keyboard
==========
There are various ways to provide input values from keyboard.

1)Command line argument

2)BufferedReader class

3)Console class

4)Scanner class

1)Command line argument

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

ex:

```
class Test
{
      public static void main(String[] args)
      {
            String name=args[0];
```

```
            System.out.println("Welcome : "+name);
      }
}
```

o/p:

```
javac  Test.java
java  Test  Alan
```

2)BufferedReader class

BufferedReader class present in java.io package.

BufferedReader class will take InputStreamReader object as a parameter which is embedded with System.in.

ex:

```
      BufferedReader br=
                  new BufferedReader
                        (new InputStreamReader(System.in));
```

To read input values from console we need to readLine() method.

ex:

```
import java.io.*;
class Test
{
      public static void main(String[] args)throws IOException
      {
            BufferedReader br=new BufferedReader(new
InputStreamReader(System.in));

            System.out.println("Enter the Name :");
            String name=br.readLine();

            System.out.println("Welcome : "+name);


      }
}
```

3)Console class

==========

Console class present in java.io package.

We can create Console class object by using console() method of System class.

ex:

```
      Console c=System.console();
```

To read inputs from console we need to use readLine() method.

ex:

```
import java.io.*;
class Test
```

```
{
	public static void main(String[] args)throws IOException
	{
		Console c=System.console();

		System.out.println("Enter the Name :");
		String name=c.readLine();

		System.out.println("Welcome : "+name);

	}
}
```

4)Scanner class
==========
Scanner class present java.util package.

We can create Scanner object class as follow.

ex:

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

We can read inputs from the console by using following methods.
ex:

```
	next()
	nextLine()
	nextInt()
	nextFloat()
	nextDouble()
	next().charAt(0);
	and etc.
```

ex:

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

		System.out.println("Enter the No :");
		int no=sc.nextInt();

		System.out.println("Enter the Name :");
		String name=sc.next();

		System.out.println("Enter the Fee :");
		double fee=sc.nextDouble();

		System.out.println(no+" "+name+" "+fee);
	}
}
```

Generics

=========
Arrays are typesafe. We can give guarantee that what type of elements are present in array.

If requirement is there to store String values then it is recommanded to use String array.

ex:

```
String[] sarr=new String[5];
sarr[0]="hi";
sarr[1]="hello";
sarr[2]="bye";
sarr[3]=10;  // invalid
```

At the time of retrieving the data from array we don't need to perform typecasting.

ex:

```
String[] sarr=new String[5];
sarr[0]="hi";
sarr[1]="hello";
sarr[2]="bye";
-
-
String value=sarr[0];
```

Collections are not typesafe. We can't give guarantee that what type of elements are present in
Collections.

If requirement is there to store String values then it is never recommended to use ArrayList because we won't get any compile time error or runtime error but sometimes our program get failure.

ex:

```
ArrayList al=new ArrayList();
al.add("hi");
al.add("hello");
al.add(10);
```

AT the time of retrieving the data from Collection , compulsary we need to perform typecasting.

ex:

```
ArrayList al=new ArrayList();
al.add("hi");
al.add("hello");
al.add(10);
-
-
String value=(String)al.get(0);
```

To overcome this above limitations Sun Micro System introduced Generics concept in 1.5v.

The main objective of generics are.

1) To make Collections as typesafe.

2) To avoid typecasting problem.

java.util package

==========

Q) What is the difference between Arrays and Collections ?

Arrays	Collections
It is a collection of homogeneous Data elements.	It is a collection of homogeneous and hetrogeneous Data elemnets
It is fixed in size.	It is growable in nature.
Performance point of view arrays are recommanded to use.	Memory point of view Collections Are recommanded to use.
Arrays are not implemented based on structure concept. Hence we can't expect concept. any readymade methods.	Collections are implemented based on data structure concept Hence we can expect ready made methods.
Arrays can hold primitive types and Object types.	Collections can hold only object types.

Collection Framework

==========

Collection framework defines several classes and interfaces to represent group of individual objects.

Collection

==========

It is a root interface for entire Collection Framework.

If we want to represent group of individual objects in a single entity then we need to use Collection.

Collection interface contains following methods which are applicable for entire Collection objects.

ex:

```
cmd> javap java.util.Collection

ex:
public abstract int size();
```

```
      public abstract boolean isEmpty();
      public abstract boolean contains(java.lang.Object);
      public abstract java.util.Iterator<E> iterator();
      public abstract java.lang.Object[] toArray();
      public abstract boolean add(E);
      public abstract boolean remove(java.lang.Object);
      public abstract boolean containsAll(java.util.Collection<?>);
      public abstract boolean addAll(java.util.Collection<? extends
E>);
      public abstract boolean removeAll(java.util.Collection<?>);
      and etc..
```

Q) What is the difference between Collection and Collections?

Collection	Collections
It is a root interface for Entire collection framework.	It is a utility class.
It is used to represent a group Of individual objects in a single unit.	It defines several utility methods that are used to operates on collection.
It contains abstract methods, Static methods and default Methods.	It contains only static methods.

Q) Write a java program to display the string in a sorting order?

```
input:
      ball  cat  dog  apple elephant

output:
      apple  ball  cat  dog  elephant

ex:
--
import java.util.*;
class Test
{
      public static void main(String[] args)
      {
            String str="ball cat dog apple elephant";

            String[] sarr=str.split(" ");
```

```
            ArrayList al=new ArrayList();

            //for each loop
            for(String s:sarr)
            {
                  al.add(s);
            }

            Collections.sort(al);

            al.forEach(element -> System.out.print(element+" "));

      }
}
```

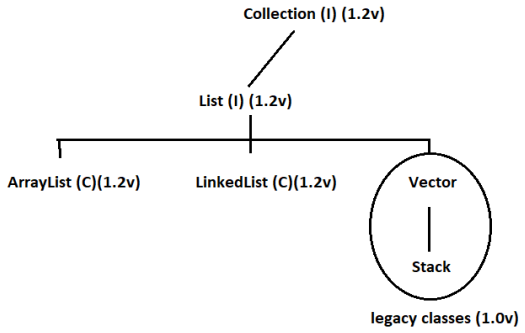
1.List

======

It is a child interface of Collection interface.

If we want to represent group of individual objects in a single entity where duplicate objects are allowed and order is preserved the we need to use List interface.

Diagram: java40.1



i)ArrayList

==========

The underlying data structure is resizable array or growable array.

Duplicate objects are allowed.

Insertion order is preserved.

Heterogeneous objects are allowed.

Null insertion is possible.

It implements Serializable, Cloneable and RandomAccess interface.

If our frequent operation is a retrieval operation then ArrayList is a best choice.

ex:

```
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:

```
import java.util.*;
class Test
{
	public static void main(String[] args)
	{
		ArrayList<String> al=new ArrayList<String>();
		al.add("one");
		al.add("two");
		al.add("three");
		System.out.println(al);//[one,two,three]

		al.add("one");
		System.out.println(al);//[one,two,three,one]

		al.add(null);
		System.out.println(al);//[one,two,three,one,null]

	}
}
```

ex:

```
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.isEmpty()); // false

            for(int i=0;i<al.size();i++)
            {
                  String s=al.get(i);
                  System.out.println(s);
            }

      }
}

ex:
---
import java.util.*;
class Test
{
      public static void main(String[] args)
      {
            ArrayList<String> al=new ArrayList<String>();
            al.add("one");
            al.add("two");
            al.add("three");

            al.add(1,"gogo");

            System.out.println(al); //[one,gogo,two,three]

            System.out.println(al.contains("gogo")); // true

            al.remove("gogo");

            System.out.println(al);//[one,two,three]

            al.clear();

            System.out.println(al);//[]

      }
}

ex:
---
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);

      }
}
```

ex:

```
import java.util.*;
class Test
{
      public static void main(String[] args)
      {
            List<String> list=Arrays.asList("one","two","three","four");

            list.forEach(element -> System.out.print(element+" "));
      }
}
```

ex:
--

```
import java.util.*;
class Test
{
      public static void main(String[] args)
      {
            List<Integer> list=Arrays.asList(6,3,9,1,4);

            list.forEach(element -> System.out.print(element+" "));
      }
}
```

ii)LinkedList
============

The underlying data structure is doubly LinkedList.

Duplicate objects are allowed.

Insertion order is preserved.

Hetrogeneous objects are allowed.

Null insertion is possible.

It implements Serializable, Cloneable and Deque interface.

If our frequent operation is adding and removing in the middle then LinkedList is a best choice.

LinkedList class contains following methods.

ex:

```
	public E getFirst();
	public E getLast();
	public E removeFirst();
	public E removeLast();
	public void addFirst(E);
	public void addLast(E);

ex:
---
import java.util.*;
class Test
{
	public static void main(String[] args)
	{
		LinkedList ll=new LinkedList();
		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(10);
		System.out.println(ll);//[one,two,three,one,10]

		ll.add(null);
		System.out.println(ll);//[one,two,three,one,10,null]


	}
}

ex:
---
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");

		ll.addFirst("gogo");
		ll.addLast("jojo");

		System.out.println(ll);//[gogo,one,three,three,jojo]

		System.out.println(ll.getFirst());//gogo
		System.out.println(ll.getLast());//jojo

		ll.removeFirst();
		ll.removeLast();

		System.out.println(ll); //[one,two,three]
	}
```

```
}
```

```
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]

      }
}
```

iii)Vector
========
The underlying data structure is resizable array or growable array.

Duplicate objects are allowed.

Insertion order is preserved.

Hetrogeneous objects are allowed.

Null insertion is possible.

It implements Serializable, Cloneable and RandomAccess interface.

Vector contains synchronized methods. Hence it is thread safe.

Vector class contains following methods.
ex:
      addElement()
      firstElement()
      lastElement()
      removeElementAt()
      removeAllElements()
      and etc.

ex:

```
----
import java.util.*;
class Test
{
	public static void main(String[] args)
	{
		Vector<Integer> v=new Vector<Integer>();
		System.out.println(v.capacity()); // 10

		for(int i=1;i<=10;i++)
		{
			v.addElement(i);
		}
		System.out.println(v);//[1,2,3,4,5,6,7,8,9,10]

		System.out.println(v.firstElement());//1
		System.out.println(v.lastElement());//10

		v.removeElementAt(5);
		System.out.println(v);//[1,2,3,4,5,7,8,9,10]

		v.insertElementAt(100,5);
		System.out.println(v);

		v.removeAllElements();
		System.out.println(v); //[]

	}
}

ex:
---
import java.util.*;
class Test
{
	public static void main(String[] args)
	{
		Vector<Integer> v=new Vector<Integer>();
		System.out.println(v.capacity()); // 10

		for(int i=1;i<=10;i++)
		{
			v.add(i);
		}
		System.out.println(v);//[1,2,3,4,5,6,7,8,9,10]

		System.out.println(v.get(0));//1
		System.out.println(v.get(v.size()-1));//10

		v.remove(5);
		System.out.println(v);//[1,2,3,4,5,7,8,9,10]

		v.add(5,100);
		System.out.println(v);

		v.clear();
		System.out.println(v); //[]
```

```
      }
}
```

Q) What is the difference between ArrayList and Vector?

ArrayList	Vector
No method is synchronized	All methods are synchronized.
At a time multiple Threads are allow to operate on ArrayList object and hence ArrayList object is not Thread safe.	At a time only one Thread is allow to operate on vector object and hence vector object is Thread safe.
Relatively performance is high Because Threads are not required to wait.	Relatively performance is low because Threads are required to wait.
It is non legacy and introduced In 1.2v	It is legacy and introduced in 1.0v

Q) What is the difference between ArrayList and LinkedList?

ArrayList	LinkedList
The underlying data structure is Resizable array or growable Array.	The underlying Data structure is doubly linked list.
Array list is better for sorting And accessing data.	Linked list is better for manipulating Data.
The memory location for the Elements of an ArrayList is Contiguous.	The memory location for the elements of an LinkedList is not contiguous.
When an ArrayList is initialized, a default capacity of 10 is assigned to the ArrayList.	There is no case of default capacity in a LinkedList.

iv)Stack
========
It is a child class of Vector class.

If we depends upon Last In First Out order then we need to use Stack.

constructor

```
      Stack s=new Stack();
```

methods
----
1) push(Object o)

	It is used to push the element in a stack.

2) pop()

	It is used to pop the element from stack.

3) peek()

	It returns toppest element from stack.

4) isEmpty()

	It is used to check stack is empty or not.

5) Search(Object o)

	It will return offset value if element is found otherwise it will return -1.

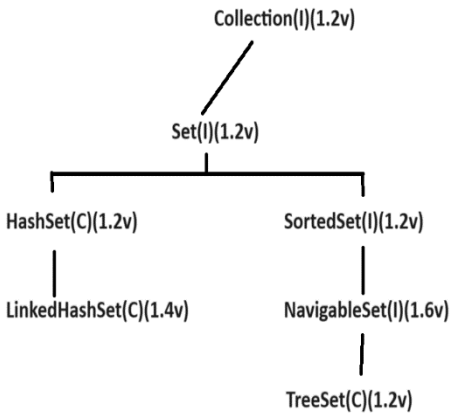
ex:

```
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.peek()); // B

		System.out.println(s.isEmpty());//false

		System.out.println(s.search("Z")); // -1

		System.out.println(s.search("A"));// 2
	}
}
```

2.Set
======
It is a child interface of Collection interface.

If we want to represent group of individual objects in a single entity where duplicate objects are
not allowed and order is not preserved then we need to use Set interface.

Diagram: java41.1



The underlying data structure is Hashtable.

Duplicate objects are not allowed.

Insertion order is not preserved because it will take hash code of an object.

Heterogeneous objects are allowed.

Null insertion is possible.

ex:

```
import java.util.*;
class Test
{
      public static void main(String[] args)
      {
            HashSet hs=new HashSet();
            hs.add("nine");
            hs.add("one");
            hs.add("six");
            System.out.println(hs);//[nine, six, one]

            hs.add("one");
            System.out.println(hs);//[nine, six, one]

            hs.add(10);
            System.out.println(hs);//[nine, six, one, 10]

            hs.add(null);
            System.out.println(hs);//[null, nine, six, one, 10]
      }
```

```
}
```

ii)LinkedHashSet
==========
LinkedHashSet is a child class of HashSet class.

LinkedHashSet is exactly same as HashSet class with following differences.

ex:

HashSet	LinkedHashSet
The underlying data structure is Hashtable.	The underlying data structure is Hashtable and LinkedList.
Insertion order is not preserved.	Insertion order is preserved.
It is introduced in 1.2v.	It is introduced in 1.4v.

ex:

```
import java.util.*;
class Test
{
      public static void main(String[] args)
      {
            LinkedHashSet lhs=new LinkedHashSet();
            lhs.add("nine");
            lhs.add("one");
            lhs.add("six");
            System.out.println(lhs);//[nine, one, six]

            lhs.add("one");
            System.out.println(lhs);//[nine, one, six]

            lhs.add(10);
            System.out.println(lhs);//[nine, one, six, 10]

            lhs.add(null);
            System.out.println(lhs);//[nine, one, six, 10, null]
      }
}
```

iii)TreeSet
==========
The underlying data structure is Balanced Tree.

Duplicate objects are not allowed.

Insertion order is not preserved because it will take sorting order of an object.

Heterogeneous objects are not allowed.

If we insert heterogeneous object then we will get runtime exception called ClassCastException.

Null insertion is not possible.

If we insert null then we will get NullPointerException.

ex:

```
import java.util.*;
class Test
{
      public static void main(String[] args)
      {
            TreeSet ts=new TreeSet();
            ts.add(10);
            ts.add(1);
            ts.add(5);
            ts.add(3);
            System.out.println(ts);//[1, 3, 5, 10]

            ts.add(1);
            System.out.println(ts);//[1, 3, 5, 10]

            //ts.add("hi");
            //System.out.println(ts); // R.E ClassCastException

            //ts.add(null);
            //System.out.println(ts); // R.E NullPointerException
      }
}
```

Q) What is the difference between Comparable and Comparator interface?

Comparable

Comparable is an interface which is present in java.lang package.

It contains only one method i.e compareTo() method.

If we depends upon default natural sorting order then we need to use Comparable interface.

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 same.
```

ex:

```
class Test
{
      public static void main(String[] args)
      {
            System.out.println("A".compareTo("Z")); // -25

            System.out.println("Z".compareTo("A")); // 25

            System.out.println("K".compareTo("K")); // 0
      }
}
```

Comparator

Comparator is an interface which is present in java.util package.

It contains two methods i.e compare() and equals() method.

If we depends upon customized sorting order then we need to use Comparator interface.e

ex:

```
      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 same.
```

Implementation of equals() method is optional because this method present in Object and it is available to the class through inheritance.

ex:

```
import java.util.*;
class Test
{
      public static void main(String[] args)
      {
      TreeSet<Integer> ts=new TreeSet<Integer>(newMyComparator());
            ts.add(10);
            ts.add(1);
            ts.add(5);
            System.out.println(ts);
      }
}
class MyComparator implements Comparator
{
      public int compare(Object obj1,Object obj2)
      {
            Integer i1=(Integer)obj1;
            Integer i2=(Integer)obj2;

            if(i1<i2)
                  return 1;
```

```
            else if(i1>i2)
                  return -1;
            else
                  return 0;
      }
}
```

```
ex:
---
import java.util.*;
class Test
{
      public static void main(String[] args)
      {
            TreeSet<Integer> ts=new TreeSet<Integer>(new
MyComparator());
            ts.add(10);
            ts.add(1);
            ts.add(5);
            System.out.println(ts);
      }
}
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;
      }
}
```

3.Map

======

It is not a child interface of Collection interface.

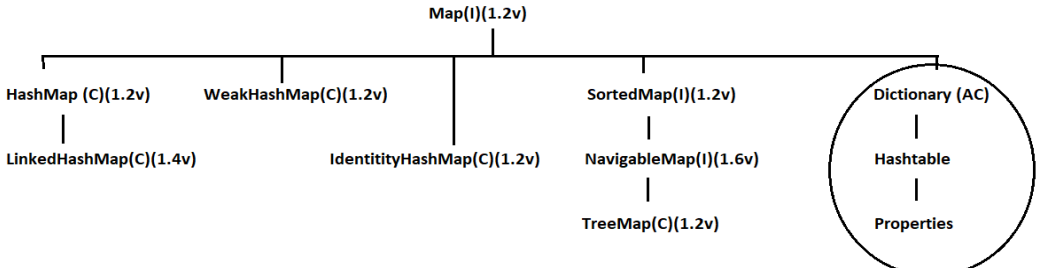
If we want to represent group of individual objects in key and value pair then we need to use Map interface.

key and value both must be objects.

key can't be duplicate but value can be duplicate.

Each key and value pair is known as single entry.

Diagram: java42.1



HashMap
========
The underlying data structure is Hashtable.

Duplicate keys are not allowed but values can be duplicates.

Insertion order is not preserved because it will take hashcode of a key.

Hetrogeneous objects are allowed for both key and value.

Null insertion is possible for both key and value.

ex:

```
import java.util.*;
class Test
{
      public static void main(String[] args)
      {
            HashMap hm=new HashMap();
            hm.put("one","raja");
            hm.put("six","alan");
            hm.put("ten","jose");
            hm.put("four","nancy");
            System.out.println(hm);//{six=alan, four=nancy, one=raja,
ten=jose}

            hm.put("one","gogo");
            System.out.println(hm);//{six=alan, four=nancy, one=gogo,
ten=jose}

            hm.put(1,100);
            System.out.println(hm);//{1=100, six=alan, four=nancy,
one=gogo, ten=jose}

            hm.put(null,null);
            System.out.println(hm);//{null=null, 1=100, six=alan,
four=nancy, one=gogo, ten=jose}
      }
}
```

ex:

```
--
import java.util.*;
class Test
{
      public static void main(String[] args)
      {
            HashMap hm=new HashMap();
            hm.put("one","raja");
            hm.put("six","alan");
            hm.put("ten","jose");
            hm.put("four","nancy");

            Set s=hm.keySet();
            System.out.println(s);//[six, four, one, ten]

            Collection c=hm.values();
            System.out.println(c);//[alan, nancy, raja, jose]

            Set s1=hm.entrySet();
            System.out.println(s1);//[six=alan, four=nancy, one=raja,
ten=jose]
      }
}
```

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 Hashtable.	The underlying data structure is Hashtable and LinkedList.
Insertion order is not preserved.	Insertion order is preserved.
Introduced in 1.2 version.	Introduced in 1.4 version.

ex:

```
import java.util.*;
class Test
{
public static void main(String[] args)
      {
            LinkedHashMap lhm=new LinkedHashMap();
            lhm.put("one","raja");
            lhm.put("six","alan");
            lhm.put("ten","jose");
            lhm.put("four","nancy");
System.out.println(lhm);//{one=raja, six=alan, ten=jose, four=nancy}

            lhm.put("one","gogo");

System.out.println(lhm);//{one=gogo, six=alan, ten=jose, four=nancy}
```

```
            lhm.put(1,100);

System.out.println(lhm);//{one=gogo, six=alan, ten=jose, four=nancy,
1=100}

            lhm.put(null,null);
System.out.println(lhm);//{one=gogo, six=alan, ten=jose, four=nancy,
1=100, null=null}
      }
}
```

TreeMap
=========
The underlying datastructure is RED BLACK TREE.

Duplicate keys are not allowed but values can be duplicate.

Insertion order is not preserved because it will take sorting order of key.

If we depends upon default natural sorting order then key must be homogeneous and Comparable.

If we depends customized sorting order then key must be hetrogeneous and Non-Comparable.

Key can't be null but value can be null.

ex:

```
import java.util.*;
class Test
{
      public static void main(String[] args)
      {
            TreeMap<Integer,String> tm=new TreeMap<Integer,String>();
            tm.put(10,"ten");
            tm.put(1,"one");
            tm.put(5,"five");
            System.out.println(tm); //{1=one, 5=five, 10=ten}

            tm.put(1,"gogo");
            System.out.println(tm); //{1=gogo, 5=five, 10=ten}

            tm.put(4,null);
            System.out.println(tm); //{1=gogo, 4=null, 5=five, 10=ten}

            tm.put(null,"four");
            System.out.println(tm); //R.E NullPointerException
      }
}
```

Hashtable
=========
The underlying data structure is Hashtable.

Keys can't be duplicate but values can be duplicate.

Insertion order is not preserved because it will take descending order of key.

Hetrogeneous objects are allowed for both key and value.

Both key and value can't be null.

ex:
--
```
import java.util.*;
class Test
{
	public static void main(String[] args)
	{
		Hashtable ht=new Hashtable();
		ht.put(10,"ten");
		ht.put(1,"one");
		ht.put(5,"five");
		System.out.println(ht); //{10=ten, 5=five, 1=one}

		ht.put(1,"gogo");
		System.out.println(ht); //{10=ten, 5=five, 1=gogo}

		ht.put("six",6);
		System.out.println(ht); //{10=ten, six=6, 5=five, 1=gogo}

		//ht.put(4,null);
		//System.out.println(ht); // R.E NullPointerException

		//ht.put(null,"four");
		//System.out.println(ht); // R.E NullPointerException
	}
}
```

Interview Questions

Q) Write a java program to check given string is balanced or not?

input:
```
	{[()]}
```

output:
```
	It is a balanced string
```

ex:

```
import java.util.*;
class Test
{
	public static void main(String[] args)
	{
		String str="{[()]}";
		//caller method
```

```
            if(isBalanced(str.toCharArray()))
                  System.out.println("It is a balanced string");
            else
                  System.out.println("It is not a balanced string");
      }

      //callie method
      public static boolean isBalanced(char[] carr)
      {
            Stack<Character> s=new Stack<Character>();

            //for each loop
            for(char ch:carr)
            {
                  if(ch=='{' || ch=='[' || ch=='(')
                  {
                        s.push(ch);
                  }
                  else if(ch==')' && !s.isEmpty() && s.peek()=='(')
                  {
                        s.pop();
                  }
                  else if(ch==']' && !s.isEmpty() && s.peek()=='[')
                  {
                        s.pop();
                  }
                  else if(ch=='}' && !s.isEmpty() && s.peek()=='{')
                  {
                        s.pop();
                  }
                  else
                  {
                        return false;
                  }

            }

            return s.isEmpty();
      }
}
```

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

```
input:
      1 2 2 3 3 3 4 4 4 4

output:
      1 2 3 4

ex:

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

```
            Set<Integer> set=new LinkedHashSet<Integer>();

            for(int i:arr)
            {
                  set.add(i);
            }

            set.forEach(element -> System.out.print(element+" "));
      }
}
```

Q) Write a java program to compare two dates?

```
ex:
import java.time.*;
class Test
{
      public static void main(String[] args)
      {
            LocalDate date1=LocalDate.of(2024,1,26);
            LocalDate date2=LocalDate.of(2024,8,15);

            if(date1.compareTo(date2)>0)
                  System.out.println("date1 is biggest");
            else if(date1.compareTo(date2)<0)
                  System.out.println("date2 is biggest");
            else
                  System.out.println("Both are same");

      }
}
```

Q) Write a java program to count occurance of a string?

```
input:
      this is is java java class

output:
      this=1 is=2 java=2 class=1

ex:

import java.util.*;
class Test
{
      public static void main(String[] args)
      {
            String str="this is is java java class";

            String[] sarr=str.split(" ");

            Map<String,Integer> map=new LinkedHashMap<String,Integer>();
```

```
            //for each loop
            for(String s:sarr)
            {
                  if(map.get(s)!=null)
                  {
                        map.put(s,map.get(s)+1);
                  }
                  else
                  {
                        map.put(s,1);
                  }
            }

      map.forEach((key,value) -> System.out.print(key+"="+value+" "));
      }
}
```

4Q) Write a java program to count occurance of a character?

```
input:
      java

output:
      j=1 a=2 v=1

ex:

import java.util.*;
class Test
{
      public static void main(String[] args)
      {
            String str="java";

            char[] carr=str.toCharArray();

            Map<Character,Integer> map=new
LinkedHashMap<Character,Integer>();

            //for each loop
            for(char c:carr)
            {
                  if(map.get(c)!=null)
                  {
                        map.put(c,map.get(c)+1);
                  }
                  else
                  {
                        map.put(c,1);
                  }
            }

map.forEach((key,value) -> System.out.print(key+"="+value+" "));
```

```
	}
}
```

Coursors & Types of Cursors

==========

Cursor is used to read the objects one by one from Collections.

We have three types of cursors.

1) Enumeration

2) Iterator

3) ListIterator

1) Enumeration

Enumeration is used to read objects one by one from legacy Collection objects.

We can create Enumeration object 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);
            }
      }
}
```

Limitation with Enumeration

Enumeration interface is used read objects only from legacy Collection objects.Hence it is not a universal cursor.

Using Enumeration we can perform read operation but not remove operation.

To overcome this limitation Sun Micro System introduced Iterator.

2) Iterator

It is used to read objects one by one from any Collection object.Hence it is a 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)
                      System.out.println(i+" ");
                else
                      itr.remove();
          }

          System.out.println(al);//[2,4,6,8,10]
     }
}
```

Limitation 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 interface we can perform read and remove operation but not adding and replacement of new object.

To overcome this limitation we need to use ListIterator.

3) ListIterator

ListIterator is a child interface of Iterator interface.

ListIterator is used to read objects one by one from List Collection object only.

ListIterator interface is used to perform read, remove , adding and replacement of new object.

We can create ListIterator object by using listIterator() method.

ex:
```
     ListIterator litr=al.listIterator();
```

ListIterator interface contains following 9 methods.

ex:
```
     public abstract boolean hasNext()
     public abstract Object next()
     public abstract boolean hasPrevious()
     public abstract Object previous()
     public abstract int nextIndex()
     public abstract int previousIndex()
     public abstract void remove()
     public abstract void add(E)
     public abstract void set(E)
```

ex:

```
import java.util.*;
class Test
{
```

```
	public static void main(String[] args)
	{
		ArrayList al=new ArrayList();
		al.add("venki");
		al.add("bala");
		al.add("nag");
		al.add("chiru");
		System.out.println(al);//[venki,bala,nag,chiru]

		ListIterator litr=al.listIterator();
		while(litr.hasNext())
		{
			String s=(String)litr.next();
			System.out.println(s);
		}
	}
}
```

ex:

```
import java.util.*;
class Test
{
	public static void main(String[] args)
	{
		ArrayList al=new ArrayList();
		al.add("venki");
		al.add("bala");
		al.add("nag");
		al.add("chiru");
		System.out.println(al);//[venki,bala,nag,chiru]

		ListIterator litr=al.listIterator();
		while(litr.hasNext())
		{
			String s=(String)litr.next();
			if(s.equals("bala"))
			{
				litr.remove();
			}
		}

		System.out.println(al);//[venki,nag,chiru]
	}
}
```

ex:

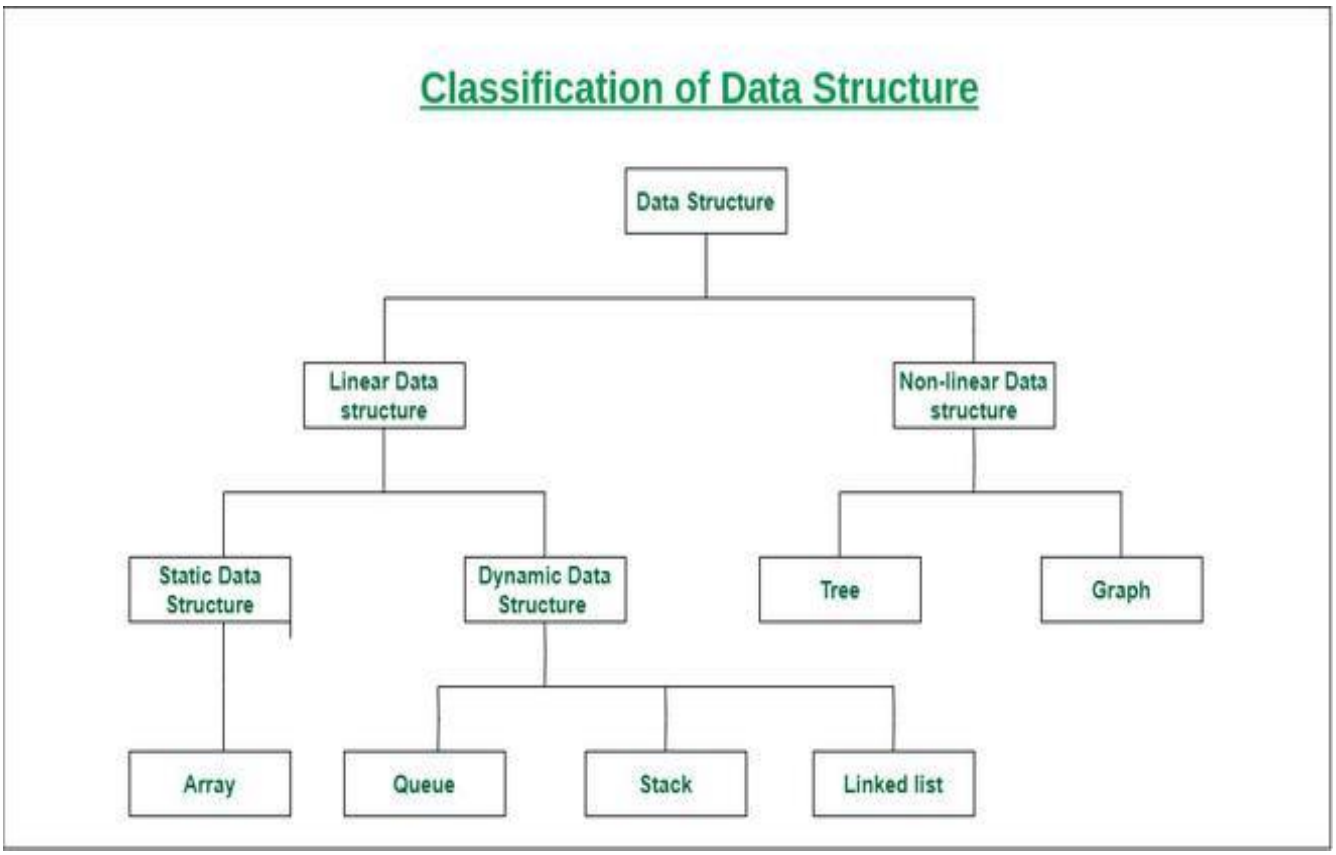
```
import java.util.*;
class Test
{
	public static void main(String[] args)
	{
```

```
            ArrayList al=new ArrayList();
            al.add("venki");
            al.add("bala");
            al.add("nag");
            al.add("chiru");
            System.out.println(al);//[venki,bala,nag,chiru]

            ListIterator litr=al.listIterator();
            while(litr.hasNext())
            {
                  String s=(String)litr.next();
                  if(s.equals("nag"))
                  {
                        litr.add("chaitanya");
                  }
            }

            System.out.println(al);//[venki, bala, nag, chaitanya,
chiru]
      }
}
```

```
ex:
---
import java.util.*;
class Test
{
      public static void main(String[] args)
      {
            ArrayList al=new ArrayList();
            al.add("venki");
            al.add("bala");
            al.add("nag");
            al.add("chiru");
            System.out.println(al);//[venki,bala,nag,chiru]

            ListIterator litr=al.listIterator();
            while(litr.hasNext())
            {
                  String s=(String)litr.next();
                  if(s.equals("chiru"))
                  {
                        litr.set("ramcharan");
                  }
            }

            System.out.println(al);//[venki,bala,nag,ramcharn]
      }
}
```

Q) Types of Data Structures in Java?

Diagram: java43.1



Q) What is the difference between Thread and Process?

Thread

A Thread is a light weight sub process.

We can run multiple threads concurently.

Once thread can communicate with another thread.

ex:

class is one thread
constructor is one thread
block is one thread

Process

A process is a collection of threads.

We can run multiple process concurently.

Once process can't communicate with another process.

ex:

typing the notes in editor is one process
taking a class using zoom meeting is one process
downloading a file from internet is one process

Multitasking

==========
Executing several task simultenously such concept is called multitasking.

We have two types of multitasking.

1) Thread based multitasking

2) Process based multitasking

1) Thread based multitasking

Executing several task simultenously where each task is a same part of a program.

It is best suitable for programmatic level.

2) Process based multitasking

Executing several task simultenously where each task is a independent process.

It is best suitable for OS level.

Multithreading
==========
Executing several threads simultenously such concept is called multithreading.

The main objective of multithreading is to improve the performance of the application/system.

The important application area of multithreading are.

1) To implements multi media graphics.

2) To develop video games.

3) To develop animations.

In multithreading only 10% of work should be done by a programmer and 90% of work will be done by JAVA API.

Ways to create a thread in java
==========
There are two ways to create a thread in java.

1) By extending Thread class

2) By implementing Runnable interface

1) By extending Thread class

```
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)
	{
		//instantiate a thread
		MyThread t=new MyThread();

		//start a thread
		t.start();

		for(int i=1;i<=5;i++)
		{
			System.out.println("Parent-Thread");
		}
	}
}
```

case1: Thread Schedular

If multiple threads are waiting for execution which thread will be executed will decided by Thread schedular.

What algorithm, behaviour or mechanism used by thread schedular is depends upon JVM vendor.

Hence we can't expect any execution order or exact output in multithreading.

case2: Difference between t.start() and to.run() method

If we invoke t.start() method then a new thread will be created which is responsible to execute run() method automatically.

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)
	{
		//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 invoke t.run() method then no new thread will be created but run() method will execute just like a normal method.

ex:

```
class MyThread extends Thread
{
	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();

		//no new thread
		t.run();

		for(int i=1;i<=5;i++)
		{
			System.out.println("Parent-Thread");
		}
	}
}
```

case3: If we won't override run() method

If we won't override run() method then start() method will execute Thread class run() method automatically.

But Thread class run() method is empty implementation.Hence we won't get any output from child thread.

ex:

```
class MyThread extends Thread
{

}
class Test
{
      public static void main(String[] args)
      {
            //instantiate a thread
            MyThread t=new MyThread();

            //new thread
            t.start();

            for(int i=1;i<=5;i++)
            {
                  System.out.println("Parent-Thread");
            }
      }
}
```

case4: If we overload run() method

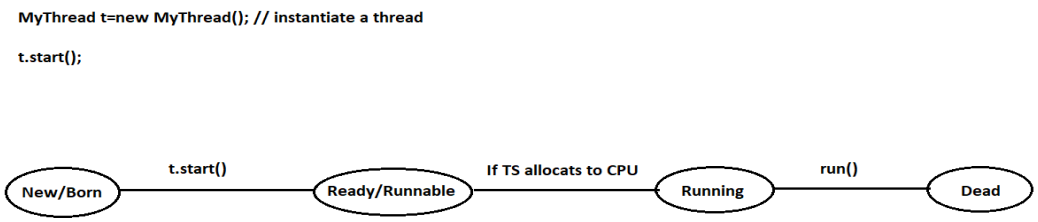
If we overload run() method then Thread class start() method always execute run() method with zero argument only.

ex:

```
class MyThread extends Thread
{
      public void run(int i)
      {
            System.out.println("int-arg method");
      }
      public void run()
      {
            System.out.println("0-arg method");
      }
}
class Test
{
      public static void main(String[] args)
      {
            //instantiate a thread
            MyThread t=new MyThread();

            //new thread
            t.start();

            for(int i=1;i<=5;i++)
            {
                  System.out.println("Parent-Thread");
            }
      }
}
```

case5: Life cycle of a thread

Diagram: java43.2



Once if we create a thread , our thread will be in new or born state.

Once if we call t.start() method our thread will be in ready or runnable state.

If thread schedular allocates to CPU our thread goes to running state.

Once run() method execution is completed our thread enters 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

		//new thread
		t.start();

		for(int i=1;i<=5;i++)
		{
```

```
                System.out.println("Parent-Thread");
            }
        }
}
```

Various ways 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()

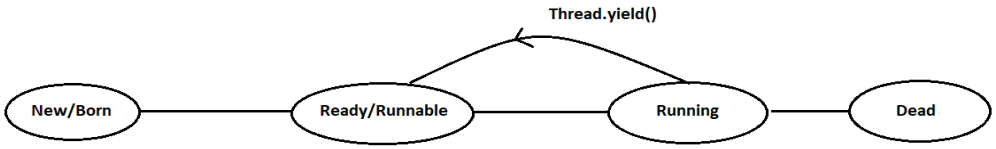
It will pause current execution thread and gives the change to other threads having same priority.

IF there is no waiting threads or low priority threads then same thread will continue it's execution.

If multiple threads having same priority then we can't expect any execution order.

ex:
```
    public static native void yield()
```

Diagram: java44.1



ex:

```
class MyThread extends Thread
{
      public void run()
      {
            for(int i=1;i<=5;i++)
            {
                  Thread.currentThread().yield();
                  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++)
            {
                  System.out.println("Parent-Thread");
            }
      }
}
```

2) join()

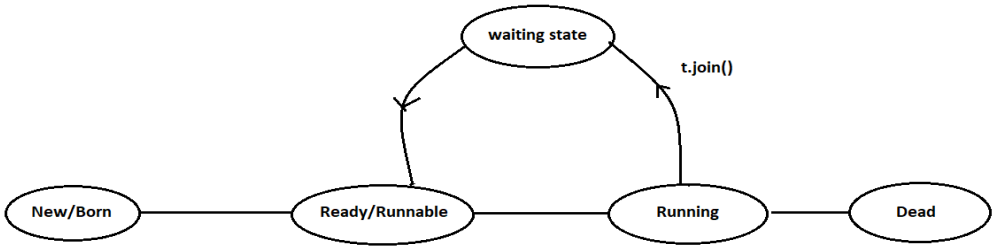
If a thread wants to wait untill the completion of some other thread then we need to use join() method.

A join() method will throw one checked exception called InterruptedException so we must and should handle that exception by using try and catch block or by using throws statement.

ex:

```
      public final void join()throws InterruptedException
      public final void join(long ms)throws InterruptedException
      public final void join(int ns)throws InterruptedException
```

Diagram: java44.2



```
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()

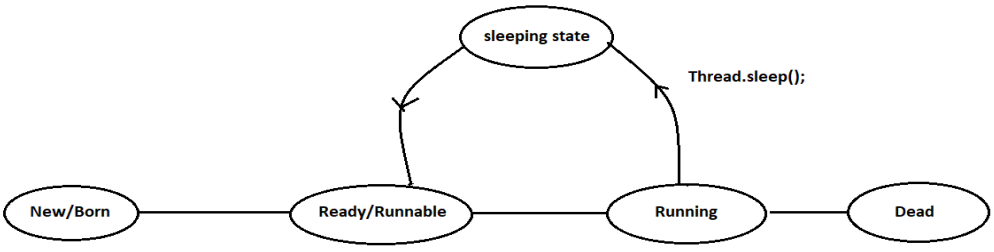
If a thread don't want to perform any operation on perticular amount of time then we need to use sleep() method.

A sleep() method will throw one checked exception so we must and should handle that exception by using try and catch block or by using throws statement.

ex:

```
  public static native void sleep()throws InterruptedException
  public static native void sleep(long ms)throws
   InterruptedException
  public static native void sleep(long ms,int ns)throws
InterruptedException
```

Diagram: java44.3



```
class MyThread extends Thread
{
	public void run()
	{
		for(int i=1;i<=5;i++)
		{
			System.out.println("Child-Thread");
			try
			{
				Thread.sleep(3000);
			}
			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, every thread has a name. Automatically generated by the JVM or explicitly provided by the programmer.

We have following methods to set and get name of a thread.
ex:

```
	public final void setName(String name)
	public final String getName()
```

ex:

```
class MyThread extends Thread
{

}
class Test
{
	public static void main(String[] args)
	{
		System.out.println(Thread.currentThread().getName()); //main

		MyThread t=new MyThread();
		System.out.println(t.getName()); // Thread-0
```

```
            Thread.currentThread().setName("Parent-Thread");

      System.out.println(Thread.currentThread().getName());//Parent-
Thread

            t.setName("Child-Thread");
            System.out.println(t.getName());//Child-Thread
      }
}
```

Thread Priority
==========
In java, every thread has a priority. Explicitly provided by the programmer or automatically generated by JVM.

The valid range of thread priority is 1 to 10. Where 1 is a least priority and 10 is a highest priority.

If we take more the 10 priority then we will get RuntimeException called IllegalArgumentException.

A Thread gave following standard constants as a thread priority.
ex:

```
      Thread.MAX_PRIORITY  - 10
      Thread.NORM_PRIORITY - 5
      Thread.MIN_PRIORITY  - 1
```

We have don't such constants like LOW_PRIORITY and HIGH_PRIORITY.

A thread which is having highest priority will be executed first.

If multiple threads having same priority then we can't expect any execution order.

A Thread schedular uses thread priority while allocating to CPU.

We have following methods to set and get thread priority.
ex:

```
      public final void setPriority(int priority)
      public final int getPriority()
```

ex:

```
class MyThread extends Thread
{

}
class Test
{
      public static void main(String[] args)
      {
            System.out.println(Thread.currentThread().getPriority());
//5

            MyThread t=new MyThread();
            System.out.println(t.getPriority()); // 5
```

```
            Thread.currentThread().setPriority(10);

      System.out.println(Thread.currentThread().getPriority());//10

            t.setPriority(4);
            System.out.println(t.getPriority());//4

            t.setPriority(11); // R.E IllegalArgumentException
      }
}
```

Deamon Thread
==========
Deamon Thread is a service provider thread which provides services to user threads.

Life of deamon thread is depends upon user threads. If user threads died then daemon thread will die automatically.

There are many daemon thread are running internally such as Garbage Collector, Finalizer and etc.

To start the daemon thread we need to use setDeamon(true) method.

To check a thread is a daemon thread or not we need to use isDaemon() method.

ex:
--

```
class MyThread extends Thread
{
      public void run()
      {
            for(int i=1;i<=5;i++)
            {
                  System.out.println(Thread.currentThread().isDaemon());
                  System.out.println("Child-Thread");
            }
      }
}
class Test
{
      public static void main(String[] args)
      {
            MyThread t=new MyThread();
            t.setDaemon(true);
            t.start();
            for(int i=1;i<=5;i++)
            {
                  System.out.println("Parent-Thread");
            }
      }
}
```

Problem without synchronization

==========
If there is no synchronization then we will face following problems.

1) Data inconsistency

2) Thread interference

ex:

```
class Table
{
	public 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)
            {
            sSystem.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.

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 8 Features
==========

Functional interface
==========
Functional interface introduced in Java 8 version.

Interface which contains only one abstract method is called functional interface.

It can have any number of default methods and static methods.

It is also known as SAM interface or Single Abstract Method interface.

The main objective of functional interface is to achieve functional programming.

ex:
```
a = f1(){}

f1(f2(){})
{
}
```

@FunctionalInterface annotation is used to declare functional interface and it is optional.

ex:

```
@FunctionalInterface
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)
	{
		A a=new B();
		a.m1();
	}
}
```

ex:

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

Lamda Expression
==========
Lamda expression introduced in Java 8 version.

Lamda expression is used to concise(reduce) the code.

We can use lamda expression when we have functional interface.

Lamda expression consider as method.

The main objective of lamda expression is to achieve functional programming.

Lamda expression does not allow name, returntype and modifier.

ex:

Java Method

```
public void m1()
{
	System.out.println("Hello World");
}
```

Lamda expression

```
()->
{
	System.out.println("Hello World");
};
```

ex:

```
@FunctionalInterface
interface A
{
	public abstract void m1();
}
class Test
{
	public static void main(String[] args)
	{
		A a=()->
			{
				System.out.println("M1 Method");
			};

		a.m1();
	}
}
```

ex:

```
@FunctionalInterface
interface A
{
	public abstract void m1(int i,int j);
}
class Test
{
	public static void main(String[] args)
	{
		A a=(int i,int j)->
			{
				System.out.println(i+j);
			};

		a.m1(10,20);
	}
}
```

ex:

```
@FunctionalInterface
interface A
{
	public abstract int m1(int i,int j);
}
class Test
{
	public static void main(String[] args)
	{
		A a=(int i,int j)->
			{
				return i+j;
			};
```

```
            System.out.println(a.m1(20,30));
      }
}
```

Default methods in interface

==========

Java provides facility to declare default methods in interface from Java 8 version.

If we declare an method inside the interface and tagged with default keyword is called default method.

Default method is a non-abstract method.

Default methods can be override.

ex:

```
interface A
{
      public abstract void m1();

      default void m2()
      {
            System.out.println("M2-Method");
      }
}
class B implements A
{
      public void m1()
      {
            System.out.println("M1-Method");
      }
}

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

ex:

```
interface A
{
      public abstract void m1();

      default void m2()
      {
            System.out.println("M2-Method");
      }
}
class B implements A
```

```
{
	public void m1()
	{
		System.out.println("M1-Method");
	}

	public void m2()
	{
		System.out.println("Override-M2-Method");
	}
}

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

To achieve multiple inheritance in java we need to use default methods of an interface.

ex:

```
---
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:

```
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:

```
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:
---
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

==========

Java provides facility to declare static methods in interface from Java 8 version.

If we declare an method inside the interface and tagged with static keyword is called static method.

Static method is a non-abstract method.

Static methods can't be override.

ex:

```
interface A
{
      static void m1()
      {
            System.out.println("M1-Method");
```

```
	}
}

class Test
{
	public static void main(String[] args)
	{
		A.m1();
	}
}
```

Stream API

==========

Stream is an interface which is present in java.util.stream package.

Stream API is used to perform bulk operations on Collections.

Stream API is used to process the objects from Collections.

ex:

```
import java.util.*;
import java.util.stream.*;
class Test
{
	public static void main(String[] args)
	{
		List<Integer> list=Arrays.asList(5,7,1,3,8,4,9);

		List<Integer> newList=list.stream().filter(i-
>i%2==0).collect(Collectors.toList());

		System.out.println(newList);
	}
}
```

ex:

```
import java.util.*;
import java.util.stream.*;
class Test
{
	public static void main(String[] args)
	{
		List<Integer> list=Arrays.asList(5,7,1,3,8,4,9);

		List<Integer> newList=list.stream().filter(i-
>i%2!=0).collect(Collectors.toList());

		System.out.println(newList);
	}
}
```

ex:

--

```
import java.util.*;
import java.util.stream.*;
class Test
{
	public static void main(String[] args)
	{
		List<Integer> list=Arrays.asList(5,7,1,3,8,4,9);

		long odd=list.stream().filter(i->i%2!=0).count();

		System.out.println(odd);
	}
}

ex:
---
import java.util.*;
import java.util.stream.*;
class Test
{
	public static void main(String[] args)
	{
		List<Integer> list=Arrays.asList(5,7,1,3,8,4,9);

		List<Integer> newList=list.stream().map(i-
>i+10).collect(Collectors.toList());

		System.out.println(newList);
	}
}

ex:
---
import java.util.*;
import java.util.stream.*;
class Test
{
	public static void main(String[] args)
	{
		List<Integer> list=Arrays.asList(5,7,1,3,8,4,9);

		List<Integer>
newList=list.stream().sorted().collect(Collectors.toList());

		System.out.println(newList);
	}
}

ex:
---
import java.util.*;
import java.util.stream.*;
class Test
{
	public static void main(String[] args)
	{
		List<Integer> list=Arrays.asList(5,7,1,3,8,4,9);
```

```
            List<Integer>
newList=list.stream().sorted(Comparator.reverseOrder()).collect(Collect
ors.toList());

            System.out.println(newList);
      }
}
```

ex:

```
---
import java.util.*;
import java.util.stream.*;
class Test
{
      public static void main(String[] args)
      {
            List<Integer> list=Arrays.asList(5,7,1,3,8,4,9);

            long minimum=list.stream().min((i1,i2)-
>i1.compareTo(i2)).get();

            System.out.println(minimum);
      }
}
```

ex:

```
---
import java.util.*;
import java.util.stream.*;
class Test
{
      public static void main(String[] args)
      {
            List<Integer> list=Arrays.asList(5,7,1,3,8,4,9);

            long maximum=list.stream().max((i1,i2)-
>i1.compareTo(i2)).get();

            System.out.println(maximum);
      }
}
```

forEach() Method

```
==================
```

A forEach() method introduced in Java 8 version.

It is used to iterate the objects from Collections.

ex:

```
---
import java.util.*;
class Test
{
      public static void main(String[] args)
      {
            List<Integer> list=Arrays.asList(5,7,1,3,8,4,9);
```

```
            list.forEach(element-> System.out.print(element+" "));
      }
}

ex:
---
import java.util.*;
class Test
{
      public static void main(String[] args)
      {
            Map<Integer,String> map=new LinkedHashMap<Integer,String>();
            map.put(1,"one");
            map.put(2,"two");
            map.put(3,"three");

            map.forEach((key,value)->
System.out.println(key+"="+value));
      }
}
```

Method Reference (::)

```
=====================
```

Method reference introduced in Java 8 version.

Method reference is a special type of lambda expression.

```
ex:
---
import java.util.*;
class Test
{
      public static void main(String[] args)
      {
            List<Integer> list=Arrays.asList(5,7,1,9,2,3);

            list.forEach(System.out::println);
      }
}
```

Interview Question

```
==================
```

Q) Write a java program to display employee records in ascending order of employee id?

```
import java.util.*;
import java.util.stream.*;

class Employee
{
      private int empId;
      private String empName;
      private double empSal;

      //parameterized constructor
```

```
	Employee(int empId,String empName,double empSal)
	{
		this.empId=empId;
		this.empName=empName;
		this.empSal=empSal;
	}

	public int getEmpId()
	{
		return empId;
	}
	public String getEmpName()
	{
		return empName;
	}
	public double getEmpSal()
	{
		return empSal;
	}
}
class Test
{
	public static void main(String[] args)
	{
		List<Employee> list=new ArrayList<Employee>();

		list.add(new Employee(104,"Jack",4000d));
		list.add(new Employee(101,"Nancy",1000d));
		list.add(new Employee(103,"Jose",3000d));
		list.add(new Employee(102,"Linda",2000d));

		List<Employee>
newList=list.stream().sorted(Comparator.comparingInt(Employee::getEmpId
)).collect(Collectors.toList());

		newList.forEach(employee->
System.out.println(employee.getEmpId()
			+" "+employee.getEmpName()+" "+employee.getEmpSal()));
	}
}
```

Q) Write a java program to display employee records in ascending order of employee name?

```
import java.util.*;
import java.util.stream.*;

class Employee
{
	private int empId;
	private String empName;
	private double empSal;

	//parameterized constructor
	Employee(int empId,String empName,double empSal)
	{
		this.empId=empId;
		this.empName=empName;
```

```
            this.empSal=empSal;
      }

      public int getEmpId()
      {
            return empId;
      }
      public String getEmpName()
      {
            return empName;
      }
      public double getEmpSal()
      {
            return empSal;
      }
}
class Test
{
      public static void main(String[] args)
      {
            List<Employee> list=new ArrayList<Employee>();

            list.add(new Employee(104,"Jack",4000d));
            list.add(new Employee(101,"Nancy",1000d));
            list.add(new Employee(103,"Jose",3000d));
            list.add(new Employee(102,"Linda",2000d));

            List<Employee>
newList=list.stream().sorted(Comparator.comparing(Employee::getEmpName)
).collect(Collectors.toList());

            newList.forEach(employee->
System.out.println(employee.getEmpId()
                  +" "+employee.getEmpName()+" "+employee.getEmpSal()));
      }
}
```

Garbage Collector

==========

It is a daemon thread which is responsible to destroy unused and useless object from java.

There are two ways to call garbage collector in java.

1) System.gc()

2) Runtime.getRuntime().gc()

ex:

```
class Test
{
      int i=10;

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

```
            Test t1=new Test();
            System.out.println(t1.i); // 10

            t1=null;
            System.gc();
      }

      public void finalize()
      {
            System.out.println("Method Called");
      }
}

ex:
---

class Test
{
      int i=10;

      public static void main(String[] args)
      {
            Test t1=new Test();
            System.out.println(t1.i); // 10

            t1=null;
            Runtime.getRuntime().gc();
      }

      public void finalize()
      {
            System.out.println("Method Called");
      }
}
```