Institute Name	IHUB TALENT MANAGEMENT
Website	www.ihubtalent.com
Location	Ameerpet , Hyderabad
Batch No	IH-JAVA-026
Course Name	Full Stack Java Development Course + AWS
Duration	4 Months
Mode	Online/Offline
Free	Recorded videos
Whatsapp group	IH-JAVA-026
Benefits	Aptitute classes , soft skills classes
Project	Employee Management System (ReactJS + Spring Boot)

```
                Full Stack Java Development Course
                  (Full Stack Developer)
                        |
            |----------------------------------------|
      FrontEnd technologies              BackEnd technologies
      (FrontEnd developer)               (BackEnd developer)

      > HTML5                            > Core Java
      > CSS3                             > Advanced Java
      > JavaScript                             > JDBC
      > Bootstrap5                             > Servlets
      > React/ReactJS                    > JSP
                                   > Oracle DB
                                         > SQL
                                         > PL/SQL
                                   > Frameworks
                                         > Spring Boot
                                         > Microservices
```

Programming language

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

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

Diagram: introduction1.1

Java

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

1995 --> James Gosling --> Sun Micro system --> Oracle Corporation
Java software --> 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 ?
Java 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.

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

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

Diagram: introduction2.1

Programming language divided into two types.

1)Low Level Language

2)High Level Language

1)Low Level Language

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

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

ex:
      Machine Language
      Assembly Language

Machine Language

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

It is also known as binary language.

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

Advantages:

> A program writtens in machine language consumes less memory.

> It does not required any translator.

> It is more efficient when compare to other languages.

Disadvantages:

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

> Whenever error raised in our program then locating and handling that error becomes difficult.

> Modifications can't be done easily.

Assembly Language

The second generation language came into an existence is called assembly language.

Assembly language is a replacement of symbols and letters for mathematical programming code i.e opcode values.

It is also known as symbolic language.

Assembly language can't understand by a computer directly.We required translator.

We have three translators.

i)Assembler

ii)Compiler

iii)Interpreter

i)Assembler

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

Merits:

> If anywhere error raised in our program locating and handling that error becomes easy.

> Modifications can be done easily.

Demerits:

> It is a mind trick to remember symbolic code.

> It requires translator.

> It is less efficient when compare to machine language.

Q)What is Debugging?

Bug is also known as Error.

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

2)High level language

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

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

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

Computer will not understand high level language directly. We required translators.

compiler

It will compile and execute our program at at time.

interpreter

It will execute our program line by line procedure.

Adantages:

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

> Debugging can be done easily.

> Modifications can be done easily.

Disadvantages:

> It requires translator.

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

> It is not efficient when compare to low level language.

Types of IT companies
=====================

1) Service Based companies

Companies those who provides services to client/customer.
ex:
Cognizent, Capgemini, TCS and etc.

2) Product Based companies

Companies those who have their own products to sell in the market.
ex:
Oracle Corporation, Microsoft, Amazon and etc.

Escape Characters / Escape Sequences
====================================
Escape characters are used to design our output in neat and clean manner.

Every escape character starts with back slash(\) followed by a character.
ex:
```
	\n
```

Mostly every escape character is placed inside 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  Ashwini
{
	public static void main(String[] args)
	{
		System.out.println("IHUB\nTALENT");
	}
}
o/p:
	IHUB
	TALENT

2) \t (horizontal tab)
------------------------
class Badrinath
{
	public static void main(String[] args)
	{
		System.out.println("IHUB\tTALENT");
	}
}
o/p:
	IHUB  TALENT



3) \b (back space)
-------------------
class Radha
{
	public static void main(String[] args)
	{
		System.out.println("I\bHUBTALENT");
	}
}
o/p:
	HUBTALENT

ex:
----
class Ajay
{
	public static void main(String[] args)
	{
		System.out.println("IHUB\b\b\bTALENT");
	}
}
o/p:
      ITALENT

4) \r (carriage return)
-------------------------
class Prakash
{
      public static void main(String[] args)
      {
            System.out.println("IHUB\rTALENT");
      }
}
o/p:
      TALENT

ex:
---
class Manisha
{
      public static void main(String[] args)
      {
            System.out.println("TALENT\rIHUB");
      }
}
o/p:
      IHUBNT

6) \\ (back slash)
--------------------
class Jyothi
{
      public static void main(String[] args)
      {
            System.out.println("IHUB\\TALENT");
      }
}
o/p:
      IHUB\TALENT


7) \" (double quote)
----------------------
class Chandu
{
      public static void main(String[] args)
      {
            System.out.println("IHUB\"TALENT");
      }
}
o/p:
      IHUB"TALENT


8)\' (single quote)
-------------------
class Doni
{
      public static void main(String[] args)
      {
            System.out.println("IHUB'TALENT");
            System.out.println("IHUB\'TALENT");
      }
}
o/p:
      IHUB'TALENT
```

C program
=========

Q)Write a c program to display %d ?

```
void main()
{
	clrscr();

	printf("%d");	//0

	getch();
}
```

ex:

```
void main()
{
	clrscr();

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

	getch();
}
```

Screening Test program
=======================
Q)What will be the output of below code?

```
class Example
{
	public static void main(String[] args)
	{
		System.out.print("\nle");
		System.out.print("\bpi");
		System.out.print("\rha");
	}
}
```

o/p:

```
	hai
```

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.

Comments will not display in output because they won't compile 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 Test
{
	//main method
	public static void main(String[] args)
	{

		//variable declaration
		int x=10;

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

Q)What is the difference between Python and Java?

Python	Java
It is a product of Microsoft.	It is a product of Oracle Corporation.
It is developed by Guido Van Rossum.	It is developed by James Gosling.
It is a scripting language.	It is a object oriented programming language.
It is a 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.
It is less secure.	It is highly secured.
Performance is low.	Performance is high.
It contains less code.	It contains more code.

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

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

Every project contains two domains.

1)Technical Domain

It describes which technology we developed our project.
ex:
Java

2)Functional Domain

It describes state of a project.
ex:
- Healthcare domain
- Banking domain
- Insaurance domain
- ERP domain
- 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 programming 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 a JVM.
It supports multiple inheritance.	It does not support multiple inheritance.
It supports pointers.	It does not support pointers.
It supports preprocessor directory(#).	It does not support preprocessor directory(#).
It contains three access specifiers i.e public,private and protected.	It contains four access modifiers i.e default,public,private and protected.
It contains three types of loops i.e do while loop, while loop and for loop.	It contains four types of loops i.e do while loop, while loop, for loop and for each loop.

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 a platform independent.
It contains less security.	It contains high security.
To develop medium scale projects we need to use .net.	To develop major scale projects we need to use java.
It contains small set of frameworks. ex: ASP.net (Actice Server Pages) ASP.net MVC (Model View Controller)	It contains large set of frameworks. ex: Hibernate Spring Framework

```
                                        Spring Boot
                                        Microservices
                                        Struts
                                        spring cloud
                                        spring security
                                       and etc.
```

Q)What are the features of Java?

We have following important features in java.

1) Simple

2) Object oriented

3) Platform independent

4) Portable

5) Highly secured

6) Architecture Neutral

7) Robust

8) Multithreaded

9) Dynamic

10) Distributed

and etc.

Q)Who is the responsible to destroy the objects in java?

Garbage collector

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

There are two ways to call garbage collector in java.

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

Q) What is package?
Package is a collection of classes and interfaces.

Modules in java
===============
We have three modules in java.

```
                        Java
      |-------------------------|------------------------|
     JSE/J2SE                  JEE/J2EE        JME/J2ME
(Java Standard Edition) (Java Enterprise Edition) (Java Micro Edition)

> Standalone app  > Distributed App       > Mobile App

> Desktop app          > Enterprises App
> Two-tier app          > ERP-App

                  > N-Tier App


> Standalone app
-------------------
A normal java program which contains main method is called standalone
application.
ex:
      class Test
      {
            public static void main(String[] args)
            {
                  -
                  -
                  -
            }
      }

> Desktop app
--------------
It is a software application which is developed 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: java3.1
Diagram: java3.2


> Distributed App
------------------
In client-server application, if multiple clients sending the request to main
server then main server will distribute the request to it's parallel servers to
reduce the burdun of main server such type of application is called distribtued
application.

Diagram: java3.3

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

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

ex:
      Facebook
      Online shopping websites

> ERP-App
-----------
ERP stands for Enterprise Resource Planning.
It is used to maintain the data in a enterprise.
```

Diagram: java3.4

> N-Tier App

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

Diagram: java3.5

> Mobile App

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

ex:
Gpay
PhonePay
TempleRun
and etc.

Now a days, To develop mobile applications we are using following technologies.
ex:
Andriod
React Native
Flutter
Swift Programming

Naming conventions in java
===========================
In java uppercase letters will consider as different and lowercase letters will consider as different that's why we consider Java is a case sensitive programming.

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 must and should starts with uppercase letter and if it contains multiple words then each inner word starts with initcap.
ex:

Predefined classes	Userdefined classes
System	Test
File	ExampleApp
FileWriter	DemoApp
BufferedReader	QualityThought
and etc.	and etc

interfaces

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

ex:

predefined interfaces	Userdefined interfaces
Serializable	ITest
Enumeration	IDemoApp
ListIterator	IExampleApp
Runnable	IQualityThought
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
out	empId
in	studName
err	deptNo
length	salary
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 starts with initcap.

ex:

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

keywords

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

ex:

predefined keywords

if, else , switch, do , while, for , public , static and etc.

packages

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

ex:

predefined packages	userdefined packages
java.lang (default pkg)	com.google.www
java.io	com.ihub.www
java.time	com.qt.www
java.util	and etc.
java.util.stream	
java.text	
java.sql	
javax.servlet	
and etc.	

```
constants
----------
In java , all constants we need to declare in uppercase letters only.
ex:
      predefined constants          userdefined constants
      -----------------       --------------------
      MAX_PRIORITY                  LIMIT
      MIN_PRIORITY                  IHUB_TALENT
      MAX_VALUE               and etc.
      MIN_VALUE
      and etc.
```

```
Interview Questions
====================

Q)Which package is a default package in java?

      java.lang is a default package in java.


Q)How many classes are there in java?

      Java 7      ------      4024 classes
      Java 8  ------    4240 classes
      Java 9  ------  6005 classes
      Java 10 ------  6002 classes

Q)What is package ?

      Package is a collection of classes and interfaces.
```

```
Assignment
============
1) class          :     QualityThought

2) Interface      :     IQualityThought

3) Variable       :     qualityThought

4) Method         :     qualityThought()

5) package        :     com.qualitythought.www

6) Constant       :     QUALITYTHOUGHT / QUALITY_THOUGHT
```

History of Java
===============
In 1990, Sun Micro System took one project to develop a software called consumer electronic device which can be controll 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.

In 1991 , they have developed a programming language called an OAK. OAK means strength , itself is a coffee seed name and it is a national tree for many contries like Germany, France , USA and etc.

In 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 symbol of java is a cup of coffee with saucer.

Interview Questions

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

JDK

JDK stands for Java Development Kit.

It is a installable software which consist Java Runtime Environment (JRE), Java Virtual Machine (JVM), Compiler (javac) , Interpreter (java), achiever (.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.
JVM is an interpreter which is used to execute our program line by line procedure.

Diagram: java5.1

Q) Is JVM platform dependent or independent?

JVM is platform dependent.

Q) In which year java was developed?

In 1995

Q) Who is the creator of Java ?

James Gosling

Q) Java originally known as ___?

OAK

Java

Version	:	Java 8
JDK	:	1.8
Open source	:	Open source

```
Creator          :     James Gosling

Vendor           :     Oracle Corporation

website          :     www.oracle.com/in/java/

Download link   :

https://drive.google.com/file/d/16fr2McV_Bex0NYlOdcVfC4k2gwUUNqzq/view?
usp=share_link


Steps to setup Java Environmental variables
===========================================
step1:
------
      Make sure JDK 1.8 installed successfully.

step2:
------
      Copy "lib" directory from java_home folder.
      ex:
            C:\Program Files\Java\jdk1.8.0_181\lib

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

            user varaibles  --> click to new button -->

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

            system variables --> click to new button -->

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

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

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

Steps to develop first application in java
============================================
step1:
-----
      Make sure JDK 1.8 installed successfully.

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

step3:
-----
```

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


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

step5:
-----
      Save above program with same name as class name inside "javaprog"
      folder.

step6:
-----
      Open the command prompt from "javaprog" location.

step7:
------
      Compile the program by using below command.
      ex:
            javaprog> javac   Test.java
                              |
                              filename

step8:
------
      Run the program by using below command.
      ex:
            javaprog> java    Test
                           |
                        classname


Internal Architecture of JVM
============================
Diagram: java6.1
```

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

JVM will invoke one module called classloader/sub system to load all the bytes 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 allocate memory.

We have five types of memories in java.

1)Method Area

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

2)Heap

Our object creations will store in heap area.

Note:

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

3)Java Stack

Java methods will store in method area but to execute those methods we required some memory.That memory will be allocated in java stack.

4)PC Register

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

5)Native Method Stack

Java methods will execute in method area.
Similarly native methods will execute in native method stack.
But native methods we can't execute directly.We need to take the support a program called Native method interface.

Execution engine

Execution engine contains interpreter and JIT compiler.

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

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

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

Q) How many memories are there in java.

We have five memories in java.

1)Method Area
2)Heap
3)Java Stack
4)PC Register
5)Native method stack

Q)What is Native method in java?

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

Q)What is JIT compiler?

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

Q)How many classloaders are there in java?

We have three predefined classloaders in java.

1)Bootstrap classloader (It loads rt.jar file)

2)Extension classloader (It loads all the jar files from ext folder)

3)Application/System classloader (It loads .class file from classpath)

Datatypes

==========

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

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

In java, datatypes are divided into two types.

Diagram: java7.1

byte

It is a smallest datatype in java.

Size: 1 byte (8 bits)

Range : -128 to 127 (-2^7 to 2^7-1)

ex:
```
      1) byte b=10;
           System.out.println(b); //10

      2) byte b=140;
           System.out.println(b); //C.T.E

      3) byte b=10.5;
           System.out.println(b); //C.T.E
```

short

It is a rarely used datatype in java.

Size : 2 bytes (16 bits)

Range : -32768 to 32767 (-2^15 to 2^15-1)

ex:
```
      1) byte b=10;
         short s=b;
         System.out.println(s); //10

      2) short s=10.5;
         System.out.println(s); //C.T.E

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

int

It is mostly used datatype in java.

Size : 4 bytes (32 bits)

Range : -2147483648 to 2147483647 (-2^31 to 2^31-1)

ex:

```
      1) int i=10.5;
```

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

      2) int i="hi";
         System.out.println(i); //C.T.E

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

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

Note:
-----
In java, For every character we have universal unicode value.
ex:
      a = 97
      A = 65

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

Size: 8 bytes (64 bits)

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

ex:

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

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

      3) long l=10.4;
         System.out.println(l); // C.T.E

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

float                         double
--------                      --------
If we need 4 to 6 decimal point of If we need 14 to 16 decimal point of
accuracy then we need to use float. accuracy then we need to use double.

Size: 4 bytes (32 bits)            Size: 8 bytes (64 bits)

Range: -3.4e38 to 3.4e38           Range: -1.7e308 to 1.7e308.

To declare a float value we need to       To declare a double value we need to
suffix with 'f'.              suffix with 'd'.
ex:                           ex:
      10.5f;                              10.5d;

ex:
---
      1) float f=10;
         System.out.println(f); //10.0

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

      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;
         System.out.println(d); //10.0

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

      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

A boolean datatype will accept boolean values either true or false.

Size: (Not Applicable)

Range : (Not Applicable)

ex:

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

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

      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 c="a";
         System.out.println(c); // C.T.E

      2) char c='a';
         System.out.println(c); // a

      3) char c=100;
         System.out.println(c); // d
```

Diagram: java7.2

Q)Write a java program to display byte range?

range : -128 to 127

ex:

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

Q)Write a java program to display int range?

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);
      }
}
```

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 and x are identifiers.

Rules to declare an identifier

Rule1:

Identifier will accept following characters.

ex:

```
      A-Z
      a-z
      0-9
      _
      $
```

Rule2:

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

```
      ex:
            int $=10;  //valid
            int _abcd; //valid
            int ab_cd; //valid
            int ab#cd; //invalid
            int @=10;  //invalid

Rule3:
-----
      Every identifier must and should starts with alphabet, underscore
      or dollar symbol but not with digit.
      ex:
            int  a1234; //valid
            int  _1234; //valid
            int  1abcd; //invalid

Rule4:
-----
      We can't take reserved words as an identifier.
      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 is 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: java8.1

Used keywords with respect to class
----------------------------------
package
import
class
interface
enum
extends
implements

Used keywords with respect to object
---------------------------------
new
```

instanceof
this
super

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

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

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

Used keywords with respect to returntype
----
void

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

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 variable / Non-static variable

2)Static variable / Global variable

3)Local variable / Temperory variable / Automatic variable

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

Instance variable will create memory 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 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
{
	public static void main(String[] args)
	{
		//calling
		Test t=new Test();
		t.m1();
	}

	//non-static method
	public void m1()
	{
		System.out.println("Instance-Method");
	}
}
```

2)Static variable

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

A 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 .class file.

Static variable will store in method area.

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

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

Static variable can access by using object reference or class name.

ex:1

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

	public static void main(String[] args)
	{
		System.out.println(i); //10

		Test t=new Test();
		System.out.println(t.i);//10

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

Note:

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

ex:2

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

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

ex:3

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

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

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

}
```

3)Local variable

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

Local variable will be created when execution block is declared 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 memory.

ex:

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

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

Note:

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

ex:2

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

            System.out.println(i);  //C.T.E
      }
}
```
o/p:
```
      variable i might not have been initialized
```

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

ex:

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

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

Interview Question
==================
Q)What is Literal?

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

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

ex:
```
      int  x = 10;
      |    |       |____ value of a variable / Literal
      |    |________ variable name / Identifier
      |_____________ datatype / Keyword
```

```
Q)Is Java purely object oriented or not?

No , Java never consider as purely object oriented programming language because
it does not support many OOPS concepts multiple inheritance, operator
overloading and more ever we depends upon primitive datatypes which are non-
objects.



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.

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 we will get runtime error
called main method not found.

public
-----
      JVM wants to call this method from anywhere.

static
------
      JVM wants to call this method without using object reference.

void
----
      Main method 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 because instead of public static we can
take static public also.
      ex:
             static public void main(String[] args)

2) We can change String[] in following acceptable formats.
      ex:
            public static void main(String[] args)
            public static void main(String  []args)
            public static void main(String  args[])

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

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

5) Main method will accept following modifiers.
```

```
      ex:
            synchronized
            strictfp
            final
```

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

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

```
ex:
      javac    Test.java

      java     Test  101  raja  M  1000.0
                  |    |        |   |____ args[3]
                  |    |        |__________ args[2]
                  |    |_______________ args[1]
                  |____________________ args[0]
```

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

      }
}
```

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

```
class Test
{
      public static  void main(String[]  args)
      {
            System.out.println("Enter the Name :");
            String name=args[0];
            System.out.println("Welcome :"+name);
      }
}

javac   Test.java

java    Test  LinusTorvalds
```

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

If we want to display any userdefined statements and data then we need to use output statement.

```
syntax:
-----
      static variable
            |
      System.out.println();
      |           |
      predefined  predefined method
      final
      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);
```

3)
```
      int i=10,j=20;
      System.out.println(i+" "+j);
      System.out.println(i+" and "+j);
```

4)
```
      int i=1,j=2,k=3;
      System.out.println(i+" "+j+" "+k);
```

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

Fully Qualified Name
====================
Full qualified name will increase readability of our code.

We will declare class name and interface name along with package name.

ex:
```
            java.lang.System
            java.lang.String
            java.lang.Object
```

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 statement then we should not use fully qualified name.

Import statement tells to the compiler, path of a class or interface.

It is pretty good and recommanded over the fully qualified name.

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.time.LocalDate;
import java.time.LocalTime;
class Test
{
      public static void main(String[] args)
      {
            LocalDate date=LocalDate.now();
            System.out.println(date); // 2023-10-30

            LocalTime time=LocalTime.now();
            System.out.println(time); // 10:51:23
      }
}
```

2)Implicit class import

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

ex:

```
import java.time.*;
class Test
{
      public static void main(String[] args)
      {
            LocalDate date=LocalDate.now();
            System.out.println(date); // 2023-10-30

            LocalTime time=LocalTime.now();
            System.out.println(time); // 10:51:23
      }
}
```

3)Static import

Using static import we can call static members 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");
	}
}
```

Basic Java Programs
===================

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

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

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

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

		//logic
		int c=a+b;

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

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

```
import java.util.Scanner;
class Example2
{
	public static void main(String[] args)
	{
```

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

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

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

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

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

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

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

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

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

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

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

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

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

Q)Write a java program to 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();

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

```
	}
}
```

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

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

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

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

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

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

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

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

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

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

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

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

	}
}
```

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

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

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

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

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

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

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

      }
}
```

Q)Write a java program to accept employee salary and find out 10 percent
  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 employee 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 find out 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  iss ="+percentage);

      }
}
```

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

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

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

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

In java, typecasting can be done in two ways.

1)Implicit typecasting

2)Explicit typecasting

1)Implicit typecasting
-----------------------
If we want to store smaller value into a bigger variable then we need to use
implicit typecasting.

A compiler is responsible to perform implicit typecasting.

There is no possibility to loss the information.

It is also known as widening or upcasting.

We can perform implicit typecasting as follow.

ex:
      byte -->short
                  -->
                        int -->     long  ---> float ---> double
                  -->
            char

ex:1
-----
class Test
{
      public static void main(String[] args)
      {
            byte b=10;

            long l=b;

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

ex:2
-----
class Test
{
      public static void main(String[] args)
      {
            char ch='a';

            int i=ch;

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

ex:3
---
class Test
{
      public static void main(String[] args)
      {
            int i=10;

            float f=i;

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

2)Explicit typecasting

If we want to store bigger value into a smaller variable then we need to use explicit typecasting.

A programmer is responsible to perform explicit typecasting.

There is a possibility to loss the information.

It is also known as Narrowing or Downcasting.

We can perform explicit typecastin as follow.

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

ex:1

```
class Test
{
      public static void main(String[] args)
      {
            float f=10.5f;

            int i=(int)f;

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

ex:2

```
class Test
{
      public static void main(String[] args)
      {
            int i=65;

            char ch=(char)i;

            System.out.println(ch);// A
      }
}
```

ex:3

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

            byte b=(byte)i;

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

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

Instance block is used to intialize the instance variables.

Instance block will execute at the time object creation.

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

syntax:

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

ex:1

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

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

ex:2

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

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

ex:3

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

	public static void main(String[] args)
	{
		Test t1=new Test();
		System.out.println("main-method");
		Test t2=new Test();
	}
}
o/p:

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

ex:4

```
class Test
{
	//instance 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 static variables.

A static block will execute at the time of class loading.

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

syntax:
```
	//static block
	static
```

```
      {
            -
            - //set of stmts
            -
      }

ex:1
----
class Test
{
      //static block
      static
      {
            System.out.println("static-block");
      }

      public static void main(String[] args)
      {
            System.out.println("main-method");
      }
}
o/p:
      static-block
      main-method

ex:2
------
class Test
{
      //instance block
      {
            System.out.println("instance-block");
      }
      //static block
      static
      {
            System.out.println("static-block");
      }

      public static void main(String[] args)
      {
            System.out.println("main-method");
            Test t=new Test();
      }
}
o/p:
      static-block
      main-method
      instance-block

ex:3
----
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 variable.

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

A local block will execute just like normal statement.

syntax:
```
	//local block
	{
		-
		- //set of stmts
		-
	}
```

ex:1

```
class Test
{
	public static void main(String[] args)
	{
		System.out.println("stmt1");
		//local block
		{
			System.out.println("stmt2");
		}
		System.out.println("stmt3");
	}
}
```
o/p:
```
	stmt1
	stmt2
	stmt3
```
ex:2

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

		//local block
		{
			i=300;
		}
		System.out.println(i);
	}
}
```

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

Yes , Till java 6 verion it possible to execute java program without main method by using static block.But from java 7 version onwards it is not possible to execute java program without main method.

```
ex:
class Test
{
      //static block
      static
      {
            System.out.println("static-block");
            System.exit(0);
      }
}
```

Operators
==========
Operator is a symbol which is used to perform some operations on operands.
ex:

```
      c = a + b;

      Here a,b & c are operands.
      Here = and + are operators.
```

It can be arithemetic operation, logical operation, bitwise operation and etc.

We have following list of operators in java.

1)Assignment operators

2)Ternary/Condtional operators

3)Bitwise operators

4)Logical operators

5)Relational operators

6)Shift operators

7)Arithmetic 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:
      Reinitialization is possible in java.

ex:

```
class Test
{
      public static void main(String[] args)
      {
            final int i=10;
            i=20;
            i=30;
            System.out.println(i);
      }
}
o/p:
      C.T.E cannot assign a value to final variable

ex:
----
class Test
{
      public static void main(String[] args)
      {
            int i=1,2,3,4,5;

            System.out.println(i);
      }
}
o/p:
      C.T.E : illegal start of expression

ex:
---
class Test
{
      //global variable
      static int i=100;

      public static void main(String[] args)
      {
            //local variable
            int i=200;

            System.out.println(i); // 200
      }
}
Note:
      Here priority goes to local variable.


ex:
----
class Test
{
      public static void main(String[] args)
      {
            int i=10/2;

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

ex:
---
class Test
{
      public static void main(String[] args)
      {
            int i=10/20;

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

ex:
---
class Test
{
      public static void main(String[] args)
      {
            int i=10%2;

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

ex:
-----
class Test
{
      public static void main(String[] args)
      {
            int i=20%100;

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

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

            i+=20; // i = i + 20;

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

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

            i*=2; // i = i * 2;

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

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

            i/=4;
            System.out.println(i);//2
      }
}
```

ex:

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

            i%=4;

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

2)Ternary operator / Conditional operator
----
syntax:
```
      (condition)?val1:value2;
```

ex:

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

ex:

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

ex:

```
class Test
{
      public static void main(String[] args)
      {
            int i=(5>2)?1:0;
            System.out.println(i);//1
      }
}
```

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();

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

      }
}
```

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

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

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

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

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

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

      }
}
```

3)Logical Operators

Logical AND operator(&&)

Truth table

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

ex:

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

            boolean b=(5>2) && (6<10);
            System.out.println(b);//true
      }
}
```

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

            boolean b=(5>20) && (6<10);
            System.out.println(b);//false
      }
}

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

            boolean b=(5>20) && (6<1);
            System.out.println(b);//false
      }
}

Logical OR operator (||)
-------------------------
Truth table
-----------
T     T     = T
T     F     = T
F     T     = T
F     F     = F

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

            boolean b=(5>20) || (6<1);
            System.out.println(b);//false
      }
}

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

            boolean b=(5>2) || (6<1);
            System.out.println(b);//true
      }
}

ex:
---
class Test
{
      public static void main(String[] args)
      {
            boolean b=(5>2) && (6<1) || true;
            System.out.println(b);// true
      }
}

Logical NOT operator (!)
-------------------------

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

            boolean b=!(5>2);

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

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

            boolean b=!(10 > 20);
            System.out.println(b);// true
      }
}

How to convert decimal to binary
---------------------------------
10   - decimal number

1010 - binary number

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

How to convert binary to decimal
--------------------------------
1010  - binary number

10    - decimal number

      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 OR operator deals with binary numbers.

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

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

            int a=10,b=5;

            int c = a | b;

            System.out.println(c); // 15
      }
}
/*
      10 - 1010
      5  - 0101
      ---------
      |  - 1111
                 <--
            1*1 + 1*2 + 1*4 + 1*8
            1 + 2 + 4 + 8 = 15

*/

Bitwise XOR operator (^)
-------------------------
Bitwise XOR operator deals with binary numbers.

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

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

            int a=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= ~23;

            System.out.println(i); // -24
      }
}
```

ex:

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

            int i= ~(-19);

            System.out.println(i); // 18
      }
}
```

5)Arithmetic operators

```
% - modules
/ - division
* - multiplication
+ - addition
- - subtraction
```

ex:

```
class Test
{
      public static void main(String[] args)
      {
            int i=4+6%3+8/2+6*6+9/100+6-20;

            System.out.println(i); // 30
      }
}
/*
      4 + 6%3 + 8/2 + 6*6 + 9/100 + 6-20

      4 + 0 + 4 + 36 +  0  -14

      44 - 14

      30

*/
```

6)Shift operators

Right shift operator (>>)

```
10 >> 1  =  10/2

10 >> 2  =  10/4

10 >> 3  =  10/8

10 >> 4  =  10/16
```

ex:

```
class Test
{
      public static void main(String[] args)
      {
            int i = 20 >> 3;

            System.out.println(i); // 20/8 = 2

      }
}
```

ex:

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

            System.out.println(i); // 100 / 32 = 3

      }
}
```

Left shift operator (<<)

```
10 << 1  =  10*2

10 << 2  =  10*4

10 << 3  =  10*8

10 << 4  =  10*16
```

```
ex:
---
class Test
{
      public static void main(String[] args)
      {
            int i = 10 << 3;

            System.out.println(i); //  10 * 8 = 80

      }
}

ex:
---
class Test
{
      public static void main(String[] args)
      {
            int i = 100 << 2;

            System.out.println(i); //  100 * 4 = 400

      }
}

7)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

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

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

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

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

      }
}

8) Unary operators
---------------------

Increment/Decrement operators (++/--)
-------------------------------------
We have two types of increment operators.

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

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

We have two types of decrement operators.

1)Pre decrement
      ex:
            --i;

2)Post decrement
      ex:
            i--;



Post Increment/Decrement
=========================
Rule1 : First Take

Rule2 : Then Change

ex:1
-----
class Test
{
      public static void main(String[] args)
      {
            int i=10;

            i++;

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

ex:2
-----
class Test
{
      public static void main(String[] args)
      {
            int i=10;

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

ex:3
-----
class Test
{
      public static void main(String[] args)
      {
            int i=10;

            int j=i++;

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

ex:4
-----
```

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

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

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

ex:5

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

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

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

Pre Increment/Decrement
=======================
Rule1: First Change

Rule2: Then Take

ex:1

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

		++i;

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

ex:2

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

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

ex:3

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

```
            int i=10;

            int j=++i;

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

ex:4

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

            int j=++i + ++i; //11 + 12

            System.out.println(i+" "+j); // 12  23
      }
}
```

ex:5

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

            int j= i++ + ++i; //10 + 12

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

ex:6

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

            100++;

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

ex:7

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

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

Control Statements

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

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

In java, We have four types of control statements.

1) Decision Making Statement

2) Selection Statement

3) Iteration Statement

4) Jump Statement

1) Decision Making Statement

It is used to declare the conditions in our program.

Decision making statement is possible by using following ways.

i) if stmt

ii) if else stmt

iii) if else if stmt

iv) nested if stmt

i) if stmt

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

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

ex:2

```
class Test
```

```
{
	public static void main(String[] args)
	{
		System.out.println("stmt1");
		if(5>20)
		{
			System.out.println("stmt2");
		}
		System.out.println("stmt3");
	}
}
o/p:
	stmt1
	stmt3

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

o/p:
	stmt2
	stmt3
Note:
-----
	Compiler will add curly braces at the time of compilation.
	Compiler will add curly brace only to first stmt.
```

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(!(5>20))
		{
			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(10!=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 find out given age is eligible to vote or not?

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

		System.out.println("Enter the age :");
		int 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 find out given number is even or odd?

even numbers : 2 4 6 8 10 . . .

odd numbers : 1 3 5 7 9 . . .

ex:

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

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

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

```
      }
}
```

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

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

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

            if(n%2==1 || n%2!=0)
                  System.out.println("It is odd number");
            else
                  System.out.println("It is not odd number");
      }
}
```

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

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

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

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

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

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

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

            if(n==0)
            {
            System.out.println("IT is not a positive or negative number");
                  System.exit(0);
            }

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

```
	}
}
```

iii)if else if ladder
====================
It will execute the source code based on multiple conditions.

syntax:

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

ex:

```
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 police number");
		else if(option==103)
			System.out.println("It is a 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 Digit");
            else
                System.out.println("It is special symbol");

      }
}
```

Q)Write a java program to find out 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')
                System.out.println("It is a vowel");
            else if(ch=='e')
                System.out.println("It is a vowel");
            else if(ch=='i')
                System.out.println("It is a vowel");
            else if(ch=='o')
                System.out.println("It is a vowel");
            else if(ch=='u')
                System.out.println("It is a vowel");
            else
                System.out.println("It is not a vowel");

      }
}
```

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

i) If average is greater then equals to 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 stmt contains another if stmt is called nested if stmt.

syntax:

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

```
				-
			}
		}

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

	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
```

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

ex:

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

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

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

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

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

It is similar to if else if ladder.

syntax:

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

Declaration of break statement is optional in switch case.If we won't declare break statement then from where our condition is satisfied from there all cases will be executed 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 police number");
                                //break;
                  case 103: System.out.println("It is enquiry number");
                                //break;
            case 108: System.out.println("It is emergency number");
                                //break;
                  default:  System.out.println("Invalid option");
            }
      }
}
```

Q)Write a java program to find given alphabet is a vowel or 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 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");
            }
      }
}
```

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

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
===================
It is used execute the source code repeatedly.

Iteration 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+" "); //infite 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

ex:

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

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

sum of 10 natural numbers : 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:
	5
Output:
	120 (5*4*3*2*1)
```

ex:

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

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

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

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

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

```
input:
	5
output:
	5 * 1 = 5
	5 * 2 = 10
	-
	-
	5 * 10 = 50
```

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

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

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

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

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

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

Q)Write a java program to perform sum of 100 natural numbers?

```
class Test
{
	public static void main(String[] args)
	{
		int i=1,sum=0;
		while(i<=100)
		{
			sum=sum+i;
			i++;
		}
		System.out.println(sum); //5050
	}
}
```

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(); //123

            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 find out given number is Armstrong or not?

```
input:
      123

output:
      It is not armstrong number(1*1*1+2*2*2+3*3*3)(1+8+27)(36)

input:
      153
output:
      It is armstrong 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(); //123

            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 armstrong number");
            else
                  System.out.println("It is not 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(); //123

            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 plaindrome 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(); //123

		int temp=n;

		int rem,rev=0;

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

Assignments
============
1) Write a java program to display 10 natural numbers?

2) Write a java program to perform sum of 100 natural numbers?

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

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

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

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

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

8) Write a java program to check given number is palindrome or not?

iii) for loop
=============
It will execute the source code untill our condition is true.

syntax:

```
	for(initialization;condition;increment/decrement)
	{
		-
		- //code to be execute
		-
	}
```

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

ex:

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

Q)Write a java program to display even numbers from 1 to 10?

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

Q)Write a java program to display number of evens and odds ?

```
class Test
{
	public static void main(String[] args)
	{
		int even=0,odd=0;
		for(int i=1;i<=10;i++)
		{
			if(i%2==0)
				even++;
			else
				odd++;
		}
		System.out.println(even); // 5
		System.out.println(odd); // 5
	}
}
```

Assignment
===========
Q)Write a java program to display reverse of a given number in words?

```
input:
      123

output:
      ThreeTwoOne
```

ex:

```
class Test
{
      public static void main(String[] args)
      {
            int sum=0;
            for(int i=1;i<=20;i++)
            {
                  if(i%2==0)
                  {
                        sum+=i;
                  }
                  i=i+2;
            }
            System.out.println(sum);//30
      }
}
```

ex:

```
class Test
{
      public static void main(String[] args)
      {
            int sum=0;
            for(int i=1;i<=20;i++)
            {
                  if(i%2==0)
                  {
                        sum+=i;
                        i=i+2;
                  }
            }
            System.out.println(sum);//50
      }
}
```

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

fibonacci series : 0 1 1 2 3 5 8

ex:

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

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

            int a=0,b=1,c;

            System.out.print(a+" "+b+" ");

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

Q)Write a java program to find out 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 prime number");
            else
                  System.out.println("It is not prime number");

      }
}
```

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

input:
      6

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

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

```
input:
      12   18

output:
      6
```

ex:

```
--class Test
{
      public static void main(String[] args)
      {
            int a=12,b=18,gcd=0;

            for(int i=1;i<=12 && i<=18;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 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+" ");
            }
      }
}
```

Various ways to declare java methods
=====================================
There are four ways to declare methods in java.

1) No returntype with No argument method

2) No returtype with Argument method

3) With returntype with No argument method

4) With returntype with Argument method

1) No returntype with No argument method

If there is no arguments then we need to ask inputs inside callie method.

Q)Write a java program to perfrom 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();
            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 swapping of two numbers using no returntype with no argument method?

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

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

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

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

2) No returtype with Argument method

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

Number of arguments 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 find out given number is even or odd by using no returntype with argument method?

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

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

		//caller method
		find(n);

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

3)With returntype with No argument method
==
A returntype is completely 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)
	{
		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 by using with returntype with argument method?

```
import java.util.Scanner;
class Test
{
	public static void main(String[] args)
	{
		Scanner sc=new Scanner(System.in);
		System.out.println("Enter the 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 check given number is even or odd using with returntype with argument method?

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

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

If we don't perticular condition assume that recursion has taken place.

Recursion is similar to loopings.

Q)Write a java program to perform sum of two numbers without using arithmetic operator?

```
import java.util.Scanner;
class Test
{
	public static void main(String[] args)
	{
		Scanner sc=new Scanner(System.in);
		System.out.println("Enter the first number :");
		int a=sc.nextInt();//5
		System.out.println("Enter the second number :");
		int b=sc.nextInt();//10

		//caller method
		System.out.println("sum of two numbers is ="+sum(a,b));
	}
	//callie method
	public static int sum(int a,int b)
	{
		if(a==0)
		{
			return b;
		}
		return sum(--a,++b);
	}
}
```

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

```
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();//5

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

	}
}
o/p:
		/*
				5*factorial(4)
				5*4*factorial(3)
				5*4*3*factorial(2)
				5*4*3*2*factorial(1)
				5*4*3*2*1*factorial(0)
				5*4*3*2*1*1
				120
		*/
```

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

fibonacci series : 0 1 1 2 3 5 8

```
input:
	4

output:
	2
```

ex:

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

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

            //caller method
            fib(n);//4
      }
      //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);
      }
}
```

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

```
class Test
{
      public static void main(String[] args)
      {
            int num=121;
            int original=num;
            int reversed=0;

            //caller method
            if(isPalindrome(num,original,reversed))
                  System.out.println("It is palindrome ");
            else
                  System.out.println("It is not palindrome");

      }
      //callie method
      public static boolean isPalindrome(int num,int original,int reversed)
      {
            if(num==0)
            {
                  return original==reversed;
            }

            reversed= reversed*10+num%10;

            return isPalindrome(num/10,original,reversed);
      }
}
```

Loop Patterns
=============

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

ex:
```
class Test
```

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

2)

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

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

3)

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

ex:

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

4)

```
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++)
            {
                  //columns
                  for(char j='A';j<='D';j++)
                  {
                        System.out.print(i+" ");
                  }
                  //new line
                  System.out.println("");
            }

      }
}
```

5)

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

ex:

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

6)

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

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

7)
```
* - - - *
- * - * -
- - * - -
- * - * -
* - - - *
```

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

8)
```
1 1 1
1 0 1
1 1 1
```

```
class Test
{
	public static void main(String[] args)
	{
		//rows
		for(int i=1;i<=3;i++)
		{
			//columns
```

```
				for(int j=1;j<=3;j++)
				{
					if(i==2 && j==2)
						System.out.print("0 ");
					else
						System.out.print("1 ");
				}
				//new line
				System.out.println("");
			}
		}
}
```

```
9)
4 4 4 4
3 3 3 3
2 2 2 2
1 1 1 1
```

ex:

```
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("");
			}
		}
}
```

```
10)
D D D D
C C C C
B B B B
A A A A
```

ex:

```
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("");
			}
		}
}
```

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

1
2 2
3 3 3
4 4 4 4

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


3)

1
1 2
1 2 3
1 2 3 4

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

4)

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

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

5)

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

```
class Test
{
	public static void main(String[] args)
	{
		//rows
		int k=1;
		for(int i=1;i<=4;i++)
		{
			//columns
			for(int j=1;j<=i;j++)
			{
				if(k<=9)
					System.out.print((k++)+" ");
				else
					System.out.print("0 ");
			}
			//new line
```

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

6)

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

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

7)

```
1
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++)
            {
                  //odd
                  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("  ");
                  }

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

2)

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

ex:

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

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

Assignment
==============
3)Write a java program for below loop pattern?

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

Pyramids
==========
1)

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

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

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

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

                  //right side elements
                  for(int j=i-1;j>=1;j--)
                  {
                        System.out.print(j+" ");
                  }

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

3)

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

ex:
--
```
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("  ");
                  }

                  //left side elements
                  for(int j=1;j<=i;j++)
                  {
                        System.out.print("* ");
                  }

                  //right side elements
                  for(int j=i-1;j>=1;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("  ");
                  }

                  //left side elements
                  for(int j=1;j<=i;j++)
                  {
                        System.out.print("* ");
                  }

                  //right side elements
                  for(int j=i-1;j>=1;j--)
                  {
                        System.out.print("* ");
                  }
                  //new line
                  System.out.println("");
            }

      }
}
```

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

```
1 2 3 4 4 3 2 1

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

            //rows
            for(int i=1;i<=rows;i++)
            {
                  //left side elements
                  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 elements
                  for(int j=i;j>=1;j--)
                  {
                        System.out.print(j+" ");
                  }
                  //new line
                  System.out.println("");

            }

      }
}
```

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

5)

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

ex:

```
class Test
{
      public static void main(String[] args)
      {
            //rows
            for(int i=1;i<=5;i++)
            {
                  //columns
                  for(int j=1;j<=5;j++)
                  {
                        if(i==3 || j==3)
                              System.out.print("* ");
                        else
                              System.out.print("  ");
                  }
                  //new line
                  System.out.println("");
            }

      }
}
```

4)Jump Statement
=================
It is used to jump from one section of code to another section.

We have two types of jump statements.

i) break

ii) continue

i) break

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

For conditional statements we can use if condition.

syntax:
```
      break;
```

ex:1

```
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:2

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

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

ex:3

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

ii) continue

It is used to continue the execution of loops.

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(!false)
            {
                  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
              }
        }
}
```

Arrays
=======
Array is a collection of homogeneous data elements.

The main objective of arrays are

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

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

2)Performance point of view arrays are recommanded to use.

The main disadvantages of arrays 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 categories into three types.

1)Single Dimensional Array

2)Double Dimensional Array / Two Dimensional Array

3)Multi Dimensional Array / Three Dimensional Array

Array Declaration

At the time of array declaration we should not specify array size.
ex:

```
                        Arrays
          |-----------------------|-----------------------------|
Single Dimensional Array  Double Dimensional Array    Multi Dimensional Array

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

Array Construction

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

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

Diagram: java19.1

Rules to constructor an array

```
Rule1:
------
      At the time of array creation compulsary we need to specify array size.
      ex:
            int[] arr=new int[3]; //valid
            int[] arr=new int[]; //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 take negative number 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 are byte,short,int and char.
      If we take other datatypes then we will get compile time error.
      ex:
            int[] arr=new int['a'];

            byte b=10;
            int[] arr=new int[b];

            int[] arr=new int[10.5f];

Rule5:
-------
      The maximum length we can't take for an array size is maximum
      length of integer.
      ex:
            int[] arr=new int[2147483647];




Array Initialization
=====================
Once if we create an array ,every array element will be initialized with default
values.

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

Diagram: java20.1


Array Declaration, Creation and Initialization in a single line
===============================================================

      int[]  arr;
      arr=new int[3];
      arr[0]=10;
      arr[1]=20;
      arr[2]=30;      ===>   int[] arr={10,20,30};
```

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

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

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

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[4];
            System.out.println(arr.length);// 4
      }
}
```

length()

It is a predefined method applicable for String objects.

It will return number of characters are present in String.

ex:

```
class Test
{
      public static void main(String[] args)
      {
            String str="bhaskar";

            System.out.println(str.length());// 7
      }
}
```

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

```
import java.util.Scanner;
class Test
{
      public static void main(String[] args)
      {
            Scanner sc=new Scanner(System.in);
            System.out.println("Enter the array size :");
            int size=sc.nextInt();//4

            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();
            }

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

Q)Write a java program to display array elements?

```
input:
      4 8 1 2 7 9

output:
      4 8 1 2 7 9
```

ex:

```
class Test
{
      public static void main(String[] args)
      {
            int[] arr={4,8,1,2,7,9};

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

approach2

```
class Test
{
      public static void main(String[] args)
      {
            int[] arr={4,8,1,2,7,9};

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

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

```
input:
      4 8 1 2 7 9

output:
      9 7 2 1 8 4
```

ex:

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

```
	{
		int[] arr={4,8,1,2,7,9};

		//reverse order
		for(int i=arr.length-1;i>=0;i--)
		{
			System.out.print(arr[i]+" ");
		}
	}
}
```

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

```
input:
	4 8 1 2 7 9

output:
	31
```

ex:

```
class Test
{
	public static void main(String[] args)
	{
		int[] arr={4,8,1,2,7,9};

		//for each loop
		int sum=0;
		for(int i:arr)
		{
			sum+=i;
		}
		System.out.println(sum);
	}
}
```

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

```
input:
	4 8 1 2 7 9

output:
	1 2 4 7 8 9
```

approach1

```
import java.util.Arrays;
class Test
{
	public static void main(String[] args)
	{
		int[] arr={4,8,1,2,7,9};

		Arrays.sort(arr);

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

```
approach2
---------
class Test
{
	public static void main(String[] args)
	{
		int[] arr={4,8,1,2,7,9};

		//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;
				}
			}
		}

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

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

```
input:
	4 8 1 2 7 9

output:
	9 8 7 4 2 1
```

```
ex:
approach1:
---------
import java.util.Arrays;
class Test
{
	public static void main(String[] args)
	{
		int[] arr={4,8,1,2,7,9};

		Arrays.sort(arr);

		//reverse order
		for(int i=arr.length-1;i>=0;i--)
		{
			System.out.print(arr[i]+" ");
		}
	}
}
```

```
approach2
------
class Test
```

```
{
	public static void main(String[] args)
	{
		int[] arr={4,8,1,2,7,9};

		//descending logic
		for(int i=0;i<arr.length;i++)
		{
			for(int j=0;j<arr.length;j++)
			{
				if(arr[i]>arr[j])
				{
					int temp=arr[i];
					arr[i]=arr[j];
					arr[j]=temp;
				}
			}
		}

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

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

```
input:
	4 8 1 2 7 9

output:
	9
```

approach1

```
import java.util.Arrays;
class Test
{
	public static void main(String[] args)
	{
		int[] arr={4,8,1,2,7,9};

		Arrays.sort(arr); // 1 2 4 7 8 90

		System.out.println(arr[arr.length-1]);
	}
}
```

approach2

```
class Test
{
	public static void main(String[] args)
	{
		int[] arr={4,8,1,2,7,9};

		int big=arr[0];

		//for each loop
		for(int i:arr)
```

```
            {
                  if(i>big)
                  {
                        big=i;
                  }
            }
            System.out.println(big);
      }
}
```

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

```
input:
      4 8 1 2 7 9

output:
      1
```

approach1

```
import java.util.Arrays;
class Test
{
      public static void main(String[] args)
      {
            int[] arr={4,8,1,2,7,9};

            Arrays.sort(arr);

            System.out.println(arr[0]);
      }
}
```

approach2

```
class Test
{
      public static void main(String[] args)
      {
            int[] arr={4,8,1,2,7,9};

            int small=arr[0];

            //for each loop
            for(int i:arr)
            {
                  if(i<small)
                  {
                        small=i;
                  }
            }
            System.out.println(small);
      }
}
```

Assignment
==========
Q)Write a java program to find out second highest elements from given array?

```
input:
      4 8 1 3 7 9
output:
      8
```

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

```
input:
      4 6 1 2 9 1 4 3 7 9

output:
      4 1 9

ex:
class Test
{
      public static void main(String[] args)
      {
            int[] arr={4,6,1,2,9,1,4,3,7,9};

            //duplicate elements
            for(int i=0;i<arr.length;i++)
            {
                  for(int j=i+1;j<arr.length;j++)
                  {
                        if(arr[i]==arr[j])
                        {
                              System.out.print(arr[i]+" ");
                        }
                  }
            }
      }
}
```

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

```
input:
      4 6 1 2 9 1 4 3 7 9

output:
      6 2 3 7

ex:

class Test
{
      public static void main(String[] args)
      {
            int[] arr={4,6,1,2,9,1,4,3,7,9};

            //unique elements
            for(int i=0;i<arr.length;i++)
            {
                  int cnt=0;
                  for(int j=0;j<arr.length;j++)
                  {
                        if(arr[i]==arr[j])
                        {
                              cnt++;
                        }
                  }
                  if(cnt==1)
                        System.out.print(arr[i]+" ");
            }
      }
}
```

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

input:
```
	4 5 1 2 3 1 7 7 9 7 3 7 5 5
```

output:
```
	7 is repeating for 4 times
```

ex:

```
class Test
{
	public static void main(String[] args)
	{
		int[] arr={4,5,1,2,3,1,7,7,9,7,3,7,5,5};

		int element=0;
		int maxCount=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(maxCount<cnt)
			{
				maxCount=cnt;
				element=arr[i];
			}
		}
		System.out.println(element+" is repeating for "+maxCount+" times");
	}
}
```

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

input:
```
	5 9 13 17 21 23  6  4
```

output:
```
	5 13 17 23
```

ex:

```
class Test
{
	public static void main(String[] args)
	{
		int[] arr={5,9,13,17,21,23,6,4};

		//for each loop
		for(int n:arr)
		{
			//prime logic
			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 segregate array elements?

```
input:
      1 0 1 1 0 0 1 0 1 0

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

ex:

```
import java.util.Arrays;
class Test
{
      public static void main(String[] args)
      {
            int[] arr={1,0,1,1,0,0,1,0,1,0};

            Arrays.sort(arr);

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

appoach2

```
import java.util.Arrays;
class Test
{
      public static void main(String[] args)
      {
            int[] arr={1,0,1,1,0,0,1,0,1,0};

            int[] newArr=new int[arr.length];


            //inserting '0'
            //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
            for(int i:newArr)
            {
                  System.out.print(i+" ");
            }

      }
}
```

Q)Write a java program find out leader elements from given array?

```
input:
      5   7   34    16   4   8

output:
      8  16   34
```

ex:

```
import java.util.Arrays;
class Test
{
      public static void main(String[] args)
      {
            int[] arr={5,7,34,16,4,8};

            int max=arr[arr.length-1];

            System.out.print(max+" ");

            //reading reverse
            for(int i=arr.length-2;i>=0;i--)
            {
                  if(arr[i]>max)
                  {
                        max=arr[i];
                        System.out.print(max+" ");
                  }
            }
      }
}
```

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

```
input:
      1 6 3 5 2

output:
      4
```

ex:

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

```
		int sum_of_arr_ele=arr.length+1;

		int max=(sum_of_arr_ele*(sum_of_arr_ele+1))/2;

		for(int i:arr)
		{
			max=max-i;
		}
		System.out.println(max);
	}
}
```

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

```
input:
	1 7 2 5 3

	9 4 8 6 10

output:
	10  11  10  11  13
```

ex:

```
import java.util.Arrays;
class Test
{
	public static void main(String[] args)
	{
		int[] arr1={1,7,2,5,3};
		int[] arr2={9,4,8,6,10};

		int[] resArr=new int[arr1.length];

		int j=0;
		for(int i=0;i<arr1.length && i<arr2.length;i++)
		{
			resArr[j++]=arr1[i]+arr2[i];
		}

		//display
		for(int i:resArr)
		{
			System.out.print(i+" ");
		}
	}
}
```

Q)Write a java program to concatinate two arrays and display them in sorting order?

```
input:
	5 1 3 2 4
	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={5,1,3,2,4};
            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++];
            }
            //sorting
            Arrays.sort(arr1);

            //display the elements
            for(int i:arr1)
            {
                  System.out.print(i+" ");
            }

      }
}
```

Q)Write a java program to delete first occurance of a given element?

```
input:
      arr = 2 5 1 3 9 5 6 5 7 5

      ele = 5

output:
      2 1 3 9 5 6 5 7 5
```

```
class Test
{
      public static void main(String[] args)
      {
            int[] arr={2,5,1,3,9,5,6,5,7,5};
            int ele=5;

            int[] resArr=new int[arr.length-1];

            int cnt=0,j=0;
            for(int i=0;i<arr.length;i++)
            {
                  if(arr[i]==ele && cnt==0)
                  {
                        cnt++;
                        continue;
                  }
                  resArr[j++]=arr[i];
            }

            //display array elements
            for(int i:resArr)
            {
                  System.out.print(i+" ");
            }
      }
}
```

Q)Write a java program to insert given element on given position in a given array?

```
input:
      arr = 2 7 1 4 5 9
      ele = 10
      position = 3
output:
      2 7 1 10 4 5 9
```

ex:

```
import java.util.Arrays;
class Test
{
      public static void main(String[] args)
      {
            int[] arr ={2,7,1,4,5,9};
            int ele = 10;
            int position = 3;

            arr=Arrays.copyOf(arr,arr.length+1);

            for(int i=arr.length-1;i>=position;i--)
            {
                  arr[i]=arr[i-1];
            }
            arr[position]=ele;

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

      }
}
```

Two Dimensional Array
=====================
Two dimensional is a combination of rows and columns.

Two dimensional array is implemented based on array or arrays approach but not matrix form.

The main objective of two dimensional array is a memory utilization.

We can use two dimensional to develop business oriented applications, gaming applications,
matrix type of applications.

We can declare two dimensional array as follow.

syntax:

```
      datatype[][] variable_name=new datatype[rows][cols];
```

ex:

```
                   rows
                          |
      int[][] arr=new int[3][3];
                           |
                        columns

      Here we can store 9 elements in array.
```

Q)Write a java program to accept array elements and display them 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();//3

            System.out.println("Enter the columns :");
            int cols=sc.nextInt();//3

            int[][] arr=new int[rows][cols];

            //inserting 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 perform sum of diagonal elements?

```
class Test
{
      public static void main(String[] args)
      {
            int[][] arr={
                                    {1,2,3},
                                    {4,5,6},
                                    {7,8,9}
                              };

            int sum=0;
            for(int i=0;i<3;i++)
            {
                for(int j=0;j<3;j++)
                {
                        if(i==j)
                        {
                                sum+=arr[i][j];
                        }
                }
        }

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

ex:

```
class Test
{
    public static void main(String[] args)
    {
        int[][] arr={
                                {1,2,3},
                                {4,5,6},
                                {7,8,9}
                        };

        int sum=0;
        for(int i=0;i<3;i++)
        {
                for(int j=0;j<3;j++)
                {
                        if(i+j==2)
                        {
                                sum+=arr[i][j];
                        }
                }
        }

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

Q)Write a java program to display sum of upper triangle elements?

```
class Test
{
    public static void main(String[] args)
    {
        int[][] arr={
                                {1,2,3},
                                {4,5,6},
                                {7,8,9}
                        };

        int sum=0;
        for(int i=0;i<3;i++)
        {
                for(int j=0;j<3;j++)
                {
                        if(i<j)
                        {
                                sum+=arr[i][j];
                        }
                }
            }

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

Q)Write a java program to display lower triangle elements?

```
class Test
{
      public static void main(String[] args)
      {
            int[][] arr={
                                    {1,2,3},
                                    {4,5,6},
                                    {7,8,9}
                              };

            int sum=0;
            for(int i=0;i<3;i++)
            {
                  for(int j=0;j<3;j++)
                  {
                        if(i>j)
                        {
                              sum+=arr[i][j];
                        }
                  }
            }

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

Assignemnt
===========
Q)Write a java program to display square of a matrix?

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}};
```

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

input:
```
      4 7 1 3 9 6
```

output:

```
	30
```

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

		//caller method
		sum(new int[]{4,7,1,3,9,6});

	}

	public static void sum(int[] arr)
	{
		int sum=0;
		for(int i=0;i<arr.length;i++)
		{
			sum+=arr[i];
		}

		System.out.println(sum);
	}

}
```

ex:

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

		//caller method
		System.out.println(sum(new int[]{4,7,1,3,9,6}));

	}

	public static int sum(int[] arr)
	{
		int sum=0;
		for(int i=0;i<arr.length;i++)
		{
			sum+=arr[i];
		}

		return sum;
	}

}
```

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[][] arr={
                                    {1,2,3},
                                    {4,5,6},
                                    {7,8,9}
                              };

            int rows=arr.length;
            int cols=arr[0].length;

            int top = 0;
        int bottom = rows - 1;
        int left = 0;
        int right = cols - 1;

        while(true)
            {
            if (left > right)
                  {
                break;
            }

            // Print top row
            for (int i = left; i <= right; i++) {
                System.out.print(matrix[top][i] + " ");
            }
            top++;


            if (top > bottom)
                  {
                break;
            }

            // Print right column
            for (int i = top; i <= bottom; i++) {
                System.out.print(matrix[i][right] + " ");
            }
            right--;


            if (left > right)
                  {
                break;
            }

            // Print bottom row
            for (int i = right; i >= left; i--)
         {
                System.out.print(matrix[bottom][i] + " ");
            }
            bottom--;

            if (top > bottom)
                  {
                break;
            }

            // Print left column
            for (int i = bottom; i >= top; i--)
          {
                System.out.print(matrix[i][left] + " ");
            }
            left++;
      }//while loop
    }
}
```

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

object oriented technology

A technology which provides very good environment to represent our data in the form objects is called object oriented technology.

A technology is said to be object oriented if it support following features.

ex:
```
            class
            object
            Abstraction
            Encapsulation
            Inheritance and
            Polymorphism
```

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

Here data members means variables, properties or fields.

Here behaviours means methods, actions or characteristics.

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

A class is a blue print of an object.

A class will accept following modifiers.
ex:
```
            default
            public
            final
            abstract
```

We can declare a class as follow.
ex:
```
                  optional
                  |
            Modifier class class_name <extends> Parent_classname
                                                                   <implements>
Interface_name
            {
                  -
                  -  // variables and methods
                  -
            }
```

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

default class

If we declare any class as default then we can access that class within the package.

ex:
```
            class  Test
            {
                  -
                  - // variables and methods
                  -
            }
```

public class

If we declare any class as public then we can access that class within the package
and outside of the package.

ex:
```
            public class  Test
            {
                  -
                  - // variables and methods
                  -
            }
```

Q)What is final class?

If we declare any class as final then extending some other class is not possible.

or

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

ex:
```
            final class A
            {
                        -
                        - //code to be execute
                        -
            }
            class B extends A    ----> invalid
            {
                        -
                        -
                        -
            }
```

Q)What is abstract class?

If we declare any class as abstract then creating object of that class is not possible.
ex:
```
            abstract class A
            {
                  -
                  - //code to be declare
                  -
            }
            A a=new A(); ---> invalid
```

```
object
=======
It is a instance of a class.

Allocating memory for our data members is called instance.

It is a out come of blue print.

Memory space will be allocated when we create an object.

We can declare object as follow.

syntax:
                  class_name  reference_variable=new  constructor();

ex:
                  Test  t = new  Test();

It is possible to create multiple objects 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@Hexadecimalno
            System.out.println(t2.toString());
            System.out.println(t3.toString());

      }
}

hashCode()
==========
It is a method present in Object class.

Whenever we create object for a class,JVM will create a unique identification
number i.e hash code.

In order to read the hash code of an object we will use hashCode() method.

Diagram: java23.1

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

Data Hiding
============
Our data should not go out directly.

It means outside person must not access our data directly.
```

Using private modifier we can achieve data hiding concept.

The main objective of data hiding is to provide security.

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

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

Using abstract classes and interfaces we will implements Abstraction.

Best of example of abstract is GUI(Graphical User Interface) ATM machine where bank people will hide internal implementation and hightlights the set of services
like banking, withdrawl, mini statement , deposit and etc.

The main advantages of abstraction are

1) It gives security because it will hide internal implementation from the outsider.

2) Enhancement becomes more easy because without effecting enduser they can perform
   any changes in our internal system.

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

4) It improves maintainability of an application.

Encapsulation
===============
The process of encapsulating or grouping variables and it's associate methods in a single entity is called encapsulation.

A class is said to be encapsulated class if it supports data hiding + abstraction.

Diagram: java23.2

The main objective of encapsulation is to protect the data and abstraction is used to hide the data.

In encapsulation , for every variable we need to write setter and getter methods.

Diagram: java23.3

The main advantages of encapsulation are

1) It gives security.

2) Enhancement becomes more easy.

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

4) It improves maintainability of an application.

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

approach1
----
ex:

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

      //setter methods
      public void setStudId(int studId)
      {
            this.studId=studId;
      }
      public void setStudName(String studName)
      {
            this.studName=studName;
      }
      public void setStudFee(double studFee)
      {
            this.studFee=studFee;
      }

      //getter methods
      public int getStudId()
      {
            return studId;
      }
      public String getStudName()
      {
            return studName;
      }
      public double getStudFee()
      {
            return studFee;
      }

      public static void main(String[] args)
      {
            Student s=new Student();
            s.setStudId(101);
            s.setStudName("Alan");
            s.setStudFee(10000d);

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

approach2

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

	//setter methods
	public void setStudId(int studId)
	{
		this.studId=studId;
	}
	public void setStudName(String studName)
	{
		this.studName=studName;
	}
	public void setStudFee(double studFee)
	{
		this.studFee=studFee;
	}

	//getter methods
	public int getStudId()
	{
		return studId;
	}
	public String getStudName()
	{
		return studName;
	}
	public double getStudFee()
	{
		return studFee;
	}
}
class Test
{
	public static void main(String[] args)
	{
		Student s=new Student();
		s.setStudId(101);
		s.setStudName("Alan");
		s.setStudFee(10000d);

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

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

By using extends keywords we can implements Is-A relationship.

The main objective of Is-A relationship is to provide reusability.

ex:

```
class  Parent
{
	public void m1()
```

```
      {
            System.out.println("Parent-M1 Method");
      }
}
class Child extends Parent
{
      public void m2()
      {
            System.out.println("Child-M2 Method");
      }
}
class Test
{
      public static void main(String[] args)
      {
            Parent p=new Parent();
            p.m1();

            Child c=new Child();
            c.m1();
            c.m2();

            Parent p1=new Child();
            p1.m1();

            //Child c1=new Parent(); //C.T.E

      }
}
```

Inheritance
=============
Inheritance is a mechanism where we will derive a class in the presence of existing classes.

Inheritance is a mechanism where one class will inherit the properties of another class.

We have five types of inheritance.

1) Single Level Inheritance

2) Multi-Level Inheritance

3) Multiple Inheritance

4) Hierarchical Inheritance

5) Hybrid Inheritance

1) Single Level Inheritance

If we derived a class in the present in one base class is called single level inheritance.
ex:

```
                  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();

      }
}
```

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.
ex:

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

ex:

```
class A
{
      public void m1()
      {
            System.out.println("m1 method");
      }
}
class B extends A
{
      public void m2()
      {
            System.out.println("m2 method");
      }
}
class C extends B
{
      public void m3()
      {
            System.out.println("m3 method");
      }
}
```

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

            B b=new B();
            b.m1();
            b.m2();

            C c=new C();
            c.m1();
            c.m2();
            c.m3();

      }
}
```

3)Multiple Inheritance

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

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

ex:

```
            class A
            {
            }
            class B
            {
            }
            class C extends A,B  ---> invalid
            {
            }
```

In java we can extends more then one interface simultenously.So we can achieve multiple
inheritance concept through interfaces.
ex:

```
            interface A
            {
            }
            interface B
            {
            }
            interface C extends A,B
            {
            }
```

Note:

```
            class     --->   class      ---> extends

            interface --->   interface  ---> extends

      class     --->    interface ---> implements
```

If our class does not extends any other class then our class is a direct child

```
class of
Object class.
ex:                                         Diag:
                  class A                               Object
                  {                                                 |
                                                                    |
                                                                    |
                  }                                                 A

If our class extends some other class then our class is a indirect child class
of Object class.
ex:                                                     Diag:
            class A                                                 Object
            {                                                           |
            }                                                           |
            class B extends A                               A  //multi level
inheritance
            {                                                           |
            }                                                           |
                                                                        B

Java does not support cyclic inheritance.
ex:
            class A extends B
            {
            }
            class B extends A
            {
            }

Q)Why Java does not support multiple inheritance?

There is a chance of raising ambiguity problem that's why java does not support
multiple
inheritance.

Diagram:
                        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:
                                          A
                                          |
                  |-----------------------|
                  B                                             C

ex:

class A
{
      public void m1()
      {
            System.out.println("m1 method");
      }
```

```
}
class B extends A
{
      public void m2()
      {
            System.out.println("m2 method");
      }
}
class C extends A
{
      public void m3()
      {
            System.out.println("m3 method");
      }
}
class Test
{
      public static void main(String[] args)
      {
            A a=new A();
            a.m1();

            B b=new B();
            b.m1();
            b.m2();

            C c=new C();
            c.m1();
            c.m3();

      }
}
```

5)Hybrid inheritance

Hybrid inheritance is a combination of more then one inheritance.

Java does not support hybrid inheritance.

Diagram:

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

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 will increase dependency between two components.

ex:

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

ex:

```
class Ihub
{
      public String courseName()
      {
            return "Full Stack Java + AWS";
      }
      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 a no chance of having contained object then
the relationship between container and contained object is called composition which is
strongly association.

Diagram: java24.1

Aggregation
============
Without existing container object there is a chance of having contained object then
the relationship between container and contained object is called aggregation which is
weakly association.

Diagram: java24.2

Q)What is the difference between POJO class and Java Bean class?

POJO
====
POJO stands for Plain Old Java Object.

A class is said to be pojo class if it supports following two properties.

1) All variables must be private.

2) All variables must and setter and getter methods.

Java Bean
==========
A class is said to be java bean class if it supports following four properties.

1)A class should be public.

2)A class should have atleast zero argument constructor

3)All variables must be private

4)All variables must have setter and getter methods.

Note:

Every java bean class is a pojo class but every pojo class is not a java bean class.

Method overloading
==================
Having same method name but different parameters 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 By voterId");
      }
      public void search(String houseNo)
      {
            System.out.println("Details Found By houseNo");
      }
      public void search(long aadharCard)
      {
            System.out.println("Details Found By aadharCard");
      }
}
class Test
{
      public static void main(String[] args)
      {
            MeeSeva ms=new MeeSeva();
            ms.search(101);
            ms.search("1-5/4/1");
            ms.search(1012l);
      }
}
```

Method overriding
==================
Having same method name with same parameters in two different classes is called method
overriding.

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

Methods which are present in child class are called overriding methods.

ex:

```
class Parent
{
      public void property()
      {
            System.out.println("Cash+Gold+Land");
      }
      //overridden methods
      public void marry()
      {
            System.out.println("Subhalakshmi");
      }
}
class Child extends Parent
{
      //overriding methods
      public void marry()
      {
            System.out.println("Trisha/Rashmika");
      }
}
class Test
{
      public static void main(String[] args)
      {
            Parent p=new Parent();
            p.property();//cash+gold+land
            p.marry();//subhalakshmi

            Child c=new Child();
            c.property();//cash+gold+land
            c.marry();//Trisha/Rashmika

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


      }
}
```

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

ex:

```
class Parent
{
      public void property()
      {
            System.out.println("Cash+Gold+Land");
      }
      //overridden methods
      public final void marry()
      {
            System.out.println("Subhalakshmi");
      }
}
class Child extends Parent
{
      //overriding methods
      public void marry()
      {
            System.out.println("Trisha/Rashmika");
      }
}
class Test
{
      public static void main(String[] args)
      {
            Parent p=new Parent();
            p.property();//cash+gold+land
            p.marry();//subhalakshmi

            Child c=new Child();
            c.property();//cash+gold+land
            c.marry();//Trisha/Rashmika

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


      }
}
```

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

```
Method overriding                  Method hiding
--------------------                     --------------
All the methods present in method overriding   All the methods present in method
hiding
must be non-static.                      must be static.

Method resolution will taken care by           Method resolution will taken care
by
JVM based on runtime object.             compiler based on reference type.

It is also known as runtime polymorphism,It is also known as compile time
polymorhpism
dynamic polymorphism or late binding.          ,static polymorphism or early
binding.
```

ex:

```
class Parent
```

```
{
      public static void property()
      {
            System.out.println("Cash+Gold+Land");
      }
      public static void marry()
      {
            System.out.println("Subhalakshmi");
      }
}
class Child extends Parent
{
      public static void marry()
      {
            System.out.println("Trisha/Rashmika");
      }
}
class Test
{
      public static void main(String[] args)
      {
            Parent p=new Parent();
            p.property();//cash+gold+land
            p.marry();//subhalakshmi

            Child c=new Child();
            c.property();//cash+gold+land
            c.marry();//Trisha/Rashmika

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


      }
}
```

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

Q)Can we overload main method in java?

Yes, it is possible to overload main method in java but JVM always execute main method with
String[] parameter only.
ex:

```
class Test
{
      public static void main(int[] iargs)
      {
            System.out.println("int-parameter");
      }
      public static void main(String[] args)
      {
            System.out.println("string-parameter");
      }
}
```

Q)Can we override main method in java?

No, we can't override main method in java because it is static and static methods can't be
override.

```
Polymorphism
=============
Polymorphism has taken from Greek Word.

Here poly means many and morhpism means forms.

The ability to represent in different forms is called polymorphism.

The main objective of polymorphism is to provide flexibility.

Diagram: java25.1

In java polymorphism is divided into two types.

1) Compile time polymorphism / Static polymorphism / Early binding

2) Runtime polymorphism / Dynamic polymorphism / Late binding

1) Compile time polymorphism
-----------------------------
A polymorphism which exhibits at compile time is called compile time
polymorphism.
ex:
      Method overloading
      Method Hiding

2)Runtime polymorphism
-------------------------
A polymorphism which exhibits at runtime is called runtime polymorphism.
ex:
      Method overriding


Summary Diagram
----------------
Diagram: java25.2


Constructors
=============
Contructor is a special method which is used to initialized an object.
ex:
      Test t=new Test();

Having same name as class name is called constructor.

A constructor does not allow any returntype.

A constructor will execute when we create an object.

A constructor will accept following modifiers.
ex:
      default
      public
      private
      protected

In java, we have two types of constructors.

1)Userdefined constructor

2)Default constructor
```

1)Userdefined constructor

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

In java userdefined constructor is divided 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 such constructor
is called 0-arg constructor.

ex:1

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

ex:2

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

ex:3

```
class Test
{
	//constructor
	public 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 such constructor
is called parameterized constructor.

ex:

```
class Employee
{
	//current class variables
	private int empId;
	private String empName;
	private double empSal;

	//constructor
	protected Employee(int empId,String empName,double empSal)
	{
		this.empId=empId;
		this.empName=empName;
		this.empSal=empSal;
	}

	//display the data
	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",10000d);
		e.getEmployeeDetails();
	}
}
```

ex:

```
class A
{
	public void m1()
	{
		System.out.println("M1-Method");
	}
}
class B
{
      A a;

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

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 use below command.
ex:

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

Diagram: java26.1

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
{
      //current class variables
      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 constructor
-------------------------------------
class A
{
      A()
```

```
	{
		System.out.println("0-arg const");
	}
	A(int i)
	{
		this();
		System.out.println("int-arg const");
	}
	A(double d)
	{
		this(10);
		System.out.println("double-arg const");
	}

}
class Test
{
	public static void main(String[] args)
	{
		A a=new A(10.5d);
	}
}
```

super keyword

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

We can utilize super keyword in following ways.

1) To refer super class variables

2) To refer super class methods

3) To refer super class constructors

1) To refer super class variables

```
class A
{
	int i=1;
	int j=2;
}
class B extends A
{
	int i=10;
	int j=20;
	B(int i,int j)
	{
		System.out.println(this.i+" "+this.j); // 10 20
		System.out.println(super.i+" "+super.j);// 1 2
		System.out.println(i+" "+j); // 100 200
	}
}
class Test
{
	public static void main(String[] args)
	{
		B b=new B(100,200);
	}
}
```

2) To refer super class methods

```
--------------------------------
class A
{
      public void m1()
      {
            System.out.println("M1-Method");
      }
}
class B extends A
{
      public void m2()
      {
            super.m1();
            System.out.println("M2-Method");
      }
}
class Test
{
      public static void main(String[] args)
      {
            B b=new B();
            b.m2();
      }
}
```

3)To refer super class constructors

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

ex:

```
class A
{
      A(int i)
      {
            System.out.println("A-const ");
      }
}
class B extends A
{
      B()
      {
            super(10);
            System.out.println("B-const");
      }
}
class Test
{
      public static void main(String[] args)
      {
            B b=new B();
      }
}
```

Interfaces
==========
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.

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.

By default every abstract method is a public and abstract.

Interface contains only constants i.e public static final.

syntax:

```
            interface  interface_name
            {
                  //constants
                  //abstract methods
            }
```

If we know Service Requirement specification then we need to use interface.

Diagram: java26.2

ex:1

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

```
	}
}

ex:2
------
interface A
{
	//abstract method
	public abstract void m1();
}

class Test
{
	public static void main(String[] args)
	{
		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 methods otherwise we will
get compile time error.

```
ex:3
----
interface A
{
	//abstract methods
	public abstract void show();
	public void display();
	abstract void view();
	void see();
}
class B implements A
{
	public void show()
	{
		System.out.println("show method");
	}
	public void display()
	{
		System.out.println("display method");
	}
	public void view()
	{
		System.out.println("view method");
	}
	public void see()
	{
		System.out.println("see method");
	}
}

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

```
            A a=new B();
            a.show();
            a.display();
            a.view();
            a.see();
      }
}
```

A class can't extends more then one class simultenously.
But interface can extends more then one interface simultenously.

ex:

```
interface A
{
      void m1();
}
interface B
{
      void m2();
}
interface C extends A,B
{
      void m3();
}
class D implements C
{
      public void m1()
      {
            System.out.println("M1 method");
      }
      public void m2()
      {
            System.out.println("M2 method");
      }
      public void m3()
      {
            System.out.println("M3 method");
      }
}
class Test
{
      public static void main(String[] args)
      {
            C c=new D();
            c.m1();
            c.m2();
            c.m3();
      }
}
```

A class can implements more then one interface.

ex:

```
interface Father
{
      float HT=6.2f;
      void height();
}
interface Mother
{
      float HT=5.8f;
      void height();
}
class Child implements Father,Mother
{
      public void height()
      {
            float height=(Father.HT+Mother.HT)/2;
            System.out.println("Child height is ="+height);
      }
}
class Test
{
      public static void main(String[] args)
      {
            Child c=new Child();
            c.height();
      }
}
```

Note:

In general, interface is a blue print of a class.

According java 8 version, interface is a collection of abstract methods, default methods
and static methods.

According java 9 version, interface is a collection of abstract methods, default methods
static methods and private methods.

Q)What is marker interface?

Interface which does not any methods or constants is called marker interface.

Empty interface is called marker interface.

Using marker interface we will get some ability to do.

We have following list of marker interfaces in java.

ex:

```
      Serializable
      Remote
      Cloneable
      and etc.
```

Abstract class
===============
Abstract class is a collection of 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 class.

To implement abstract methods of a abstract class we will use sub classes.

By default every abstract is a public and abstract.

Abstract class contains only instance variables.

syntax:

```
-------
      abstract class <class_name>
      {
            -
            - //instance variables
            - //abstract methods
            - //concrete methods
            -
      }
```

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();

      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=2.5f;
      }
}
class CommercialPlan extends Plan
{
      public void getRate()
      {
            rate=5.0f;
      }
}
class  Test
{
      public static void main(String[] args)
      {
            DomesticPlan dp=new DomesticPlan();
            dp.getRate();
            dp.calculateBillAmt(250);

            CommercialPlan cp=new CommercialPlan();
            cp.getRate();
            cp.calculateBillAmt(250);
      }
}
```

Q)What is the difference between interface and abstract class?

```
Interface                           Abstract class
--------------                            ---------------
To declare interface we will use interface     To declare abstract class we will 
use
keyword.                            abstract keyword.
```

It is a collection of abstract methods, default methods and static methods.	It is a collection of abstract methods and concrete methods.
It contains only constants.	It contains only instance variables.
We can achieve multiple inheritance.	We can't achieve multiple inheritance.
To write the implementation of abstract methods of an interface we will use implementation class.	To write the implementation of abstract methods of a abstract class we will use sub classes.
If we know specification then we need to use interface.	If we know partial implementation then we need to use abstract class.

API
====
API stands Application Programming Interface.

It is a collection of packages.

It is a base for the programmer to develop software applications.

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) Userdefined API

API which is created by the user based on the application requirement.

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.

Package is also known as folder or a directory.

In java packages are divided into two types.

1)Predefined packages

2)Userdefined packages

```
1)Predefined packages
-------------------
Built-In packages are called predefined packages.
ex:
      java.lang
      java.util
      java.time
      java.io
      java.util.stream
      java.text
      and etc.

2)Userdefined packages
------------------------
Packages which are created by the user based on the application requirement are
userdefined packages.

To declare userdefined package we need to use "package" keyword.

syntax:
            package package_name;

It is recommanded to use package name in the reverse order of url.
ex:
            package  com.ihub.www;

ex:
-----
package com.ihub.www;
import java.util.Calendar;
class  Test
{
      public static void main(String[] args)
      {
            Calendar c=Calendar.getInstance();

            //convert time to 24 hours
            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 execute above program by using below command.
ex:
      java   com.ihub.www.Test
                  |
```

            package name

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

Using a class if we call any method and that methods returns same class object then that
class is called singleton class.

We have following list of singleton classes in java.

ex:
      LocalDate
      LocalTime
      Calendar
      and etc.

To create singleton class we required private constructor and factory method.

ex:

```
class Singleton
{

      static Singleton singleton=null;

      //private constructor
      private Singleton()
      {
            //empty
      }

      //factory 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());

            Singleton s3=Singleton.getInstance();
            System.out.println(s3.hashCode());
      }
}
```

Inner classes
==============
Sometimes we will declare a class inside another class such concept is called

inner class.
ex:
```
      class Outer_Class
      {
            class Inner_Class
            {
                  -
                  - //code to be execute
                  -
            }
      }
```

Inner class must be with in the scope of outer class or enclosing class.

Inner class 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.

In java 8 version, We can't declare static members in inner class.

Accessing inner class data from static area of outer class

```
class Outer
{
      class Inner
      {
            //non-static methods
            public void m1()
            {
                  System.out.println("M1-Method");
            }
      }

      public static void main(String[] args)
      {
            Outer.Inner i=new Outer().new Inner();
            i.m1();
      }
}
```

If we compile above program we will get two .class files i.e
Outer.class and Outer$Inner.class.

ex:2

```
class Outer
{
      class Inner
      {
            //non-static methods
            public void m1()
            {
                  System.out.println("M1-Method");
            }
      }

      public static void main(String[] args)
      {
            new Outer().new Inner().m1();
      }
}
```

```
Accessing inner class data from non-static area of outer class
----------------------------------------------------------
class Outer
{
      class Inner
      {
            //non-static methods
            public void m1()
            {
                  System.out.println("M1-Method");
            }
      }
      public void m2()
      {
            Inner i=new Inner();
            i.m1();
      }

      public static void main(String[] args)
      {
            Outer o=new Outer();
            o.m2();
      }
}
```

If we compile above program we will get two .class files i.e
Outer.class and Outer$Inner.class.

Enum
=====
Enum is a group of named constants.

Enum concept introduced in 1.5v.

Using enum we can create our own datatype called enumerated datatype.

When compare to old language enum , java enum is more powerful.

syntax:

```
      enum  type_name
      {
            val1,val2,....,valN
      }
```

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 a reference variable of enum type.

```
enum Months             public final class Months extends java.lang.Enum
{                       {
      JAN,FEB,MAR  ==>        public static final Months JAN=new Months();
}                             public static final Months FEB=new Months();
                              public static final Months MAR=new Months();
                        }
```

Declaration and Usage of enum

```
enum Months
{
            JAN,FEB,MAR
}
class Test
{
      public static void main(String[] args)
      {
            Months m=Months.JAN;
            System.out.println(m);
      }
}
```

Enum vs Switch case

From java 1.5v onwards it is possible pass enum to the switch case.

Diagram : java28.1

ex:

```
enum Months
{
            JAN,FEB,MAR
}
class Test
{
      public static void main(String[] args)
      {
            Months m=Months.FEB;
            switch(m)
            {
                  case JAN: System.out.println("January"); break;
                  case FEB: System.out.println("February"); break;
                  case MAR: System.out.println("March"); break;
            }
      }
}
```

Enum vs inheritance

Every enum internally consider as final so we can't create child enum.

Every enum extended with java.lang.Enum class so it is not possible to extend some other class.

But enum can implements interface.So we conclude by say this inheritance concept is not
applicable for enum.

ex:

```
      enum A
      {
      }
      enum B extends A
      {
      }
      o/p: invalid

ex:
---
      class A
      {
      }
      enum B extends A
      {
      }
      o/p: invalid
ex:
---
      interface A
      {
      }
      enum B implements A
      {
      }
      o/p: valid
```

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 Week
{
            MON,TUE,WED,THU,FRI,SAT,SUN
}
class Test
{
      public static void main(String[] args)
      {
            Week[] w=Week.values();

            //for each loop
            for(Week day:w)
            {
                  System.out.println(day+" ------------- "+day.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 Drinks
{
            COLA,PEPSI,SPRITE;
            //constructor
            Drinks()
            {
                  System.out.println("constructor");
            }
}
class Test
{
      public static void main(String[] args)
      {
            Drinks d=Drinks.COLA;
      }
}
o/p:
      constructor
      constructor
      constructor
```

ex:

```
enum Drinks
{
            COLA,PEPSI,SPRITE;

            //static variable
            static int i=10;

            public static void main(String[] args)
            {
                  System.out.println(i); //10
            }
}
o/p:
      javac Drinks.java
      java  Drinks
```

Wrapper classes
===============

The main objective of wrapper classes are

1) To wrap primitive type to wrapper object and vice versa.

2) To defined several utility methods.

primitive type	wrapper class
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
boolean	Boolean
char	Character

constructor

Every wrapper class contains following two constructors.One will take corresponding
primitive as an argument and another will take corresponding String as an argument.
ex:

Wrapper class	constructor
Byte	byte or String
Short	short or String
Integer	int or String
Long	long or String
Float	float or String
Double	double or String
Boolean	boolean or String
Character	char

ex:1

```
class Test
{
      public static void main(String[] args)
      {
            Integer i1=new Integer(10);

            Integer i2=new Integer("20");

            System.out.println(i1+" "+i2);
      }
}
```

ex:2

```
class Test
{
      public static void main(String[] args)
      {
            Boolean b1=new Boolean(true);

            Boolean b2=new Boolean("false");

            System.out.println(b1+" "+b2);
      }
}
```

ex:3

```
class Test
{
      public static void main(String[] args)
      {
            Character c=new Character('a');

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

Utility methods
----
1) valueOf()

      It is similar to constructor.
      It converts primitive type to wrapper object.
      ex:
```
            class Test
            {
                  public static void main(String[] args)
                  {
                        Integer i1=Integer.valueOf(10);
                        Long l1=Long.valueOf(10001);

                        Float f1=Float.valueOf(10.5f);

                        System.out.println(i1+" "+l1+" "+f1);
                  }
            }
```

2) xxxValue()

```
      It is used to convert wrapper object to primitive type.
      ex:
            class Test
            {
                  public static void main(String[] args)
                  {
                        Integer i=new Integer(10);

                        byte b=i.byteValue();

                        short s=i.shortValue();

                        System.out.println(b+" "+s);
                  }
            }
```

3) parseXxx()

```
      It is used to convert string to primitive type .
      ex:
            class Test
            {
                  public static void main(String[] args)
                  {
                        String str="35";

                        int i=Integer.parseInt(str);
                        System.out.println(i);

                        long l=Long.parseLong(str);
                        System.out.println(l);

                        float f=Float.parseFloat(str);
                        System.out.println(f);
                  }
            }
```

4) 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 s=i.toString();

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

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

Q)Write a query to display 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 number :");
            String binary1=sc.next();

            System.out.println("Enter the second binary number :");
            String binary2=sc.next();

            //convert binary to decimal
            int a=Integer.parseInt(binary1,2);
            int b=Integer.parseInt(binary2,2);

            //sum
            int c=a+b;

            //convert decimal to binary
            String result=Integer.toBinaryString(c);
            System.out.println(result);
      }
}
```

Types of objects in Java
=========================
We have two types of objects in java.

1)Immutable object

After object creation if we perform any changes then for every change a new object will be created such type of object is called immutable object.
ex:
      String and Wrapper classes

2)Mutable object

After object creation if we perform any changes then all the changes will be reflected
in a single object such type of object is called mutable object.
ex:
      StringBuffer and StringBuilder

String
=======
A String is a collection of characters which is enclosed in a double quotations.

case1:

Once if we create String object we can't perform any changes.If we perform any changes then
for every change a new object will be created such behaviour is called immutability of an
object.

Diagram: java28.2

case2:

Q)What is the difference between == and .equals() method?

==

It is a equality operator or comparision operator which always return boolean value.

It is used for reference comparision or address comparision.

ex:

```
class Test
{
      public static void main(String[] args)
      {
            String s1=new String("bhaskar");
            String s2=new String("solution");
            System.out.println(s1==s2); //false
      }
}
```

.equals()

It is a method present in String which always return boolean value.

It is used to content comparision and it is a case sensitive.

ex:

```
class Test
{
      public static void main(String[] args)
      {
            String s1=new String("bhaskar");
            String s2=new String("bhaskar");
            System.out.println(s1.equals(s2)); // true
      }
}
```

case3:

Once if we create a String object.Two objects will be created one is on heap and another
is on SCP (String Constant Pool) area.But 's' always points to heap area only.

ex:
```
      String s=new String("bhaskar");
```

Diagram: java29.1

Object creation in SCP area is always optional.First JVM will check is there any object
is created with same content or not.If it is created then it simply refers to

that object.
If it is not created then it will create a new object.Hence there is no chance of having
duplicate objects in SCP area.

SCP objects do not have any reference even though garbage collector can't access them.

SCP objects will destroy at the time when JVM shutdowns or terminated.

Diagram: java29.2

Interning of String object

With the help of heap object reference if we need corresponding SCP object reference then
we need to use intern() method.

Diagram: java29.3

String important methods
========================

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

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

            System.out.println("Enter the string :");
            String str=sc.next();

            System.out.println("Welcome :"+str);

      }
}
```

approach2

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

            System.out.println("Enter the string :");
            String str=sc.nextLine();

            System.out.println("Welcome :"+str);

      }
}
```

Q)Write a java program to remove special characters from given string?

input:
```
      Ih@ub_Tale$nt
```

output:
```
      IhubTalent
```

ex:

```
class Test
{
	public static void main(String[] args)
	{
		String str="Ih@ub_Tale$nt12";

		str=str.replaceAll("[^A-Za-z0-9]","");

		System.out.println(str);

	}
}
```

Q)Write a java program to display given character which is present in given index?

```
input:
	str = "bhaskar";

	index = 3;

output:
	s
```

ex:

```
class Test
{
	public static void main(String[] args)
	{
		String str="bhaskar";

		int index=3;

		char ch=str.charAt(index);

		System.out.println(ch);

	}
}
```

Q)Write a java program to check given word present in a given string or not?

```
input:
	str= "This is java class";

	word= "java";

output:
	It is present
```

ex:

```
class Test
{
	public static void main(String[] args)
	{
		String str="This is java class";

		String word="java";
```

```
            if(str.contains(word))
                  System.out.println("It is present");
            else
                  System.out.println("It is not present");

      }
}
```

Q)Write a java program to convert lowercase string to uppercase string?

```
input:
      bhaskar

output:
      BHASKAR
```

ex:

```
class Test
{
      public static void main(String[] args)
      {
            String str="bhaskar";

            str=str.toUpperCase();

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

Q)Write a java program to convert uppercase string to lowercase string?

```
input:
      BHASKAR
output:
      bhaskar
```

ex:

```
class Test
{
      public static void main(String[] args)
      {
            String str="BHASKAR";

            str=str.toLowerCase();

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

Q)Write a java program to find out length of string ?

```
input:
      bhaskar

output:
      7
```

ex:

```
class Test
```

```
{
	public static void main(String[] args)
	{
		String str="bhaskar";

		int len=str.length();

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

Q)Write a java program to check both strings are equals or not?

```
input:
	str1 = "bhaskar"
	str2 = "bhaskar"
output:
	Both are equals
```

ex:

```
class Test
{
	public static void main(String[] args)
	{
		String str1="bhaskar";

		String str2="bhaskar";

		if(str1.equals(str2))
			System.out.println("Both are equals");
		else
			System.out.println("Both are not equals");
	}
}
```

approach2

```
class Test
{
	public static void main(String[] args)
	{
		String str1="bhaskar";

		String str2="BHASKAR";

		if(str1.equalsIgnoreCase(str2))
			System.out.println("Both are equals");
		else
			System.out.println("Both are not equals");
	}
}
```

Q)Write a java program display the first occurance index number of a given character?

```
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 display the last occurance index number of a given character?

```
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 display reverse a string?

```
input:
	hello

output:
	olleh

ex:
---

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(); //  h   e   l   l   o

            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 find out 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="";

            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 each 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

            //for each loop
            String rev="";

            for(String s:sarr)
            {
                  //convert the string to char array
                  char[] carr=s.toCharArray(); // T   h   i   s

                  for(int i=carr.length-1;i>=0;i--)
                  {
                        rev+=carr[i];
                  }
                  //space
                  rev+=" ";

            }
            System.out.print(rev);

      }
}
```

Q)Write a java program to find out number of uppercase letters,lowercase letters, digits,
words and spaces?

```
input:
      This Is Java Class29
output:
      uppercase letters : 4
      lowecase letters  : 11
      digits              : 2
      words               : 4
      spaces        : 3
```

ex:

```
class Test
{
	public static void main(String[] args)
	{
		String str="This Is Java Class29";

		char[] carr=str.toCharArray();

		int upper=0,lower=0,digit=0,word=1,space=0;

		for(char ch:carr)
		{
			if(ch>='A' && ch<='Z')
			{
				upper++;
			}
			else if(ch>='a' && ch<='z')
			{
				lower++;
			}
			else if(ch>='0' && ch<='9')
			{
				digit++;
			}
			else if(ch==' ')
			{
				word++;
				space++;
			}
		}
		System.out.println("Uppercase letters : "+upper);
		System.out.println("Lowercase letters : "+lower);
		System.out.println("Digits : "+digit);
		System.out.println("Words : "+word);
		System.out.println("Spaces : "+space);
	}
}
```

Q)Write a java program to display duplicate characters from given String?

```
input:
	google

output:
	og
```

```
class Test
{
	public static void main(String[] args)
	{
		String str="google";

		String duplicates="";
		String characters="";

		for(int i=0;i<str.length();i++)
		{
			String current=Character.toString(str.charAt(i));

			if(characters.contains(current))
			{
				if(!duplicates.contains(current))
				{
					duplicates+=current;
					continue;
				}
```

```
                  }
                  characters+=current;
            }
            System.out.println(duplicates);
      }
}
```

Q)Write a java program to display unique/distinct characters from given String?

```
input:
      google

output:
      gole

ex:
---
class Test
{
      public static void main(String[] args)
      {
            String str="google";

            String duplicates="";
            String characters="";

            for(int i=0;i<str.length();i++)
            {
                  String current=Character.toString(str.charAt(i));

                  if(characters.contains(current))
                  {
                        if(!duplicates.contains(current))
                        {
                              duplicates+=current;
                              continue;
                        }
                  }
                  characters+=current;
            }
            System.out.println(characters);
      }
}
```

Q)Write a java program to display the string in below 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 most repeating character from given string?

```
input:
	googleformat

output:
	o is repeating for 3 times
```

ex:

```
class Test
{
	public static void main(String[] args)
	{
		String str="googleformat";

		int maxCount=0;
		char character=' ';

		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;
				character=str.charAt(i);
			}
		}
		System.out.println(character+" is repeating for "+maxCount+"
times");

	}
}
```

Q)Write a java program to perform right rotation of a string?

input:

```
      str = ihubtalent

      cnt = 2

output:
      ubtalentih

ex:
---
class Test
{
      public static void main(String[] args)
      {
            String str ="ihubtalent";

            int cnt =2;

            String str1=str.substring(0,2);

            String str2=str.substring(cnt,str.length());

            System.out.println(str2+str1);


      }
}
```

Q)Write a java program to perform left rotation of a string?

```
input:
      str = ihubtalent

      cnt = 2

output:
      ntihubtale

ex:
---
class Test
{
      public static void main(String[] args)
      {
            String str ="ihubtalent";

            int cnt =2;

            String str1=str.substring(0,str.length()-cnt);

            String str2=str.substring(str.length()-cnt,str.length());

            System.out.println(str2+str1);
      }
}
```

Q)Write a java program to display the strings starts with uppercase letters?

```
input:
      This is Java session For upcoming Students
output:
      This Java For Students

ex:
```

```
---
class Test
{
      public static void main(String[] args)
      {
            String str ="This is Java session For upcoming Students";

            String[] sarr=str.split(" ");

            //for each loop
            String rev="";
            for(String s:sarr)
            {
                  if(s.charAt(0)>='A' && s.charAt(0)<='Z')
                  {
                        rev+=s+" ";
                  }
            }
            System.out.println(rev);
      }
}
```

Q)Write a java to display permutations of a given string?

```
input:
      ABC

output:
      ABC
      ACB
      BAC
      BCA
      CBA
      CAB
```

ex:

```
public class Test
{
    public static void main(String[] args)
      {
            String str="ABC";
            permutations(str.toCharArray(),0);
    }
      public static void permutations(char[] ar, int fi)
      {
            if(fi==ar.length-1)
            {
                  System.out.println(ar);
                  return;
            }
            for(int i=fi;i<ar.length;i++)
            {
                  swap(ar,i,fi);
                  permutations(ar,fi+1);
                  swap(ar,i,fi);
            }
      }
      public static void swap(char[] ar,int i,int fi)
      {
            char temp=ar[i];
            ar[i]=ar[fi];
            ar[fi]=temp;
      }
}
```

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

Q)Write a java program to display the string in below format?

```
input:
      ABBCCCDDDD

output:
      A1B2C3D4
```

Q)Write a java program to find out given String is Anagram or not?

```
input:
      str1 = silent
      str2 = 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


            if(carr1.length!=carr2.length)
            {
                  System.out.println("It is not an anagram string");
                  System.exit(0);
            }

            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 anagram string");
            else
                  System.out.println("It is not anagram string");
      }
}
```

StringBuffer

=============
If our content is not fixed then it is never recommanded to use String because for every change a new object will be created.

To overcome this limitation Sun Micro System introduced StringBuffer.

In StringBuffer 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 capacity reaches to maximum capacity then new capacity will be created with below formulea.
ex:
```
      new capacity = current_capacity+1*2;
```

ex:

```
class Test
{
      public static void main(String[] args)
      {
            StringBuffer sb=new StringBuffer();

            System.out.println(sb.capacity()); // 16

            sb.append("abcdefghijklmnop");

            System.out.println(sb.capacity()); // 16

            sb.append("qr");

            System.out.println(sb.capacity()); // 16+1*2=34
      }
}
```

2)StringBuffer sb=new StringBuffer(int initialcapacity)

It will create StringBuffer object with specified initial capacity.

ex:

```
class Test
{
      public static void main(String[] args)
      {
            StringBuffer sb=new StringBuffer(19);

            System.out.println(sb.capacity()); // 19

      }
}
```

3)StringBuffer sb=new StringBuffer(String str)

It will create StringBuffer object which is equivalent to String.

Here capacity will be created with below formulea.

```
      capacity = str.length() + 16
```

ex:

```
class Test
{
      public static void main(String[] args)
      {
            StringBuffer sb=new StringBuffer("bhaskar");

            System.out.println(sb.capacity()); // 7 + 16 = 23

      }
}
```

Q)Write a java program to display reverse of a string?

```
class Test
{
      public static void main(String[] args)
      {
            String str="hello";

            String rev="";

            StringBuffer sb=new StringBuffer(str);

            rev=sb.reverse().toString();

            System.out.println(rev);

      }
}
```

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

```
class Test
{
      public static void main(String[] args)
      {
            String str="madam";

            String rev="";

            StringBuffer sb=new StringBuffer(str);

            rev=sb.reverse().toString();

            if(str.equals(rev))
                  System.out.println("It is palindrome string");
            else
                  System.out.println("It is not palindrome string");

      }
}
```

Q)Write a java program to display the string in given format?

```
input:
      ABBCCCDDDD
output:
      A1B2C3D4
```

ex:

```
public class Test
{
    public static void main(String[] args)
      {
            String str = "ABBCCCDDDD";

            StringBuffer sb = new StringBuffer();

        int count = 1;
        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());
    }
}
```

StringBuilder
==============
StringBuilder is exactly same as StringBuffer with following differences.

StringBuffer	StringBuilder
All the methods present in StringBuffer are synchronized.	No method present in StringBuilder are synchronized.
At a time only one thread is allowed to execute.Hence we can achieve thread safety.	Multiple threads are allowed to execute. Hence we can't achieve thread safety.
Waiting time of a thread will increase effectively performance is low.	There is no waiting threads effectively performance is high.
It is introduced in 1.0v.	It is introduced in 1.5v.

Note:

If our content not change frequently then we need to use String object.

If our content change frequently where thread safety is required then we need to use StringBuffer object.

If our content change frequently where thread safety is not required then we need to use
StringBuilder object.

StringTokenizer
================
A 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 class object as follow.

syntax:

```
      StringTokenizer st=new StringTokenizer(String str,RegEx reg);
```

StringTokenizer class contains following methods.
ex:

```
      public int countTokens()
      public boolean hasMoreTokens()
      public String nextToken()
      public boolean hasMoreElements()
      public Object nextElement
```

ex:

```
import java.util.StringTokenizer;
public class Test
{
    public static void main(String[] args)
      {
            StringTokenizer st=new StringTokenizer("This is java class");

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

Note:

      Here default regular expression is space.

ex:2

```
import java.util.StringTokenizer;
public class Test
{
    public static void main(String[] args)
      {
            StringTokenizer st=new StringTokenizer("This is java class"," ");

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

ex:3

```
import java.util.StringTokenizer;
public class Test
{
    public static void main(String[] args)
      {
            StringTokenizer st=new StringTokenizer("This is java class"," ");

            while(st.hasMoreTokens())
            {
                  String str=st.nextToken();
                  System.out.println(str);
            }

    }
}
```

```
ex:4
-----
import java.util.StringTokenizer;
public class Test
{
    public static void main(String[] args)
      {
            StringTokenizer st=new StringTokenizer("This is java class"," ");

            while(st.hasMoreElements())
            {
                  String str=(String)st.nextElement();
                  System.out.println(str);
            }

    }
}
```

```
ex:5
----
import java.util.StringTokenizer;
public class Test
{
    public static void main(String[] args)
      {
            StringTokenizer st=new StringTokenizer("9,99,999",",");

            while(st.hasMoreElements())
            {
                  String str=(String)st.nextElement();
                  System.out.println(str);
            }

    }
}
```

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
      IllegalArgumentException
      NegativeArraySizeException
      and etc.

Error

Error is a problem for which we can't provide solution programmatically.

Error will raise due to lack of system resources.

ex:
      OutOfMemoryError
      StackOverFlowError
      LinkageError

```
      and etc.

As a part of java application development, it is a responsibility of a
programmer to
provide smooth termination for every java program.

We have two types of terminations.

1) Smooth termination / Graceful termination
-------------------------------------------
During the program execution suppose if we are not getting any interruption in
the middle
of the program such type of termination is called smooth termination.


2) Abnormal termination
----------------------
During the program execution suppose if we are getting some interruptions in the
middle
of the program such type of termination is called abnormal termination.
ex:
public class Test
{
    public static void main(String[] args)
    {
            System.out.println(10/0);
    }
}

If any exception raised in our program we must and should handle that exception
otherwise
our program will terminate abnormally.

Here exception will display name of the exception, description of the exception
and
line number of the exception.

Exception
==========
It is a unwanted , unexpect event which disturbs normal flow of our program.

Exceptions always raise at runtime so they are also known as runtime events.

The main objective of exception handling is to provide graceful termination.

In java exceptions are divided into two types.

1)Predefined exceptions

2)Userdefined exceptions

1)Predefined exceptions
------------------------
Built-In exceptions are called predefined exceptions.

It is divided into two types.

i) Checked Exceptions
--------------------
Exceptions which are checked by the compiler at the time of compilation are
called
checked exceptions.
ex:
      FileNotFoundException
      InterruptedException
      IOException
      EOFException

ii) Unchecked Exceptions
--------------------------
Exceptions which are checked by the JVM at the time of runtime are called
unchecked exceptions.
ex:
      ClassCastException
      ArithmeticException
      IllegalArgumentException

Diagram: java32.1

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 always associate with catch block.

A try block is used to throw the exception to catch block.

If any exception raised in try block then try won't be executed , it will jump
catch block.

If any exception raised in the middle of the try block then rest of the code
won't be executed.

catch block
===========
It is a block which contains error handling code.

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 ae)
{
      -
      - // Error Handling Code
      -
}

ex:1
-----
class Test
{
	public static void main(String[] args)
	{
		try
		{
			System.out.println("try-block");
		}
		catch (Exception e)
		{
			System.out.println("catch-block");
		}
	}
}
o/p:
	try-block

ex:2
----
class Test
{
	public static void main(String[] args)
	{
		try
		{
			System.out.println(10/0);
		}
		catch (Exception e)
		{
			System.out.println("catch-block");
		}
	}
}
o/p:
	catch-block

ex:3
-----
class Test
{
	public static void main(String[] args)
	{
		try
		{
			System.out.println("stmt1");
			System.out.println(10/0);
			System.out.println("stmt2");
		}
		catch (Exception e)
		{
			System.out.println("catch-block");
		}
	}
}
o/p:
	stmt1
	catch-block

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

A try with multiple catch blocks

A try block can have multiple catch blocks.

If a try block contains multiple catch blocks 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");
            }
      }
}
o/p:
      from AE
```

Various methods to display exception details

Throwable class defines following methods to display exception details.

1) printStackTrace()

It will display name of the exception, description of the exception and line number of the exception.

2) toString()

It will display name of the exception and description of the exception.

3) getMessage()

It will display description of the exception.

ex:

```
class Test
{
      public static void main(String[] args)
      {
            try
            {
                  System.out.println(10/0);
            }
            catch (ArithmeticException ae)
            {
                  ae.printStackTrace();

                  System.out.println("=====================");

                  System.out.println(ae.toString());

                  System.out.println("=====================");

                  System.out.println(ae.getMessage());
            }

      }
}
```

Q)How to handle multiples exceptions in catch block?

```
class Test
{
      public static void main(String[] args)
      {
            try
            {
                  System.out.println(10/0);
            }
            catch (ArithmeticException | ClassCastException |
NullPointerException  e)
            {
                  e.printStackTrace();
            }

      }
}
```

finally block
===============
It is never recommanded 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 recommanded to maintain cleanup code in catch block because if there is no exception in try block then catch block won't be executed.

We need a place where we can maintain the cleanup code and it should execute irrespective of exception is raised 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 (ArithmeticException ae)
            {
                  ae.printStackTrace();
            }
            finally
            {
                  System.out.println("finally-block");
            }

      }
}
```

o/p:
```
      try-block
      finally-block
```

ex:

```
class Test
{
      public static void main(String[] args)
      {
            try
            {
                  System.out.println(10/0);
            }
            catch (ArithmeticException ae)
            {
                  ae.printStackTrace();
            }
            finally
            {
                  System.out.println("finally-block");
            }

      }
}
```
o/p:
```
      java.lang.ArithmeticException: / by zero
        at Test.main(Test.java:7)
      finally-block
```

Note:

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");
            }

      }
}
```
o/p:
```
      try-block
      finally-block
```

Q)What is the difference between final,finally and finalized method ?

final
------
final is a modifier which is applicable for variables, methods and classes.

If we declare any variable as final then reassignment of that variable is not possible.

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

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

finally

It block which contains cleanup code and it will execute irrespective of exception raised or not.

finalized method

It is a method called by garbage collector just before destroying an object for cleanup activity.

throw statement
================
Sometimes we will create exception object explicitly and handover to JVM manually
by using throw statement.
ex:
```
      throw new ArithmeticException("don't divide by zero");
```

ex:

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

Here exception object is created and hover to JVM by using main method.

ex:

```
class Test
{
      public static void main(String[] args)
      {
                  throw new ArithmeticException("don't divide by zero");
      }
}
```

Here exception object is created and handover to JVM by using throw statement.

throws statement
================
If any checked exception raised in our program the we must and should handle that exceptions by using try and catch block or by using throws statement.

ex:1

```
class Test
{
      public static void main(String[] args)
      {
                  try
                  {
                        Thread.sleep(3000);
                        System.out.println("Welcome to Java");
                  }
                  catch (InterruptedException ie)
                  {
                        ie.printStackTrace();
                  }
      }
}
```

ex:2

```
class Test
{
      public static void main(String[] args)throws InterruptedException
      {
                  Thread.sleep(5000);
                  System.out.println("Welcome to Java");
      }
}
```

2)Userdefined exceptions
=======================
Exceptions which are created by the user based on the application requirements are called customized exceptions.
ex:
```
      NoInterestInJavaException
      NoPracticeInLabException
      ClassGettingBoreException
      TooYoungException
      TooOldException
      and etc.
```

```
ex:
---
import java.util.Scanner;

class TooYoungException extends RuntimeException
{
	TooYoungException(String s)
	{
		super(s);
	}
}
class TooOldException extends RuntimeException
{
	TooOldException(String s)
	{
		super(s);
	}
}
class Test
{
	public static void main(String[] args)
	{
		Scanner sc=new Scanner(System.in);

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

		if(age<18)
			throw new TooYoungException("U r not eligible to vote");
		else
			throw new TooOldException("U r eligible to vote");

	}
}
```

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
----
ex:
----
import java.io.*;
class  Test
{
      public static void main(String[] args)throws IOException
      {
            FileReader fr=new FileReader("aaa.txt");

            char[] ch=new char[255];

            fr.read(ch);

            //for each loop
            for(char c:ch)
            {
                  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 keyboard
==
There are many ways to provide input values from keyboard.

1) command line argument

2) Console class

3) BufferedReader class

4) Scanner class

1) command line argument

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

            System.out.println("Welcome :"+name);
      }
}
```

```
o/p:
      javac   Test.java

      java    Test   DennisRitchie


2) Console class
-----------------
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);
      }
}

3) BufferedReader class
------------------------
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);
      }
}

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

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

            System.out.println("Enter the employee name :");
            String name=sc.next();

            System.out.println("Enter the employee salary :");
            float sal=sc.nextFloat();

            System.out.println(id+" "+name+" "+sal);
      }
}
```

Standard procedure to write java program
==

```
import java.io.*;
class  Test
{
      public static void main(String[] args)
      {
            BufferedReader br=null;
            try
            {
                  br=new BufferedReader(new FileReader("aaa.txt"));

                  String line=br.readLine();

                  while(line!=null)
                  {
                        System.out.println(line);
                        line=br.readLine();
                  }
            }
            catch (IOException ioe)
            {
                  ioe.printStackTrace();
            }
            finally
            {
                  try
                  {
                        br.close();
                  }
                  catch (IOException ioe)
                  {
                        ioe.printStackTrace();
                  }
            }
      }
}
```

Try with Resources
===================

In Java, the try-with-resources statement is, a try statement that declares one or more resources.

The try-with-resources statement ensures that each resource is closed at the end of the statement execution.Hence we don't need to use finally block to close the objects.

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

            try(BufferedReader br=new BufferedReader(new 
FileReader("aaa.txt"));)
            {
                  String line=br.readLine();

                  while(line!=null)
                  {
                        System.out.println(line);
                        line=br.readLine();
                  }
            }
            catch (IOException ioe)
            {
                  ioe.printStackTrace();
            }

      }
}
```

Generics
=========
Arrays are typesafe.It means we can provide guarantee that what type of elements are present in arrays.

If requirement is there to store String values then it is recommanded to use String[] array.
ex:

```
      String[] sarr=new String[10];
      sarr[0]="hi";
      sarr[1]="hello";
      sarr[2]="bye";
      sarr[3]=10; // invalid
```

At the time retrieving the data we don't need to perform any typecasting.
ex:

```
      String[] sarr=new String[10];
      sarr[0]="hi";
      sarr[1]="hello";
      sarr[2]="bye";
      -
      -
      String val1=sarr[0];
```

Collections are not typesafe.It means we can't provide guaranatee that what type of elements are present in Collections.

If requirement is there to store String values then it is never recommanded 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("bye");
      al.add(10);
```

At the time of retrieving the data compulsary we need to perform typecasting.
ex:

```
      ArrayList al=new ArrayList();
      al.add("hi");
      al.add("hello");
      al.add("bye");
      al.add(10);
      -
      -
      String str=(String)al.get(0);
```

To overcome this limitation 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 elements.
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.
It holds primitive types and object types.	It holds only object types.
Arrays are not implemented based on data structure concept.So we can't expect ready made methods.For every logic we need to write the code explicitly.	Collections are implemented based on data structure concept.So we can expect ready made methods.

Collection

It is an interface which is present in java.util package.

It is a root interface for entire Collection framework.

If we want to represent group of individual objects in a single entity then we need to use Colection interface.

Collection interface contains following methods.

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 <T> T[] toArray(T[]);
  public abstract boolean add(E);
  public abstract boolean remove(java.lang.Object);
  public abstract boolean containsAll(java.util.Collection<?>);
  public abstract boolean addAll(java.util.Collection<? extends E>);
  public abstract boolean removeAll(java.util.Collection<?>);
  public boolean removeIf(java.util.function.Predicate<? super E>);
  public abstract boolean retainAll(java.util.Collection<?>);
  public abstract void clear();
  public abstract boolean equals(java.lang.Object);
  public abstract int hashCode();
  and etc.
```

List

======
It is a child interface of Collection interface.

If we want to represent group of individual objects in a single entity where duplicate objects are allowed and order is preserved then we need to use List interface.

Diagram: java34.1

ArrayList
=========
The underlying data structure is resizable array or growable array.

Duplicate objects are allowed.

Order is preserved.

Hetrogeneous 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:1

```
import java.util.*;
class Test
{
      public static void main(String[] args)
      {
            ArrayList al=new ArrayList();
            al.add("one");
            al.add("two");
            al.add("three");
            System.out.println(al);//[one,two,three]
            al.add("one");
            System.out.println(al);//[one,two,three,one]
            al.add(10);
            System.out.println(al);//[one,two,three,one,10]
            al.add(null);
            System.out.println(al);//[one,two,three,one,10,null]
      }
}
```

ex:2

```
import java.util.*;
class Test
{
      public static void main(String[] args)
      {
            ArrayList<String> al=new ArrayList<String>();
            al.add("one");
            al.add("two");
            al.add("three");
            System.out.println(al);//[one,two,three]
            al.add("one");
            System.out.println(al);//[one,two,three,one]
            al.add(null);
            System.out.println(al);//[one,two,three,one,null]
      }
}

ex:3
----
import java.util.*;
class Test
{
      public static void main(String[] args)
      {
            ArrayList<String> al=new ArrayList<String>();

            System.out.println(al.isEmpty()); //true
            al.add("one");
            al.add("two");
            al.add("three");

            for(int i=0;i<al.size();i++)
            {
                  String s=al.get(i);
                  System.out.println(s);
            }

      }
}

ex:4
----
import java.util.*;
class Test
{
      public static void main(String[] args)
      {
            ArrayList<String> al=new ArrayList<String>();
            al.add("one");
            al.add("two");
            al.add("three");

            System.out.println(al.contains("one")); //true

            al.remove("three");

            System.out.println(al);//[one,two]

            al.clear();
            System.out.println(al);//[]

      }
}

ex:5
-----
import java.util.*;
class Test
{
      public static void main(String[] args)
      {
            List<String> list=new ArrayList<String>();
            list.add("one");
            list.add("two");
            list.add("three");

            System.out.println(list); //[one,two,three]
      }
```

```
}

ex:6
----
import java.util.*;
class Test
{
      public static void main(String[] args)
      {
            List<String> list=new ArrayList<String>();
            list.add(new String("one"));
            list.add(new String("two"));
            list.add(new String("three"));
            System.out.println(list); //[one,two,three]
      }
}

ex:7
----
import java.util.*;
class Test
{
      public static void main(String[] args)
      {
            ArrayList<String> al1=new ArrayList<String>();
            al1.add("one");
            al1.add("two");
            al1.add("three");
            System.out.println(al1);//[one,two,three]

            ArrayList<String> al2=new ArrayList<String>();
            al2.add("raja");
            System.out.println(al2); //[raja]

            al2.addAll(al1);
            System.out.println(al2);//[raja,one,two,three]

            System.out.println(al2.containsAll(al1)); // true

            al2.removeAll(al1);
            System.out.println(al2);//[raja]

      }
}

ex:8
-----
import java.util.*;
class Test
{
      public static void main(String[] args)
      {
            List<String> list=Arrays.asList("one","two","three");

            System.out.println(list);//[one,two,three]

      }
}

ex:
---
import java.util.*;
class Test
{
	public static void main(String[] args)
	{
		List<Integer> list=Arrays.asList(10,20,30,40);

		System.out.println(list);//[10,20,30,40]

	}
}
```

LinkedList
==========
The underlying data structure is doubly LinkedList.

Duplicate objects are allowed.

Order is preserved.

Hetrogeneous objects are allowed.

Null insertion is possible.

It implements Serializable, Cloneable and Deque interface.

LinkedList contains following methods.
ex:
```
	public E getFirst();
	public E getLast();
	public E removeFirst();
	public E removeLast();
	public void addFirst(E);
	public void addLast(E);
```

If our frequent operation is a insertion and deletion in the middle then LinkedList is a best choice.

ex:1

```
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:2

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

```
      {
            LinkedList<String> ll=new LinkedList<String>();
            ll.add("one");
            ll.add("two");
            ll.add("three");
            System.out.println(ll);//[one,two,three]
            ll.add("one");
            System.out.println(ll);//[one,two,three,one]
            ll.add(null);
            System.out.println(ll);//[one,two,three,one,null]

      }
}
```

ex:3

```
import java.util.*;
class Test
{
      public static void main(String[] args)
      {
            LinkedList<String> ll=new LinkedList<String>();
            ll.add("one");
            ll.add("two");
            ll.add("three");
            System.out.println(ll);//[one,two,three]

            ll.addFirst("gogo");
            ll.addLast("jojo");
            System.out.println(ll);//[gogo,one,two,three,jojo]

            System.out.println(ll.getFirst());//gogo
            System.out.println(ll.getLast());//jojo

            ll.removeFirst();
            ll.removeLast();
            System.out.println(ll);//[one,two,three]

      }
}
```

ex:

```
import java.util.*;
class Test
{
      public static void main(String[] args)
      {
            LinkedList<String> ll=new LinkedList<String>();
            ll.add("one");
            ll.add("two");
            ll.add("three");
            System.out.println(ll);//[one,two,three]

            ll.add(1,"raja");
            System.out.println(ll);//[one,raja,two,three]

      }
}
```

Assignment
==========

Method overloading vs Method overriding

interface vs Abstract class

StringBuffer vs StringBuilder

Arrays vs Collections

JDK vs JRE vs JVM

final vs finally vs finalized method

length vs length()

== vs .equals()

Vector
======
The underlying data structure is resizable array or growable array.

Insertion order is preserved.

Duplicate objectes are allowed.

Hetrogeneous objects are allowed.

Null insertion is possible.

It implements Serializable,Cloneable and RandomAccess interface.

All the methods present in Vector are synchronized.Hence we achieve thread safety.

We have following methods in Vector class.
ex:
```
      addElement()
      removeElementAt()
      removeAllElements()
      firstElement()
      lastElement()
      and etc.
```

ex:

```
import java.util.*;
class Test
{
      public static void main(String[] args)
      {
            Vector<Integer> v=new Vector<Integer>();

            System.out.println(v.capacity());

            for(int i=1;i<=10;i++)
            {
                  v.addElement(i);
            }
            System.out.println(v);//[1,2,3,4,5,6,7,8,9,10]

            System.out.println(v.firstElement());//1
            System.out.println(v.lastElement());//10

            v.removeElementAt(5);
            System.out.println(v);//[1, 2, 3, 4, 5, 7, 8, 9, 10]
            v.removeAllElements();
            System.out.println(v); //[]


      }
}
```

ex:

```
import java.util.*;
class Test
{
      public static void main(String[] args)
      {
            Vector<Integer> v=new Vector<Integer>();

            System.out.println(v.capacity());

            for(int i=1;i<=10;i++)
            {
                  v.add(i);
            }
            System.out.println(v);//[1,2,3,4,5,6,7,8,9,10]

            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.clear();
            System.out.println(v); //[]
      }
}
```

Stack
=======
It is a child class of Vector class.

If we depend upon last in firt out(LIFO) order then we need to use stack.

constructor

```
Stack s=new Stack();
```

Methods
----
push(Object obj)

      It is used to push the element to stack.
pop()

      It is used to delete the element from stack.

peek()

      It will return toppest element from stack.

isEmpty()

      It will check stack is empty or not.

search(Object obj)

      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("S")); // -1

		System.out.println(s.search("A")); // 2

	}
}
```

Set
====
It is a child interface of Collection interface.

If we want to represent group of individual objects in a single entity where duplicate objects are not allowed and order is not preserved.

Diagam: java35.1

HashSet

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.

Hetrogeneous objects are allowed.

Null insertion is possible.

It implements Serializable and Cloneable interface.

ex:

```
import java.util.*;
class Test
{
	public static void main(String[] args)
	{
		HashSet hs=new HashSet();
		hs.add("one");
```

```
            hs.add("five");
            hs.add("nine");
            System.out.println(hs); //[nine, one, five]
            hs.add("one");
            System.out.println(hs); //[nine, one, five]
            hs.add(10);
            System.out.println(hs); //[nine, one, 10, five]
            hs.add(null);
            System.out.println(hs); //[null, nine, one, 10, five]

      }
}
```

LinkedHashSet

It is a child class of HashSet class.

LinkedHashSet is exactly same as HashSet class with following differences.

HashSet	LinkedHashSet
The underlying data structure is Hashtable.	The underlying data structure is Hashtable and LinkedList.
Insertion order is not preserved.	Insertion order is preserved.
It is introduced in 1.2v.	It is introduced in 1.4v.

ex:

```
import java.util.*;
class Test
{
      public static void main(String[] args)
      {
            LinkedHashSet lhs=new LinkedHashSet();
            lhs.add("one");
            lhs.add("five");
            lhs.add("nine");
            System.out.println(lhs); //[one, five, nine]
            lhs.add("one");
            System.out.println(lhs); //[one, five, nine]
            lhs.add(10);
            System.out.println(lhs); //[one, five, nine, 10]
            lhs.add(null);
            System.out.println(lhs); //[one, five, nine, 10, null]

      }
}
```

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.

Hetrogeneous objects are not allowed otherwise we will get ClassCastException.

Null insertion is not possible otherwise 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(7);
            System.out.println(ts); //[1, 5, 7, 10]

            ts.add(1);
            System.out.println(ts); //[1, 5, 7, 10]

            //ts.add("hi");
            //System.out.println(ts); //R.E ClassCastException

            //ts.add(null);
            //System.out.println(ts); // R.E NullPointerException

      }
}
```

Q)What is the difference between Comparable and Comparator interface?

Comparable

Comparable is an interface which is present in java.lang package.

It contains only one method i.e compareTo() method.

If we depend 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 interface present in java.util package.

Comparator interface contains following two methods i.e compare() and equals() method.

```
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.

ex:

      public boolean equals(Object o)
```

Implemention of equals() method is optional because it is present in Object class so it is available through inheritance.

If we depend upon customized sorting order then we need to use Comparator interface.

```
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(5);
            ts.add(1);
            ts.add(7);
            System.out.println(ts); //[10,7,5,1]
      }
}
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(5);
            ts.add(1);
            ts.add(7);
            System.out.println(ts); //[1,5,7,10]
      }
}
class MyComparator implements Comparator
{
      public int compare(Object obj1,Object obj2)
      {
            Integer i1=(Integer)obj1;
            Integer i2=(Integer)obj2;

            if(i1<i2)
                  return -1;
            else if(i1>i2)
                  return 1;
            else
                  return 0;

      }
}
```

Interview Question
==================
Q)Write a java program to check given String is balanced or not?

input:
      ({[]})

output:
      It is balanced string

ex:

```
import java.util.*;
class Test
{
      public static void main(String[] args)
      {
            String str="({[[]})";

            Stack<Character> stack=new Stack<Character>();

            //for each loop
            for(char ch:str.toCharArray())
            {
                  if(ch=='(' || ch=='{' || ch=='[')
                  {
                        stack.push(ch);
                  }
                  else if(ch==')' && !stack.isEmpty() && stack.peek()=='(')
                  {
                        stack.pop();
                  }
                  else if(ch=='}' && !stack.isEmpty() && stack.peek()=='{')
                  {
                        stack.pop();
                  }
                  else if(ch==']' && !stack.isEmpty() && stack.peek()=='[')
                  {
                        stack.pop();
                  }
            }

            if(stack.isEmpty())
                  System.out.println("It is  balanced String");
            else
                  System.out.println("It is not balanced String");
      }
}
```

Map
====
It is a not a child interface of Collection interface.

If we want to represent group of individual objects in key,value pair then we need to use Map interface.

Key and value both must be objects.

Key can't duplicate but value can be duplicate.

Each key and value pair is called one entry.

Diagram: java36.1

HashMap

The underlying data structure is Hashtable.

Key can't be duplicated but value can be duplicated.

Insertion is not preserved because it will take hashcode of the 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("five","alan");
            hm.put("nine","jose");
            hm.put("six","nelson");
            System.out.println(hm);//{nine=jose, six=nelson, one=raja, 
five=alan}
            hm.put("one","rani");
            System.out.println(hm);//{nine=jose, six=nelson, one=rani, 
five=alan}
            hm.put(1,10);
            System.out.println(hm);//{nine=jose, 1=10, six=nelson, one=rani, 
five=alan}
            hm.put(null,null);
            System.out.println(hm);//{null=null, nine=jose, 1=10, six=nelson, 
one=rani, five=alan}
      }
}
```

ex:

```
import java.util.*;
class Test
{
      public static void main(String[] args)
      {
            HashMap<String,String> hm=new HashMap<String,String>();
            hm.put("one","raja");
            hm.put("five","alan");
            hm.put("nine","jose");
            hm.put("six","nelson");

            Set s=hm.keySet();
            System.out.println(s);//[nine, six, one, five]

            Collection c=hm.values();
            System.out.println(c);//[jose, nelson, raja, alan]

            Set s1=hm.entrySet();
            System.out.println(s1);//[nine=jose, six=nelson, one=raja, 
five=alan]
      }
}
```

LinkedHashMap

It is a child class of HashMap class.

LinkedHashMap is exactly as HashMap class with following differences.

HashMap	LinkedHashMap
The underlying datastructure is Hashtable.	The underlying datastructure is Hashtable and LinkedList.
Insertion order is not preserved.	Insertion order is preserved.
Introduced in 1.2v.	Introduced in 1.4v.

ex:

```
import java.util.*;
class Test
{
      public static void main(String[] args)
      {
            LinkedHashMap<String,String> lhm=new LinkedHashMap<String,String>();
            lhm.put("one","raja");
            lhm.put("five","alan");
            lhm.put("nine","jose");
            lhm.put("six","nelson");
            System.out.println(lhm);//{one=raja, five=alan, nine=jose, 
six=nelson}
            lhm.put("one","rani");
            System.out.println(lhm);//{one=rani, five=alan, nine=jose, 
six=nelson}
            lhm.put(null,null);
            System.out.println(lhm);//{one=rani, five=alan, nine=jose, 
six=nelson, null=null}
      }
}
```

TreeMap

The underlying data structure is RED BLACK TREE.

Key can't be duplicate but value can be duplicated.

Insertion order is not preserved because it will take sorting order of the key.

If we depend upon default natural sorting order then key must be homogeneous and Comparable.

If we depend upon 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");
            tm.put(3,"three");
            System.out.println(tm);//{1=one, 3=three, 5=five, 10=ten}
            tm.put(1,"hundred");
            System.out.println(tm);//{1=hundred, 3=three, 5=five, 10=ten}
            tm.put(4,null);
            System.out.println(tm);//{1=hundred, 3=three, 4=null, 5=five, 
10=ten}
            tm.put(null,"six");
            System.out.println(tm);//R.E NullPointerException
      }
}
```

Hashtable

The underlying data structure is Hashtable.

Key can't be duplicate but value can be duplicated.

Insertion order is not preserved because it will take descending order of the key.

Hetrogeneous objects are allowed for both key and value.

Null insertion is not possible for both key and value.

ex:

```
import java.util.*;
class Test
{
      public static void main(String[] args)
      {
            Hashtable<Integer,String> ht=new Hashtable<Integer,String>();
            ht.put(10,"ten");
            ht.put(1,"one");
            ht.put(5,"five");
            ht.put(3,"three");
            System.out.println(ht);//{10=ten, 5=five, 3=three, 1=one}

            ht.put(1,"hundred");
            System.out.println(ht);//{10=ten, 5=five, 3=three, 1=hundred}

            //ht.put(4,null);
            //System.out.println(ht);//NullPointerException

            //ht.put(null,"four");
            //System.out.println(ht);//NullPointerException
      }
}
```

Types of Cursors
================
Cursor is used to read 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 as follow.
ex:
```
      Enumeration e=v.elements();
```

Enumeration interface contains following methods.
ex:
```
      public boolean hasMoreElements()
      public Object nextElement()
```

ex:

```
import java.util.*;
class Test
{
      public static void main(String[] args)
      {
            Vector v=new Vector();
            for(int i=1;i<=10;i++)
            {
                  v.add(i);
            }
            System.out.println(v);//[1,2,3,4,5,6,7,8,9,10]

            Enumeration e=v.elements();

            while(e.hasMoreElements())
            {
                  Integer i=(Integer)e.nextElement();
                  System.out.println(i);
            }


      }
}
```
Limitations with Enumeration

Using Enumeration interface we can read objects only from Legacy Collection objects.Hence it is not a universal cursor.

Using Enumeration , we can perform only read operation but not remove operation.

To overcome these limitations , Sun Micro System introduced Iterator interface.

2)Iterator

It is used to read the objects one by one from any Collection object.Hence it is a universal cursor.

Iterator interface can perform read and remove operations.

Iterator object can be created as follow.
ex:
```
      Iterator itr=al.iterator();
```

Iterator interface contains following methods.
ex:
```
      public boolean hasNext()
      public Object next()
      public void remove()
```

ex:

```
import java.util.*;
class Test
{
      public static void main(String[] args)
      {
            ArrayList al=new ArrayList();
            for(int i=1;i<=10;i++)
            {
                  al.add(i);
            }
            System.out.println(al);//[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

            Iterator itr=al.iterator();
            while(itr.hasNext())
            {
                  Integer i=(Integer)itr.next();
                  if(i%2==0)
                  {
                        itr.remove();
                  }
            }
            System.out.println(al);//[1,3,5,7,9]
      }
}
```

Limitations with Iterator

Using Enumeration and Iterator we can read objects only in forward direction but 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 objects.

To overcome these limitations Sun Micro System introduced ListIterator.

3)ListIterator

IT is used to read objects one by one from List Collection objects.

ListIterator interface can perform read, remove , adding and replacement of new objects.

We can create ListIterator interface object as follow.

```
ex:
      ListIterator litr=al.listIterator();
```

ListIterator contains following methods.

```
ex:
      public boolean hasNext()
      public Object next()
      public boolean hasPrevious()
      public Object previous()
      public void remove();
      public void nextIndex();
      public void previousIndex();
      public void set(Object o);
      public void add(Object o);
```

```
ex:
---
import java.util.*;
class Test
{
      public static void main(String[] args)
      {
            ArrayList al=new ArrayList();
            al.add("bala");
            al.add("nag");
            al.add("venki");
            al.add("chiru");
            System.out.println(al);//[bala,nag,venki,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("bala");
            al.add("nag");
            al.add("venki");
            al.add("chiru");
            System.out.println(al);//[bala,nag,venki,chiru]

            ListIterator litr=al.listIterator();

            while(litr.hasNext())
            {
                  String s=(String)litr.next();
                  if(s.equals("nag"))
                  {
                        litr.remove();
                  }
            }
            System.out.println(al);//[bala,venki,chiru]

      }
}
```

ex:

```
import java.util.*;
class Test
{
      public static void main(String[] args)
      {
            ArrayList al=new ArrayList();
            al.add("bala");
            al.add("nag");
            al.add("venki");
            al.add("chiru");
            System.out.println(al);//[bala,nag,venki,chiru]

            ListIterator litr=al.listIterator();

            while(litr.hasNext())
            {
                  String s=(String)litr.next();
                  if(s.equals("nag"))
                  {
                        litr.add("chaitanya");
                  }
            }
            System.out.println(al);//[bala,nag,chaitanya,venki,chiru]

      }
}
```

ex:

```
import java.util.*;
class Test
{
      public static void main(String[] args)
      {
            ArrayList al=new ArrayList();
            al.add("bala");
            al.add("nag");
            al.add("venki");
            al.add("chiru");
            System.out.println(al);//[bala,nag,venki,chiru]

            ListIterator litr=al.listIterator();

            while(litr.hasNext())
            {
                  String s=(String)litr.next();
                  if(s.equals("nag"))
                  {
                        litr.set("chaitanya");
                  }
            }
            System.out.println(al);//[bala, chaitanya, venki, chiru]

      }
}
```

Interview Questions
==================
Q)Write a java program to display number of words present in a given String?

input:

```
	This is is java java class
```

output:

```
	This=1, is=2, java=2, class=1
```

```
import java.util.*;
class Test
{
	public static void main(String[] args)
	{
		String str="This is is java java class";

		Map<String,Integer> map=new LinkedHashMap<String,Integer>();

		String[] sarr=str.split(" ");

		//for each loop
		for(String s:sarr)
		{
			if(map.get(s)!=null)
			{
				map.put(s,map.get(s)+1);
			}
			else
			{
				map.put(s,1);
			}
		}
		System.out.println(map);
	}
}
```

Q)Write a java program to display number of characters present in a given String?

input:

```
	java
```

output:

```
	j=1,a=2,v=1
```

ex:

```
import java.util.*;
class Test
{
	public static void main(String[] args)
	{
		String str="java";

		Map<Character,Integer> map=new LinkedHashMap<Character,Integer>();

		char[] carr=str.toCharArray();

		//for each loop
		for(char ch:carr)
```

```
            {
                  if(map.get(ch)!=null)
                  {
                        map.put(ch,map.get(ch)+1);
                  }
                  else
                  {
                        map.put(ch,1);
                  }
            }
            System.out.println(map);
      }
}
```

Q)Types of Data structures in java?

We have two types of data structures in java.

Diagram: java37.1

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

```
input:
      2 4 6 7 9 1 3 3 6 7 2 5 7

output:
      duplicates= 2 6 7 3

      unique= 2 4 6 7 9 1 3 5
```

ex:

```
import java.util.*;
class Test
{
      public static void main(String[] args)
      {
            int[] arr={2,4,6,7,9,1,3,3,6,7,2,5,7};

            Set<Integer> unique=new LinkedHashSet<Integer>();
            Set<Integer> duplicate=new LinkedHashSet<Integer>();

            //for each loop
            for(int i:arr)
            {
                  if(!unique.add(i))
                  {
                        duplicate.add(i);
                  }
                  unique.add(i);
            }
            System.out.println(duplicate);
            System.out.println(unique);
      }
}
```

Multithreading
===============

Q)What is the difference between Thread and Process?

Thread

A thread is a leight weight sub-process.

We can run multiple threads concurently.

One thread can communicate with another thread.

Process

A process is a collection of threads.

We can run multiple process concurently.

One process can't communicate with another process.

Multitasking

Executing several task simultenously such concept is called multitasking.

We have two types of 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.

In multithreading only 10% of work should be done by a programmer and 90% of work will be done by a JAVA API.

The main important application area of multithreading are.

1) To implements multimedia graphics.

2) To develop video games.

3) To develop Animations.

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
{
	//work of a thread
	public void run()
	{
		for(int i=1;i<=5;i++)
		{
			System.out.println("Child-Thread");
		}
	}
}
class Test
{
	public static void main(String[] args)
	{
		//instantiate a thread
		MyThread t=new MyThread();

		//start a thread
		t.start();

		for(int i=1;i<=5;i++)
		{
			System.out.println("Parent-Thread");
		}
	}
}
```

case1: Thread Schedular

If multiple threads are waiting for execution which thread will be executed will decided by thread schedular. What algorithm, mechanism or behaviour 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 t.run()

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
{
	//work of a thread
	public void run()
	{
		for(int i=1;i<=5;i++)
		{
			System.out.println("Child-Thread");
		}
	}
}
class Test
{
	public static void main(String[] args)
	{
		//instantiate a thread
		MyThread t=new MyThread();

		//start a thread
		t.start();
```

```
            for(int i=1;i<=5;i++)
            {
                  System.out.println("Parent-Thread");
            }
      }
}
```

If we invoke t.run() method then no new thread will be created but run() method will execute just like normal method.

ex:

```
class MyThread extends Thread
{
      //work of a thread
      public void run()
      {
            for(int i=1;i<=5;i++)
            {
                  System.out.println("Child-Thread");
            }
      }
}
class Test
{
      public static void main(String[] args)
      {
            //instantiate a thread
            MyThread t=new MyThread();

            //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 Thread class run() method will execute automatically.

Thread class run() method is a 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();

            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 t.start() method always execute run() method with no parameters only.

ex:

```
class MyThread extends Thread
{
        public void run()
        {
                System.out.println("0-arg method");
        }
        public void run(int i)
        {
                System.out.println("int-arg method");
        }
}
class Test
{
        public static void main(String[] args)
        {
                //instantiate a thread
                MyThread t=new MyThread();

                t.start();

                for(int i=1;i<=5;i++)
                {
                        System.out.println("Parent-Thread");
                }
        }
}
```

case 5: Life cycle of a thread

Diagram: java37.2

Once if we create a thread then our thread will be in new or born state.

Once if we call t.start() method then out thread goes to ready or runnable state.

If Thread Schedular allocates to CPU then our thread enters to running state.

Once the run() method execution is completed our thread will goes to dead state.

2) By implementing Runnable interface

```
class MyRunnable implements Runnable
{
        public void run()
        {
                for(int i=1;i<=5;i++)
                {
                        System.out.println("Child-Thread");
                }
        }
}
class Test
{
      public static void main(String[] args)
      {
            MyRunnable r=new MyRunnable();

            Thread t=new Thread(r); // r is a targatable interface

            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 ,explicitly provided by the programmer or automatically generated by JVM.

We have following methods to set and get name of a thread.

ex:

```
      public final void setName(String name)
      public final String getName()
```

ex:
--

```
class MyThread extends Thread
{

}
class Test
{
      public static void main(String[] args)
      {
            System.out.println(Thread.currentThread().getName()); // Main

            MyThread t=new MyThread();
            System.out.println(t.getName());// Thread-0

            Thread.currentThread().setName("Parent-Thread");
            System.out.println(Thread.currentThread().getName());//Parent-Thread

            t.setName("child-thread");
            System.out.println(t.getName());//Child-Thread

      }
}
```

Thread priority
================
In jav 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 take more then 10 priority then we will get IllegalArgumentException.

Thread class defines following standard constants as thread priorities.

```
ex:
      Thread.MAX_PRIORITY   - 10
      Thread.NORM_PRIORITY  - 5
      Thread.MIN_PRIORITY   - 1
```

We don't have such constant 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.

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(9);
            System.out.println(Thread.currentThread().getPriority());//9

            System.out.println(t.getPriority());// 5

            t.setPriority(4);
            System.out.println(t.getPriority());//4

            //t.setPriority(11);//IllegalArgumentException

      }
}
```

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 others having same priority.

If multiple threads having same priority then we can't expect any execution order.

If there is no waiting threads or low priority threads then same thread will continue it's execution.

The thread yielded when it will get a chance of execution is depends upon mercy of thread schedular.

ex:
```
	publi static native void yield();
```

Diagram: java38.1

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 waiting 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(long ms,int ns)throws InterruptedException
```

Diagram: java38.2

ex:
```
class MyThread extends Thread
{
	public void run()
	{
		for(int i=1;i<=5;i++)
		{
			System.out.println("child-thread");
```

```
            }
      }
}
class Test
{
      public static void main(String[] args)throws InterruptedException
      {
            MyThread t=new MyThread();
            t.start();
            t.join();
            for(int i=1;i<=5;i++)
            {
                  System.out.println("parent-thread");
            }
      }
}
```

3)sleep()

If thread don't want to perform any operation on perticular amount of time then we need to use sleep().

A sleep() method will throw one checked exception called InterruptedException so we must and should handle that exception by using try and catch block or by using throws statement.

ex:
```
      public static native void sleep()throws InterruptedException
      public static native void sleep(long ms)throws InterruptedException
      public static native void sleep(long ms,int ns)throws InterruptedException
```

Diagram: java38.3

ex:

```
class MyThread extends Thread
{
      public void run()
      {
            for(int i=1;i<=5;i++)
            {
                  System.out.println("child-thread");
                  try
                  {
                        Thread.sleep(2000);
                  }
                  catch (InterruptedException ie)
                  {
                        ie.printStackTrace();
                  }
            }
      }
}
class Test
{
      public static void main(String[] args)
      {
            MyThread t=new MyThread();
            t.start();
            for(int i=1;i<=5;i++)
            {
                  System.out.println("parent-thread");
            }
      }
}
```

Deamon Thread
==============
It is a service provider thread which provides services to user threads.

Life of daemon thread is depends upon user threads because when user threads died then deamon thread will terminate automatically.

There are many daemon thread are running internaly like Garbage collector, Finalizer and etc.

To start a daemon thread we need to use setDaemon(true) method.

To check thread is a daemon or not we need to use isDaemon() method.

ex:

```
class MyThread extends Thread
{
      public void run()
      {
            for(int i=1;i<=5;i++)
            {
                  System.out.println(Thread.currentThread().isDaemon());
                  System.out.println("child-thread");
            }
      }
}
class Test
{
      public static void main(String[] args)
      {
            MyThread t=new MyThread();
            t.setDaemon(true);
            t.start();
            for(int i=1;i<=5;i++)
            {
                  System.out.println("parent-thread");
            }
      }
}
```

Problem without synchronization
===============================
If there is no synchronization then we will face following problems.

1) Data inconsistency

2) Thread interference

ex:

```
class Table
{
      void printTable(int n)
      {
            for(int i=1;i<=5;i++)
            {
                  System.out.println(n*i);
                  try
                  {
                        Thread.sleep(2000);
                  }
                  catch (InterruptedException ie)
                  {
                        ie.printStackTrace();
                  }
            }
      }
}
class MyThread1 extends Thread
{
      Table t;
      MyThread1(Table t)
      {
            this.t=t;
      }
      public void run()
      {
            t.printTable(5);
      }
}
class MyThread2 extends Thread
{
      Table t;
      MyThread2(Table t)
      {
            this.t=t;
      }
      public void run()
      {
            t.printTable(10);
      }
}
class Test
{
      public static void main(String[] args)
      {
            Table obj=new Table();
            MyThread1 t1=new MyThread1(obj);
            MyThread2 t2=new MyThread2(obj);
            t1.start();
            t2.start();
      }
}
```

```
synchronization
=============
â(¢ Synchronized is the keyword applicable for methods and blocks but not for
classes and
  variables.

â(¢ If a method or block declared as the synchronized then at a time only one
Thread is allow
  to execute that method or block on the given object.

â(¢ The main advantage of synchronized keyword is we can resolve date
inconsistency problems.

â(¢ But the main disadvantage of synchronized keyword is it increases waiting
time of the
  Thread and effects performance of the system.

â(¢ Hence if there is no specific requirement then never recommended to use
synchronized
```

keyword.

â(¢ Internally synchronization concept is implemented by using lock concept.

â(¢ Every object in java has a unique lock. Whenever we are using synchronized keyword then
only lock concept will come into the picture.

â(¢ If a Thread wants to execute any synchronized method on the given object 1st it has to get
the lock of that object. Once a Thread got the lock of that object then itâ(s allow to execute
any synchronized method on that object. If the synchronized method execution completes
then automatically Thread releases lock.

â(¢ While a Thread executing any synchronized method the remaining Threads are not allowed
execute any synchronized method on that object simultaneously. But remaining Threads
are allowed to execute any non-synchronized method simultaneously. [lock concept is
implemented based on object but not based on method].

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:
====================
â(¢ If very few lines of the code required synchronization then itâ(✉s never
recommended to
declare entire method as synchronized we have to enclose those few lines of the
code
with in synchronized block.
â(¢ The main advantage of synchronized block over synchronized method is it
reduces
waiting time of Thread and improves performance of the system.

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();
                  }
            }
            }//block
      }
}
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();
      }
}
```

static synchronization
=======================
Every class in java has a unique lock. If a Thread wants to execute a static synchronized
method then it required class level lock.

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();
      }
}
```

Java 8 Features
===============
We have following features in Java 8.

1) java.time package

2) Functional interface

3) Lamda Expression

4) Default methods in interface

5) Static methods in interface

6) Stream API

7) forEach()

8) Method reference

and etc.

Functional interface

An interface which contains only one abstract method is called functional interface.

We have following list of functional interfaces.

ex:
```
      Runnable   -----> run()
      Comparable -----> comapareTo()
      ActionListener --->     actionPerformed()
      and etc.
```

It can have any number of default methods and static methods.

Functional interface is also known as SAM or Single Abstract Method interface.

Functional interface is used to achieve functional programming.
ex:
```
      a=f1(){}

      f1(f2(){})
      {
      }
```
@FunctionalInterface annotation is used to declare the 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.

Lamda expression is used to concise the code.

We can use Lamda expression when we have functional interface.

The main objective of lamda expression is to achieve functional programming.

Lamda expression consider as method not a class.

Lamda expression does not allow modifier,returntype and name.

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(10,20));
      }
}
```

Default methods in interface
=============================

```
Default methods introduced in java 8.

Default methods are non-abstract methods.

We can override default methods in java.

To declare default methods we need to use "default" keyword.

syntax:
      default  returntype  method_name()
      {
            -
            - //code to be execute
            -
      }

ex:
---
@FunctionalInterface
interface A
{
      //abstract method
      public abstract void m1();

      //default method
      default void m2()
      {
            System.out.println("Default-Method");
      }
}
class B implements A
{
      public void m1()
      {
            System.out.println("Abstract-Method");
      }
}
class Test
{
      public static void main(String[] args)
      {
            A a=new B();
            a.m1();
            a.m2();
      }
}

ex:
---
@FunctionalInterface
interface A
{
      //abstract method
      public abstract void m1();

      //default method
      default void m2()
      {
            System.out.println("Default-Method");
      }
}
class B implements A
{
      public void m1()
```

```
	{
		System.out.println("Abstract-Method");
	}
	public void m2()
	{
		System.out.println("Override-Default-Method");
	}
}
class Test
{
	public static void main(String[] args)
	{
		A a=new B();
		a.m1();
		a.m2();
	}
}
```

In java we can achieve multiple inheritance through default methods in interface.

ex:

```
interface Right
{
	default void m1()
	{
		System.out.println("From Right");
	}
}
interface Left
{
	default void m1()
	{
		System.out.println("From Left");
	}
}
class Middle implements Right,Left
{
	public void m1()
	{
		System.out.println("From Middle");
	}
}
class Test
{
	public static void main(String[] args)
	{
		Middle m=new Middle();
		m.m1();
	}
}
```

ex:

```
interface Right
{
	default void m1()
	{
		System.out.println("From Right");
	}
}
interface Left
{
      default void m1()
      {
            System.out.println("From Left");
      }
}
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("From Right");
      }
}
interface Left
{
      default void m1()
      {
            System.out.println("From Left");
      }
}
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("From Right");
      }
}
interface Left
{
      default void m1()
      {
            System.out.println("From Left");
      }
}
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
============================
A static method introduced in java 8.

A static method is a non-abstract method.

A static method can't be override.

To declare static method we need to use static keyword.

syntax:
```
      static  returntype  method_name()
      {
            -
            - //code to be execute
            -
      }
```

ex:

```
interface A
{
      static void m1()
      {
            System.out.println("From M1 Method");
      }
}
class Test
{
      public static void main(String[] args)
      {
            A.m1();
      }
}
```

Stream API
==========
Stream API introduced in java 8.

A Stream is an interface which is present in java.util.stream package.

Stream API is used to perform bulk operations on Collections.

```
ex:1
-----
import java.util.*;
import java.util.stream.*;
class Test
{
      public static void main(String[] args)
      {
            List<Integer> list=Arrays.asList(2,6,9,1,3,5,7,4);

            List<Integer> evens=list.stream().filter(i->i
%2==0).collect(Collectors.toList());

            System.out.println(evens);
      }
}

ex:
----
import java.util.*;
import java.util.stream.*;
class Test
{
      public static void main(String[] args)
      {
            List<Integer> list=Arrays.asList(2,6,9,1,3,5,7,4);

            List<Integer> odds=list.stream().filter(i->i
%2==1).collect(Collectors.toList());

            System.out.println(odds);
      }
}

ex:
----
import java.util.*;
import java.util.stream.*;
class Test
{
      public static void main(String[] args)
      {
            List<Integer> list=Arrays.asList(2,6,9,1,3,5,7,4);

            List<Integer>
ascending=list.stream().sorted().collect(Collectors.toList());

            System.out.println(ascending);
      }
}

ex:
---
import java.util.*;
import java.util.stream.*;
class Test
{
      public static void main(String[] args)
      {
            List<Integer> list=Arrays.asList(2,6,9,1,3,5,7,4);

            List<Integer>
descending=list.stream().sorted(Comparator.reverseOrder()).collect(Collectors.to
List());

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

ex:

```
import java.util.*;
import java.util.stream.*;
class Test
{
      public static void main(String[] args)
      {
            List<Integer> marks=Arrays.asList(29,61,49,51,73,75,67,44);

            List<Integer> newMarks=marks.stream().map(i-
>i+10).collect(Collectors.toList());

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

ex:

```
import java.util.*;
import java.util.stream.*;
class Test
{
      public static void main(String[] args)
      {
            List<Integer> marks=Arrays.asList(29,61,49,15,73,75,67,44);

            long failed=marks.stream().filter(i-> i<35).count();

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

forEach() method
================
Java provides a new method forEach() to iterate the elements.

ex:
```
import java.util.*;
import java.util.stream.*;
class Test
{
      public static void main(String[] args)
      {
            List<Integer> list=Arrays.asList(6,8,1,3,4,9);

            list.forEach( ele ->  System.out.println(ele));
      }
}
```

Method reference
=================
Method reference is used to refer method of functional interface.

ex:

```
import java.util.*;
import java.util.stream.*;
class Test
{
      public static void main(String[] args)
      {
            List<Integer> list=Arrays.asList(6,8,1,3,4,9);

            list.forEach(System.out::println);
      }
}
```

Q)Write a java program to display employees information based on sorting order of employee Id ?

```
import java.util.*;
import java.util.stream.*;
class Employee
{
      int empId;
      String empName;
      double empSal;

      //parameterized constructor
      public Employee(int empId,String empName,double empSal)
      {
            this.empId=empId;
            this.empName=empName;
            this.empSal=empSal;
      }

      //getter methods
      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(105,"Alan",10000d));
            list.add(new Employee(101,"Jose",20000d));
            list.add(new Employee(103,"Nelson",30000d));
            list.add(new Employee(104,"Kelvin",40000d));

            List<Employee> 
newList=list.stream().sorted(Comparator.comparingInt(Employee::getEmpId)).collec
t(Collectors.toList());

            newList.forEach(employee -> System.out.println(employee.empId+" 
"+employee.empName+" "+employee.empSal));
      }
}
```

Q)Write a java program to display employees information based on sorting order of employee names ?

```
import java.util.*;
import java.util.stream.*;
class Employee
{
	int empId;
	String empName;
	double empSal;

	//parameterized constructor
	public Employee(int empId,String empName,double empSal)
	{
		this.empId=empId;
		this.empName=empName;
		this.empSal=empSal;
	}

	//getter methods
	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(105,"Alan",10000d));
		list.add(new Employee(101,"Jose",20000d));
		list.add(new Employee(103,"Nelson",30000d));
		list.add(new Employee(104,"Kelvin",40000d));

		List<Employee> 
newList=list.stream().sorted(Comparator.comparing(Employee::getEmpName)).collect
(Collectors.toList());

		newList.forEach(employee -> System.out.println(employee.empId+" 
"+employee.empName+" "+employee.empSal));
	}
}
```