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 a 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 operator overloading.	It does not support operator overloading.
It supports preprocessor directory(#).	It des not support preprocessor directory.
It supports access specifiers i.e public,private and protected.	It supports access modifiers i.e default,public,private and protected.
We have three types of loops i.e do while loop, while loop and for loop.	We have four types of loops. i.e do while loop, while loop, for loop and for each loop.

We can save C++ program by using .cpp extension.	We can save java program by using .java extension.

Q)What is the difference between Python and Java?

Python	Java
It is developed by Guido Van Rossum.	It is developed by James Gosling.
It is a product of Microsoft.	It is a product of Oracle Corporation.
It is a scripting language.	It is a object oriented programming language.
It is a interpreted language.	It is a compiled language.
It contains PVM.	It contains JVM.
It is a dynamically typed language.	It is a statically typed language.
Performance is low.	Performance is high.
It gives low security.	It is Highly secured.
It contains less code.	It contains more code.

Project

It is an individual or collaborative enterprise that is carefully planned to achieve a particular aim.

A project is a collection of modules.

Every project contains two domains.

1)Technical Domain

Using which technology we developed our project.

ex: Java

2)Functional Domain

It describes state of a project.

ex: Healthcare domain

Banking domain

Insuarance domain

ERP domain

Modules in Java

We have three modules in java.

Modules in JAVA

|----------|----------|----------|

JSE/J2SE	**JEE/J2EE**	**JME/J2ME**
(Java Standard Edition)	(Java Enterprise Edition)	(Java Micro Edition)
Standalone App	Distributed App	Mobile App
Desktop App	Enterprise 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){
                -
```

- //code to be execute

-

}

}

Desktop App

It is a software application which is specially designed to perform perticular task.

ex:

Control Panel

Recycle Bin

VLC Media player

and etc.

Two-tier App

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

Diagram:java2.1

Diagram:java2.2

Distributed App

In client-server application, if multiple clients are giving the request to main server then main server will distributes these requests to its parallel servers to reduce the burdun of main server such type of application is called distributed application.

Diagram:java2.3

Enterprise App

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

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

ex: Facebook

Online Shopping websites

ERP App

ERP stands for Enterprise Resource Planning.

ERP is used to maintain the data in a enterprise.

Diagram: java2.4

N-tier App

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

Diagram: java2.5

Mobile App

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

ex: PhonePay

Gpay

TempleRun

Currently we are not using JME module to develop mobile applications.

We are using alternate technologies like Android, React Native, Flutter and Swift programming.

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 it won't compiled by the compiler.

We have two types of comments in java.

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
        -
*/
```

Diagram: java3.1

ex:

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

Interview Questions

Q) What is Java?

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

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) Multithreaded

8) Dynamic

9) High level

10) Distributed

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), archiever (.jar),document generator(javadoc) and other tools need for java application development.

JRE

JRE stands for Java Runtime Evironment.

It provides very good environment to run java applications only.

JVM

JVM stands for Java Virtual Machine.

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

Diagram: java3.2

Types of IT companies

Mainly we have two types of IT companies.

1)Service Based company

ex: Cognizent

Capgemini

TCS

2)Product Based company

ex: Oracle Corporation

Microsoft

IBM

Amazon

JPMorgan

Naming conventions in java

In java, uppercase letters will consider as different and lowercase letters will consider as different.Hence we consider java is a case sensitive programming language.

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

ex: classes

interfaces

variables

methods

keywords

packages

constants

classes

In java, a class name must and should starts with uppercase letter and if it contains multiple words then each inner words starts with init cap.

ex:

Predefined classes	**Userdefined classes**
System	Gopal
StringBuffer	GopalaKrishna
FileWriter	DemoApp
Date	Example
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 must and should starts with init cap.

ex:

Predefined interfaces	userdefined interfaces
Runnable	IGopalaKrishna
ListIterator	IDemoApp
Enumeration	IExample
Serializable	IQualityThought

variables

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

ex:

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

methods

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

ex:

predefined methods	userdefined methods
getPriority()	getDetails()
hashCode()	setBill()
getClass()	getEmployeeInfo()
toString()	calculatBillAmt()
and etc.	and etc.

keywords

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

ex:

predefined keywords

if

else

for

switch

public

static

void

packages

In java, all packages must and should declare under lowercase letters only.

ex:

predefined packages	userdefined packages
java.lang	ihub
java.io	com.ihub.www
java.util	com.quality.www
java.sql	com.student.www
java.util.stream	student
java.text	and etc.
and etc.	

constants

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

ex:

predefined constants	userdefined constants
MAX_PRIORITY	LIMIT=10;
MIN_PRIORITY	
HOUR_OF_DAY	
MAX_VALUE	

Assignment

class	:	GaneshMadda
interface	:	IGaneshMadda
variable	:	ganeshMadda
method	:	ganeshMadda()
package	:	com.ganeshmadda.www
constant	:	GANESHMADDA / GANESH_MADDA

Interview questions

Q)How many classes are there in java?

According to

Java 7 ------- 4024 classes

Java 8 ------- 4240 classes

Java 9 ------- 6005 classes

Java 10 ------- 6002 classes

Diagram: java4.1

Q)What is garbage collector ?

It is used to destroy unused or useless objects from java.

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

There are two ways to call garbage collection in java.

1) System.gc();

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

History of Java

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

James Gosling, Mike Sheradin and Patrick Naughton were to develop the project and they met in a placed called Aspan/Colarado to start the work with Graphic System.James Gosling decided to

use C and C++ languages to develop the project.But the problem what they have faced is C and C++ languages are system dependent.Then James Gosling decided why don't we create our own programming language which is system independent.

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

Identifiers

A name in java is called identifier.

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

ex:

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

Here Demo,main,args and x are identifiers.

Rules to declare an identifiers

Rule1:

Identifier will accept following characters.

ex: A-Z

a-z

0-9

_ (underscore)

$

Rule2:

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

ex: int emp_no; //valid

int emp#no; //invalid

double emp$al; //valid

Rule3:

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

ex: int $=10; //valid

int _abcd; // valid

int a1234; //valid

int 1abcd; //invalid

Rule4:

Every identifier is a case sensitive.

ex: int number;

int NUMBER;

int NumBer;

Rule5:

We can't take reserved words as an identifier.

ex: int if; //invalid

int else;//invalid

int for;//invalid

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 funcationality or meaning

Such type of identifiers are called reserved words.

Java support 53 reserved words and it is divided into two types.

Diagram: java5.1

Used keywords with respect to class

package

import

enum

class

interface

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 flow control

if

else

switch

case

for

while

do

break

continue

Used keywords with respect to return type

void

Used keywords with respect to modifiers

default

public

private

protected

abstract

final

static

native

synchronized

strictfp

transient

volatile

Used keywords with respect to exception handling

try

catch

throw

throws

finally

assert

Java

Version : Java 8

JDK : 1.8v

Creator: James Gosling

Open Source : Open source

website : www.oracle.com/java

tutorials : www.javatpoint.com

www.tutorialspoint.com

www.geeksforgeeks.org

www.w3school.com

Download link

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

Steps to setup java environmental variables

step1:

Make sure JDK 1.8 installed successfully.

step2:

Copy a "lib" directory from "JAVA_HOME" folder.

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

step3:

Paste "lib" directory in environmental variables

ex: Right click to my pc --> properties --> advanced system settings -->

environmental variables -->

user variables --> click to new button -->

variable Name : CLASSPATH

variable value: C:\Program Files\Java\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 java application

step1:

Make sure JDK 1.8 install successfully.

step2:

Make sure environmental setup has done perfectly.

step3:

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

step4:

Open the notepad and write simple Hello world program.

ex:

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

step5:

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

step6:

Open the command prompt from javaprog location.

step7:

Compile the above program by using below command.

ex: cmd/javaprog> javac Test.java

|

filename

step8:

Run/Execute the above program by using below command.

ex: cmd/javaprog> java Test

|

classname

Internal Architecture of JVM

Diagram: java6.1

Our java program contains java code instructions.Once if we compiled then java code instructions convert to byte code instructions in class file.

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

1) Method Area

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

2) Heap

Our object creation will store in heap area.

Note:

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

3) Java Stack

Java methods will execute 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 the address of an instructions.

5) Native Method Stack

Java methods will execute in method area.

Similary native methods will execute in native methods stack.

But native methods we can't execute directly.We required a program called Native method interface.

Execution Engine

Execution engine contains interpreter and JIT compiler.

Whenever JVM loads byte code instructions from .class file.It simultenously uses 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.

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 a class from classpath)

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 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.

Datatypes

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

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

In java datatypes are divided into two types.

Diagram: jaav7.1

byte

It is a smallest datatype in java.

Size: 1 byte (8 bits)

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

ex:

1) byte b=10;

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

2) byte b=130;

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

3) byte b=10.5;

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

short

It is a rarely used datatype in java.

Size: 2 bytes(16 bits)

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

ex:

1) byte b=10;

```
short s=b;
    System.out.println(s); // 10
```

2) byte b=10.5;

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

3) byte b="hello";

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

int

It is a mostly used datatype in java.

Size: 4 bytes (32 bits)

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

ex: 1) int i="true";

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

2) int i=true;

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

3) int i='a';

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

Note:

In java, every character contains 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=10.5;

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

3) long l='A';

System.out.println(l); // 65

float

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

Size: 4 bytes (32 bits)

Range: -3.4e38 to 3.4e38

To declare float value we need to suffix with 'f' or 'F'.

ex:

```
10.5f
```

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="hello";
   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;
```

double

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

Size: 8 bytes (64 bits)

Range: -1.7e308 to 1.7e308

To declare double value we need to suffix with 'd' or 'D'.

ex:

```
10.5d
```

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

3) double d='a';

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

4) double d="hello";

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

5) double d=true;

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

boolean

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

Size: (Not Applicable)

Range: (Not Applicable)

ex:

1) boolean b="true";

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

2) boolean b=FALSE;

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

3) boolean b=true;

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

char

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

Size: 2 bytes (16 bits)

Range: 0 to 65535

ex:

1) char ch='a';

```
System.out.println(ch); //a
```

2) char ch=97;

```
System.out.println(ch); //a
```

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

Diagram: java7.2

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

byte = -128 to 127

ex:

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

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

range : -2147483648 to 2147483647

ex:

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

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

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

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

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

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

ex:1

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

ex:2

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

If we won't initialize any value to instance variable then JVM will initialized 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");
	}
}
```

ex:5

```
class	Test{
	//instance variable
	int i=10;
	public static void main(String[] args){
		Test t1=new Test();
		Test t2=new Test();
		t2.i=20;
		System.out.println(t1.i); //10
		System.out.println(t2.i); //20
	}
}
```

2) Static variable

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

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

Static variable will store in method area.

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

Static variable we can access directly from instance area and static area.

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

ex:1

```
class	Test{
	//static variable
```

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

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

ex:2

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

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");
        }
}
```

ex:4

```
class   Test{
        //static variable
        static int i=10;
        public static void main(String[] args){
                Test t1=new Test();
                Test t2=new Test();
                t2.i=20;
                System.out.println(t1.i);//20
                System.out.println(t2.i);//20
        }
}
```

3)Local variable

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

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

Local variable will store in Java stack memory.

ex:1

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

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

Note:

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

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
        }
}
```

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 always at runtime.

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

JVM always looks for main method with following signature.

signature

public static void main(String[] args)

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

public

JVM wants to call this main method from anywhere.

static

JVM wants to call main method without using object reference.

void

Main method does not return any value 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 are not important.Incase of public static we can declare static public also.

ex:

static public void main(String[] args)

2) We can write 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 and final.

command line argument

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

In command line arguments we need to pass out arguments at runtime command.

ex:

```
javac       Test.java


java        Test   101   Alan   M   1000.0
                   |     |  |    |____args[3]
                   |     |  |_________args[2]
                   |     |____________args[1]
                   |__________________________args[0]
```

ex:

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

```
    }
}
```

o/p:

```
    javac   Test.java
    java    Test   101 Alan M 1000.0
```

System.out.println()

It is a output statement in java.

Whenever we want to display any userdefined statement or data then we need to use System.out.println() stmt.

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 in java

1)

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

2)

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

3)

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

4)

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

Fully Qualified Name

In fully qualified name we will declare our class or interface along with package.

Using fully qualified name we can achieve readability of our code.

ex:

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

ex:

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

Import statements

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

Using short name also we can achieve.

In java, we have three 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);

                LocalTime time=LocalTime.now();

                System.out.println(time);

        }

}
```

2) Implicit class import

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

ex:

```
import   java.time.*;

class Test {
```

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

3) Static import

Using static import we can access static members directly.

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

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("stmt3");
	}
```

```
}
```

Basic Java Programs

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

```
import java.util.Scanner;
class Example1 {
	public static void main(String[] args) {
		Scanner sc=new Scanner(System.in);
		System.out.println("Enter the First number :");
		int a=sc.nextInt();
		System.out.println("Enter the Second number :");
		int b=sc.nextInt();
		//logic
		int c=a+b;
		System.out.println("sum of two numbers is ="+c);
	}
}
```

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

```
import java.util.Scanner;
class Example2{
	public static void main(String[] args) {
		Scanner sc=new Scanner(System.in);
		System.out.println("Enter the First number :");
		int a=sc.nextInt();
		System.out.println("Enter the Second number :");
		int b=sc.nextInt();
		System.out.println("sum of two numbers is ="+(a+b));
```

```
	}
}
```

Q)Write a java program to display 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 find out 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);
		//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);
		//logic
		a=a+b;
		b=a-b;
		a=a-b;
		System.out.println("After swapping a ="+a+" and b ="+b);
	}
}
```

Q)Write a java program to find out CGPA to percentage?

ex:

```
import java.util.Scanner;
class Example9{
	public static void main(String[] args) {
		Scanner sc=new Scanner(System.in);
		System.out.println("Enter the CGPA    :");
		float cgpa=sc.nextFloat();
		float percentage=cgpa*9.5f;
		System.out.println("cgpa to percentage is ="+percentage);
	}
}
```

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

```
import java.util.Scanner;
class Example10{
	public static void main(String[] args) {
		Scanner sc=new Scanner(System.in);
		System.out.println("Enter the Salary     :");
		int salary=sc.nextInt();
```

```
        //logic
        float tds=(float)salary*10/100;
        System.out.println("10 percent of TDS is ="+tds);
    }
}
```

Assignments

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

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

3) 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 small value into a bigger variable then we need to use implicit typecasting.

A compiler is responsible to perform implicit typecasting.

There is no possibility to loss the information.

It is also known as Widening or Upcasting.

We can perform implicit typecasting as follow.

ex:

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

ex:1

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

ex:2

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

ex:3

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

2)Explicit typecasting

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

A programmer is responsible to perform explicit typecasting.

There is a possibility to loss the information.

It is also known as Narrowing or Downcasting.

We can perform explicit typecasting as follow.

ex:

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

ex:1

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

ex:

```
class Test  {
	public static void main(String[] args) {
		int m1=98,m2=45,m3=39,m4=67,m5=72,m6=81;
		int total=m1+m2+m3+m4+m5+m6;
		float avg=(float)total/6;
		System.out.println("Total :"+total);
		System.out.println("Average :"+avg);
	}
}
```

Types of blocks in java

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

We have three types of blocks in java.

1)Instance block

2)Static block

3)Local block

1)Instance block

Instance block is used to initialize the instance variables.

Instance block will execute when we create an object.

Instance block can be declared inside the class but not inside the methods and constructors.

We can declare instance block as follow.

syntax:

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

ex:1

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

o/p:

```
    main-method
```

ex:2

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

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

o/p:

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

ex: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
	}
}
```

ex:5

```
class Test {
	//instance variable
	int i=100;
	//instance block
	{
		i=200;
	}
	public static void main(String[] args) {
		Test t=new Test();
		System.out.println(t.i);//200
	}
}
```

2)static block

A static block is used to initialize the static variables.

A static block will execute at the time of classloading.

A static block can be declared inside the class but not inside the methods and constructors.

We can declare static block as follow.

syntax:

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

ex:

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

o/p:

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

ex:

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

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

o/p:

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

ex:

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

ex:

```
class Test {
        //static variable
        static int i=100;
        //static block
        static
        {
                i=200;
        }

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

3)Local block

A local block is used to initialize the local variables.

A local block will execute just like normal statement.

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

We can declare local block as follow.

syntax:

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

ex:

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

ex:

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

Java Source File Structure

case1:

A java program can have multiple classes.

case2:

If a java program contains multiple classes then we need to check which class contains

main method and that class will treated as main class.

ex:

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

case3:

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

ex:

Ihub.java

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

```
    }
}
```

Note: if we compile above program we will get three .class files i.e A.class, B.class and C.class.

case4:

If a java program contains multiple classes with main method then we need to declare atleast one class as public.

ex:

A.java

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

Operators

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

ex:

```
c=a+b
```

Here = and + are operators.

Here a,b and c are operands.

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

We have following list of operators in java.

1) Assignment operators

2) Conditional/Ternary operators

3) Logical operators

4) Bitwise operators

5) Arithmetic operators

6) Relational operators

7) Unary operators

1) Assignment operators

ex:

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

Note:

Reinitialization is possible in java.

ex:

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

```
            final int i=10;
            i=20;
            i=30;
            System.out.println(i);//C.T.E
        }
}
```

Note: C.T.E cannot assign a value to final variable i

ex:

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

Note: C.T.E : Illegal start of expression

ex:

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

ex:

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

```
		i+=2; // i = i + 2
		System.out.println(i); //12
	}
}
ex:
class Test {
	public static void main(String[] args) {
		int i=10;
		i%=20; // i = i % 20
		System.out.println(i); //10
	}
}
ex:
class Test {
	public static void main(String[] args) {
		int i=20;
		i/=10; // i = i / 10
		System.out.println(i); //2
	}
}
ex:
class Test {
	public static void main(String[] args) {
		int i=10;
		i/=20; // i = i / 20
		System.out.println(i); //0
```

```
	}
}
```

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 {
	//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.

2) Conditional operators/ Ternary operators

syntax:

(condition)?val1:value2;

ex:

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

ex:

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

ex:

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

ex:

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

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

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

		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();
		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 (&&)

Logical AND operator will return boolean values either true or false.

Truth table

T T = T

T F = F

F T = F

F F = F

ex:

```
class Test {
        public static void main(String[] args) {
                boolean b = (5>2) && (10<15);
                        System.out.println(b);//true
        }
}
```

ex:

```
class Test {
        public static void main(String[] args) {
                boolean b = (5>20) && (10<15);
                System.out.println(b);//false
        }
}
```

ex:

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

Logical OR operator (||)

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

Truth table

T T = T

T F = T

F T = T

F F = F

ex:

```
class Test {
	public static void main(String[] args) {
		boolean b = (5>2) || (6<10);
		System.out.println(b);//true
	}
}
```

ex:

```
class Test {
	public static void main(String[] args) {
		boolean b = (5>2) || (6<4);
		System.out.println(b);//true
	}
}
```

ex:

```
class Test {
	public static void main(String[] args) {
		boolean b = (5>20) || (6<4);
		System.out.println(b);//false
	}
}
```

ex:

```
class Test {
	public static void main(String[] args) {
		boolean b = (5>20) && (6<10) || (5==5);
		System.out.println(b);//true
	}
}
```

Logical NOT operator(!)

ex:

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

ex:

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

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

4)Bitwise operators

How to convert decimal to binary number

10 - decimal number

1010 - binary number

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

How to convert binary to decimal number

1010 - binary number

10 - decimal number

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

Bitwise AND operator(&)

Bitwise AND operator deals with binary numbers.

Truth table

T T = T

T F = F

F T = F

F F = F

ex:

```
class Test {
	public static void main(String[] args) {
		int a=10,b=15;
		int c= a & b;
		System.out.println(c); //10
	}
}
/*
	10 - 1010
	15 - 1111
	----------
	&  - 1010
			<----
	0*1 + 1*2 + 0*4 + 1*8
	0 + 2 + 0 + 8 = 10
*/
```

ex:

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

```
		int a=10,b=5;
		int c= a & b;
		System.out.println(c); //0
	}
}
/*
	10 - 1010
	5  - 0101
	----------
	&  - 0000

*/
```

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); //15
        }
}
```

```
/*
        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= ~(-9);
		System.out.println(i); // 8
	}
}
```

5)Arithmetic operators

% - modules

/ - division

*** - multiplication**

+ - addition

- - subtraction

ex:

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

	5 + 0 + 0   + 16 – 10
	21-10
	11
*/
```

6)Relational operators

ex:

```
class Test {
	public static void main(String[] args) {
		System.out.println(10 > 20); //false
		System.out.println(10 < 20); //true
		System.out.println(10 >= 5); //true
		System.out.println(10 <= 10);//true
	}
}
```

ex:

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

Right Shift operators (>>)

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
	}
}
```

ex:

```
class Test {
	public static void main(String[] args) {
		int i = 10 >> 7;
		System.out.println(i); // 10 / 128 = 0
	}
}
```

Left Shift operators (<<)

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 << 5;
		System.out.println(i); // 100 * 32 = 3200
	}
}
```

7)Unary operators

Increment/Decrement operators(++/--)

We have two types of increment operators.

i) Post-increment

ex:

```
i++
```

ii) Pre-increment

ex:

```
++i;
```

We have two types of decrement operators.

i) post-decrement

ex:

```
i--
```

ii) pre-decrement

ex:

```
--i
```

Post increment/decrement operators

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:

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

ex:

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

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

ex:

```
class Test {
	public static void main(String[] args) {
		int i = 10;
		int j = i-- + i--; //10 + 9
		System.out.println(i+" "+j);// 8 19
	}
}
```

ex:

```
class Test {
	public static void main(String[] args) {
		int i = 10;
		int j = i++ + i++ + i++; //10 + 11 + 12
		System.out.println(i+" "+j);// 13 33
	}
}
```

Pre increment/decrement operators

Rule1: First Change

Rule2: Then Take

ex:

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

ex:

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

ex:

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

ex:

```
class Test {
	public static void main(String[] args) {
		int i = 10;
		int j= --i + --i; // 9 + 8
```

```
                System.out.println(i+" "+j); // 8 17
        }
}
ex:
class Test {
        public static void main(String[] args) {
                int i = 10;
                int j= i++ + ++i; // 10 + 12
                System.out.println(i+" "+j); // 12   22
        }
}
ex:
class Test {
        public static void main(String[] args) {
                int i=100;
                100++;
                System.out.println(i); //C.T.E
        }
}
ex:
class Test {
        public static void main(String[] args) {
                int i=10;
                System.out.println(++(i++)); //C.T.E
        }
}
```

Control Statements

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

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

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 conditions in our program.

Decision making statement is possible in following ways.

i) if stmt

ii) if else stmt

iii) if else if ladder

iv) nested if stmt

i) if stmt

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

syntax:

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

ex:

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

```
                System.out.println("stmt1");
                if(!(5>10)){
                        System.out.println("stmt2");
                }
                System.out.println("stmt3");
        }
}
```

ex:

```
class Test {
        public static void main(String[] args) {
                System.out.println("stmt1");
                if(!(5>1)){
                        System.out.println("stmt2");
                }
                System.out.println("stmt3");
        }
}
```

ex:

```
class Test {
        public static void main(String[] args) {
                System.out.println("stmt1");
                if(5,4,3,2,1,0){
                        System.out.println("stmt2");
                }
                System.out.println("stmt3");
        }
}
```

o/p:

C.T.E

ex:

```
class Test {
	public static void main(String[] args) {
		if((5>2) && (6<1))
			System.out.println("stmt1");
			System.out.println("stmt2");
			System.out.println("stmt3");
	}
}
```

o/p:

stmt2

stmt3

ex:

```
class Test {
	public static void main(String[] args) {
		if((5>2) && (6<10))
			System.out.println("stmt1");
			System.out.println("stmt2");
			System.out.println("stmt3");
	}
}
```

o/p:

stmt1

stmt2

stmt3

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

```
import java.util.Scanner;
class Test {
	public static void main(String[] args) {
		Scanner sc=new Scanner(System.in);
		System.out.println("Enter the first number :");
		int a=sc.nextInt();
		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>3)
        {
            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(5>30)
                {
                        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 +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");
    }
}
```

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("Both are same");
            System.exit(0);
        }
        if(a>b)
            System.out.println(a+" is greatest ");
        else
            System.out.println(b+" is greatest ");
    }
}
```

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

```
import java.util.Scanner;
class Test {
	public static void main(String[] args) {
		Scanner sc=new Scanner(System.in);
		System.out.println("Enter the number :");
		int n=sc.nextInt();
		if(n%2==0)
			System.out.println("It is even number");
		else
			System.out.println("It is odd number");

	}
}
```

Q)Write a java program to 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!=0 || n%2==1)
			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");
	}
}
```

iii) if else if ladder

It will execute the source code based on multiple conditions.

syntax:

```
	if(cond1)
	{
		- //execute the code
	}
	else   if(cond2)
	{
		- //execute the code
	}
	else   if(cond3)
	{
		- //execute the code
```

```
        }
        else
        {
                - //execute the code 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 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 find out given alphabet is a uppercase letter, lower case 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' || 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 six marks of a student then find out total, average and grade?

i) if average is greater then equals to 70 then A grade

ii) if average is greater then equals to 50 then B grade

iii) if average is greater then equals to 35 then C grade

iv) if average is less then 35 then Failed.

iv) nested if stmt

If 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>20) {
			System.out.println("stmt2");
			if(true)
			{
				System.out.println("stmt3");
			}
			System.out.println("stmt4");
		}
		System.out.println("stmt5");
	}
}
```

o/p:

```
	stmt1
	stmt5
```

ex:

```
class Test {
	public static void main(String[] args) {
		System.out.println("stmt1");
		if(5>2){
			System.out.println("stmt2");
			if(false){
				System.out.println("stmt3");
			}
			System.out.println("stmt4");
```

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

o/p:

```
stmt1
stmt2
stmt4
stmt5
```

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

```
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 a positive number");
                                System.exit(0);
                        }
                        System.out.println("It is a negative number");
                }
        }
}
```

2) Selection statement

switch case

It will the source code based on multiple conditions.

It is similar to if else if ladder.

syntax:

```
switch(condition)
{
        case val1:
                //code to be execute
                break stmt;
        case val2:
                //code to be execute
                break stmt;
        default:
                //code to be execute if all cases are false
}
```

ex:1

```
import java.util.Scanner;
class Test {
        public static void main(String[] args) {
                Scanner sc=new Scanner(System.in);
                System.out.println("Enter the option :");
                int 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.If we won't declare break stmt then from where our condition is satisfied from there all cases will be executed that state is called fall through state of switch case.

ex:

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

        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");
            }
        }
}
```

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

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

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

```
	public static void main(String[] args) {
		Scanner sc=new Scanner(System.in);
		System.out.println("Enter the String :");
		String str=sc.next();
		switch(str)
		{
			case "one": System.out.println("January"); break;
			case "two": System.out.println("February"); break;
			case "three": System.out.println("March"); break;
			case "four": System.out.println("April"); break;
			case "five": System.out.println("May"); break;
			default:   System.out.println("Coming Soon...");
		}
	}
}
```

ex:

```
class Test {
	public static void main(String[] args) {
		float f=10.3f;
		switch(f)
		{
			case 10.1f: System.out.println("stmt1"); break;
			case 10.2f: System.out.println("stmt2"); break;
			case 10.3f: System.out.println("stmt3"); break;
			default:   System.out.println("Not Found");
		}
	}
```

}

o/p:

C.T.E we can't take decimal numbers.

3)Iteration statement

If we want to execute the code repeatedly then we need to use iteration stmt.

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);
```

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

ex:

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

```
                System.out.print(i+" "); //infinite 1
            }
            while (i<=10);
        }
}
ex:
class Test {
        public static void main(String[] args) {
            int i=11;
            do
            {
                System.out.print(i+" ");//11
            }
            while (i<=10);
        }
}
ex:
class Test {
        public static void main(String[] args) {
            int i=1;
            do
            {
                System.out.print(i++ +" "); // infinite 1
                i--;
            }while (i<=10);
        }
}
```

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

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

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

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

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

```
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); // 55
	}
}
```

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

input:

	n=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(); // 5
		int i=n,fact=1;
		do
```

```
		{
			fact=fact*i;
			i--;
		}while (i>=1);
		System.out.println(fact);
	}
}
```

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

```
import java.util.Scanner;
class Test {
	public static void main(String[] args) {
		Scanner sc=new Scanner(System.in);
		System.out.println("Enter the number :");
		int n=sc.nextInt(); // 5
		int i=1;
		do
		{
			System.out.println(n+" * "+i+" = "+n*i);
			i++;
		}while (i<=10);
	}
}
```

ii) while loop

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

syntax:

```
	while(condition)
	{
```

```
        -
        - //code to be execute
        -
    }
ex:
class Test{
    public static void main(String[] args) {
        int i=1;
        while(i<=10)
        {
            System.out.print(i+" "); // infinite 1
        }
    }
}
ex:
class Test{
    public static void main(String[] args) {
        int i=11;
        while(i<=10)
        {
            System.out.print(i+" "); // no output
        }
    }
}
```

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 10 natural numbers?

```
class Test{
	public static void main(String[] args) {
		int i=1,sum=0;
		while(i<=10)
		{
			sum=sum+i;
			i++;
		}
		System.out.println(sum);
	}
}
```

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

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

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

`       int i=n,fact=1;
        while(i>=1)
        {
            fact=fact*i;
            i--;
        }
        System.out.println(fact);
    }
}
```

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

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

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

```
	}
}
```

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

input:

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

output:

It is an armstrong 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(); //153

		int temp=n;

		int rem,sum=0;
		while(n>0)
		{
			rem=n%10;
			sum=sum+rem*rem*rem;
			n=n/10;
		}
		if(sum==temp)
			System.out.println("It is a armstrong number");
		else
			System.out.println("It is not a 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 palindrome 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(); //121

		int temp=n;

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

}

iii) for loop

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

syntax:

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

If number of iterations are known by the user then we need to use for loop.

If number of iterations are not known by the user then we need to use while loop.

If number of iterations are not known by the user but code must execute atleast for one time then we need to use do while loop.

ex:

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

ex:

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

ex:

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

ex:

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

		for(int i=1;i<=20;i++)
		{
			if(i%2==0)
			{
				sum=sum+i;
				i=i+2;
			}
		}
```

```
            System.out.println(sum);// 2 + 6 + 10 + 14 + 18 =50
        }
}
```

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

fibonacci series : 0 1 1 2 3 5 8

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

                int a=0,b=1,c;
                System.out.print(a+" "+b+" ");
                for(int i=2;i<=n;i++)
                {
                        c=a+b;
                        System.out.print(c+" ");
                        a=b;
                        b=c;
                }
        }
}
```

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

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 display prime numbers from 1 to 100?

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:

```
class Test {
	public static void main(String[] args) {
		for(int n=2;n<=100;n++)
		{
			boolean flag=true;
			for(int i=2;i<=n/2;i++)
			{
				if(n%i==0)
				{
					flag=false;
					break;
				}
			}
			if(flag==true)
				System.out.print(n+" ");
		}
	}
}
```

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

input:

	6

output:

It is a perfect number

ex:

```
import java.util.Scanner;
class Test {
	public static void main(String[] args) {
		Scanner sc=new Scanner(System.in);
		System.out.println("Enter the number :");
		int n=sc.nextInt();   // 6

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

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

input:

12 18

output:

6

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);
	}
}
```

Various ways to write methods in java

There are four ways to write methods in java.

1) No returntype with No argument method

2) No returntype with Argument method

3) With returntype with No argument method

4) With returntype with Argument method

1) No returntype with No argument method

If we don't have arguments then we need to ask our inputs inside callie method.

Q)Write a java program to perform 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
	//static 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();
		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 using no return type with no argument method?

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

```
	//callie method
	public static void find(){
		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");
	}
}
```

2) No returntype with Argument method

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

Number of arguments depends upon number of inputs.

Q)Write a java program to perform sum of two numbers with 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 convert decimal to octa with no return type 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 Decimal number :");
		int decimal=sc.nextInt();

		//caller method
		find(decimal);
	}
	//callie method
	public static void find(int decimal){
		String str=Integer.toOctalString(decimal);
		System.out.println(str);
	}
}
```

Assignment

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

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 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();

		int c=a+b;
		return c;
	}
}
```

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

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

		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 find out 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 call itself for many number of times is called recursion.

Recursion is similar to loopings.

If we use recursion , we should not use loops.

Q)Write a java program 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+" "); //1 2 3 4 5 6 7 8 9 10
            display(i+1);
        }
    }
}
```

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();

		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){

		if(a==0)

		{

			return b;

		}

		return sum(--a,++b);

	}

}
```

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

```
import java.util.Scanner;

class Test {

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

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

		//caller method
		System.out.println("Factorial of a given number is ="+fact(n));
	}
	//callie method
	public static int   fact(int n) {
		if(n<0)
			return -1;
		if(n==0)
			return 1;
		return n*fact(n-1);
	}
}
```

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
		System.out.println(fib(n));
	}
	//callie method
	public static int fib(int n) {
		if(n == 0 || n==1)
			return 0;
		else if(n==2)
			return 1;
		else
			return fib(n-1)+fib(n-2);
	}
}
```

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

```
import java.util.Scanner;
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 number");
```

```
            else
                System.out.println("It is not palindrome number");
        }
        //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);
        }
}
```

Assignment

Q)Write a java program to find out decimal to binary ?

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

```
import java.util.*;
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");
    }
}
```

LOOP Patterns

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

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

2)

1 2 3 4

1 2 3 4

1 2 3 4

1 2 3 4

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

3)

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

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

4)

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

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();
		}
	}
}
```

5)

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

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

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

6)

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

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

7)

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

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

8)

A A A A

B B B B

C C C C

D D D D

ex:

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

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();
		}
	}
}
```

10)

1 1 1

1 0 1

1 1 1

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

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

Left Side Loop patterns

1)

1

2 2

3 3 3

4 4 4 4

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

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

2)

4 4 4 4

3 3 3

2 2

1

ex:

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

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

3)

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

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

```
                }
                //descending
                //rows
                for(int i=3;i>=1;i--)
                {
                        //cols
                        for(int j=1;j<=i;j++)
                        {
                                System.out.print("* ");
                        }
                        //new line
                        System.out.println();
                }
        }
}
```

4)

1

2 3

4 5 6

7 8 9 0

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

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

5)

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++)
        {
            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();
            }
        }
    }
}
```

6)

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++)
```

```
        {
            if(i%2!=0)
            {
                for(int j=1;j<=i;j++)
                {
                    if(j>1)
                        System.out.print("#"+j);
                    else
                        System.out.print(j);
                }
                //new line
                System.out.println();
            }
            else
            {
                for(int j=i;j>=1;j--)
                {
                    if(j>1)
                        System.out.print(j+"#");
                    else
                        System.out.print(j);
                }
                //new line
                System.out.println();
            }
        }
    }
```

}

Right side loop patterns

1)

1

2 2

3 3 3

4 4 4 4

ex

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

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

```
        }
}
```

2)

4 4 4 4

3 3 3

2 2

1

ex:

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

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

```
        }
}
```

3)

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

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

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

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

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

Assignment

Q)Write a java program to display even number loop patter?

2

4 6

8 10 12

14 16 18 20

Q)Write a java program to display prime number loop pattern?

2

3 5

7 11 13

17 19 23 29

Pyramid loop patterns

1)

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

ex:

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

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

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

2)

1 2 3 4 3 2 1

** 1 2 3 2 1**

**   1 2 1**

**   1**

ex:

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

```
                for(int i=4;i>=1;i--)
                {
                        //space
                        for(int j=4;j>i;j--)
                        {
                                System.out.print("    ");
                        }
                        //left side elements
                        for(int j=1;j<=i;j++)
                        {
                                System.out.print(j+" ");
                        }

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

3)

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

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

ex:

```
class Test {
        public static void main(String[] args) {
                //ascending order
                //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 order
            //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();
            }
        }
}
```

Interview Questions

Q)Write a java program display below loop pattern?

1 1

1 2 2 1

1 2 3 3 2 1

1 2 3 4 4 3 2 1

ex:

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

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

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

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

ex:

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

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

Q)Write a java program to display prime number loop pattern?

2

3 5

7 11 13

17 19 23 29

ex:

```
class Test {
    public static void main(String[] args) {
        int num=2;
        //rows
        for(int i=1;i<=4;i++)
        {
            //cols
            for(int j=1;j<=i;j++)
            {
                while(true)
                {
                    boolean flag=true;
```

```
                    for(int k=2;k<=num/2;k++)
                    {
                        if(num%k==0)
                        {
                            flag=false;
                            break;
                        }
                    }
                    if(flag)
                    {
                        System.out.print(num+" ");
                        num++;
                        break;
                    }
                    num++;
                }
            }
            //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 stmt

ii) continue stmt

i) break stmt

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

For conditional statement 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(true)
		{
			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 stmt

A continue statement is used to continue the execution of loops.

For conditional statement we can use if condition.

syntax:

```
	continue;
```

ex:1

```
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:2

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

ex:3

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

Arrays

Arrays is a collection of homogeneous data elements.

The main advatages of arrays are

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

ex:

```
int[] arr={10,20,30,40};
```

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

The main disadvantages of arrays are

1) It is 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 classified 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.

Arrays

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

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

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

Array Creation

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

ex:

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

Diagram: java21.1

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

Rule to construct an array

Rule1:

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

ex:

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

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 numbers as an array size otherwise we will get runtime exception called NegativeArraySizeException.

ex:

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

int[] arr=new int[-3]; //invalid
```

Rule4:

The allowed datatype for an array size is byte,short,int and char.

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

ex:

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

int[] arr=new int['a'];

int[] arr=new int[5.5f]; //invalid
```

Rule5:

The maximum length we can take for an arrays size is maximum length of int datatype.

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.

ex:

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

Diagram: java21.2

Array Declaration, Creation and Initialization using single line

```
int[] arr;
arr=new int[3];
arr[0]=10;
arr[1]=20;
arr[2]=30;      ==>   int[] arr={10,20,30};
                ==>   char[] carr={'a','b','c'};
                ==>   String[] sarr={"hi","hello","bye"};
```

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

length

It is a final variable which is applicable only for arrays.

It will return size of an array.

ex:

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

length()

It is a predefined method which is applicable for String objects.

It will return number of characters present in String.

ex:

```
class Test {
	public static void main(String[] args) {
		String str="bhaskar";
		System.out.println(str.length());//7
	}
}
```

Single Dimensional Array Programs

Q) Write a java program to accept array elements and dispaly 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();
		int[] arr=new int[size];

		//insert the 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 7 1 3 9 6

ex:

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

output:

6 9 3 1 7 4

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

```
		int[] arr={4,7,1,3,9,6};
		//reading reverse
		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 7 1 3 9 6

output:

30

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

output:

1 3 4 6 7 9

approach1

```
import java.util.Arrays;
class Test {
	public static void main(String[] args) {
		int[] arr={4,7,1,3,9,6};
		Arrays.sort(arr); // 1 3 4 6 7 9
		for(int i:arr)
		{
			System.out.print(i+" ");
		}
	}
}
```

approach2

```
class Test {
	public static void main(String[] args) {
		int[] arr={4,7,1,3,9,6};
		//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;
                    }
                }
            }
            //reading the elements
            for(int i:arr)
            {
                System.out.print(i+" ");
            }
        }
}
```

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

input:

4 7 1 3 9 6

output:

9 7 6 4 3 1

```
import java.util.Arrays;
class Test {
    public static void main(String[] args) {
        int[] arr={4,7,1,3,9,6};
        Arrays.sort(arr); // 1 3 4 6 7 9
        //reading reverse
        for(int i=arr.length-1;i>=0;i--)
        {
```

```
                System.out.print(arr[i]+" ");
            }

        }
}
```

approach2

```
class Test {
    public static void main(String[] args) {
        int[] arr={4,7,1,3,9,6};
        //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;
                }
            }
        }
        //reading the elements
        for(int i:arr)
        {
            System.out.print(i+" ");
        }
```

```
	}
}
```

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

input:

	4 7 1 3 9 6

output:

	9

ex:

```
import java.util.Arrays;
class Test {
	public static void main(String[] args) {
		int[] arr={4,7,1,3,9,6};
		Arrays.sort(arr); // 1 3 4 6 7 9
		System.out.println(arr[arr.length-1]);//9
	}
}
```

approach2

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

output:

1

ex:

```
import java.util.Arrays;
class Test   {
    public static void main(String[] args) {
        int[] arr={4,7,1,3,9,6};
        Arrays.sort(arr);//1 3 4 6 7 9
        System.out.println(arr[0]);//1
    }
}
```

approach2

```
class Test {
    public static void main(String[] args) {
        int[] arr={4,7,1,3,9,6};
        int small=arr[0];
        //for each loop
        for(int i:arr)
        {
```

```
                    if(i<small)
                    {
                        small=i;
                    }
                }
                System.out.println(small);
        }
}
```

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

input:

4 6 1 2 2 9 6 5 3 9

output:

6 2 9

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

```
    }
}
```

Assignment

Q)Write a java program to find out second highest element from given array without using Arrays class?

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

input:

4 6 1 2 3 9 6 5 2 3

output:

4 1 9 5

ex:

```
class Test {
    public static void main(String[] args) {
        int[] arr={4,6,1,2,3,9,6,5,2,3};
        //unique elements
        for(int i=0;i<arr.length;i++)
        {
            int cnt=0;
            for(int j=0;j<arr.length;j++)
            {
                if(arr[i]==arr[j])
                {
                    cnt++;
                }
            }
            if(cnt==1)
```

```
                {
                        System.out.print(arr[i]+" ");
                }
        }
    }
}
```

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

input:

4 6 1 2 3 9 6 5 2 3 2 8 2

output:

2 is repeating for 4 times

ex:

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

```
                    if(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:

9 7 13 2 5 6 8

output:

7 13 2 5

ex:

```
class Test {
        public static void main(String[] args) {
                int[] arr={9,7,13,2,5,6,8};
                //for each loop
                for(int n:arr)
                {
                    boolean flag=true;
                    for(int i=2;i<=n/2;i++)
                    {
                        if(n%i==0)
                        {
                            flag=false;
```

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

Q)Write a java program to seggregate 0's and 1's?

input:

1 0 1 1 0 0 0 1 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={1,0,1,1,0,0,0,1,1,0};
                Arrays.sort(arr);
                //display
                for(int i:arr)
                {
                        System.out.print(i+" ");
                }
        }
```

```
}
```

approach2

```
class Test {
	public static void main(String[] args) {
		int[] arr={1,0,1,1,0,0,0,1,1,0};    //    1    0    1    1    0    0    0    1    1    0
		int[] resArr=new int[arr.length];
		//inserting 0
		int j=0;
		for(int i=0;i<arr.length;i++)
		{
			if(arr[i]==0)
			{
				resArr[j++]=arr[i];
			}
		}
		//inserting 1
		while(j<arr.length)
		{
			resArr[j++]=1;
		}
		//display
		for(int i:resArr)
		{
			System.out.print(i+" ");
		}
	}
}
```

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

input:

6 64 14 9 3 1 5

output:

5 9 14 64

ex:

```
class Test {
	public static void main(String[] args) {
		int[] arr={6,64,14,9,3,1,5};
		int max=arr[arr.length-1];
		System.out.print(max+" ");
		//moving 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 find out missing element from given array?

input:

6 1 3 5 7 2

output:

4

```
class Test {
	public static void main(String[] args) {
		int[] arr={6,1,3,5,7,2};
		int sum_of_arr_ele=arr.length+1;
		int sum=(sum_of_arr_ele*(sum_of_arr_ele+1))/2;
		//for each loop
		for(int i:arr)
		{
			sum=sum-i;
		}
		System.out.println("Missing element is ="+sum);
	}
}
```

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

input:

arr1 = 5 9 2 1 3

arr2 = 6 7 9 8 2

output:

11 16 11 9 5

ex:

```
class Test {
	public static void main(String[] args) {
		int[] arr1 = {5,9,2,1,3};
		int[] arr2 = {6,7,9,8,2};
		int[] resArr=new int[arr1.length];
		for(int i=0;i<arr1.length || i<arr2.length;i++)
		{
```

```
                resArr[i]=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:

arr1 = 5 1 3 2 4

arr2 = 9 7 8 6 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 ={9,7,8,6,10};
        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
                for(int i:arr1)
                {
                        System.out.print(i+" ");
                }
        }
}
```

Q)Write a java program to display triplet of array elements equals to given element?

input:

arr = 6 8 1 3 5 4 2

sum = 9

output:

6 1 2

1 3 5

3 4 2

ex:

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

```
            int sum=9;
            for(int i=0;i<arr.length;i++)
            {
                        for(int j=i+1;j<arr.length;j++)
                        {
                                    for(int k=j+1;k<arr.length;k++)
                                    {
                                                if(arr[i]+arr[j]+arr[k]== sum)
                                                {
                                    System.out.println(arr[i]+" "+arr[j]+" "+arr[k]);
                                                }
                                    }
                        }
                }
    }
}
```

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

input:

arr = 6 3 2 1 9 2 7 4 2

element = 2

output:

6 3 1 9 2 7 4 2

ex:

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

```
            int[] resArr=new int[arr.length-1];
            int cnt=0,j=0;
            for(int i=0;i<arr.length;i++)
            {
                if(arr[i]==element && cnt==0)
                {
                    cnt++;
                    continue;
                }
                resArr[j++]=arr[i];
            }
            //display
            for(int i:resArr)
            {
                System.out.print(i+" ");
            }
        }
}
```

Q)Write a java program to insert given element in a given position/index ?

input:

arr = 5 8 2 9 7 3

position = 3

element = 100

output:

5 8 2 100 9 7 3

```
import java.util.Arrays;
class Test {
	public static void main(String[] args) {
		int[] arr={5,8,2,9,7,3};
		int position = 3;
		int element = 100;
		arr=Arrays.copyOf(arr,arr.length+1);
		for(int i=arr.length-1;i>=position;i--)
		{
			arr[i]=arr[i-1];
		}
		arr[position]=element;
		//display elmements
		for(int i:arr)
		{
			System.out.print(i+" ");
		}
	}
}
```

Q)Write a java program to find out second highest element from given array without using Arrays class?

input:

2 7 9 4 6 3 8 1

output:

8

```
public class Test {
    public static void main(String[] args) {
        int[] array = {2, 7, 9, 4, 6, 3, 8, 1};
        int highest = Integer.MIN_VALUE; // -2147483648
        int secondHighest = Integer.MIN_VALUE; //-2147483648

        // Find the highest and second-highest elements
        for (int i = 0; i < array.length; i++)
        {
            if (array[i] > highest)
            {
                secondHighest = highest;
                highest = array[i];
            }
            else if (array[i] > secondHighest && array[i] < highest)
            {
                secondHighest = array[i];
            }
        }
        System.out.println(secondHighest);
    }
}
```

Double Dimensional Array

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

Two dimensional is a combination of rows and columns.

The main objective of two dimensional array is a just for instance use.

We can use two dimensional array 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:

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

Here we can store 9 elements.

Q)Write a java program to display array elements in matrix form?

```
import java.util.Scanner;
public class Test {
    public static void main(String[] args) {
        Scanner sc=new Scanner(System.in);
            System.out.println("Enter the rows :");
            int rows=sc.nextInt();
            System.out.println("Enter the columns :");
            int cols=sc.nextInt();
            int[][] arr=new int[rows][cols];
            //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 left side diagonal elements?

```
import java.util.Scanner;
public 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 sum=0;
                for(int i=0;i<rows;i++)
                {
```

```
                for(int j=0;j<cols;j++)
                {
                        if(i==j)
                        {
                                sum+=arr[i][j];
                        }
                }
        }
        System.out.println("sum of diagonal elements is ="+sum);
    }
}
```

Q)Write a java program to perform sum of right side diagonal elements?

```
import java.util.Scanner;
public 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 sum=0;
        for(int i=0;i<rows;i++)
        {
                for(int j=0;j<cols;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 ?

```
import java.util.Scanner;
public 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 sum=0;
        for(int i=0;i<rows;i++)
        {
                for(int j=0;j<cols;j++)
                {
```

```
                    if(i<j)
                    {
                        sum+=arr[i][j];
                    }
                }
            }
            System.out.println("sum of upper triangle elements is ="+sum);
    }
}
```

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

```
import java.util.Scanner;
public 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 sum=0;
            for(int i=0;i<rows;i++)
            {
                for(int j=0;j<cols;j++)
                {
                    if(i>j)
```

```
                    {
                        sum+=arr[i][j];
                    }
                }
            }
            System.out.println("sum of lower triangle elements is ="+sum);
    }
}
```

Assignment

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

```
public class Test {
    public static void main(String[] args) {
        int[][] matrix = {
                    {1, 2, 3},
                    {4, 5, 6},
                    {7, 8, 9}
                };
        int rows = matrix.length;
        int cols = matrix[0].length;

        int 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
    }
}
```

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 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}};
```

```
class Test {
	public static void main(String[] args){
		//caller method
		sum(new int[]{10,20,30});
	}
	//callie method
	public static void sum(int[] arr){
		//for each loop
		int sum=0;
		for(int i:arr)
		{
			sum+=i;
		}
		System.out.println(sum);
	}
}
class Test {
	public static void main(String[] args)
	{
		//caller method
		System.out.println(sum(new int[]{10,20,30}));
	}
	//callie method
	public static int sum(int[] arr)
	{
		//for each loop
		int sum=0;
```

```
        for(int i:arr)
        {
            sum+=i;
        }
        return sum;
    }
}
```

OOPS

OOPS stands for Object Oriented Programming System/Structure.

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 supports following features.

ex:

class

object

Abstraction

Encapsulation

inheritance

and

polymorphism.

class

A class is a collection of data members and behaviours.

Here data members means variables or properties or fields.

Here behaviours means methods or actions or characteristics.

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

A class is a blue print of an object.

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

A default class we can access within the package.

ex:

class A {

-

- //some logic

-

}

public class

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

ex:

```
public class A
{
    -
    - //some logic
    -
}
```

Q)What is final class?

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

If we declare any class as final then extending anyother class is not possible.

ex:

```
final class A
{

}
class B extends A     ---> invalid
{

}
```

Q)What is abstract class?

If we declare any class abstract then creating object of that class is not possible.

ex:

```
abstract class A
{
}
```

A a=new A(); --> invalid

object

It is a outcome of a blue print.

It is a instance of a class.

Here instance means allocating memory of our data members.

Memory will be created when we create an object.

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

We can create object as follow.

ex:

```
                operator
                   |
        Test    t  =  new     Test();
        |       |              |
classname  reference_var    constructor
```

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.

For every object , jvm will create a unique identification number i.e hash code.

In order to read the hash code of an object we need to use hashCode() method.

Diagram: java24.1

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.

Object class

Object class present in java.lang package.

Object class will consider as parent class for every java class.

Object class contains following methods.

ex:

```
cmd> javap     java.lang.Object


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

Data Hiding

Our internal 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 the internal implementation and highlighting the set of services is called abstraction.

Using abstract classes and interfaces we can implements Abstraction.

The best example of abstract is GUI(Graphical User Interface) ATM machine where bank people will hide internal implementation and hihlights the set of services like banking, withdrawl, ministatement, 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 the 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.

Diagram: java25.1

A class is said to be encapsulated class if it supports data hiding + abstraction.

In encapsulation ,for every variable we need to write setter and getter method.

Diagram: java25.2

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.

Note:

Abstraction is used to hide the data and Encapsulation is used to protect the data.

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("Jose");
        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("Jose");
		s.setStudFee(10000d);

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

Q)What is tighty encapsulated class?

A class is said to be tightly encapsulated class if all the variables of that class must be private. Here we don't need to check these variables having setter and getter methods or not.

ex:

```
	class A
	{
		private int i=10;
```

```
}
//tightly encapsulated class
```

ex:

```
class A
{
    int i=10;
}
class B extends A
{
    private int j=20;
}
```

Note:

If no parent is tightly encapsulated then no child is tightly encapsulated.

Is-A relationship

Is-a relationship is also known as inheritance.

By using extends keyword 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();
		p.m1();

		//Child c1=new Parent(); //invalid
	}
}
```

Conclusion

Whatever our parent contains bydefault it goes to child but whatever our child contains it nevers goes back to parent.

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

Inheritance

Inheritance is a mechanism where our class will inherit the properties of another class.

or

Inheritance is a mechanism where we can derived a class in the presence of existing class.

The main objective of inheritance is reusability.

Diagram: java26.1

We have five types of inheritance.

1) Single level inheritance

2) Multilevel inheritance

3) Multiple inheritance

4) Hierarchical inheritance

5) Hybrid inheritance

1) Single level inheritance

If we derived a class in the presence of one base class is called single level inheritance.

ex:

A (Parent / Super / Base class)

|

|

|

B (Child / Sub / Derived class)

ex:

```
class A {
	public void m1()
	{
		System.out.println("A-m1 method");
	}
}
class B extends A
{
	public void m2()
```

```
    {
        System.out.println("B-m2 method");
    }
}
class Test  {
    public static void main(String[] args)
    {
        A a=new A();
        a.m1();

        B b=new B();
        b.m1();
        b.m2();
    }
}
```

2)Multilevel inheritance

If a class is derived from one base class and that class is derived from another base class is called multilevel inheritance.

Ex:

```
    A
    |
    B
    |
    C
```

ex:

```
class A
{
```

```
	public void m1()
	{
		System.out.println("A-m1 method");
	}
}
class B extends A
{
	public void m2()
	{
		System.out.println("B-m2 method");
	}
}
class C extends B
{
	public void m3()
	{
		System.out.println("B-m3 method");
	}
}
class Test
{
	public static void main(String[] args)
	{
		A a=new A();
		a.m1();

		B b=new B();
```

```
        b.m1();
        b.m2();

        C c=new C();
        c.m1();
        c.m2();
        c.m3();
    }
}
```

3)Multiple inheritance

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

ex:

```
    class A
    {
    }
    class B
    {
    }
    class C extends A,B   --> invalid
    {
    }
```

Interface can extends more then one interface so we can achieve multiple inheritance concept through interfaces.

ex:

```
    interface A
    {
    }
```

```
interface B
{
}
interface C extends A,B
{
}
```

If our class does not extends any other class then it is a direct child class of Object class.

ex:

```
class A
{
}
```

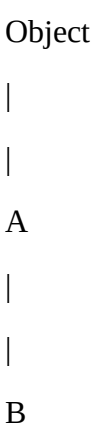
Diag:

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

ex:

```
class A
{
}
class B extends A
{
}
```

Diag:



Java does not support cyclic inheritance.

ex:

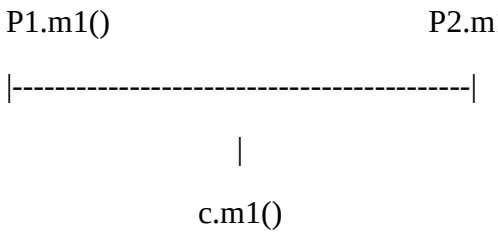
```
class A extends B
{
}
class B extends A
```

```
{
}
```

Q)Why java does not support multiple inheritance?

There is a chance of raising ambiguity problem that's why java does not support multiple inheritance.

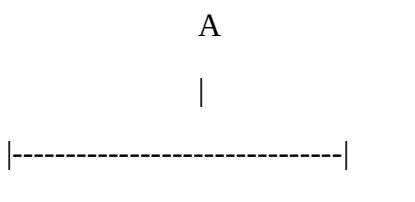
ex:



4)Hierarchical inheritance

If we derived multiple classes in the presence of one base class is called hierarchical inheritance.

ex:



ex:

```
class A {
	public void m1()
	{
		System.out.println("A-m1 method");
	}
}
class B extends A
{
	public void m2()
```

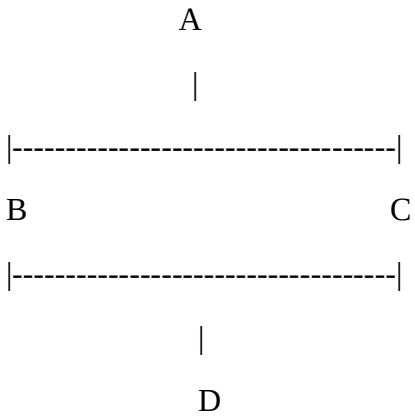
```
	{
		System.out.println("B-m2 method");
	}
}
class C extends A
{
	public void m3()
	{
		System.out.println("B-m3 method");
	}
}
class Test
{
	public static void main(String[] args) {
		A a=new A();
		a.m1();

		B b=new B();
		b.m1();
		b.m2();

		C c=new C();
		c.m1();
		c.m3();
	}
}
```

5)Hybrid inheritance

It is a combination of more then one inheritance.

Java does not support hybrid inheritance.

ex:



Has-A relationship

Has-A relationship is also known as composition and aggregation.

There is no specific keyword to 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 With 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 no chance of having contained object then the relationship between container and contained object is called composition which is strongly association.

Diagram: java26.2

aggregation

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

Diagram: java26.3

Q)What is the difference between POJO class and Java Bean class?

POJO

A class said to be pojo class if it supports following two properties.

1) All variables must be private.

2) All variables must have setter and getter methods.

Java Bean

A class 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 with difference parameters in a single class is called method overloading.

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

Method overloading will reduce complexity the programming.

ex:

```
class MeeSeva{
	//overloaded methods
	public void search(int voterId)
	{
		System.out.println("Details Found - VoterId");
	}
	public void search(String houseNo)
	{
		System.out.println("Details Found - houseNo");
	}
	public void search(long aadharCard)
	{
		System.out.println("Details Found - aadharCard");
	}
}
class Test
{
```

```
    public static void main(String[] args)
    {
        MeeSeva ms=new MeeSeva();
        ms.search(1001);
        ms.search("1-4-676");
        ms.search(18954L);
    }
}
```

Method overriding

Having same method name with same parameters in two different classes is called method overriding.

Methods present in parent class are called overridden methods.

Methods 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("Rashmika");
```

```
	}
}
class Test {
	public static void main(String[] args){
			Parent p=new Parent();
			p.property(); //cash+gold+land
			p.marry(); // Subhalakshmi

			Child c=new Child();
			c.property(); // cash+gold+land
			c.marry(); // Rashmika

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

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

ex:

```
class Parent
{
	public void property()
	{
		System.out.println("Cash+Gold+Land");
	}
	//overridden method
	public final void marry()
```

```
	{
		System.out.println("subhalakshmi");
	}
}
class Child extends Parent
{
	//overriding method
	public void marry()
	{
		System.out.println("Rashmika");
	}
}
class Test
{
	public static void main(String[] args)
	{
			Parent p=new Parent();
			p.property(); //cash+gold+land
			p.marry(); // Subhalakshmi

			Child c=new Child();
			c.property(); // cash+gold+land
			c.marry(); // Rashmika

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

```
    }
}
```

Method Hiding

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

Method overriding	**Method hiding**
All the methods present in method overriding must be non-static.	All the methods present in method hiding must be static.
Method resolution will taken care by JVM based on runtime object.	Method resolution will taken care by compiler based on reference type.
It is also known as runtime polymorphism, dynamic polymorphism or late binding.	It is also known as compile time polymorphism, static polymorphism, early binding.

ex:

```
class Parent
{
    public static void property()
    {
        System.out.println("Cash+Gold+Land");
    }
    public static void marry()
    {
        System.out.println("subhalakshmi");
    }
}
```

```
class Child extends Parent
{
	public static void marry()
	{
		System.out.println("Rashmika");
	}
}
class Test
{
	public static void main(String[] args)
	{
			Parent p=new Parent();
			p.property(); //cash+gold+land
			p.marry(); // Subhalakshmi

			Child c=new Child();
			c.property(); // cash+gold+land
			c.marry(); // Rashmika

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

Q) Can we overload main method in java?

Yes, we can 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[] iargs");
	}
	public static void main(String[] args)
	{
			System.out.println("string[] args");
	}
}
```

Q)Can we override main method in java?

No, we can't override main method in java because it is static.

In java static methods we can't override.

Polymorphism

Polymorphism has taken from Greek word.

Here poly means many and morphism means forms.

The ability to represent in different forms is called polymorphism.

Diagram: java27.1

The main objective of polymorphism is to provide flexibility.

In java, polymorhism is divided into two types.

1) Compile time polymorphism / Static polymorphism / Early binding

2) Runtime polymorhism / 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 run time is called runtime polymorphism.

ex:

Method overriding

Diagram: java27.2

Constructors

Constructor is a special method which is used to intialized an object.

ex:

Test t=new Test();

Having same name as class name is called constructor.

Constructor does not allow any returntype.

Constructor will execute when we create an object.

The allowed modifiers for constructor are.

ex:

default

public

private

protected

In java constructors are divided into two types.

1) Userdefined constructor

2) Default constructor

1) Userdefined constructor

A constructor which is created by the user based on the application requirements is called

userdefined constructor.

It is classified into two types.

i) Zero Argument constructor

ii) Parameterized constructor

i) Zero Argument constructor

Suppose if we won't pass any argument to userdefined constructor then that constructor is called zero argument constructor.

ex:

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

o/p:

main-method

ex:

```
class Test
{
	Test()
	{
		System.out.println("constructor");
```

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

o/p:

```
	main-method
	constructor
```

ex:

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

o/p:

```
	constructor
	main-method
```

constructor

ii) Parameterized constructor

Suppose if we are passing atleast one argument to userdefined constructor then such constructor is called parameterized constructor.

ex:

```
class Employee
{
	private int empId;
	private String empName;
	private double empSal;

	//parameterized constructor
	public Employee(int empId,String empName,double empSal)
	{
		this.empId=empId;
		this.empName=empName;
		this.empSal=empSal;
	}
	public void getEmployeeDetails()
	{
		System.out.println("Employee Id :"+empId);
		System.out.println("Employee Name :"+empName);
		System.out.println("Employee Salary :"+empSal);
	}
}
class Test
{
```

```
    public static void main(String[] args)
    {
        Employee e1=new Employee(101,"Alan Morries",10000d);
        e1.getEmployeeDetails();

        Employee e2=new Employee(102,"Erick Anderson",20000d);
        e2.getEmployeeDetails();
    }
}
```

2) Default constructor

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

We can see default constructor by using below command.

ex:

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

Diagram: java27.3

Constructor overloading

Having same constructor name with different parameters in a single class is called constructor overloading.

ex:

```
class A
{
    A()
    {
        System.out.println("0-arg const");
    }

    A(int i)
```

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

this keyword

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

We can utilize this keyword in following ways.

i) To refer current class variables

ii) To refer current class methods

iii) To refer current class constructors

i) To refer current class variables

```
class A
{
	int i=10;
	int j=20;

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

ii) To refer current class methods

```
class A
{
	public void m1()
	{
		System.out.println("From M1 Method");
		this.m2();
	}
```

```
	public void m2()
	{
		System.out.println("From M2 Method");
	}
}
class Test
{
	public static void main(String[] args)
	{
		A a=new A();
		a.m1();
	}
}
```

iii)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.

i) To refer super class variables.

ii) To refer super class methods.

iii) To refer super class constructors.

i) To refer super class variables

```
class A
{
    int i=10;
    int j=20;
}
class B extends A
{
    int i=100;
    int j=200;
```

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

ii)To refer super class methods

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

```
            System.out.println("M2-Method");
        }
}
class Test
{
        public static void main(String[] args)
        {
            B b=new B();
            b.m2();
        }
}
```

iii)To refer super class constructor

```
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();
    }
}
```

Interface

Interface is a collection of zero or more abstract methods.

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

It is not possible to create object for interfaces without having body.

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

It is possible to create object for interface 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
    -
}
```

Note:

If we know Service Requirement Specification then we need to use interface.

Diagram: java28.1

ex:

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

ex:2

```
interface A
{
	//abstract method
	public abstract void m1();
}
```

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

```
interface A
{
	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 see()
        {
            System.out.println("see-method");
        }
        public void view()
        {
            System.out.println("view-method");
        }
}
class Test
{
        public static void main(String[] args)
        {
            A a=new B();
            a.show();
            a.display();
            a.see();
            a.view();
        }
}
```

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

But interface can extends more then one interface.

ex:

```
interface A
{
	void m1();
}
interface B
{
	void m2();
}
interface C extends A,B
{
	void m3();
}
class D implements C
{
	public void m1()
	{
		System.out.println("M1-Method");
	}
	public void m2()
	{
		System.out.println("M2-Method");
	}
	public void m3()
	{
```

```
            System.out.println("M3-Method");
        }
}
class Test
{
        public static void main(String[] args)
        {
            C c=new D();
            c.m1();
            c.m2();
            c.m3();
        }
}
```

In java , a class can implements more then one interface.

```
interface Father
{
        float HT=6.2f;
        void height();
}
interface Mother
{
        float HT=5.8f;
        void height();
}
class Child implements Father,Mother
{
        public void height()
```

```
        {
                float height=(Father.HT+Mother.HT)/2;
                System.out.println("Child Height :"+height);
        }
}
class Test
{
        public static void main(String[] args)
        {
                Child c=new Child();
                c.height();
        }
}
```

Java 8

Interface is a collection of abstract methods, default methods and static methods.

Java 9

Interface is a collection of abstract methods, default methods, static methods and private methods.

Q) What is marker interface?

An interface which does not have any constants and methods is called marker interface.

In general, empty interface is called marker interface.

Using marker interfaces we will get some ability to do.

We have following list of marker interface in java.

ex:

Serializable

Cloneable

Remote

Abstract class

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

A abstract keyword is applicable for classes and methods but not for variables.

It is not possible to create object for abstract classes.

To write the implementation of abstract methods of an abstract class we will use sub classes.

By default every abstract method is a public and abstract.

Abstract class contains only instance variables.

syntax:

```
abstract class <class_name>
{
        -
        - //instance variables
        - //abstract methods
        - //concrete methods
        -
}
```

Note:

If we know partial implementation then we need to use abstract class.

ex:

```
abstract class Plan
{
        //instance variable
        protected double rate;

        //abstract method
        public abstract void getRate();
```

```
	//concrete method
	public void calculateBillAmt(int units)
	{
		System.out.println("Total Units :"+units);
		System.out.println("Total Bill :"+(units*rate));
	}
}
class DomesticPlan extends Plan
{
	public void getRate()
	{
		rate=2.5d;
	}
}
class CommercialPlan extends Plan
{
	public void getRate()
	{
		rate=5.0d;
	}
}
class Test
{
	public static void main(String[] args)
	{
		DomesticPlan dp=new DomesticPlan();
		dp.getRate();
```

```
        dp.calculateBillAmt(250);

        CommercialPlan cp=new CommercialPlan();
        cp.getRate();
        cp.calculateBillAmt(250);
    }
}
```

Q)What is the difference between interface and abstract class?

interface	abstract class
To declare interface we will use interface keyword.	To declare abstract class we will use abstract keyword.
It is a collection of abstract methods, default methods and static methods.	It is a collection of abstract methods and concrete methods.
We can achieve multiple inheritance.	We can't achieve multiple inheritance.
It contains constants.	It contains instance variables.
To write the implementation of abstract methods we will use implementation class.	To write the implementation of abstract methods we will use sub classes.
If we know only specification then we need to use interface.	If we know partial implementation then we need to use abstract class.
It does not support constructor.	It supports constructor.

It does not allow blocks.	It allows blocks.

Singleton class

A singleton is one of the design pattern.

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

If we call any static method using class name and that method returns same class object is called singleton class.

ex:

```
LocalDate date=LocalDate.now();

LocalTime time=LocalTime.now();

Calendar c=Calendar.getInstance();
```

We have following list of predefined singleton classes.

ex:

LocalDate

LocalTime

Calendar

and etc.

To create a singleton class we need to take private constructor and factory method.

ex:

```
class Singleton
{
    //static variable
    static Singleton singleton=null;
    private Singleton()
    {


    }
```

```
	//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());
	}
}
```

API

API stands for Application Programming Interface.

API is a collection of packages.

It is a base for the programmer to develop software applications.

In java API is divided in two three types.

1) Predefined API

Built-In API is called predefined API.

ex:

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

2) Userdefined API

API which is created by the user based on the application requirement.

3) Third party API

API which is given by third party vendor.

ex:

JAVAZOOM API

iText API

and etc.

Packages

A package is a collection of classes,interfaces,enums and annotations.

Here enum is special class and annotation is special interface.

In general, package is a collection of classes and interfaces.

A package is also known as folder or a directory.

In java ,we have two types of packages.

1) Predefined packages

2) Userdefined packages

1) Predefined packages

Built-in packages are called predefined packages.

ex:

java.lang

java.io

java.util

java.time

java.util.stream

java.text

java.sql

javax.servlet

and etc.

2) Userdefined packages

A package which is created by the user based on the application requirement is called userdefined package or customized package.

To declare userdefined package we need to use package keyword.

syntax:

```
package     <package_name>;
```

ex:

```
package   com.ihub.www;

import java.util.Calendar;

class   Test

{

        public static void main(String[] args)

        {

                Calendar c=Calendar.getInstance();

                //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
```

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");

        }
}
```

Assigment

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

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

		int cube=(int)Math.pow(n,3);
		System.out.println("cube of a given number is ="+cube);
	}
}
```

o/p:

```
javac	-d	.	Test.java
java	com.ihub.www.Test
```

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 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); // JAN
        }
}
```

Enum vs switch case

From java 1.5 version onwards , it is possible to pass enum to switch case.

Diagram: java30.1

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

Enum is a final so we can't create child enum.

Enum extends with java.lang.Enum class so it is not possible to extends some other class.

Hence we conclude 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 constructor, variables and methods.

ex:

```
enum   Cloths{
        SILK,COTTON,KHADI;
        Cloths(){
                System.out.println("constructor");
        }
}
class Test {
        public static void main(String[] args) {
                Cloths c=Cloths.SILK;
        }
}
```

ex:

```
enum   Cloths{
        SILK,COTTON,KHADI;
        static int i=10;

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

Wrapper classes

The main objective of wrapper classes are .

1) To wrap primitive type to wrapper object and vice versa.

2) To 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

There are two ways to create objects for wrapper class.One will take corresponding primitive as an argument and second will take corresponding String as an argument.

ex:

wrapper class	constructor
Byte	byte or String
Short	short or String
Integer	int or String
Long	long or String
Float	float or String
Double	double or String
Boolean	boolean or String
Character	char

ex:

```
class Test   {
        public static void main(String[] args) {
                Integer i1=new Integer(10);
                System.out.println(i1);
                Integer i2=new Integer("20");
                System.out.println(i2);
        }
}
```

ex:

```
class Test {
        public static void main(String[] args) {
                Boolean b1=new Boolean(true);
                System.out.println(b1);

                Boolean b2=new Boolean("false");
                System.out.println(b2);
        }
}
```

ex:

```
class Test {
        public static void main(String[] args) {
                Character 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 i=Integer.valueOf(10);
                System.out.println(i);

                Long l=Long.valueOf("200");
                System.out.println(l);
        }
}
```

2) xxxValue()

It will convert wrapper object to primitive type.

ex:

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

                byte b= i.byteValue();
                System.out.println(b);
                short s= i.shortValue();
                System.out.println(s);
        }
}
```

3) parseXxx()

It will convert string to primitive type.

ex:

```
class Test {
	public static void main(String[] args) {
		String str="10";
		int i=Integer.parseInt(str);
		System.out.println(i);

		float f=Float.parseFloat(str);
		System.out.println(f);
	}
}
```

4) toString()

It will 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
	}
}
```

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

input:

	1010

	0101

output:

1111

ex:

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

		System.out.println("Enter the first binary :");
		String binary1=sc.next(); //1010

		System.out.println("Enter the second binary :");
		String binary2=sc.next(); //0101

		//convert binary to decimal
		int a=Integer.parseInt(binary1,2);
		int b=Integer.parseInt(binary2,2);
		int c=a+b;

		//convert decimal to binary
		String result=Integer.toBinaryString(c);
		System.out.println(result);
	}
}
```

Inner classes

Sometimes we will declare a class inside another class such concept is called inner class.

ex:

```
class Outer_class
{
    class Inner_class
    {
        -
        -
        -
    }
}
```

Inner classes introduced as a part of event handling to remove GUI bugs.

But due to powerful features and benefits of inner classes. Programmers started to use inner classes in regular programming.

Accessing inner class data from static area of outer class

```
class Outer
{
    class Inner
    {
        public void m1()
        {
            System.out.println("Inner-M1 Method");
        }
    }
```

```
        public static void main(String[] args) {
            Outer.Inner i=new Outer().new Inner();
            i.m1();
    }
}
```

Note:

Here we will get two .clas files i.e Outer.class and Outer$Inner.class.

Accessing inner class data from non-static area of outer class

```
class Outer
{
    class Inner
    {
        public void m1()
        {
            System.out.println("Inner-M1 Method");
        }
    }
    public void m2()
    {
        Inner i=new Inner();
        i.m1();
    }
    public static void main(String[] args)
    {
        Outer o=new Outer();
        o.m2();
    }
```

```
}
```

According java 8 version, it is not possible to declare static members in inner class.

ex:

```
class Outer
{
	class Inner
	{
		public static void m1()
		{
			System.out.println("Inner-M1 Method");
		}
	}
	public void m2()
	{
		Inner i=new Inner();
		i.m1();
	}
	public static void main(String[] args)
	{
		Outer o=new Outer();
		o.m2();
	}
}
```

o/p:

C.T.e illegal static declaration

Interview Question

Q)Write a java program to multiply two arrays and merge them in third array?

input:

2 5 1 3 4

5 8 6 4 9

output:

10 40 6 12 36

ex:

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

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

		for(int i=0;i<arr1.length && i<arr2.length;i++)
		{
			resArr[i]=arr1[i]*arr2[i];
		}
		//display
		for(int i:resArr)
		{
			System.out.print(i+" ");
		}
```

```
    }
}
```

Q) Types of objects in java?

We have two types of objects in java.

1) Immutable object

2) Mutable object

1) Immutable object

After object creation if we perform any changes then for every change a new object will be created such type of object is called immutable object.

ex:

String and Wrapper clases.

2)Mutable object

After object creation if we perform any changes then all the changes will be done in a single object only.

ex:

StringBuffer and StringBuilder

String

It is a collection of characters which is enclosed in a double quotation.

case1:

Once if we create a String object we can't perform any changes.If we perform any changes then for every change a new object will be created such behaviour is called immutability of an object.

ex:

Diagram: java31.1

case2:

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

length

A length is a final variable which is applicable for arrays.

It will return size of an array.

ex:

```
class Test {
	public static void main(String[] args)
	{
		int[] arr=new int[5];
		System.out.println(arr.length);
	}
}
```

length()

It is a predefined method which applicable for String objects.

It will return number of characters present in String.

ex:

```
class Test
{
	public static void main(String[] args)
	{
		String str="bhaskar";
		System.out.println(str.length());
	}
}
```

case3:

Once if we create a string object.Two objects will be created, One is on heap and another is on SCP(String Constant Pool) area.But 's' always points to heap area only.

Diagram: java31.2

Object creation in SCP area is always optional.

First JVM will check is there any object is created with same content or not.

If it is created then JVM simply refers to that object.If it is not creatd then JVM will create a new object.Hence there is a no chance of having duplicate objects in SCP Area.

Garbage collector can't access SCP area objects even though they don't any object reference.

Ex:

Diagram: java31.3

Q)What is the difference between == and .equals() ?

==

It is a equality 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("bhaskar");
		System.out.println(s1==s2);//false
	}
}
```

.equals()

It is a predefined method present in String class which always returns boolean values.

It is used for 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
    }
}
```

Interning of String object

With the help of heap object reference if we need corresponding SCP object reference then we need to use intern() method.

ex:

Diagram: java31.4

Important string methods

Q)Write a java program to accept one 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 Name :");
        String name=sc.next();

        System.out.println("welcome :"+name);
    }
}
```

approach2

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


		System.out.println("Enter the Name :");
		String name=sc.nextLine();


		System.out.println("welcome :"+name);
	}
}
```

Q)Write a java program to remove special characters from given string?

input:

	Ihub@Ta#len$t

output:

	IhubTalent

ex:

```
class Test
{
	public static void main(String[] args)
	{
		String str="Ihub@Ta#len$t";
		str=str.replaceAll("[^A-Za-z0-9]","");
```

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

Q)Write a java program to find out length of the string?

```
class Test
{
        public static void main(String[] args)
        {
                String str="bhaskar";
                System.out.println(str.length());
        }
}
```

Q)Write a java program to convert given string to upper case letter?

```
class Test
{
        public static void main(String[] args)
        {
                String str="bhaskar";
                System.out.println(str.toUpperCase());
        }
}
```

Q)Write a java program to convert given string to lower case letter?

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

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

Q)Write a java program to check given word present in 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";

        if(str.contains("java"))
            System.out.println("It is present");
        else
            System.out.println("It is not present");
    }
}
```

Q)Write a java program to check given strings are equals or not?

input:

str1 ="bhaskar"

str2 ="bhaskar"

output:

Both are equals

```
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 java program to display the character present in a given index?

input:

    str = bhaskar

    index = 2

output:

    a

```
class Test
{
    public static void main(String[] args)
    {
        String str="bhaskar";
        System.out.println(str.charAt(2));
    }
}
```

Q)Write a java program to display first occurance index of a given character?

input:

    str = bhaskar

    ch ='a'

output:

    2

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

```
        char ch='a';
        System.out.println(str.indexOf(ch));
    }
}
```

Q)Write a java program to display last occurance index 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';
        System.out.println(str.lastIndexOf(ch));
    }
}
```

Q)Write a java program to display the string in reverse order?

input:

    hello

output:

    olleh

```
class Test
{
	public static void main(String[] args)
	{
		String str="hello";
		char[] carr=str.toCharArray(); // h  e  l  l  o
		String rev="";
		//reading reverse
		for(int i=carr.length-1;i>=0;i--)
		{
			rev+=carr[i];
		}
		System.out.println(rev);
	}
}
```

Q)Write a java program to display given string is palindrome or not?

input:

racar

output:

It is a palindrome string

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

```
		char[] carr=str.toCharArray();

		String rev="";

		//reading reverse
		for(int i=carr.length-1;i>=0;i--)
		{
			rev+=carr[i];
		}
		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 reverse of a sentence?

input:

This is java class

output:

class java is This

```
class Test
{
	public static void main(String[] args)
	{
		String str="This is java class";
		String[] sarr=str.split(" ");
```

```
		String rev="";

		//reading reverse
		for(int i=sarr.length-1;i>=0;i--)
		{
			rev+=sarr[i]+" ";
		}
		System.out.println(rev);
	}
}
```

Q)Write a java program to display reverse of a word in a given string?

input:

This is java class

output:

sihT si avaj ssalC

ex:

```
class Test {
	public static void main(String[] args) {
		String str="This is java class";
		String[] sarr=str.split(" "); // This is java class
		String rev="";

		//for each loop
		for(String s:sarr)
		{
			char[] carr=s.toCharArray(); // i s
```

```
                    //reading reverse
                    for(int i=carr.length-1;i>=0;i--)
                    {
                            rev+=carr[i];
                    }
                    //add the space
                    rev+=" ";
            }
            System.out.println(rev);
    }
}
```

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 uniques="";
            String duplicates="";

            for(int i=0;i<str.length();i++)
            {
                    String current=Character.toString(str.charAt(i));
```

```
                if(uniques.contains(current))
                {
                        if(!duplicates.contains(current))
                        {
                                duplicates+=current;
                                continue;
                        }
                }
                uniques+=current;
        }
        System.out.println(duplicates);
    }
}
```

Q)Write a java program to display unique characters from given string?

input:

google

output:

gole

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

        String uniques="";
        String duplicates="";

        for(int i=0;i<str.length();i++)
        {
```

```
                String current=Character.toString(str.charAt(i));


                if(uniques.contains(current))
                {
                        if(!duplicates.contains(current))
                        {
                                duplicates+=current;
                                continue;
                        }
                }
                uniques+=current;
        }
        System.out.println(uniques);
    }
}
```

Q)Write a java program to find out most repeating character from given string?

input:

ihubtalentinstitute

output:

t is repeating for 5 times

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


        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 display the string in given format?

input:

A1B2C3D4

output:

ABBCCCDDDD

```
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)); // A B
			}
			else
			{
				int j=Character.getNumericValue(str.charAt(i)); //2
				for(int k=1;k<j;k++)
				{
					System.out.print(str.charAt(i-1));
				}
			}
		}
	}
}
```

Q) Write a java program to check given string is anagram or not?

input:

str1="listen";

str2="silent";

output:

It is a anagram string

```
import java.util.Arrays;
class Test {
	public static void main(String[] args) {
		String str1="listen";
		String str2="silent";

		//convert string to array
		char[] carr1=str1.toCharArray();
		char[] carr2=str2.toCharArray();

		Arrays.sort(carr1); // e i l n s t
		Arrays.sort(carr2); // e i l n s t

		boolean flag=true;
		//checking the letters are same or not
		for(int i=0;i<carr1.length && i<carr2.length;i++){
			if(carr1[i]!=carr2[i]){
				flag=false;
				break;
			}
		}
		if(flag)
			System.out.println("It is Anagram String");
		else
			System.out.println("It is not Anagram String");
	}
}
```

Q)Write a java program to display number of palindromes present in given string ?

input:

racar is madam for java

output:

racar madam

ex:

```
import java.util.Arrays;
class Test {
	public static void main(String[] args) {
		String str="racar is madam for java";

		String[] sarr=str.split(" "); //racar	is	madam	for	java

		//for each loop
		for(String s:sarr)
		{
			//convert string to char array
			char[] carr=s.toCharArray();

			String rev="";

			//reading reverse
			for(int i=carr.length-1;i>=0;i--)
			{
				rev+=carr[i];
			}
```

```
                if(s.equals(rev))
                        System.out.print(s+" ");
                }
        }
}
```

Q)Write a java program to display the strings starting with capital letters?

input:

This is Java student Class

```
import java.util.Arrays;
class Test {
        public static void main(String[] args) {
                String str="This is Java student Class";

                //split the string
                String[] sarr=str.split(" ");

                //for each loop
                for(String s:sarr)
                {
                        char ch=s.charAt(0);

                        if(ch>='A' && ch<='Z')
                                System.out.print(s+" ");
                }
        }
}
```

Q)Write a java program to perform right rotation of a string?

input:

ihubtalent

2

output:

ubtalentih

ex:

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

		int cnt=2;

		//convert string to char array
		char[] carr=str.toCharArray();  // i h u b t a l e n t

		String res1=str.substring(2,carr.length);

		String res2=str.substring(0,cnt);

		System.out.println(res1+res2);
	}
}
```

Q)Write a java program to perform left rotation of a string?

input:

ihubtalent

2

output:

ntihubtale

ex:

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

		int cnt=2;

		//convert string to char array
		char[] carr=str.toCharArray();  // i h u b t a l e n t

		String str1=str.substring(0,carr.length-cnt);

		String str2=str.substring(carr.length-cnt,carr.length);

		System.out.println(str2+str1);
	}
}
```

Q)Write a java program to display permutation of given string?

input:

ABC

output:

ABC

ACB

BAC

BCA

CBA

CAB

ex:

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

		//caller method
		permutation(str.toCharArray(),0);
	}
	//callie method
	public static void permutation(char[] 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);
			permutation(ar,fi+1);
			swap(ar,i,fi);
		}
	}
```

```
	//callie method
	public static void swap(char[] ar,int i,int fi){
		char temp=ar[i];
		ar[i]=ar[fi];
		ar[fi]=temp;
	}
}
```

StringBuffer

If our content change frequently then it is never recommanded to use String object because for every change a new object will be created such behaviour is called immutability of an object.

To overcome this limitation Sun Micro System introduced StringBuffer object.

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(initial capacity);

It will create StringBuffer object with specified initial capacity.

ex:

```
class Test {
        public static void main(String[] args) {
                StringBuffer sb=new StringBuffer(19);
                System.out.println(sb.capacity()); //19
        }
}
```

3)StringBuffer sb=new StringBuffer(String str)

It will create StringBuffer object which is equivalent to String.

Here capacity will be created with below formulea.

ex:

```
        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?

input:

hello

output:

olleh

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

		StringBuffer sb=new StringBuffer(str);

		String rev=sb.reverse().toString();

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

Q)Write a java program to check given string is palindrome or not?

input:

racar

output:

It is a palindrome string

ex:

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

```
        StringBuffer sb=new StringBuffer(str);

        String rev=sb.reverse().toString();

        if(str.equals(rev))
            System.out.println("It is palindrome string");
        else
            System.out.println("It is not palindrome string");
    }
}
```

Q)Write a java program to multiply of two arrays?

input:

    arr1={3,4,9};

    arr2={2,5};

output:

    8725 (349*25)

ex:

```
class Test
{
    public static void main(String[] args)
    {
        int[] arr1={3,4,9};
        int[] arr2={2,5};

        String res1=arrayToString(arr1);
        String res2=arrayToString(arr2);
```

```
            //converting string to int
            int a=Integer.parseInt(res1);
            int b=Integer.parseInt(res2);
            System.out.println(a*b);


        }
        //callie method
        public static String arrayToString(int[] arr)
        {
            StringBuffer sb=new StringBuffer();

            //for each loop
            for(int i:arr)
            {
                sb.append(i);
            }
            return sb.toString();
        }
}
```

StringBuilder

StringBuilder is exactly same as StringBuffer with following difference.

StringBuffer	StringBuilder
All the methods present in StringBuffer are synchronized.	No method present in StringBuilder is synchronized.
At at time only one thread is allowed to execute StringBuffer object.Hence we can	Multiple threads are allowed to execute at a time StringBuilder object.Hence we can't

achieve thread safety.	achieve thread safety.
Waiting time of a thread will increase effectively performance is low.	There is no waiting thread effectively performance is high.
It is introduced in 1.0v.	It is introduced in 1.5v.

Note:

If our content not change fequently then it is recommanded to use String object.

If our content change frequently where thread safety is required then it is recommanded to use StringBuffer object.

If our content change frequently where thread safety is not required then it is recommanded

to use StringBuilder object.

Q)Write a java program to display given string in below format?

input:

ABBCCCDDDD

output:

A1B2C3D4

ex:

```
public class Test{
    public static void main(String[] args) {
        String input = "ABBCCCDDDD";
            StringBuilder sb = new StringBuilder();
            int count = 1;

        for (int i = 0; i < input.length(); i++) {
            // Check if the current character is the same as the next one
            if (i < input.length() - 1 && input.charAt(i) == input.charAt(i + 1)) {
```

```
                count++;
            }
            else {
                // If the next character is different, append the current character and its count
                sb.append(input.charAt(i)).append(count);
                count = 1; // Reset count for the new character
            }
        }
        System.out.println(sb.toString());
    }
}
```

StringTokenizer

StringTokenizer is a class which is present in java.util package.

It is used to tokenized the string irrespective of regular expression.

We can create StringTokenizer class object as follow.

ex:

```
StringTokenizer st=new StringTokenizer(String str,RegEx reg);
```

StringTokenizer class contains following five methods.

ex:

```
public boolean hasMoreTokens()
public String nextToken()
public boolean hasMoreElements()
public Object nextElement();
pubilc int countTokens();
```

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:

The dafault regular expression is space.

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());
    }
}
```

ex:

```
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:
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:
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);
            }
    }
}
```

Interview Question

Q)Java program to compare two dates ?

```
import java.time.*;
class Test {
	public static void main(String[] args) {
		LocalDate d1=LocalDate.now(); // current data
		LocalDate d2=LocalDate.of(2023,9,11); // given date

		int result=d1.compareTo(d2);

		if(result<0)
			System.out.println("Date1   less then Date2 ");
		else if(result>0)
			System.out.println("Date1 greater then Date2 ");
		else
			System.out.println("Both Dates are same");
	}
}
```

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:

ArithmeticException

FileNotFoundException

IllArgumentException

Error

Error is a problem for which we can't provide solution programmatically.

Error will raise due to lack of system resources.

ex:

LinkageError

StackOverFlowError

OutOfMemoryError

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 any interruption in the middle

of the program such type of termination is called abnormal termination.

ex:

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

If any exception raised in our program then we must and should handle that exception

otherwise our program will terminates abnormally.

Here exception will display name of the exception, description of the exception and

line number of the exception.

Exception

Exception is a unwanted, unexpected 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 classified into two types.

i) Checked exceptions

Exception which is checked by the compiler at the time of compilation is called checked exception.

ex:

FileNotFoundException

InterruptedException

EOFException

and etc.

ii) Unchecked exceptions

Exception which is checked by the JVM at the time of runtime is called unchecked exception.

ex:

ArithmeticException

IllegalArgumentException

ClassCastException

and etc.

Diagram: java34.1

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 catch block.

If any exception raise in try block then try block won't be executed.

catch block

It is a block which contains Error Handling Code.

A catch block always associate with try block.

A catch block is used to catch the exception which is thrown by try block.

A catch block will take exception name as a parameter and that name must match with exception class name.

If there is no exception in try block then catch block won't be executed.

syntax:

```
try
{
        -
        - //Risky Code
        -
}
catch(ArithmeticException ae)
{
        -
        - //Error Handling Code
        -
}
```

ex:

```
class Test {
	public static void main(String[] args) {
		try
		{
			System.out.println("try-block");
		}
		catch(Exception e)
		{
			System.out.println("cathc-block");
		}
	}
}
```

o/p:

```
	try-block
```

ex:2

```
class Test {
	public static void main(String[] args) {
		try
		{
			System.out.println(10/0);
		}
		catch(ArithmeticException ae)
		{
			System.out.println("catch-block");
		}
	}
}
```

o/p:

```
catch-block
```

ex:3

```
class Test {
	public static void main(String[] args) {
		try
		{
			System.out.println("stmt1");
			System.out.println(10/0);
			System.out.println("stmt2");
		}
		catch(ArithmeticException ae)
		{
			System.out.println("catch-block");
		}
	}
}
```

o/p:

```
stmt1
catch-block
```

Try with multiple catch block

A try block can have multiple catch blocks.

If try block contains multiple catch blocks then order of catch block is very important it should be from child to parent but not from parent to child.

ex:

```
class Test {
	public static void main(String[] args) {
		try
		{
			System.out.println(10/0);
		}
		catch(ArithmeticException ae)
		{
			System.out.println("from ae");
		}
		catch(RuntimeException re)
		{
			System.out.println("from re");
		}
		catch(Exception e)
		{
			System.out.println("from e");
		}
	}
}
```

Various methods to display exception details

Throwable class defines following three methods to display exception details.

1) printStackTrace()

It will display name of the exception, description of the exception and line number of the exception.

2) toString()

It will display name of the exception and description of the exception.

3) getMessage()

It will display description of the exception.

```
class Test {
	public static void main(String[] args) {
		try
		{
			System.out.println(10/0);
		}
		catch(ArithmeticException ae)
		{
			ae.printStackTrace();

			System.out.println("======================");

			System.out.println(ae.toString());

			System.out.println("======================");

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

finally block

It is never recommanded to maintain cleanup code in try block because if any exception raised 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 cleaup code and it should execute irrespective of exception raised or not.such block is called finally block.

syntax:

```
try
{
        - // Risky Code
}
catch(Exception e)
{
        - // Error Handling
}
finally
{
        - // Cleanup Code
}
```

ex:1

```
class Test {
        public static void main(String[] args) {
                try
                {
                        System.out.println("try-block");
                }
                catch(Exception e)
                {
                        e.printStackTrace();
                }
                finally
                {
```

```
                System.out.println("finally-block");
            }
        }
}
```

o/p:

```
        try-block
        finally-block
ex:2
class Test {
        public static void main(String[] args) {
            try
            {
                System.out.println(10/0);
            }
            catch(Exception e)
            {
                e.printStackTrace();
            }
            finally
            {
                System.out.println("finally-block");
            }
        }
}
```

o/p:

```
java.lang.ArithmeticException: / by zero
         at Test.main(Test.java:7)
```

finally-block

A try with finally combination is valid in java.

ex:

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

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 is a 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 number with zero");
```

ex:

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

Here exception object will be created and handover to JVM by main method.

ex:

```
class Test {
	public static void main(String[] args)
	{
			throw new ArithmeticException("don't divided by zeroooooo");
	}
}
```

Here exception object is created and hover to JVM by throw statement.

throws statement

If any checked exception raised in our program we must and should handle that exception by using try and catch block or by using throws statement.

ex:

```
class Test {
	public static void main(String[] args) {
		try
		{
			Thread.sleep(3000);
			System.out.println("Welcome to Java");
		}
		catch (InterruptedException ie)
		{
			ie.printStackTrace();
		}
	}
}
```

ex:

```
class Test {
	public static void main(String[] args)throws InterruptedException {
		Thread.sleep(5000);
		System.out.println("Welcome to Java");
	}
}
```

2)Userdefined exceptions

Exception which are given by the user based on the application requirements are called userdefined exceptions or customized exceptions.

ex:

NotPracticingException

NoInterestInJobException

NoInterviewsLaterException

TooYoungException

TooOldException

```
import java.util.Scanner;
class TooYoungException extends RuntimeException {
	TooYoungException(String msg){
		super(msg);
	}
}
class TooOldException extends RuntimeException {
	TooOldException(String msg){
		super(msg);
	}
}
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");
	}
}
```

Interview Question

Q)Write a java program to find out given number is even or odd without using modules(%) ?

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


		if((n & 1)==0)
			System.out.println("It is even number");
		else
			System.out.println("It is odd number");
	}
}
/*
	10 - 1010
	  1 - 0001
	----------
	&   - 0000
*/
```

Q)Write a java program for reverse of a given number and do sum of digits of that number?

input:

	123

output:

	reverse : 321

	sum of digits : 6

```
class Test {
	public static void main(String[] args){
		int num=123;

		String str=Integer.toString(num);

		StringBuffer sb=new StringBuffer(str);

		str=sb.reverse().toString();

		int n=Integer.parseInt(str);

		System.out.println("Reverse No :"+n);

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

		System.out.println("sum of digits :"+sum);
	}
}
```

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.

```
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");
        }
}
```

ex:

```
import java.io.*;
class Test {
        public static void main(String[] args){
                FileWriter fw=null;
                try
                {
                        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();
```

```
                System.out.println("Please check the location....");
            }
            catch (IOException ioe)
            {
                ioe.printStackTrace();
            }
            finally
            {
                try
                {
                    fw.close();
                }
                catch (IOException ioe)
                {
                    ioe.printStackTrace();
                }
            }
        }
}
```

FileReader

It is used to read character oriented data from a file.

constructor

FileReader fr=new FileReader(String s);

FileReader fr=new FileReader(File f);

ex:

```
FileReader fr=new FileReader("aaa.txt");
```

or

File f=new File("aaa.txt");

FileReader fr=new FileReader(f);

Methods

1)read()

It will read next character from a file and return unicode value.

If next character is not available then it will return -1.

2)read(char[] ch)

It will read collection of characters from a file.

3)close()

It is used to close FileReader object.

ex:1

```
import java.io.*;
class Test {
	public static void main(String[] args)throws IOException{
		FileReader fr=new FileReader("aaa.txt");

		int i=fr.read();
		while(i!=-1)
		{
			System.out.print((char)i);
			i=fr.read();
		}
		fr.close();
	}
}
```

ex:2

```
import java.io.*;
class   Test{
        public static void main(String[] args)throws IOException {
                FileReader fr=new FileReader("aaa.txt");

                char[] 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.

```
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);
        }
}
```

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.

ex:

```
import java.io.*;
class Test {
        public static void main(String[] args){
            try(FileWriter fw=new FileWriter("xyz.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();
                System.out.println("Please check the location....");
            }
            catch (IOException ioe)
            {
                ioe.printStackTrace();
            }
        }
}
```

Q) How can we handle multiple exceptions in a single catch block?

```
import java.io.*;
class Test {
    public static void main(String[] args){
        try
        {
            System.out.println(10/0);
        }
    catch(ArithmeticException | IllegalArgumentException | ClassCastException   e ){
            e.printStackTrace();
        }
    }
}
```

Generics

Arrays are typesafe.It means we can provide gurantee that what type of elements are present in array.

If requirement is there to store String values then it is recommanded to use String array.

ex:

```
String[] sarr=new String[10];
sarr[0]="hi";
sarr[1]="hello";
sarr[2]=10; //C.T.E
```

At the time of retrieving the data from array we don't need to perform any typecasting.

ex:

```
String[] sarr=new String[10];
sarr[0]="hi";
sarr[1]="hello";
-
-
String val=sarr[0];
```

But Collections are not typesafe.It means we can't provide guarantee 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.Here we won't get any compile time error or runtime error but sometimes our program get failure.

ex:

```
ArrayList al=new ArrayList();
al.add("hi");
al.add("hello");
al.add(10);
```

At the time of retrieving the data from Collection , compulsary we need to perform typecasting.

ex:

```
ArrayList al=new ArrayList();
al.add("hi");
al.add("hello");
-
-
```

String val=(String)al.get(0);

To overcome above limitations , Sun Micro System introduced Generics in 1.5v.

The main objective of Generics are

1) To make collection 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 recommanded to use.	Memory point of view Collections are use.
Arrays are not implemented based on data structure concept.Hence we can't expect any readymade methods.For every logic we need to write the code explicitly.	Collections are implemented based on data structure concept.Hence we can expect ready made methods.
It holds primitive type and object type.	It holds only object type.

Collection

Collection is an interface which is present in java.util package.

It is a root interface for entire Collection Framework.

If we want to represent group of individual objects in a single entity then we need to use Collection.

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: java37.1

ArrayList

The underlying datastructure is resizable or growable array.

Duplicate objects are allowed.

Insertion order is preserved.

Hetrogeneous objects are allowed.

Null insertion is possible.

It implements Serializable, Cloneable and RandomAccess interface.

If our frequent operation is a retrieval operation then ArrayList is a best choice.

ex:

```
import java.util.ArrayList;
class Test {
	public static void main(String[] args){
		ArrayList al=new ArrayList();
		al.add("one");
		al.add("two");
		al.add("three");
		System.out.println(al);//[one, two, three]
		al.add("one");
		System.out.println(al);//[one, two, three, one]
		al.add(10);
		System.out.println(al);//[one,two,three,one,10]
		al.add(null);
		System.out.println(al);//[one,two,three,one,10,null]
	}
}
```

ex:

```
import java.util.ArrayList;
class Test {
	public static void main(String[] args){
		ArrayList<String> al=new ArrayList<String>();
		al.add("one");
		al.add("two");
		al.add("three");
		System.out.println(al);//[one, two, three]
		al.add("one");
		System.out.println(al);//[one, two, three, one]
		al.add(null);
		System.out.println(al);//[one,two,three,one,null]
	}
}
```

ex:

```
import java.util.ArrayList;
class Test {
	public static void main(String[] args){
		ArrayList<String> al=new ArrayList<String>();
		al.add("one");
		al.add("two");
		al.add("three");

		al.add(0,"four");

		for(int i=0;i<al.size();i++)
		{
```

```
                String s=al.get(i);
                System.out.println(s); // four   one    two   three
            }
        }
}
ex:
import java.util.ArrayList;
class Test {
        public static void main(String[] args){
            ArrayList<String> al=new ArrayList<String>();
            al.add("one");
            al.add("two");
            al.add("three");

            System.out.println(al.isEmpty()); // false

            System.out.println(al.contains("one")); //true

            al.remove("two");
            System.out.println(al);//[one,three]

            al.clear();
            System.out.println(al); //[]
        }
}
ex:
```

```
import java.util.ArrayList;
class Test {
	public static void main(String[] args){
		ArrayList<String> al=new ArrayList<String>();
		al.add(new String("one"));
		al.add(new String("two"));
		al.add(new String("three"));
		System.out.println(al);
	}
}
```

ex:

```
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:

```
import java.util.*;
class Test {
	public static void main(String[] args){
		List<Integer> list=Arrays.asList(6,2,8,1,9,7,4);
```

```
                System.out.println(list);//[6,2,8,1,9,7,4]
        }
}
ex:
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]
        }
}
```

LinkedList

The underlying datastructure is a doubly linkedlist.

Duplicate objects are allowed.

Insertion order is preserved.

Hetrogeneous objects are allowed.

Null insertion is possible.

It implements Serializable, Cloneable and Deque interface.

If our frequent operation is a adding and removing in the middle then LinkedList is a best choice.

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);
```

ex:

```
import java.util.*;
class Test {
        public static void main(String[] args){
                LinkedList ll=new LinkedList();
                ll.add("five");
                ll.add("six");
                ll.add("seven");
                System.out.println(ll);//[five,six,seven]
                ll.add("five");
                System.out.println(ll);//[five,six,seven,five]
```

```
                ll.add(10);
                System.out.println(ll);//[five,six,seven,five,10]
                ll.add(null);
                System.out.println(ll);//[five,six,seven,five,10,null]
        }
}
ex:
import java.util.*;
class Test {
        public static void main(String[] args){
                LinkedList<String> ll=new LinkedList<String>();
                ll.add("five");
                ll.add("six");
                ll.add("seven");
                System.out.println(ll);//[five,six,seven]
                ll.add("five");
                System.out.println(ll);//[five,six,seven,five]
                ll.add(null);
                System.out.println(ll);//[five,six,seven,five,null]
        }
}
ex:
import java.util.*;
class Test {
        public static void main(String[] args){
                LinkedList<String> ll=new LinkedList<String>();
                ll.add("five");
```

```
        ll.add("six");
        ll.add("seven");
        System.out.println(ll);//[five,six,seven]
        ll.addFirst("gogo");
        ll.addLast("jojo");
        System.out.println(ll);//[gogo,five,six,seven,jojo]

        System.out.println(ll.getFirst());
        System.out.println(ll.getLast());

        ll.removeFirst();
        ll.removeLast();
        System.out.println(ll);//[five,six,seven]
    }
}
```

Vector

The underlying data structure is resizable array or growable array.

Duplicates objects are allowed.

Insertion order is preserved.

Hetrogeneous objects are allowed.

Null insertion is possible.

All the methods present in Vector are synchronized.Hence we can achieve thread safety.

We have following methods present in Vector class.

ex:

addElement()

removeElement()

firstElement()

lastElement()

removeAllElements();

and etc.

ex:

```
import java.util.*;
class Test {
	public static void main(String[] args) {
		Vector<Integer> v=new Vector<Integer>();

		System.out.println(v.capacity());//10

		for(int i=1;i<=10;i++)
		{
			v.addElement(i);
		}
		System.out.println(v);//[1,2,3,4,5,6,7,8,9,10]

		v.removeElement(5);
		System.out.println(v);//[1, 2, 3, 4, 6, 7, 8, 9, 10]

		System.out.println(v.firstElement()); //1
		System.out.println(v.lastElement()); //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());//10

		for(int i=1;i<=10;i++)
		{
			v.add(i);
		}
		System.out.println(v);//[1,2,3,4,5,6,7,8,9,10]

		v.remove(4);
		System.out.println(v);//[1, 2, 3, 4, 6, 7, 8, 9, 10]

		System.out.println(v.get(0)); //1
		System.out.println(v.get(v.size()-1)); //10

		v.clear();
		System.out.println(v); //[]
	}
}
```

Q)What is the difference between ArrayList vs Vector?

ArrayList	Vector
It is a non-legacy class.	It is a legacy class.
It is introduced in 1.2v.	It is introduced in 1.0v.
No method is synchronized.	All methods are synchronized.
Multiple threads are allowed to execute at a time.Hence we can't achieve thread safety.	At a time only one thread is allowed to execute.Hence we can achieve thread safety.
There is no waiting time of a thread so performance is high.	There is a waiting time of a thread so perforamance is low.

Stack

It is a child class of Vector class.

If we depend upon last in first out (LIFO) order then we need to use stack.

constructor

```
Stack s=new Stack();
```

methods

1) push(Object o)

It is used to push the element to stack.

2) pop()

It is used to pop the element from stack.

3) peek()

It will return toppest element of a stack.

4) isEmpty()

It is used to check stack is empty or not.

5) search(Object o)

It will return offset value if element found otherwise it will return -1.

ex:

```
import java.util.*;
class Test {
	public static void main(String[] args) {
		Stack<String> s=new Stack<String>();
		s.push("A");
		s.push("B");
		s.push("C");
		System.out.println(s);//[A,B,C]

		s.pop();
		System.out.println(s);//[A,B]

		System.out.println(s.peek());//B

		System.out.println(s.isEmpty());//false

		System.out.println(s.search("Z")); //-1

		System.out.println(s.search("A")); //2

	}
}
```

Interview Question

Q) Write a java program to check given string is balanced or not?

input:

{[()]}

output:

It is a balanced string

ex:

```
import java.util.*;
class Test {
	public static void main(String[] args) {
		String str="{[()]}";

		if(isBalanced(str.toCharArray()))
			System.out.println("It is balanced string");
		else
			System.out.println("It is not balanced string");
	}
	//callie method
	public static boolean isBalanced(char[] carr){
		Stack<Character> s=new Stack<Character>();

		for(char ch:carr)
		{
			if(ch=='{' || ch=='[' || ch=='('
			{
				s.push(ch);
			}
```

```
                else if(ch=='}' && !s.empty() && s.peek()=='{')
                {
                        s.pop();
                }
                else if(ch==']' && !s.empty() && s.peek()=='[')
                {
                        s.pop();
                }
                else if(ch==')' && !s.empty() && s.peek()=='(')
                {
                        s.pop();
                }
                else
                {
                        return false;
                }
        }
        return s.isEmpty();
    }
}
```

Q) How to convert Collections to array?

ex:

```
import java.util.*;
class Test {
    public static void main(String[] args) {
        List<Integer> list=Arrays.asList(5,7,1,2,9,7);
```

```
		Integer[] iarr=new Integer[list.size()];

		int j=0;
		for(Integer i:list)
		{
			iarr[j++]=i;
		}
		//display
		for(int i:iarr)
		{
			System.out.print(i+" ");
		}
	}
}
```

Q)How to convert arrays to collections?

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

		List<Integer> list=new ArrayList<Integer>();

		for(int i:arr)
		{
			list.add(i);
		}
		System.out.println(list);
```

```
    }
}
```

Set

It is a child interface of Collection interface.

If we want to represent group of individual objects in a single entity where duplicate objects are not allowed and order is not preserved then we need to use Set interface.

Diagram: java38.1

HashSet

The underlying datastructure is Hashtable.

Duplicate objects are not allowed.

Insertion order is not preserved.

Hetrogeneous objects are allowed.

Null insertion is possible.

ex:

```
import java.util.*;
class Test {
    public static void main(String[] args) {
        HashSet hs=new HashSet();
        hs.add("one");
        hs.add("six");
        hs.add("nine");
        System.out.println(hs);//[nine, six, one]

        hs.add("one");
        System.out.println(hs);//[nine,six,one]

        hs.add(10);
```

```
            System.out.println(hs);//[nine, six, one, 10]


            hs.add(null);
            System.out.println(hs);//[null, nine, six, one, 10]
        }
}
```

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("six");
        lhs.add("nine");
```

```
            System.out.println(lhs);//[one,six,nine]

            lhs.add("one");
            System.out.println(lhs);//[one,six,nine]

            lhs.add(10);
            System.out.println(lhs);//[one,six,nine,10]

            lhs.add(null);
            System.out.println(lhs);//[one,six,nine,10,null]
        }
}
```

Q)Write a java program to display distinct elements?

input:

2 6 7 2 8 5 5 2 9

output:

2 6 7 8 5 9

ex:

```
import java.util.*;
class Test
{
    public static void main(String[] args)
    {
        int[] arr={2,6,7,2,8,5,5,2,9};

        Set<Integer> set=new LinkedHashSet<Integer>();
```

```
            for(int i:arr)
            {
                  set.add(i);
            }
            System.out.println(set);
      }
}
```

TreeSet

The underlying data structure is Balanced Tree.

Duplicates are not allowed.

Order is not preserved because it will take sorting order of an object.

Hetrogenous objects are not allowed otherwise we will get ClassCastException.

Null insertion is not possible otherwiser 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(10);
            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.

Comparable interface 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 obj1 comes before obj2

        It will return +ve obj1 comes after obj2

        It will return 0 if both objects are same

ex:

```
class   Test {
        public static void main(String[] args) {
            System.out.println("A".compareTo("Z"));      //-25

            System.out.println("Z".compareTo("A"));      //25

            System.out.println("K".compareTo("K"));   //0
        }
```

}

Comparator

Comparator is an interface which is present in java.util package.

Comparator interface contains two methods i.e compare() and equals() method.

If we depend upon customized sorting order then we need to use Comparator interface.

ex:

```
public int compare(Object obj1,Object obj2)
        It will return +ve obj1 comes before obj2
        It will return -ve obj1 comes after obj2
        It will return 0 if both objects are same
```

ex:

```
public boolean equals(Object o)
```

Implementation of equals() method is optional because it is present in Object class which is available to our program through inheritance.

Implementation of compare() is mandatory.

ex:1

```
import java.util.*;
class   Test
{
    public static void main(String[] args)
    {
        TreeSet ts=new TreeSet(new MyComparator());
        ts.add(10);
        ts.add(3);
        ts.add(1);
        ts.add(7);
        System.out.println(ts);//[10, 7, 3, 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:2

```
import java.util.*;
class   Test
{
	public static void main(String[] args)
	{
		TreeSet ts=new TreeSet(new MyComparator());
		ts.add(10);
		ts.add(3);
		ts.add(1);
		ts.add(7);
```

```
                System.out.println(ts);//[1, 3, 7, 10]
        }
}
class MyComparator implements Comparator
{
        public int compare(Object obj1,Object obj2)
        {
                Integer i1=(Integer)obj1;
                Integer i2=(Integer)obj2;

                if(i1<i2)
                        return -1;
                else if(i1>i2)
                        return 1;
                else
                        return 0;
        }
}
```

Map

It is not a child interface of Collection interface.

If we want to represent group of objects in key,value pair then we need to use Map interface.

Both key and value must be objects.

Key can't be duplicate but value can be duplicate.

Each key and value pair is called one entry.

Diagram: java39.1

HashMap

The underlying data structure is Hashtable.

Duplicate keys not allowed but values can be duplicate.

Insertion order is not preserved because it will take hashcode of the key.

Hetrogenous objects are allowed for both key and value.

Null insertion is possible for key and value.

ex:

```
import java.util.*;
class   Test
{
        public static void main(String[] args)
        {
                HashMap hm=new HashMap();
                hm.put("one","raja");
                hm.put("six","alan");
                hm.put("nine","jack");
                System.out.println(hm);//{nine=jack, six=alan, one=raja}
                hm.put("one","gogo");
                System.out.println(hm);//{nine=jack, six=alan, one=gogo}
                hm.put(1,10);
                System.out.println(hm);//{nine=jack, 1=10, six=alan, one=gogo}
                hm.put(null,null);
                System.out.println(hm);//{null=null, nine=jack, 1=10, six=alan, one=gogo}
        }
}
```

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("six","alan");
                hm.put("nine","jack");
                System.out.println(hm);//{nine=jack, six=alan, one=raja}
                hm.put("one","gogo");
                System.out.println(hm);//{nine=jack, six=alan, one=gogo}
                hm.put(null,null);
                System.out.println(hm);//{null=null, nine=jack, six=alan, one=gogo}
        }
}
```

LinkedHashMap

It is a child class of HashMap class.

LinkedHashMap is exactly same as HashMap with following differences.

HashMap	LinkedHashMap
The underlying data structure is Hashtable.	The underlying data structure is Hashtable and LinkedList.
Insertion order is not preserved.	Insertion order is preserved.
Introduced in 1.2v.	Introduced in 1.4v.

```
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("six","alan");
                lhm.put("nine","jack");
                System.out.println(lhm);//{one=raja, six=alan, nine=jack}
                lhm.put("one","gogo");
                System.out.println(lhm);//{one=raja, six=alan, nine=jack}
                lhm.put(null,null);
                System.out.println(lhm);//{one=raja, six=alan, nine=jack, null=null}
        }
}
```

TreeMap

The underlying data structure is RED BLACK TREE.

Duplicate keys are not allowed but value can be duplicate.

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(1,"one");
                tm.put(10,"ten");
                tm.put(5,"five");
                tm.put(7,"seven");
                System.out.println(tm);//{1=one,5=five,7=seven,10=ten}
                tm.put(1,"gogo");
                System.out.println(tm);//{1=gogo,5=five,7=seven,10=ten}
                //tm.put(null,"hi");
                //System.out.println(tm);//R.E NullPointerException
        }
}
```

Hashtable

The underlying data structure is hashtable.

Duplicate keys are not allowed but values can be duplicate.

Insertion order is not preserved because it will display descending order.

Hetrogeneous objects are allowed for both key and value.

Key and value can't be null.

ex:

```
import java.util.*;
class   Test
{
        public static void main(String[] args)
        {
                Hashtable<Integer,String> ht=new Hashtable<Integer,String>();
                ht.put(1,"one");
                ht.put(10,"ten");
                ht.put(5,"five");
                ht.put(7,"seven");
                System.out.println(ht);//{10=ten, 7=seven, 5=five, 1=one}
                ht.put(1,"gogo");
                System.out.println(ht);//{10=ten, 7=seven, 5=five, 1=gogo}
                //ht.put(null,"eight");
                //System.out.println(ht);//R.E NullPointerException

                //ht.put(8,null);
                //System.out.println(ht);//R.E NullPointerException
        }
}
```

Interview Questions

Q)Write a java program to display number of words present in a given string?

input:

this is is java java

output:

this=1, is=2, java=2

```
import java.util.*;
class Test
{
	public static void main(String[] args)
	{
		String str="this is is java java";

		String[] sarr=str.split(" ");

		Map<String,Integer> map=new LinkedHashMap<String,Integer>();

		//for each loop
		for(String s:sarr)
		{
			if(map.get(s)!=null)
			{
				map.put(s,map.get(s)+1);
			}
			else
			{
				map.put(s,1);
			}
		}
		System.out.println(map);
	}
}
```

Q)Write a java program to check number of letters 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";

		char[] carr=str.toCharArray();

		Map<Character,Integer> map=new LinkedHashMap<Character,Integer>();

		//for each loop
		for(char c:carr)
		{
			if(map.get(c)!=null)
			{
				map.put(c,map.get(c)+1);
			}
			else
			{
				map.put(c,1);
```

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

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

input:

2 5 7 1 3 9 7 5 2 6

output:

duplicates : 7 5 2

uniques : 2 5 7 1 3 9 6

ex:

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

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

Types of Cursors in Java

Cursor is used to read the objects one by one from Collections.

We have three types of cursors.

1) Enumeration

2) Iterator

3) ListIterator

1) Enumeration

It is used to read objects one by one from legacy Collection objects.

We can create Enumeration object as follow.

ex:

```
Enumeration e=v.elements();
```

Enumeration interface contains following two methods.

ex:

```
public boolean hasMoreElements()
public Object nextElement()
```

ex:

```
import java.util.*;
class Test
{
	public static void main(String[] args)
	{
		Vector v=new Vector();
		for(int i=1;i<=10;i++)
		{
			v.add(i);
		}
		System.out.println(v);//[1,2,3,4,5,6,7,8,9,10]

		Enumeration e=v.elements();
		while(e.hasMoreElements())
		{
			Integer i=(Integer)e.nextElement();
			System.out.println(i);
		}
	}
}
```

Limitations with Enumeration

Using Enumeration interface we can read objects one by one from legacy Collection objects.Hence it is not a universal cursor.

Using Enumeration we can perform read operation but not remove operation.

To overcome this limitation Sun Micro System introduced Iterator.

2)Iterator

Iterator interface is used to read objects one by one from any Collection object.Hence it is a universal cursor.

Using Iterator interface we can perform read and remove operations.

We can create Iterator object as follow.

ex:

```
Iterator itr=al.iterator();
```

Iterator interface contains following three methods.

ex:

```
public boolean hasNext()
public Object next()
public void remove()
```

ex:

```
import java.util.*;
class Test
{
    public static void main(String[] args)
    {
        ArrayList al=new ArrayList();
        for(int i=1;i<=10;i++)
        {
            al.add(i);
        }
        System.out.println(al);//[1,2,3,4,5,6,7,8,9,10]

        Iterator itr=al.iterator();
        while(itr.hasNext())
        {
            Integer i=(Integer)itr.next();
            if(i%2==0)
```

```
                {
                        System.out.println(i);
                }
                else
                {
                        itr.remove();
                }
        }
        System.out.println(al);
    }
}
```

Limitations with Iterator

Using Enumeration and Iterator we can read objects only in forward direction but not in backward direction.Hence there are not bi-directional cursors.

Using Iterator interface we can perform read and remove operation but not adding and replacement of new objects.

To overcome this limitation Sun Micro system introduced ListIterator.

3) ListIterator

ListIterator 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 object as follow.

ex:

```
ListIterator litr=al.listIterator();
```

ListIterator interface contains following 9 methods.

ex:

```
public boolean hasNext()
public Object next()
public boolean hasPrevious()
```

```
        public Object previous()
        public void remove()
        public int nextIndex()
        public int 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("venki");
                al.add("chiru");
                al.add("nag");
                al.add("bala");
                System.out.println(al);//[venki,chiru,nag,bala]

                ListIterator litr=al.listIterator();

                while(litr.hasNext())
                {
                        String s=(String)litr.next();
                        System.out.println(s);
                }
        }
}
```

```
import java.util.*;
class Test
{
	public static void main(String[] args)
	{
		ArrayList al=new ArrayList();
		al.add("venki");
		al.add("chiru");
		al.add("nag");
		al.add("bala");
		System.out.println(al);//[venki,chiru,nag,bala]

		ListIterator litr=al.listIterator();

		while(litr.hasNext())
		{
			String s=(String)litr.next();
			if(s.equals("bala"))
			{
				litr.remove();
			}
		}

		System.out.println(al);//[venki,chiru,nag]
	}
}
```

```
import java.util.*;
class Test
{
	public static void main(String[] args)
	{
		ArrayList al=new ArrayList();
		al.add("venki");
		al.add("chiru");
		al.add("nag");
		al.add("bala");
		System.out.println(al);//[venki,chiru,nag,bala]

		ListIterator litr=al.listIterator();

		while(litr.hasNext())
		{
			String s=(String)litr.next();
			if(s.equals("nag"))
			{
				litr.add("chaitanya");
			}
		}

		System.out.println(al);//[venki,chiru,nag,chaitanya,bala]
	}
}
```

```
import java.util.*;
class Test
{
	public static void main(String[] args)
	{
		ArrayList al=new ArrayList();
		al.add("venki");
		al.add("chiru");
		al.add("nag");
		al.add("bala");
		System.out.println(al);//[venki,chiru,nag,bala]

		ListIterator litr=al.listIterator();

		while(litr.hasNext())
		{
			String s=(String)litr.next();
			if(s.equals("nag"))
			{
				litr.set("chaitanya");
			}
		}
		System.out.println(al);//[venki,chiru,chaitanya,bala]
	}
}
```

Interview Question

Q)Types of Data structures?

We have two types of data structures in java.

1) Primitive Datastructure

2) Non-Primitive Data structure

Diagram: java40.1

Multithreading

Q)What is the difference between Thread and Process?

Thread

It is a light weight sub process.

We can run multiple threads simultenously.

One thread can communicate with another thread.

ex:

class is one thread

constructor is one thread

block is one thread

and etc.

Process

A process is a collection of threads.

We can run multiple processes simultenously.

One process can't communicate with another process.

ex:

downloading a file from internet is one process

typing a notes in editor is one process

taking a class using zoom meeting is one process

Q)What is multitasking?

Executing several task simultenously such concept is called multitasking.

We have two types of multitasking.

1) Process based multitasking

Executing several task simultenously where each task is a independent process.

It is best suitable for OS level.

2) Thread based multitasking

Executing several task simultenously where each task is a same part of a program.

It is best suitable for programmatic level.

Q)What is 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 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
{
    public void run()
    {
        for(int i=1;i<=5;i++)
        {
            System.out.println("Child-Thread");
        }
```

```
        }
}
class Test
{
        public static void main(String[] args)
        {
                //instantiate a thread
                MyThread t=new MyThread();

                //start a thread
                t.start();

                for(int i=1;i<=5;i++)
                {
                        System.out.println("Parent-Thread");
                }
        }
}
```

case1: Thread Schedular

If multiple threads are waiting for execution which thread will be executed will decided by thread schedular.

What algorithm , behaviour or mechanism used by thread schedular is depends upon JVM vendor.

Hence we can't expect any execution order or exact output in multithreading.

case2: Difference between t.start() and 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
{
	public void run()
	{
		for(int i=1;i<=5;i++)
		{
			System.out.println("Child-Thread");
		}
	}
}
class Test
{
	public static void main(String[] args)
	{
		//instantiate a thread
		MyThread t=new MyThread();

		//start a thread
		t.start();

		for(int i=1;i<=5;i++)
		{
			System.out.println("Parent-Thread");
		}
	}
}
```

If we invoke t.run() method then no new thread will be created but run() method will execute just like a normal method.

ex:

```
class MyThread extends Thread
{
	public void run()
	{
		for(int i=1;i<=5;i++)
		{
			System.out.println("Child-Thread");
		}
	}
}
class Test
{
	public static void main(String[] args)
	{
		//instantiate a thread
		MyThread t=new MyThread();

		//no new thread
		t.run();

		for(int i=1;i<=5;i++)
		{
			System.out.println("Parent-Thread");
		}
	}
}
```

case3: If we won't override run() method

If we won't override run() method then 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();

		//start a thread
		t.start();

		for(int i=1;i<=5;i++)
		{
			System.out.println("Parent-Thread");
		}
	}
}
```

case4: If we overload run() method

If we overload run() method then Thread class start() method will execute run() method with no parameter only.

ex:

```
class MyThread extends Thread
```

```
{
	public void run()
	{
		System.out.println("0-arg method");
	}
	public void run(int i)
	{
		System.out.println("int-arg method");
	}
}
class Test
{
	public static void main(String[] args)
	{
		//instantiate a thread
		MyThread t=new MyThread();

		//start a thread
		t.start();

		for(int i=1;i<=5;i++)
		{
			System.out.println("Parent-Thread");
		}
	}
}
```

case5: If we override start() method

If we override start() method then no new thread will be created but start() method will execute just like normal method.

ex:

```
class MyThread extends Thread
{
	public void run()
	{
		System.out.println("run method");
	}
	public void start()
	{
		System.out.println("start method");
	}
}
class Test
{
	public static void main(String[] args)
	{
		//instantiate a thread
		MyThread t=new MyThread();

		//no new thread
		t.start();

		for(int i=1;i<=5;i++)
		{
```

```
                System.out.println("Parent-Thread");
            }
        }
}
```

case 5: Life cycle of a thread

Diagram: java41.1

nce if we create a thread object then our thread will be in new/born state.

Once if we call t.start() method then our thread goes to ready/runnable state.

If Thread Schedular allocates to CPU then our thread enters to running state.

Once run() method execution is completed then our thread goes to dead state.

2) By implementing Runnable interface

```
class MyRunnable implements Runnable
{
    public void run()
    {
        for(int i=1;i<=5;i++)
        {
            System.out.println("Child-Thread");
        }
    }
}
class Test
{
    public static void main(String[] args)
    {
        MyRunnable   r=new MyRunnable();
```

```
            Thread t=new Thread(r); // r is a targatable interface


            //new thread
            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 can set and get name of a thread as follow.

ex:

```
      public final void setName(String name)
      public final String getName()
```

```
class MyThread extends Thread
{
}
class Test
{
      public static void main(String[] args)
```

```
    {
        System.out.println(Thread.currentThread().getName()); // main

        MyThread t=new MyThread();
        System.out.println(t.getName()); //Thread-0

        Thread.currentThread().setName("Parent-thread");
        System.out.println(Thread.currentThread().getName()); // Parent-Thread

        t.setName("Child-Thread");
        System.out.println(t.getName()); //Child-Thread
    }
}
```

Thread Priority

In java, every thread has a priority explicitly provided by the programmer or automatically generated by JVM.

The valid range of thread priority is 1 to 10.Where 1 is a least priority and 10 is highest priority.

If take priority more then 10 then we will get IllegalArgumentException.

Thread class defines following standard constants as a thread priorities.

ex:

Thread.MAX_PRIORITY - 10

Thread.NORM_PRIORITY - 5

Thread.MIN_PRIORITY - 1

We don't have such constants like LOW_PRIORITY and HIGH_PRIORITY.

A thread which is having highest priority will be executed first.

Thread schedular uses thread priority while allocating to CPU.

If multiple threads having same priority then we can't expect any execution order.

We have following methods to set and get thread priority.

ex:

```
public final void setPriority(int priority)
public final int getPriority()
```

ex:

```
class MyThread extends Thread
{
}
class Test
{
    public static void main(String[] args)
    {
        System.out.println(Thread.currentThread().getPriority()); // 5

        MyThread t=new MyThread();
        System.out.println(t.getPriority()); //5

        Thread.currentThread().setPriority(10);
        System.out.println(Thread.currentThread().getPriority()); // 10

        System.out.println(t.getPriority()); //5

        t.setPriority(4);
        System.out.println(t.getPriority()); //4
        t.setPriority(11); // R.E IllegalArgumentException
    }
}
```

Various ways to prevent a thread in java

There are three ways to prevent(stop) a thread in java.

1)yield()

2)join()

3)sleep()

1)yield()

It will pause current execution thread and gives the chance to other threads having same priority.

If there is low priority thread or no waiting thread then same thread will continue it's execution.

If multiple threads having same priority then we can't expect any execution order.

The thread which is yielded when it will get a chance for execution is depends upon mercy of thread schedular.

ex:

public static native void yield()

Diagram: java41.2

ex:

```
class MyThread extends Thread
{
	public void run()
	{
		for(int i=1;i<=5;i++)
		{
			System.out.println("Child-Thread");
		}
	}
}
class Test
{
```

```
    public static void main(String[] args)
    {
        MyThread t=new MyThread();
        t.start();

        for(int i=1;i<=5;i++)
        {
            Thread.currentThread().yield();
            System.out.println("Parent-Thread");
        }
    }
}
```

2)join()

If a thread wants to wait 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 throw statement.

ex:

```
    public final void join()throws InterruptedException
    public final void join(long ms)throws InterruptedException
    public final void join(long ms,int ns)throws InterruptedException
```

Diagram: java41.3

```
class MyThread extends Thread
{
    public void run()
    {
        for(int i=1;i<=5;i++)
        {
```

```
                System.out.println("Child-Thread");
            }
        }
}
class Test
{
    public static void main(String[] args)throws InterruptedException
    {
        MyThread t=new MyThread();
        t.start();
        t.join();

        for(int i=1;i<=5;i++)
        {
            System.out.println("Parent-Thread");
        }
    }
}
```

3)sleep()

If a thread don't want to perform any operation on perticular amount of time then we need to use sleep() method.

A sleep() method will throw one checked exception i.e 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: java41.4

```
class MyThread extends Thread
{
	public void run()
	{
		for(int i=1;i<=5;i++)
		{
			System.out.println("Child-Thread");
			try
			{
				Thread.sleep(2000);
			}
			catch (InterruptedException ie)
			{
				ie.printStackTrace();
			}
		}
	}
}
class Test
{
	public static void main(String[] args)
	{
		MyThread t=new MyThread();
		t.start();
		for(int i=1;i<=5;i++)
		{
			System.out.println("Parent-Thread");
```

```
        }
    }
}
```

Daemon Thread

Daemon thread is a service provider thread which provides services to user threads.

Life of daemon thread is depends upon user threads.If user threads died then daemon thread will terminated automatically.

There are many daemon threads are running 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

A synchronized keyword is applicable for methods and blocks.

A synchronization is allowed one thread to execute given object.Hence we achieve thread safety.

The main advantage of synchronization is we solve data inconsistence problem.

The main disadvantage of synchronization is ,it will increase waiting time of a thread which reduce the performance of the system.

If there is no specific requirement then it is never recommanded to use synchronization concept.

synchronization internally uses lock mechanism.

Whenever a thread wants to access object , first it has to acquire lock of an object and thread will release the lock when it completes it's task.

When a thread wants to execute synchronized method.It automatically gets the lock of an object.

When one thread is executing synchronized method then other threads are not allowed to execute other synchronized methods in a same object concurently.But other threads are allowed to execute non-synchronized method concurently.

ex:

```
class Table
{
```

```
        synchronized void printTable(int n)
        {
                for(int i=1;i<=5;i++)
                {
                                System.out.println(n*i);
                                try
                                {
                                        Thread.sleep(2000);
                                }
                                catch (InterruptedException ie)
                                {
                                        ie.printStackTrace();
                                }
                }
        }
}
class MyThread1 extends Thread
{
        Table t;
        MyThread1(Table t)
        {
                this.t=t;
        }
        public void run()
        {
                t.printTable(5);
        }
```

```
}
class MyThread2 extends Thread
{
	Table t;
	MyThread2(Table t)
	{
		this.t=t;
	}
	public void run()
	{
		t.printTable(10);
	}
}
class Test
{
	public static void main(String[] args)
	{
		Table obj=new Table();
		MyThread1 t1=new MyThread1(obj);
		MyThread2 t2=new MyThread2(obj);

		t1.start();
		t2.start();
	}
}
```

synchronized block

If we want to perform synchronization on specific resource of a program then we need to use

synchronization.

ex:

If we have 100 lines of code and if we want to perform synchronization only for 10 lines then we need to use synchronized block.

If we keep all the logic in synchronized block then it will act as a synchronized method.

ex:

```
class Table
{
     void printTable(int n)
    {
        synchronized(this)
        {
        for(int i=1;i<=5;i++)
        {
                System.out.println(n*i);
                try
                {
                    Thread.sleep(2000);
                }
                catch (InterruptedException ie)
                {
                    ie.printStackTrace();
                }
        }
        }//sync
    }
}
```

```
class MyThread1 extends Thread
{
	Table t;
	MyThread1(Table t)
	{
		this.t=t;
	}
	public void run()
	{
		t.printTable(5);
	}
}
class MyThread2 extends Thread
{
	Table t;
	MyThread2(Table t)
	{
		this.t=t;
	}
	public void run()
	{
		t.printTable(10);
	}
}
class Test
{
	public static void main(String[] args)
```

```
	{
		Table obj=new Table();
		MyThread1 t1=new MyThread1(obj);
		MyThread2 t2=new MyThread2(obj);

		t1.start();
		t2.start();
	}
}
```

3)Static synchronization

In static synchronization the lock will be on class but not on object.

If we declare any static method as synchronized then it is called static synchronization method.

ex:

```
class Table
{
	static synchronized void printTable(int n)
	{
		for(int i=1;i<=5;i++)
		{
				System.out.println(n*i);
				try
				{
					Thread.sleep(2000);
				}
				catch (InterruptedException ie)
				{
					ie.printStackTrace();
```

```
                                }
                        }

                }
        }
class MyThread1 extends Thread
{
        public void run()
        {
                Table.printTable(5);
        }
}
class MyThread2 extends Thread
{
        public void run()
        {
                Table.printTable(10);
        }
}
class Test
{
        public static void main(String[] args)
        {
                MyThread1 t1=new MyThread1();
                MyThread2 t2=new MyThread2();

                t1.start();
```

```
        t2.start();
    }
}
```

Inter-Thread Communication

Two threads can communicate with one another by using wait(),notify() and notifyAll() method.

The Thread which is expecting updations it has to wait() method and the thread which is performing updations it has to call notify() method.

wait(),notify() and notifyAll() method present in Object class but not in Thread class.

To call wait(),notify() and notifyAll() method our current thread must be in a synchronized area otherwise we will get IllegalMonitorStateException.

Once a thread calls wait() method on a given object ,1st it will release the lock of that object immediately and entered into waiting state.

Once a thread calls notify() and notifyAll() method on a given object.It will release the lock of that object but not immediately.

Except wait(),notify() and notifyAll() method ,there is no such concept where lock release can happen.

ex:

```
class MyThread extends Thread
{
    int total=0;
    public void run()
    {
        synchronized(this)
        {
            System.out.println("Child Thread started calculation");
            for(int i=1;i<=10;i++)
            {
                total=total+i;
            }
```

```
                    System.out.println("Child thread giving notification");
                    this.notify();
                }
        }
}
class Test
{
        public static void main(String[] args)throws InterruptedException
        {
                MyThread t=new MyThread();
                t.start();
                synchronized(t)
                {
                        System.out.println("Main Thread waiting for updating");
                        t.wait();
                        System.out.println("Main -Thread got notification ");
                        System.out.println(t.total);
                }
        }
}
```

DeadLock in java

DeadLock will occur in a suitation when one thread is waiting to access object lock which is acquired by another thread and that thread is waiting to access object lock which is acquired by first thread.

Here both the threads are waiting release the thread but no body will release such situation is called DeadLock.

ex:

```
class Test
{
	public static void main(String[] args)
	{
		final String res1="hi";
		final String res2="bye";

		Thread t1=new Thread()
		{
			public void run()
			{
				synchronized(res1)
				{
					System.out.println("Thread1: Locking Resource 1");
					synchronized(res2)
					{
					System.out.println("Thread1: Locking Resource2");
					}
				}
			}
		};
		Thread t2=new Thread()
		{
			public void run()
			{
				synchronized(res2)
				{
```

```
                                System.out.println("Thread2: Locking Resource 2");
                                synchronized(res1)
                                {
                                System.out.println("Thread1: Locking Resource 1");
                                }
                        }
                }
        };
        t1.start();
        t2.start();
    }
}
```

END