Institute Name : IHUB TALENT MANAGEMENT

Website : www.ihubtalent.com

Location : Ameerpet , Hyderabad.

Trainer Name : Niyaz Sir

Experience : 13+ years of experience in development and training

Course Name : Full Stack Java Development Course with AWS

Batch No : IH-JAVA-025

Duration : 4 Months

Mode : Online/Offline

Full Stack Java Development Course

(Full stack Developer)

|

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

FrontEnd technologies

(FrontEnd Developer)

> HTML5

> CSS3

> JavaScript

> Bootstrap

BackEnd technologies

(BackEnd Developer)

> Core Java

> Advanced Java

> JDBC

> Servlets

> React/ReactJS

> JSP

> Oracle DB

> SQL

> PL/SQL

> Frameworks

> Spring Boot

> Microservices

+

AWS Cloud

+

Aptitute classes

+

Softskills classes

+

Free Recorded videos

Programming language

==========

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

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

Diagram: introduction1.1

Java

======

Object oriented programming language

Platform independent programming language.

Case sensitive programming language

Strongly typed checking language.

High level programming language.

Open Source programming language.

1995 --> Sun Micro system --> Oracle Corporation

Java software --> JDK software

C

===

Procedure oriented programming language.

Platform dependent programming language.

Case sensitive programming language

Lossely typed checking language.

Middle level language (LOW + HIGH)

Interview Question

==========

Q)What is Java?

Java is a object oriented , platform independent, case sensitive,

strongly typed checking, high level , open source programming language

developed by James Gosling in the year of 1995.

Programming Language

==========

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

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

Diagram: introduction2.1

Programming language is divided into two types.

1)Low Level Language

2)High Level Language

1)Low Level Language

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

In general, a language which is computer dependent is called low level language.

ex:

Machine Lanuage

Assembly Language

Machine Lanuage

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

It is also known as binary language.

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

Advantages:

> A program writtens in machine language consumes less memory.

> It does not required any translator.

> It is more efficient when compare to other languages.

Diadvantages:

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

> If anywhere error raised in our program then locating and handling that

error becomes difficult.

> Modifications can't be done easily.

Assembly Language

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

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

It is also known as symbolic language.

Assembly language can't understand by a computer directly. We need to use translator.

We have three translators.

i) Assembler

ii) Compiler

iii)Interpreter

i) Assembler

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

Merits:

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

> Modifications can be done easily.

Demerits:

> It is a mind trick to remember all symbolic code.

> It requires translator.

> It is less efficient when compare to machine language.

Q) What is Debugging?

Bug is also known as Error.

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

2)High Level Language

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

In general, a language which is user dependent is called high level language.

Ex:

C++, C#, Java, .net ,Python and etc.

High level language can't understand by a computer directly.We need to take the support of translators.

compiler

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

interpreter

It will execute our program line by line procedure.

Advantages:

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

> Debugging can be done easily.

> Modifications can be done easily.

Disadvantages:

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

> It requires translator.

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

Escape Characters or Escape Sequences

==========

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

Every escape character or sequence starts with back slash(\) followed by a character.

ex:

```
\n
```

Mostly escape characters are placed inside output statement in java.

ex:

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

We have following list of escape characters or escape sequences in java.

1) \n (new line)

2) \t (horizontal tab)

3) \b (back space)

4) \r (carriage return)

5) \f (form feeding)

6) \\ (back slash)

7) \" (double quote)

8) \' (single quote)

and etc.

1) \n (new line)

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

o/p:

	IHUB

TALENT

2) \t (horizontal tab)

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

o/p:

IHUB TALENT

3) \b (back space)

```
class Vishnu
{
	public static void main(String[] args)
	{
		System.out.println("IHUBTAL\bENT");
```

```
        }
}
```

o/p:

IHUBTAENT

ex:

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

o/p:

ITALENT

4) \r (carriage return)

```
class Lahari
{
```

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

o/p:

	TALENT

ex:

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

o/p:

	IHUBNT

6) \\ (back slash)

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

o/p:

IHUB\TALENT

7) \" (double quote)

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

o/p:

IHUB"TALENT

8)\' (single quote)

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

o/p:

IHUB'TALENT

IHUB'TALENT

C program

=========

Q)Write a c program to display %d ?

```
void main()
{
```

```
        clrscr();

        printf("%d"); //0

        getch();
}
```

ex:

--

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

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

        getch();
}
```

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.	Perfomance is high.
Low security.	Highly secured.
It contains less code.	It contains more code.

Naming Conventions in java

==========

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

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

ex:

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

classes

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

ex:

Predefined classes Userdefined classes

System	Test
FileWriter	DemoApp
BufferedReader	QualityThought
PrintWriter	JavaDemo
and etc.	and etc.

interfaces

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

ex:

Predefined interfaces	Userdefined interfaces
Runnable	ITest
Serializable	IDemoApp
Cloneable	IQualityThought
Statement	and etc.
ListIterator	
and etc.	

variables

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

ex:

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

methods

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

ex:

predefined methods	userdefined methods
hashCode()	calculateBillAmt()

toString() getInfo()

getMessage() setDetails()

setPriority() and etc.

and etc.

keywords

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

ex:

predefined keywords

---

public , static , void , class , if, else , do , for , while and etc.

packages

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

ex:

predefined packages userdefined packages

--- ---

java.lang ihubtalent

java.io com.ihubtalent.www

java.util com.google.www

java.text and etc.

java.util.stream

and etc.

constants

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

ex:

predefined constants	Userdefined constants
MAX_PRIORITY	LIMIT
NORM_PRIORITY	DEGREE
MIN_PRIORITY	and etc.
MAX_VALUE	
MIN_VALUE	
and etc.	

Assignment

==========

Class : GopiNath

Interface : IGopiNath

Variable : gopiNath

Method : gopiNath()

Package : com.gopinath.www

Constant : GOPINATH/GOPI_NATH

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)Highly secured

5)Architecture Neutral

6)Robust

7)Multithreaded

8)Dynamic

9)Distributed

and etc.

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

Garbage Collector

Q) Who is the responsible to execute the java program?

JVM (Java Virtual Machine)

Q) Where our java program will execute?

JRE (Java Runtime Environment)

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

There are two ways to call garbage collector 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 control by a remote like setup box.

That time project was called Stealth project and later it was renamed to Green project.

James gosling, Mike Sheradin and Patrick Naughton were there to develop the project and they have met in a place called Aspan/Colarado to start with 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 one programming language called an OAK. OAK means strength, itself is a coffee seed name and it is a national tree for many

countries like Germany, France, USA and etc.

Later in 1995, They have renamed OAK to Java. Java is island of an Indonasia

where first coffee of seed was produced and during the development of project they were consuming lot of coffee's.Hence symbol of java is a cup of coffee with saucer.

Identifiers

==========

A name in java is called identifier.

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

ex:

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

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

Here Test, main , args and i are identifiers.

Rules to declare an identifiers

Rule1:

Identifier will accept following characters.

ex:

A-Z

a-z

0-9

_

$

Rule2:

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

ex:

```
int  empId; //valid
int  emp_id; //valid
int  emp$sal; //valid
int  dept#No; //invalid
```

Rule3:

Every identifier must and should starts with alphabet, underscore or

dollar symbol but not with digits.

ex:

```
int  a1234; //valid
int  _abcd; //valid
int  $=20;  //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  for; //invalid
int public; //invalid
```

Rule6:

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

Rule7:

Identifier can be alpha numeric character also.

ex:

```
int  emp_Id1;
```

Reserved Words

==========

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

Java supports 53 reserved words.

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

In java reserved words are divided into two types.

Diagram: java5.1

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 datatypes

---

byte

short

int

long

float

double

boolean

char

Used keywords with respect to modifiers

---

default

public

private

protected

final

abstract

static

strictfp

synchronized

native

transient

volatile

Used keywords with respect to return type

---

void

Used keywords with respect to flow control

---

if

else

switch

case

break

continue

do

while

for

Used kewords with respect to exception handling

---

try

catch

finally

throw

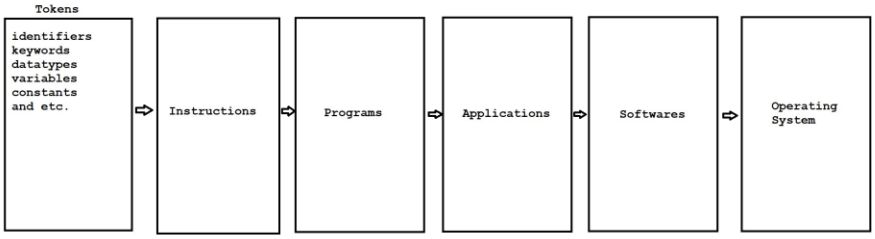
throws

assert

Interview questions

==========

Diagram : java5.2



Java

=======

JDK : 1.8v

Version : Java 8

Creator : James Gosling

Vendor : Oracle Corporation

Open source : Open source

website : www.oracle.com/in/java

Download link :

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

Steps to setup Java Environmental variables

==========

step1:

Make sure JDK 1.8 version installed successfully.

step2:

Copy "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/Mycomputer --> properties -->

Advanced system settings --> environrmental 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 application in java

==========

step1:

Make sure JDK 1.8 version installed successfully.

step2:

Make sure environmental setup done perfectly.

step3:

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

step4:

Open the notepad and develop simple Hello World.

ex:

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

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

step5:

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

step6:

Open the command prompt from javaprog folder.

step7:

------

compile the java program by using below command.

ex:

```
javac   Test.java
          |
        file name
```

step8:

Run the java program by using below command.

ex:

java Test

|

classname

Internal Architecture of JVM

==========

Diagram: java6.1

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

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

We have five types of memories in java.

1)Method Area

Method are contains code of a class, code of a method and code of a variable.

2)Heap area

Our object creations will store in heap area.

Note:

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

3)Java Stack

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.

Similarly native methods will execute in native method stack.

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 will use interpreter and jit compiler simultenously.

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

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

Interview Questions

==========

Q)How many memories are there in java?

We have five memories in java.

1)Method Area

2)Heap

3)Java Stack

4)PC Register

5)Native Method Stack

Q)What is Native method in java?

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

Q)What is JIT compiler?

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

Q)How many classloaders are there in java?

We have three predefined classloaders in java.

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

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

4)Application/System classloader (It loads the .class file from CLASSPATH)

Q)What is package?

A package is a collection of classes and interfaces.

Q)What is Literal ?

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

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

ex:

```
int  i = 10;
 |   |   |___value of a variable | Literal
 |   |_______variable name | identifier
 |___________datatype | keyword
```

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, we have two types of datatypes.

Diagram: java7.1

byte

It is 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=20;

short s=b;

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

2) short s="hi";

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

3) short s=true;

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

int

It is mostly used datatype in java.

Size: 4 bytes (32 bits)

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

ex:

1) int i="false";

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

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

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

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

Note:

In java , for every character we have Universal Unicode value.

ex:

a --- 97

A --- 65

long

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

Size: 8 bytes (64 bits)

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

ex:

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

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

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

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

Q)Difference between float and double?

float	double

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

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

Size : 4 bytes (32 bits)

Size : 8 bytes (64 bits)

Range: -3.4e38 to 3.4e38

Range: -1.7e308 to 1.7e308

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

ex:

10.5f

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

ex:

10.5d

ex:

---

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

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

3) float f='a';

System.out.println(f); // 97.0

4) float f="hi";

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

5) float f=true;

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

boolean

A boolean datatype 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='ab';

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

3) char ch="a";

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

Diagram: java7.2

Interview Questions

==========

Q) Is java purely object oriented or not?

No, Java will not consider as purely object oriented programming language because java does not support many OOPS concepts like multiple inheritance, operator overloading and more ever we depends upon primitive datatypes which are non-objects.

Types of variables

==========

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

Purpose of variable is used to store the data.

In java variables are divided into two types.

1)Primitive variables

It is used to represent primitive values.

2)Reference variables

It is used to represent object reference.

ex:

```
Employee e=new Employee();
          |
      reference variable
```

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

1)Instance variables / Non-static variables

2)Static variables / Global variables

3)Local variables / Temperory variables / Automatic variables

1)Instance variables

A value of a variable which is varied 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 desctructions.Hence scope of instance variable is same as scope of an object.

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

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

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

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

ex:1

```
class Test
{
```

```
	//instance variable
	int i=10;

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

ex:2

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

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

Note:

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

ex:3

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

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

ex:4

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

2)static variable

==========

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

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 .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 can access directly from instance area and static area.

Static variable can access by using object reference and classname.

ex:

```
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 intialized default values.

ex:

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

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

ex:

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

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

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

3)Local variables

==========

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

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

Local variables will store in Java stack memory.

ex:

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

}
```

Note:

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

ex:

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

}
```

o/p:

C.T.E variable i might not have been initialized

A local variable will accept only final modifier.

ex:

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

}
```

ex:

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

```
    }

}
```

Note:

We can't assign any value to final variable.

Main method

==========

Our program contains main method or not.Either it is properly declare or not. It is not a responsibility of a compiler to check. It is a liability of a JVM to look for main method always at runtime.If JVM won't find main method then it will throw one runtime error called main method not found.

JVM always look for main method with following signature.

ex:

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

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

public

JVM wants to call this method from anywhere.

static

JVM wants to call this method without using object reference.

void

Main method does not return any value to JVM.

main

It is an identifier given to main method.

String[] args

It is known as command line arguments.

We can perform following changes in main method.

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

ex

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

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

ex:

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

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

ex:

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

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

ex:

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

5)Main method will accept following modifiers.

ex:

```
synchronized
```

strictfp

final

Command line arguments

==========

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

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

ex:

javac Test.java

java Test 101 raja M 1000.0

| | | |_____args[3]

| | |_____args[2]

| |_____args[1]

|_____args[0]

ex:

class Test

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

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

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

```
javac  Test.java
```

java Test ElonMusk

System.out.println()

==========

It is a output statement in java.

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

syntax:

static variable

|

System.out.println()

| |

predefinedpredefined method.

final

class

Diagram: java8.1

ex:

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

		System.out.printf("stmt2");

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

Various ways to display the data

==========

1)

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

2)

```
	int i=10;
	System.out.println(i);
```

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

3)

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

4)

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

Fully Qualified Name

==========

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

It improves readability of our code.

ex:

java.text.SimpleDateFormat(C)

java.util.Iterator(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 statement we should not use fully qualified name.

Using short name also we can achieve.

We have three types of import statements in java.

1)Explicit class import

2)Implicit class import

3)Static import

1)Explicit class import

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

ex:

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

		LocalTime t=LocalTime.now();
		System.out.println(t);
```

```
    }
}
```

2)Implicit class import

This type of import statement is it 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 d=LocalDate.now();
        System.out.println(d);

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

3)static import

Using static import we can call static members directly.

Often use of static import makes our program less redable and complex.

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");
	}
}
```

Editplus Editor

==========

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

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();

		int c=a+b;

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

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

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

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

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


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

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

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

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

		//logic
		int square=n*n;

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

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

```
import  java.util.Scanner;
class  Example4
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

```
import  java.util.Scanner;
class  Example7
{
        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);
        }
}
```

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

```
import java.util.Scanner;
class Example8
{
        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 perform swapping of two numbers?

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

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

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

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

                //swapping
```

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

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

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

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

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


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

    }
}
```

Typecasting

==========

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

In java , typecasting can be performed 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:

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

ex:

```
class Test
{
```

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

		float f=ch;

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

ex:

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

		double d=i;

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

}

2)Explicit typecasting

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

A programmer is responsible to perform explicit typecasting.

There is a possibility to loss the information.

It is also know as Narrowing or Downcasting.

We can perform explicit typcasting as follow.

ex:

byte <---short

<----

int <-- long <--- float <--- double

<----

char

ex:

```
class Test
{
	public static void main(String[] args)
	{
		float f=10.56f;

		int i=(int)f;

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

	}
}
```

ex:

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

```
        int i=65;

        char ch=(char)i;

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

ex:

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

        byte b=(byte)i;

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

Types of blocks in java

==========

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

In Java , We have three types of blocks.

1)Instance block

2)Static block

3)Local block

1)Instance block

Instance block is used to initialize the values to instance variables.

Instance block will execute when we create an object.

We can declare instance block as follow.

syntax:

```
    //instance block
```

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

ex:

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

ex:

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

o/p:

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

ex:

```
class Test
{
	//instance block
```

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

o/p:

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

ex:

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

```
	//instance block
	{
		i=100;
	}
	public static void main(String[] args)
	{
		Test t=new Test();

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

2)Static block

A static block is used to initialize the value to static variable.

A static block will execute at the time of classloading.

We can declare static block as follow.

syntax:

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

ex:

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

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

```
	}
}
```

o/p:

	static-block
	main-method

ex:

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

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

```
	{
		Test t=new Test();
		System.out.println("main-method");
	}
}
```

o/p:

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

ex:

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

	//static block
	static
	{
		i=200;
	}
```

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

3)Local block

A local block is used to initialize the local variables.

A local block will execute just like normal statement.

We can declare local block as follow.

syntax:

```
    //local block
    {
        -
        - //set of stmt
        -
```

```
        }
```

ex:

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

o/p:

```
stmt1
stmt2
stmt3
```

ex:

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

		//local block
		{
			i=300;
		}

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

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 consider as main class.

ex:

A.java

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

case3:

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

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");
        }
}
```

o/p:

javac A.java (Here three .class files will be created)

java A

java B

java C

Assignments

==========

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

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

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

Operators

==========

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

ex:

a + b

Here + is a operator

Here a and b are operands.

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

We have following list of operators in java.

1) Assignment Operators

2) Conditional/ Ternary Operators

3) Logical Operators

4) Bitwise Operators

5) Relational operators

6) Arithmetic 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 of a variable is possible

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: cannot assign a value to final variable

ex:

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

```
            int i=1,2,3,4,5;

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

Note : C.T.E Illegal start of expression

ex:

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

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

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

Note: Here priority goes to local variable

ex:

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

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

ex:

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

```
        int j= 10 % 20;
        System.out.println(j); //10

    }
}
```

ex:

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

        int j= 10 / 20;
        System.out.println(j); //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*=3; // i = i * 3

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

	}
}
```

ex:

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

		i%=3; //

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

	}
}
```

ex:

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

		i/=3; //

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

	}
}
```

2) Conditional/ Ternary Operators

---

syntax:

	(condition)?value1:value2;

ex:

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

ex:

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

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

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

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

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

		//logic
		int max=(a>b)?a:b;
		System.out.println(max+" is greatest");
	}
}
```

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

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

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

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

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

		//logic

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

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

3)Logical operators

Logical AND operator (&&)

Truth table

T T = T

T F = F

F T = F

F F = F

ex:

```
class  Test
{
	public static void main(String[] args)
	{
		boolean b= (5>2) && (6<10);

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

ex:

```
class  Test
{
	public static void main(String[] args)
	{
		boolean b= (5>20) && (6<10);

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

ex:

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

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

Logical OR operator (||)

Truth table

T	T	= T

T	F	= T

F	T	= T

F	F	= F

ex:

```
class  Test
{
	public static void main(String[] args)
	{
		boolean b= (5>20) || (6<3);

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

ex:

```
class  Test
{
	public static void main(String[] args)
	{
		boolean b= (5>2) || (6<30);

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

```
}

ex:
----
class  Test
{
        public static void main(String[] args)
        {
                boolean b= (5>2) || (6<3);

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

ex:
---
class  Test
{
        public static void main(String[] args)
        {
                boolean b= (5>20) && (6<2) || true;
```

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

Logical NOT operator (!)

---

ex:

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

ex:

```
class  Test
```

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

Assignment

==========

Q)Write a java program to accept one employee salary then display Basic salary , 10% of tax deduction and Actual salary ?

inputs:

100000

outputs:

Basic Salary : 100000

Tax Deduction: 10000

Actual Salary: 900000

ex:

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

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

		float tax=(float)salary*10/100;
		System.out.println("Tax Deduction:"+tax);

		float actualSal = salary - tax;
		System.out.println("Actual Salary :"+actualSal);
	}
}
```

Converting decimal to binary number

==========

Decimal number : 10

Binary number : 1010

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

Converting binary to decimal number

==========

Binary number : 0101

Decimal number : 5

0101

<----

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

1 + 0 + 4 + 0 = 5

4)Bitwise Operators

==========

Bitwise AND operator (&)

Bitwise AND operator deals with binary numbers.

Truth table

T T = T

T F = F

F T = F

F F = F

ex:

```
---
class  Test
{
	public static void main(String[] args)
	{
		int a=10,b=15;
		int c= a & b;
		System.out.println(c);//10
	}
}
/*
		10 - 1010
		11 - 1111
		----------
		&  - 1010
				<----
		0*1 + 1*2 + 0*4 + 1*8

		0 + 2 + 0 + 8 =10

*/
```

ex:

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

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

Bitwise OR operator(|)

Bitwise OR operator deals with binary numbers.

Truth table

T T = T

T F = T

F T = T

F F = F

ex:

```
class  Test
{
	public static void main(String[] args)
	{
		int a=10,b=15;
		int c= a | b;
		System.out.println(c);// 15
	}
}
```

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


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

Bitwise XOR operator (^)

Bitwise XOR operator deals with binary numbers.

Truth table

T T = F

T F = T

F T = T

F F = F

ex:

```
class  Test
{
	public static void main(String[] args)
	{
		int a=10,b=15;
		int c= a ^ b;
		System.out.println(c);// 5
	}
}
/*
		10 - 1010
		15 - 1111
		-----------
		^  - 0101
				<----
		1*1 + 0*2 + 1*4 + 0*8

		1 + 0 + 4 + 0  = 5
```

```
*/

Bitwise NOT operator (~)
-------------------------
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)Relational operators

```
class  Test
```

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

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

		System.out.println(2 <= 10); //true

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

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

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

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

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


	}
}
```

Right Shift operators (>>)

---

10 >> 1 = 10/2

10 >> 2 = 10/4

10 >> 3 = 10/8

10 >> 4 = 10/16

10 >> 5 = 10/32

ex:

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

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

```
        }
}
```

ex:

```
class  Test
{
        public static void main(String[] args)
        {
                int i=10 >> 5;

                System.out.println(i); // 10 / 32 = 0
        }
}
```

Left Shift operators (<<)

---

10 << 1 = 10*2

10 << 2 = 10*4

10 << 3 = 10*8

10 << 4 = 10*16

10 << 5 = 10*32

ex:

```
class  Test
{
	public static void main(String[] args)
	{
		int i=10 << 4;

		System.out.println(i); // 10 * 16 = 160
	}
}
```

ex:

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

```
	{
		int i=100 << 5;

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

6)Arithmetic operators

==========

% - modules

/ - division

* - multiplication

+ - addition

- - subtraction

ex:

```
class  Test
{
	public static void main(String[] args)
	{
		int i=10+5%10+6/2+9*2+6-4;
```

```
            System.out.println(i);
        }
}
/*
        10 + 5%10 + 6/2 + 9*2 + 6-4

        10 + 5   + 3   + 18  +  2

        38
 */
```

Increment/Decrement operators(++/--)

==========

We have two types of increment operators.

i)Pre-increment

ex:

```
            ++i;
```

2)post-increment

ex:

```
        i++;
```

We have two types of decrement operators.

i)Pre-decrement

ex:

```
        --i;
```

ii)Post-decrement

ex:

```
        i--;
```

Post increment/decrement

Rule 1 : First Take

Rule 2 : Then Change

ex:

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

```
	{
		int i=10;

		i++;

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

ex:

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

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

ex:

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

		int j=i++;

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

ex:

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

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

```
            System.out.println(i+" "+j);//10 21
        }
}
```

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
        }
}
```

Pre increment/decrement

Rule 1 : First Change

Rule 2 : 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; // 11 10

		System.out.println(i+" "+j);// 10  21
	}
}
```

ex:

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

		int j=--i + --i + --i; // 9 + 8 + 7
```

```
            System.out.println(i+" "+j);//7 24
        }
}
```

ex:

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

            System.out.println(i++ + ++i);//10 + 12
        }
}
```

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 repeatedly.

In java, we have four control statements.

1) Decision making statement

2) Selection statement

3) Iteration statement

4) Jump statement

1) Decision making statement

It is used to create conditions in our programs.

Decision making statement is possible by using following ways.

i) if stmt

ii) if else stmt

iii) if else if ladder

iv) nested if stmt

i) if stmt

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

syntax:

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

ex:

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

o/p:

```
stmt1
stmt2
stmt3
```

ex:

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

```
        {
                System.out.println("stmt1");
                if(10 == 20)
                {
                        System.out.println("stmt2");
                }
                System.out.println("stmt3");
        }
}
```

o/p:

```
stmt1
stmt3
```

ex:

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

```
                System.out.println("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(2>1)
                {
                        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(2>10)
                {
                        System.out.println("stmt2");
                }
                else
                {
                        System.out.println("stmt3");
                }
```

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

o/p:

```
stmt1
stmt3
stmt4
```

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

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

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

                if(age>=18)
                        System.out.println("U r eligible to vote");
```

```
            else
                System.out.println("U r not eligible to vote");
        }
}
```

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

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

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

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

```
	}
}
```

Q)Write a java program to 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 umber");
	}
}
```

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

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

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

		if(n==0)
		{
			System.out.println("It is not a +ve or -ve 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 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 leap year");
                else
                        System.out.println("It is not leap year");
        }
```

}

iii) if else if ladder

==========

It will execute the source code based on multiple conditions.

syntax:

```
if(cond1)
{
    - //code to be execute cond1 is true
}
else if(cond2)
{
    - //code to be execute cond2 is true
}
else if(cond3)
{
    - //code to be execute cond3 is true
}
else
{
```

```
            - //code to be execute if all conditions are false
        }


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

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

                if(option==100)
                {
                        System.out.println("It is police number");
                }
                else  if(option==103)
                {
```

```
                System.out.println("It is 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 vowel or not?

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

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

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

    }
}
```

Q)Write a java program to check given alphabet is a upper case letter, lower case letter, digit or 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");

	}
}
```

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(true)
		{
			System.out.println("stmt2");
			if(!(5>20))
			{
				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(false)
		{
			System.out.println("stmt2");
			if(!(5>20))
			{
				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(true)
                {
                        System.out.println("stmt2");
                        if(5>20)
                        {
                                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 by using nested if stmt?

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

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

		if(n!=0)
		{
```

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

2)Selection Statement

==========

switch case

It will execute the source code based on multiple conditions.

It is similar to if else if ladder.

syntax:

```
    switch(condition/expression)
```

```
{
        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.
}
```

Declaration of break statement is optional in switch case.If we won't define break statement then from where our condition is satisfied from there all

cases will be executed , that state is called "fall through state of switch case".

ex:

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

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

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

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

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");

			case 103:
				System.out.println("It is enquiry number");

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

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

Q)Write a java program to 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 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...");
            }
        }
}
```

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

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

ex:

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

```
        float f=10.3f;

        switch(f)
        {
            case 10.1 : System.out.println("January"); break;
            case 10.2: System.out.println("February"); break;

            case 10.3: System.out.println("March"); break;
            case 10.4: System.out.println("April"); break;
            case 10.5 : System.out.println("May"); break;
            default : System.out.println("coming soon...");
        }
    }
}
```

Assignment

==========

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

if else stmt?

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

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

i) If average greater then equals to 75 then A grade.

ii) If average greater then equals to 50 then B grade.

iii) If average greater then equals to 35 then C grade.

iv) If average is less then 35 then Failed.

3)Iteration statement

==========

Iteration statement is used to execute the code repeatedly.

Iteatation statement is possible by using LOOPS.

We have four types of loops.

i)do while loop

ii) while loop

iii) for loop

iv) for each loop

i)do while loop

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

syntax:

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

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

ex:

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

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

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

ex:

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

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

```
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?

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

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

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

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

input:

5

output:

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

ex:

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

```
            int n=sc.nextInt();

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

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

Q)Write a java program to find out 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+" ");// nothing
		}
	}
}
```

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

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

		while(i<=100)
```

```
            {
                System.out.print(i+" ");// 1 2 3  ... 100

                i++;
            }
        }
}
```

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

input:

123

output:

6

ex:

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

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

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

        int rem,sum=0;

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

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

input:

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

output:

It is a armstrong number

ex:

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

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

		int temp=n;

		int rem,sum=0;

		while(n>0)
```

```
        {
            rem=n%10;
            sum=sum+rem*rem*rem;
            n=n/10;
        }
        if(temp==sum)
            System.out.println("It is armstrong number");
        else
            System.out.println("It is not armstrong number");
    }
}
```

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

inpupt:

123

output:

321

ex:

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

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

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

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

input:

121

output:

It is palindrome number

ex:

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

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

		int temp=n;
```

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

        if(temp==rev)
            System.out.println("It is palindrome number");
        else
            System.out.println("It is not palindrome number");
    }
}
```

iii)for loop

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

syntax:

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

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

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

ex:
----
class Test
{
```

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

			i--;
		}
	}
}
```

ex:

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

ex:

---

class Test

```
{
	public static void main(String[] args)
	{
		for(int i=1;i<=20;i++)
		{
			if(i%2==0)
			{
				System.out.print(i+" "); //4 10 16
			}
			i=i+2;
		}
	}
}
```

ex:

```
class Test
{
	public static void main(String[] args)
	{
		for(int i=1;i<=20;i++)
		{
```

```
                if(i%2==0)
                {
                        System.out.print(i+" ");//2 6 10 14 18
                        i=i+2;
                }
            }
        }
}
```

Assignment

==========

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

input:

123

output:

ThreeTwoOne

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

fibonacci series : 0 1 1 2 3 5 8

ex:

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

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

		int a=0,b=1,c;

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

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

```
                a=b;
                b=c;
            }


        }
}
```

Q)Write a java program to find out given number is 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 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(); //5

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

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

Q)Write a java program to find out 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);
	}
}
```

Assignment

==========

Q)Write a java program to display list of prime numbers between two given numbers?

input:

2 20

output:

2 3 5 7 11 13 17 19

Various ways to declare the methods in java

==========

We can declare a method in java by using following ways.

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 any 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
	public static void sum()
	{
		Scanner sc=new Scanner(System.in);
		System.out.println("Enter the first number :");
		int a=sc.nextInt();
```

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

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

Q)Write a java program to find out given number is even or odd by using no returntype 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");
	}
}
```

3)With returntype With No argument method

A returntype is completely depends upon datatype of output variable.

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

ex:

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

```
class Test
{
	public static void main(String[] args)
	{
		//caller method
		int k=sum();
		System.out.println("sum of two numbers is ="+k);
	}

	//callie method
	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 by 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;
    }

}
```

2)No returntype With argument method

Number of arguments is depend upon number of inputs.

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

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

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

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

        //caller method
        sum(a,b);
    }

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

}
```

Q)Write a java program to find out given number is even or odd using 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 number :");
		int n=sc.nextInt();

		//caller method
		find(n);
	}

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

```
}
```

4)With returntype With argument method

Q)Write a java program to find out 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;
        }
}
```

Assignment

==========

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

Q)Write a java program to find out list of 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+" ");
		}
	}
}
```

Java Recursion

==========

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

Recursion is similar to loopings.

Whenever we use recursion , we should not use loops.

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

```
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 find out factorial of a given number using recursion?

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

```
        //caller method
        System.out.println(factorial(n));
    }

    //callie method
    public static int factorial(int n)
    {
        if(n<0)
            return -1;

        if(n==0)
            return 1;

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

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

fibonacci series : 0 1 1 2 3 5 8

input:

4

output:

3

ex:

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

		//caller method
		System.out.println(fib(n));
	}

	//callie method
	public static int fib(int n)
```

```
	{
		if(n==1 || n==2)
			return 1;

		if(n==0)
			return 0;

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

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

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

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

        //caller method
        System.out.println(sum(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 given number is palindrome or not using recursion?

input:

121

output:

It is palidrome number

ex:

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

		//caller
		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);

	}
}
```

LOOP Patterns

==========

1)

1 1 1 1

2 2 2 2

3 3 3 3

4 4 4 4

ex:

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

```
}
```

2)

1 2 3 4

1 2 3 4

1 2 3 4

1 2 3 4

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

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

3)

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

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

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

4)

4 4 4 4

3 3 3 3

2 2 2 2

1 1 1 1

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

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

5)

A A A A

B B B B

C C C C

D D D D

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("");
		}
	}
}
```

6)

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("");
		}
	}
}
```

Assignment

==========

1)

1 1 1

1 0 1

1 1 1

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

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++)
			{
				if(i==j)
					System.out.print("* ");
				else
					System.out.print("- ");
			}
			//new line
			System.out.println("");
		}
	}
}
```

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==j || i+j==6)
					System.out.print("* ");
				else
					System.out.print("- ");
			}
```

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

Left Side LOOP patterns

==========

1)

1

2 2

3 3 3

4 4 4 4

ex:

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

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

2)

1

1 2

1 2 3

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

3)

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("");
		}
	}
}
```

4)

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

ex

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

5)

1

2 3

4 5 6

7 8 9 0

ex:

```
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");
				k++;
			}
```

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

        }
}
```

6)

A

B B

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<=i;j++)
                {
                        System.out.print(i+" ");
                }
                //new line
                System.out.println("");
            }

        }
}
```

7)

1

2 1

1 2 3

4 3 2 1

ex:

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

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

8)

1

2#1

1#2#3

4#3#2#1

ex:

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

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

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

Assignment

Q)Write a java program to display loop patter for prime numbers?

2

3 5

7 11 13

17 19 23 29

Right Side loop patterns

==========

1)

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

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

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

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

2)

4 4 4 4

 3 3 3

  2 2

   1

ex:

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

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

3)

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

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

ex:

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

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

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

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

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

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)
	{
		//rows
		for(int i=1;i<=4;i++)
		{

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

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

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

//descending
//rows
for(int i=3;i>=1;i--)
{

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

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

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

        }
}
```

Note:

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

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

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

4)Jump statement

==========

Jump statement is used to jump from one section of code to another section.

We have two types of jump statements.

1) break statement

2) continue statement

1) break statement

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

For conditional statement we can use if condition.

syntax:

```
        break;
```

ex:

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

o/p:

        C.T.E : break outside switch or loop

ex:

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

o/p:

C.T.E : break outside switch or loop

ex:

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

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

2) continue statement

A continue stmt is used to continue the execution of loop.

For conditional statement we can use if condition.

syntax:

```
      continue;
```

ex:

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

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

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

o/p:

C.T.E continue outside of loop

ex:

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

```
            }
        }
}
```

Arrays

========

In a normal variable we can store only one value at time.

To store more then one value in a single variable then we need to use Arrays.

Array is a collection of homogeneous data elements.

The main advantages of arrays are

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

ex:

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

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

The main disadvantages of arrays are

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

2) To use array concept in advanced we should know what is the size of an array which is also 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.

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

Single Dimensional Array Double Dimensional Array Multi Dimensional Array

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

Array Creation

==========

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

Diagram: java18.1

Rules to constructor an array

Rule1:

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

ex:

```
int[] arr=new int[3];  //valid
int[] arr=new int[]; //C.T.E array dimension missing
```

Rule2:

It is legal to have an array size with zero.

ex:

```
int[] arr=new int[0];
System.out.println(arr.length);//0
```

Rule3:

We can't give negative number as an array size otherwise we will get one exception called NegativeArraySizeException.

ex:

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

```
System.out.println(arr.length); // R.E NegativeArraySizeException
```

Rule4:

The allowed datatype for an array size is byte,short,int and char.If we take other datatypes then we will get compile time error.

ex:

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

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

int[] arr=new int[10.5d];// C.T.E
```

Rule5:

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

ex:

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

Array Initialization

At the time of array creation ,every array elements will be initialized with default values.

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

Diagram: java18.2

Array Declaration, creation and Intialization 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 for arrays.

It will return size of an array.

ex:

```
class Test
{
	public static void main(String[] args)
	{
		int[] arr=new int[6];

		System.out.println(arr.length);//6
	}
}
```

length()

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

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

Array which contains only one dimension is called single dimensional array.

syntax: optional

|

datatype[] variable_name=initialization;

ex:

int[] arr;

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

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

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

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

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

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

approach2

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

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



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

Approach3

```
class Test
{
      public static void main(String[] args)
      {
            int[] arr={10,20,30,40};

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

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

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

input:

4 7 1 3 9 2

output:

26

ex:

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

        int sum=0;
        for(int ele:arr)
```

```
            {
                sum+=ele;
            }
            System.out.println(sum);
        }
}
```

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

input:

4 7 1 3 9 2

output:

2 9 3 1 7 4

ex:

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

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

      }
}
```

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

input:

5 9 1 3 7 6

output:

1 3 5 6 7 9

approach1

```
import java.util.Arrays;
class Test
{
```

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

		Arrays.sort(arr); // 1 3 5 6 7 9

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

approach2

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

```
        //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;
                        }
                }
        }

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

```
}
```

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

input:

5 9 1 3 7 6

output:

9 7 6 5 3 1

approach1

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

		Arrays.sort(arr);//1 3 5 6 7 9

		//reading reverse
```

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

apporach2

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


		//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;
                        }
                }
        }

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

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

input:

5 9 1 3 7 6

output:

1

approach1

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

		Arrays.sort(arr);//1 3 5 6 7 9

		System.out.println(arr[0]);//1
	}
}
```

approach2

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

		int small=arr[0];

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

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

input:

5 9 1 3 7 6

output:

9

approach1

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

		Arrays.sort(arr);//1 3 5 6 7 9

		System.out.println(arr[arr.length-1]);//9
	}
}
```

approach2

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

		int big=arr[0];

		//for each loop
		for(int ele:arr)
		{
			if(ele>big)
			{
				big=ele;
			}
		}
		System.out.println(big);
	}
```

```
}
```

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

input:

4 6 2 3 1 1 6 9 7 2

output:

6 2 1

ex:

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

		//duplicate elements
		for(int i=0;i<arr.length;i++)
		{
```

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

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

input:

4 6 2 3 1 1 6 9 7 2

output:

4 3 9 7

ex:

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

		//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:

3 5 6 1 2 1 9 3 3 5 2 3 7 1

output:

3 is repeating for 4 times

ex:

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

```
int maxCount=0;
int element=0;

//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(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:

2 4 9 5 13 17 6 29

output:

2 5 13 17 29

ex:

```
class Test
{
        public static void main(String[] args)
        {
                int[] arr={2,4,9,5,13,17,6,29};
```

```
            for(int n:arr)
            {
                boolean flag=true;

                //prime logic
                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 find out leader elements from given array?

input:

4 8 32 16 9 12 5

output:

5 12 16 32

ex:

```
class Test
{
	public static void main(String[] args)
	{
		int[] arr={4,8,32,16,9,12,5};

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

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

		for(int i=arr.length-2;i>=0;i--)
		{
			if(arr[i]>max)
			{
```

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

        }
}
```

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

input:

5 7 1 2 3 6

output:

4

ex:

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

```
	{
		int[] arr={5,7,1,2,3,6};

		int sum_arr_ele=arr.length+1;


		int sum=(sum_arr_ele*(sum_arr_ele+1))/2;


		for(int ele:arr)
		{
			sum=sum-ele;
		}

		System.out.println("Missing element is ="+sum);
	}
}
```

Assignment

==========

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

input:

arr1 : 2 4 6 8 10

arr2 : 1 3 5 7 9

output:

3 7 11 15 19

Q)Write a java program to seggreate array elements?

input:

0 1 1 0 0 1 0 1 1 0

output:

0 0 0 0 0 1 1 1 1 1

approach1

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

```
		Arrays.sort(arr);//0 0 0 0 0 1 1 1 1 1

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

approach2

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

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

		//inserting '0'
		//for each loop
		int j=0;
```

```
            for(int ele:arr)
            {
                if(ele==0)
                {
                    resArr[j++]=ele;
                }
            }

            //inserting '1'
            while(j<arr.length)
            {
                resArr[j++]=1;
            }

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

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

input:

arr1 = 5 4 1 3 2

arr2 = 9 6 7 8 10

output:

1 2 3 4 5 6 7 8 9 10

ex:

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

        int[] arr2 ={9,6,7,8,10};
```

```
        int size1=arr1.length;
        int size2=arr2.length;

        arr1=Arrays.copyOf(arr1,size1+size2);

        int j=0;
        for(int i=size1;i<arr1.length;i++)
        {
            arr1[i]=arr2[j++];
        }

        //sorting
        Arrays.sort(arr1);

        //for each loop
        for(int ele:arr1)
        {
            System.out.print(ele+" ");
        }
    }
}
```

Q)Write a java program to delete first occurance from given array?

input:

arr = 4 3 9 6 3 7 3 1 2

ele = 3

output:

4 9 6 3 7 3 1 2

ex:

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

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

```
        int ele=3;

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

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

    }
}
```

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

input:

arr = 4 7 9 1 2 6

ele = 5

position = 3

output:

4 7 9 5 1 2 6

ex:

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

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

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

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

Two Dimensional Array

==========

Two dimensional array is a combination of rows and columns.

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

The main objective of two dimensional array is memory utilization.

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

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?

ex:

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

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

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

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

		//inserting the elements
		for(int i=0;i<rows;i++)
		{
			for(int j=0;j<cols;j++)
			{
```

```
                System.out.println("Enter the element :");
                arr[i][j]=sc.nextInt();
            }
        }

        //display the 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 display square of a matrix ?

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

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

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

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

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

        //display the elements
        for(int i=0;i<rows;i++)
        {
                for(int j=0;j<cols;j++)
                {
                        System.out.print(arr[i][j] * arr[i][j] +" ");
                }
                //new line
                System.out.println("");
        }

}
```

```
}
```

Q)Write a java program to find out sum of diagonal elements?

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

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

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

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

		//inserting the elements
		for(int i=0;i<rows;i++)
```

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

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

```
}
```

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

ex:

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

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

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

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

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

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

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

Q)Write a java program to find out sum of lower triangle elements?

ex:

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

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

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

```
int cols=sc.nextInt();//3

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

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

//display the elements
int sum=0;
for(int i=0;i<rows;i++)
{
	for(int j=0;j<cols;j++)
	{
		if(i>j)
		{
```

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

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

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

input:

1 2 3

4 5 6

7 8 9

output:

1 2 3 6 9 8 7 4 5

ex:

```
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.

syntax:

```
new int[]{10,20,30};
```

```
        new int[][]{{10,20,30},{40,50,60}};


ex:
----
public class Test
{
   public static void main(String[] args)
   {
                //caller method
                sum(new int[]{10,20,30});

        }
        //callie method
        public static void sum(int[] arr)
        {
                int sum=0;
                for(int i:arr)
                {
                        sum+=i;
                }
                System.out.println(sum);
        }
```

```
}

ex:
----
public class Test
{
   public static void main(String[] args)
   {
            //caller method
            System.out.println(sum(new int[]{10,20,30}));

      }
      //callie method
      public static int sum(int[] arr)
      {
            int sum=0;
            for(int i:arr)
            {
                  sum+=i;
            }
            return sum;
      }
```

}

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 said to be object oriented if it supports following features.

ex:

class

object

abstraction

encapsulation

inheritance &

polymorphism

class

=======

A class is a collection of data members and behaviours.

Here data members means variables or fields or properties.

Here bahaviour means methods or actions or characteristics.

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

It is a blue print of an object.

We can declare a class as follow.

syntax:

optional

|

modifier class class_name <extends> parent_classname

<implements> interface_name

{

-

- // variables and methods

-

}

A class will accept following modifiers.

ex:

default

public

final

abstract

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

default class

If we declare any class as default then we can access that class within the package.

ex:

class A

{

-

- //logic

-

```
}
```

public class

If we declare any class as public then we can access that class within the package and outside the package.

ex:

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

Q)What is final class?

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

ex:

```
final class A
{
}
class B extends A  --> invalid
```

{

}

Q)What is abstract class?

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

ex:

```
abstract class A
{
    -
}
A a=new A(); --> invalid
```

object

==========

It is a instance of a class.

Here instance means allocating memory for our data members.

It is a outcome of a blue print.

We can create object as follow.

syntax:

Class_Name reference_variable=new Constructor();

ex:

Test t=new Test();

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

Memory space will be created when create an object.

ex:

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

```
        Test t1=new Test();
        Test t2=new Test();
        Test t3=new Test();

        System.out.println(t1.hashCode());
        System.out.println(t2.hashCode());
        System.out.println(t3.hashCode());

        System.out.println(t1); //Test@Hexadecimal
        System.out.println(t2); //Test@Hexadecimal
        System.out.println(t3); //Test@Hexadecimal

    }
}
```

hashCode()

==========

It is a method present in Object class.

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

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

Diagram: java21.1

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.

Q)What is Object class?

It is a parent class for every java class.

It is present in java.lang package.

Object class contains following methods.

ex:

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

getClass()

wait()

notify()

notifyAll()

clone()

hashCode()

toString()

and etc.

ex:

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

Data Hiding

==========

Our internal data should not go out directly.

It means outside perform must not access our data directly.

Using private modifier we can implements data hiding.

The main objective of data hiding is to provide security.

ex:

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

Abstraction

==========

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

Using interfaces and abstract classes we can implements abstraction.

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

The main advantages of abstraction are.

1) It gives security because it wil 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.

Q)Write a java program to display below 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++)
			{
				//prime logic
				while(true)
```

```
        {
            boolean flag=true;

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

    }
    //new line
```

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

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

input:

ABC

output:

ABC

ACB

BAC

BCA

CAB

CBA

ex:

```
class Test
{
```

```
public static void main(String[] args)
{
        String str="ABC";
        //caller method
        permutations(str.toCharArray(),0);
}

//callie method
public static void permutations(char[] arr,int fi)
{
        if(fi==arr.length-1)
        {
                System.out.println(arr);
                return;
        }
        for(int i=fi;i<arr.length;i++)
        {
                swap(arr,i,fi);
                permutations(arr,fi+1);
                swap(arr,i,fi);
        }
}
```

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

Encapsulation

==========

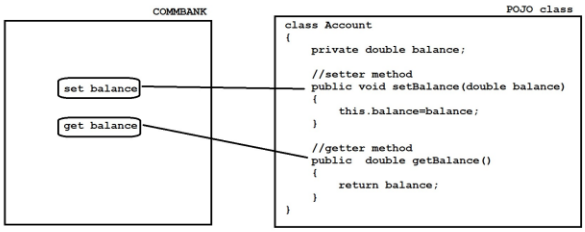
The process of encapsulting or grouping variables and it's assoicate methods in a single entity is called encapsulation.

Diagram: java22.1

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

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

Diagram: java22.2



The main advantages of encapsulation are.

1) It gives security.

2) Enhancement becomes more easy.

3) It provides flexibility to the enduser.

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 will hide the data and Encapsulation will protect the data.

Approach1

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

	//setter methods
	public void setStudId(int studId)
	{
		this.studId=studId;
	}
	public void setStudName(String studName)
	{
		this.studName=studName;
	}
	public void setStudFee(double studFee)
	{
```

```
        this.studFee=studFee;
}

//getter methods
public int getStudId()
{
        return studId;
}
public String getStudName()
{
        return studName;
}
public double getStudFee()
{
        return studFee;
}
public static void main(String[] args)
{
        Student s=new Student();

        s.setStudId(101);
        s.setStudName("Alan");
```

```
            s.setStudFee(1000d);

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

Approach2

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

        //setter methods
        public void setStudId(int studId)
        {
            this.studId=studId;
        }
        public void setStudName(String studName)
```

```
{
        this.studName=studName;
}
public void setStudFee(double studFee)
{
        this.studFee=studFee;
}

//getter methods
public int getStudId()
{
        return studId;
}
public String getStudName()
{
        return studName;
}
public double getStudFee()
{
        return studFee;
}
```

```
}
class Test
{
	public static void main(String[] args)
	{
		Student s=new Student();

		s.setStudId(101);
		s.setStudName("Alan");
		s.setStudFee(1000d);

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

Interview Question

==========

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

POJO class (Plain Old Java Object)

A class is said to be a pojo class if it supports following two properities.

1) All variables must be private.

2) All variables must have setter and getter method.

Java Bean class

A class is said to be a java bean class if it supports following four properities.

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 method.

Note:

Every java bean class is a pojo class but every pojo class is not a java bean class.

Q)What is tightly encapsulated class?

A class is said to be tightly encapsulated class if and only if all the variables of that class must be private.We should not see , either these variables have setter and getter methods or not.

Q)Java program to convert octa to decimal?

```
class Test
{
	public static void main(String[] args)
	{
		int i=45;

		int j=045; //octa value

		System.out.println(i); // 45

		System.out.println(j); // 37
```

```
		//convert octa to decimal
		System.out.println(Integer.parseInt("46",8)); //38
	}
}
```

Is-A relationship

==========

Is-A relationship is also known as inheritance.

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

The main objective of Is-A relationship is to provide reusability.

ex:

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

```
}
class Child extends Parent
{
	public void m2()
	{
		System.out.println("M2-Method");
	}
}
class Test
{
	public static void main(String[] args)
	{
		Parent p=new Parent();
		p.m1();

		Child c=new Child();
		c.m1();
		c.m2();

		Parent p1=new Child();
		p1.m1();
```

```
            //Child c1=new Parent(); // C.T.E parent can't convert to
child
        }
}
```

conclusion

Whatever the properties are there with parent it comes to child but child properties will never go backs to parent.

A parent reference can hold child object but child reference can't hold parent object.

inheritance

==========

Inheritance is a mechanism where we will derive class in the present of existing class.

or

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

Inheritance deals with reusability.

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.

Diagram:

A (Parent/Base/Super class)

|

|

|

B (Child/Derived/sub class)

ex:

```
class A
{
	public void m1()
	{
		System.out.println("M1 method");
	}
}
class B extends A
{
	public void m2()
	{
		System.out.println("M2 method");
	}
}
class Test
{
```

```
    public static void main(String[] args)
    {
        A a=new A();
        a.m1();

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

2) Multilevel inheritance

If we derived a class in the presence of one base class and that class is derived from another base class is called multilevel inheritance.

Diagram:

```
        A
        |
        |
        B
        |
```

|

C

ex:

```
class A
{
	public void m1()
	{
		System.out.println("M1 method");
	}
}
class B extends A
{
	public void m2()
	{
		System.out.println("M2 method");
	}
}
class C extends B
{
	public void m3()
	{
```

```
        System.out.println("M3 method");
    }
}
class Test
{
    public static void main(String[] args)
    {
        A a=new A();
        a.m1();

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

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

3) Multiple inheritance

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

ex:

```
class A {}

class B {}

class C extends A,B  --> invalid

{

}
```

But interface can extends more then one interface so we can achive multiple inheritance concept through interfaces.

ex:

```
interface A {}

interface B {}

interface C extends A,B  --> valid

{}
```

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

ex:

```
class A
{
}
```

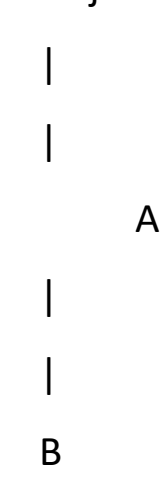
Diag:

If our class extends some other class then it 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.

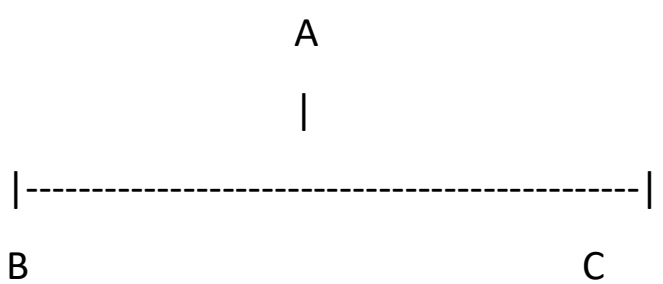
ex:

```
p1.m1()                    p2.m1()

  |-----------------------------|
            |
          c.m1()
```

4) Hierarchical inheritance

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

Diagram:



ex:

```
class A
{
	public void m1()
	{
		System.out.println("M1 method");
	}
}
class B extends A
{
	public void m2()
	{
		System.out.println("M2 method");
	}
```

```
}
class C extends A
{
	public void m3()
	{
		System.out.println("M3 method");
	}
}
class Test
{
	public static void main(String[] args)
	{
		A a=new A();
		a.m1();

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

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

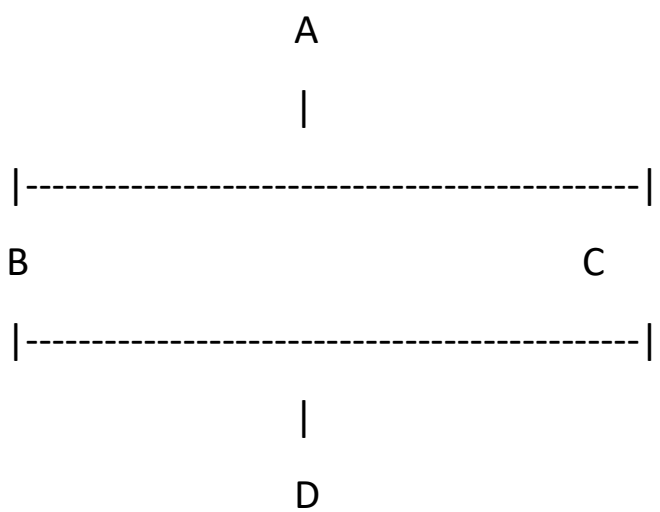
```
        }
}
```

5) Hybrid inheritance

Hybrid inheritance is a combination of more then one inheritance.

Java does not support hybrid inheritance.

Diagram:



Has-A relationship

==========

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

There is no specific keyword to implements Has-A relationship but mostly

we will use new operator.

The main objective of Has-A relationship is to provide reusability.

Has-A relationship will increase dependency between two components.

ex:

```
class Engine
{
    -
    - //engine specific functionality
    -
}
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";
	}
	public int courseDuration()
	{
		return 120;
	}
}
```

```
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

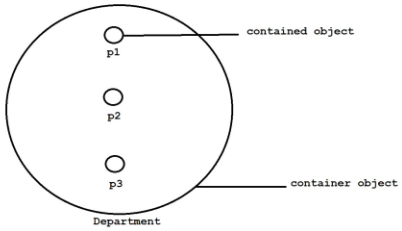
Without existing container object there is no chance of having contained object then the relationship between container and contained object is called composition which is strongly association.

Diagram: java23.1

aggregation

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

Diagram: java23.2



Method overloading

==========

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

All the methods present in class are called overloaded methods.

Method overloading will reduce complexity of the programming.

ex:

```
class MeeSeva
{
	//overloaded methods
	public void search(int voterId)
	{
		System.out.println("Found By VoterId");
	}
	public void search(String houseNo)
	{
		System.out.println("Found By HouseNo");
	}
	public void search(long aadharNo)
	{
```

```
            System.out.println("Found AadharNo");
        }
}
class Test
{
        public static void main(String[] args)
        {
            MeeSeva  ms=new MeeSeva();
            ms.search(1001);
            ms.search("1-4-6/4/1");
            ms.search(1234567l);
        }
}
```

Method overriding

==========

Having same method name with same parameters in a 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");
	}
	public void marry()
	{
		System.out.println("subhalakshmi");
	}
}
class Child extends Parent
{
	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 methods as final then overriding of that methods 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

==========

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

Method overriding	Method hiding

All the methods present in method overriding must be non-static.	All the methods present in method hiding must be static.
Method 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 dynamic polymorhpism, runtime polymorphism or late binding.	IT is also known as static polymorphism, compiletime polymorphism or early binding.

ex:

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

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

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

```
        c.marry();// Rashmika

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

Interview Questions

==========

Q)Can we overload main method in java?

Yes, it is possible to overload main method in java.But JVM always execute main method with String[] parameter only.

ex:

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

```
    }
    public static void main(int[] iargs)
    {
        System.out.println("int-arg method");
    }
}
```

Q)Can we override main method in java?

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

Polymorphism

==========

Polymorphism has taken from Greek word.

Here poly means many and morphism means forms.

The ability to represent in different forms is called polymorphism.

The main objective of polymorphism is to provide flexibility.

Diagram: java24.1

In java, polymorhpism is divided into two types.

1)Compile time polymorphism / static polymorphism / early binding

2)Runtime polymorphism / dynamic polymorphism / late binding

1)Compile time polymorphism

A polymorphism which exhibits at compile time is called compile time polymorphism.

ex:

Method overloading

Method hiding

2)Runtime polymorphism

A polymorphism which exhibits at run time is called run time polymorphism.

ex:

Method overriding

Summary Diagram : java24.2

Constructor

==========

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

ex:

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

Having same name as class name is called constructor.

A constructor does not allow any returntype.

A constructor will following modifiers.

ex:

default

public

private

protected

Constructor will execute when we create an object.

In java , we have two types of constructors.

1)Userdefined constructor

2)Default constructor

1)Userdefined constructor

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

It is classified into two types.

i)Zero-Argument constructor

ii)Parameterized constructor

i)Zero-Argument constructor

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

ex:

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

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

}
```

o/p:

	Main

ex:2

```
class Test
{
	//constructor
	public Test()
	{
		System.out.println("0-arg const");
	}

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

}
```

ex:

```
class Test
{
	//constructor
	private Test()
```

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

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

}
```

o/p:

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

ex:

```
class Test
{
```

```
	//constructor
	protected Test()
	{
		System.out.println("0-arg const");
	}

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

}
```

o/p:

0-arg const

Main-Method

0-arg const

ii)Parameterized constructor

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

ex:

```
class Employee
{
	//instance variable
	//current class variables
	private int empId;
	private String empName;
	private double empSal;

	//parameterized constructor
	public Employee(int empId,String empName,double empSal)
	{
		this.empId=empId;
		this.empName=empName;
		this.empSal=empSal;
	}
```

```
	public void getEmployeeDetails()
	{
		System.out.println("Employee Id :"+empId);
		System.out.println("Employee Name :"+empName);
		System.out.println("Employee Salary :"+empSal);
	}
}
class Test
{
	public static void main(String[] args)
	{
		Employee e=new Employee(101,"Alan Morries",1000d);
		e.getEmployeeDetails();
	}
}
```

2)Default constructor

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

A default constructor is a empty implementation.

To see the default constructor we will use below command.

ex:

```
javap  -c  Test
```

Diagram: java24.3

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.

1) To refer current class variables

2) To refer current class methods

3) To refer current class constructors

1) To refer current class variables

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

	A(int i,int j)
	{
		System.out.println(i+" "+j); // 100  200
		System.out.println(this.i+" "+this.j); //10  20
	}
}
```

```
class Test
{
	public static void main(String[] args)
	{
		A a=new A(100,200);
	}
}
```

2) To refer current class methods

```
class A
{
	public void m1()
	{
		System.out.println("M1-Method");
		this.m2();
	}
	public void m2()
	{
		System.out.println("M2-Method");
	}
}
```

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

3) To refer current class constructors

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

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

super keyword

==========

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

We can utilize super keyword in following ways.

1) To refer super class variables

2) To refer super class methods

3) To refer super class constructors

1) To refer super class variables

```
class A
{
	int i=10;
	int j=20;
}
class B extends A
{
	int i=100;
	int j=200;
	B(int i,int j)
	{
		System.out.println(super.i+" "+super.j); // 10 20
		System.out.println(this.i+" "+this.j); //100 200
		System.out.println(i+" "+j); //1000 2000
	}
```

```
}
class Test
{
	public static void main(String[] args)
	{
		B b=new B(1000,2000);
	}
}
```

2) To refer super class methods

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

```
        super.m1();
        System.out.println("M2-Method");
    }
}
class Test
{
    public static void main(String[] args)
    {
        B b=new B();
        b.m2();
    }
}
```

3) To refer super class constructors

```
class A
{
    A()
    {
        System.out.println("A-constructor");
    }
}
```

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

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.

ex:

```
void m1();
```

It is not possible to create object for interfaces.

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

We can create object for implementation class because it contains method with body.

By default every abstract method is a public and abstract.

Interface contains only constants i.e public static final.

syntax:

```
optionnal
|
modifier interface   <interface_name>
{
        -
        - //abstract methods
```

```
        - //constants
        -
    }
```

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

Diagram: java25.1

ex:1

```
interface A
{
    //abstract method
    public abstract void m1();
}
class B implements A
{
    public void m1()
    {
        System.out.println("M1-Method");
```

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

ex:2

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

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

```
	{
		A a=new A()
		{
			public void m1()
			{
				System.out.println("From M1 Method");
			}
		};
		a.m1();
	}
}
```

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

ex:

```
interface A
{
	//abstract methods
	public abstract void view();
	public void show();
```

```
        abstract void display();
        void see();
}
class B implements A
{
        public void view()
        {
                System.out.println("From view method");
        }
        public void show()
        {
                System.out.println("From show method");
        }
        public void display()
        {
                System.out.println("From display method");
        }
        public void see()
        {
                System.out.println("From see method");
        }
}
```

```
class Test
{
	public static void main(String[] args)
	{
		A a=new B();
		a.view();
		a.show();
		a.display();
		a.see();
	}
}
```

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

But interface can extends more then one interface simultenously.

ex:

--

```
interface A
{
	void m1();
```

```
}
interface B
{
	void m2();
}
interface C extends A,B
{
	void m3();
}
class D implements C
{
	public void m1()
	{
		System.out.println("M1-Method");
	}
	public void m2()
	{
		System.out.println("M2-Method");
	}
	public void m3()
	{
		System.out.println("M3-Method");
```

```
        }
}
class Test
{
        public static void main(String[] args)
        {
                C c=new D();
                c.m1();
                c.m2();
                c.m3();
        }
}
```

A class can implements more then one interface.

ex:

```
interface Father
{
        float HT=6.2f;
        void height();
```

```
}
interface Mother
{
	float HT=5.8f;
	void height();
}
class Child implements Father,Mother
{
	public void height()
	{
		float height=(Father.HT+Mother.HT)/2;
		System.out.println("Child height :"+height);
	}
}
class Test
{
	public static void main(String[] args)
	{
		Child c=new Child();
		c.height();
	}
}
```

Note:

According Java 8 version , Interface is a collection of abstract methods, default methods and static methods.

Q)What is marker interface?

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

In general, empty interface is called marker interface.

ex:

Serializable

Remote

Cloneable

and etc.

Using marker interface we will get some ability to do.

Abstract class

==========

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

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

It is not possible to create object for abstract class.

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

By default every abstract method is a public and abstract.

Abstract class contains only instance variables.

syntax:

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

```
    }
```

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

ex:

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

    //abstract method
    public abstract void getRate();

    //concrete method
    public void calculateBillAmt(int units)
    {
        System.out.println("Total Units :"+ units);
        System.out.println("Total Bill  :"+ rate*units);
    }
}
class DomesticPlan extends Plan
```

```
{
        public void getRate()
        {
                rate=2.5d;
        }
}
class CommercialPlan extends Plan
{
        public void getRate()
        {
                rate=5.0d;
        }
}
class Test
{
        public static void main(String[] args)
        {
                DomesticPlan dp=new DomesticPlan();
                dp.getRate();
                dp.calculateBillAmt(250);

                CommercialPlan cp=new CommercialPlan();
```

```
            cp.getRate();
            cp.calculateBillAmt(250);
        }
}
```

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, static methods and default methods.	It is a collection of abstract methods and concrete methods.
We can achieve multiple inheritance.	We can't achieve multiple inheritance.
It contains only constants.	It contains only instance variables.

If we know specification then we need to use interface.

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

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

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

It does not allow constructor.

It allows constructor.

It does not allow blocks.

It allows blocks.

Singleton class

==========

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

Using a class if we call any method and that methods returns same class object then it is called singleton class.

ex:

LocalDate

LocalTime

Calendar

and etc.

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

ex:

```
class Singleton
{
	static Singleton singleton=null;

	//private constructor
	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());
        }
}
```

API

=====

API stands for Application Programming Interface.

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

API is a collection of packages.

In java, we have three set of API's.

1)Predefined API

Built-in API is called predefined API.

ex:

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

2)Userdefined API

API which is created by the user based on the application requirements is

called userdefined API.

3)Third party API

API which is given by third party vendor is called third party API.

ex:

javazoom API

iText API

and etc

Packages

========

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

Here enum is a special class and annotation is a special interface.

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

Package is also known as folder or a directory.

In java, 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.util

java.io

java.text

java.time

java.sql

and etc.

2)Userdefined packages

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

To declare userdefined package we will 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();

		//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 run above program by using below command.

ex:

```
        java   com.ihub.www.Test
                  |          |
            pkg name  classname
```

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 part of event handling to remove GUI bugs.

Due to powerful features and benefits of inner classes.Our programmers started to use inner classes in regular programming.

Inner classes do not support static members.

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 two .class files will be created i.e Outer.class and Outer$Inner.class.

ex:1

```
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();
	}
}
```

ex:2

```
class Outer
```

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

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

ex:3

```
class Outer
{
	class Inner
	{
		public static void m1()
```

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

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

Note:

C.T.E : Illegal static declaration in 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();
        }
}
```

Enum

=====

Enum is a group of named constants.

Enum concept is introduced in Java 5 version.

Using enum we can created 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

In java, enum will consider as class concept and extends 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);
```

```
        }
}
```

ex:

```
enum Months
{
        JAN,FEB,MAR
}
class Test
{
        public static void main(String[] args)
        {
                Months m=Months.FEB;
                switch(m)
                {
                        case JAN: System.out.println("January"); break;
                        case FEB: System.out.println("February"); break;
                        case MAR: System.out.println("March"); break;
                }
        }
}
```

java.lang.Enum

==========

Power to enum will be inherited from java.lang.Enum class.

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 w1:w)
			{
			System.out.println(w1+"==========="+w1.ordinal());
			}

	}
}
```

When compare to old language enum, java enum is more powerful because in addition to constants we can declare variables, methods and constructors.

ex:

```
enum 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=100;

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

Wrapper classes

==========

The main objective of wrapper classes are.

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

2) To defined several utility methods.

primitive type	wrapper class
byte	Byte
short	Short
int	Integer

long	Long
float	Float
double	Double
boolean	Boolean
char	Character

constructor

Every wrapper class contains following two constructors . One will take corresponding primitive as an argument and another will take corresponding String as an argument.

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

ex:1

```
class Test
{
	public static void main(String[] args)
	{
		Integer i1=new Integer(10);

		Integer i2=new Integer("20");

		System.out.println(i1+" "+i2);
	}
}
```

ex:2

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

```
		Boolean b2=new Boolean("false");

		System.out.println(b1+" "+b2);//true   false
	}
}
```

ex:3

```
class Test
{
	public static void main(String[] args)
	{
		Character c=new Character('a');
		System.out.println(c);
	}
}
```

Utility methods

==========

We have following utility methods.

1) valueOf()

It is similar to constructor.

It will convert primitive type to wrapper object.

ex:

```
class Test
{
	public static void main(String[] args)
	{
		Integer i1=Integer.valueOf(10);

		Integer i2=Integer.valueOf("20");

		System.out.println(i1+" "+i2);
	}
}
```

2) xxxValue()

It is used to convert wrapper object to primitive type.

ex:

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

		byte b=i.byteValue();
		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="56";

		int i=Integer.parseInt(str);

		long l=Long.parseLong(str);

		float f=Float.parseFloat(str);

		System.out.println(i+" "+l+" "+f);//56 56 56.0

	}
}
```

4) toString()

It will convert Wrapper object to String type.

ex:

```
class Test
{
	public static void main(String[] args)
	{
		Integer i1=new Integer(100);

		String str=i1.toString();

		System.out.println(str);//100
	}
}
```

Interview Program

==========

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

input:

	1010

0101

output:

1111

ex:

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

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

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

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

```
        //logic
        int c=a+b;

        //convert decimal to binary
        String result=Integer.toBinaryString(c);

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

Q) Types of objects in java ?

We have two types of objects in java.

1)Immutable object

2)Mutable object

1)Immutable object

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

ex:

String

Wrapper classes

2)Mutable object

After object creation if we perform any changes then all the changes will be reflected to single object only such type of object is called mutable object.

ex:

StringBuffer

StringBuilder

String

=======

A String is a collection of characters which is enclosed in a double quotation.

case1:

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

Diagram: java27.1

case2:

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

==

It is a equality operator which always return boolean value.

It is used for reference 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);
	}
}
```

.equals()

It is a method present String class which always returns boolean value.

It is used for content comparision and it is case sensitive.

ex:

```
class Test
{
	public static void main(String[] args)
	{
		String s1=new String("bhaskar");
```

```
        String s2=new String("bhaskar");
        System.out.println(s1.equals(s2));//true
    }
}
```

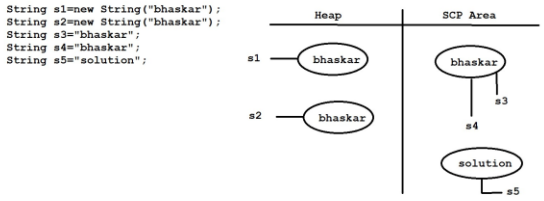
case3:

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

ex:

```
    String s=new String("bhaskar");
```

Diagram: java27.2

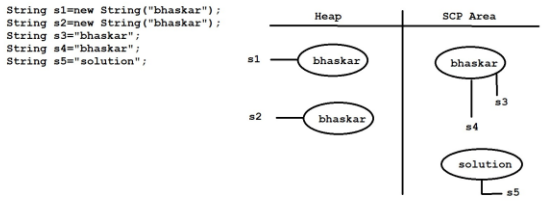


Object creation in SCP area is always optional. First JVM will check is there any object is created with same content or not.If already created then it simply refers to that object.If it is not created then JVM will create a new object.Hence there is no chance of having duplicate objects in SCP area.

Even though SCP objects do not have any reference , garbage collector can't access them.

SCP objects will destroy when JVM shutdowns or terminated.

Diagram: java27.3

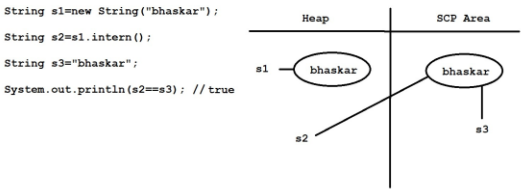


Interning of String object

==========

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

Diagram: java27.4



String important methods

==========

class Test

{

public static void main(String[] args)

{

String str="bhaskar";

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

```
        System.out.println(str.charAt(3)); // s

        System.out.println(str.toUpperCase());// BHASKAR

        System.out.println(str.toLowerCase());// bhaskar

        System.out.println(str.concat("solution")); //bhaskarsolution

        System.out.println(str.replaceAll("a","A")); //bhAskAr

        System.out.println(str.indexOf("a")); //2

        System.out.println(str.lastIndexOf("a")); //5

        System.out.println(str.equals("BHASKAR")); //false

        System.out.println(str.equalsIgnoreCase("BHASKAR")); //
true


    }
}
```

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 string :");
		String str=sc.next();

		System.out.println("Welcome :"+str);

	}
}
```

approach2

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

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

		System.out.println("Enter the string :");
		String str=sc.nextLine();

		System.out.println("Welcome :"+str);

	}
}
```

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

input:

	hello

output:

	olleh

ex:

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

		//convert string to char array
		char[] carr=str.toCharArray();

		//reading reverse
		String rev="";

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

		System.out.println(rev);

	}
}
```

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

input:

racar

output:

It is a palindrome string

ex:

--

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

		//convert string to char array
		char[] carr=str.toCharArray();

		//reading reverse
		String rev="";
```

```
            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

ex:

```
class Test
{
	public static void main(String[] args)
	{
		String str="This Is Java Class";

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

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

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

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(" ");

		String rev="";

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

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

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

input:

google

output:

o g

ex:

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

		String characters="";
		String duplicates="";

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

			if(characters.contains(current))
			{
				if(!duplicates.contains(current))
				{
					duplicates+=current;
					continue;
				}
			}
```

```
                characters+=current;
            }

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

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

input:

google

output:

g o l e

ex:

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

```
{
        String str="google";

        String characters="";
        String duplicates="";

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

                if(characters.contains(current))
                {
                        if(!duplicates.contains(current))
                        {
                                duplicates+=current;
                                continue;
                        }
                }
                characters+=current;
        }

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

```
    }
}
```

Q)Write a java program to display most repeating character in a given string?

input:

ihubtalentingoogletest

output:

t repeating for 4 times

ex:

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

        int maxCount=0;
        char alphabet=' ';
```

```
            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(cnt>maxCount)
                {
                    maxCount=cnt;
                    alphabet=str.charAt(i);
                }
            }
            System.out.println(alphabet+" is repeating for
"+maxCount+" times");
        }
}
```

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

input:

str1 = silent

str2 = listen

output:

It is a Anagram String

ex:

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

		//converting string to char 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;

        for(int i=0;i<carr1.length && i<carr2.length;i++)
        {
                if(carr1[i]!=carr2[i])
                {
                        flag=false;
                        break;
                }
        }

        if(flag==true)
                System.out.println("It is Anagram string");
        else
                System.out.println("It is not Anagram string");
}
```

```
}
```

Q)Write a java program to display the string in a given format?

input:

A1B2C3D4

output:

ABBCCCDDDD

ex:

```
import java.util.Arrays;
class Test
{
	public static void main(String[] args)
	{
		String str="A1B2C3D4";

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

```
                if(Character.isAlphabetic(str.charAt(i)))
                {
                        System.out.print(str.charAt(i));
                }
                else
                {
                int a=Character.getNumericValue(str.charAt(i));

                        for(int j=1;j<a;j++)
                        {
                                System.out.print(str.charAt(i-1));
                        }
                }
            }
        }
}
```

Assignment

==========

Q)Write a java program to display the string in given format?

input:

ABBCCCDDDD

output:

A1B2C3D4

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

input:

str = ihubtalent

cnt = 2

output:

ubtalentih

ex:

```
class Test
{
	public static void main(String[] args)
	{
		String str="ihubtalent";
		int cnt=2;

		String str1=str.substring(0,cnt);
```

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

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

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

input:

str = ihubtalent

cnt = 2

output

ntihubtale

ex:

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

```
            int cnt=2;

            String str1=str.substring(0,str.length()-2);

            String str2=str.substring(str.length()-2,str.length());

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

Q)Write a java program to display the strings starting with Uppercase letter?

input:

This is Java class For freshers

output:

This Java For

ex:

```
class Test
{
	public static void main(String[] args)
	{
		String str="This is Java class For freshers";

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

		String result="";

		//for each loop
		for(String s:sarr)
		{
			if(s.charAt(0)>='A' && s.charAt(0)<='Z')
			{
				result+=s+" ";
			}
		}
		System.out.println(result);
	}
}
```

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.

To overcome this limitation, Sun Micro system introduced StringBufffer.

In StringBuffer is mutable and all the changes will be done in a single object only.

constructors

---

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:

```
capacity = current_capacity+1*2;
```

ex:

--

```
class Test
{
	public static void main(String[] args)
	{
		StringBuffer sb=new StringBuffer();

		System.out.println(sb.capacity()); //16

		sb.append("abcdefghijklmnop");

		System.out.println(sb.capacity()); //16

		sb.append("qr");

		System.out.println(sb.capacity()); // 16+1*2 = 34

	}
}
```

2)StringBuffer sb=new StringBuffer(int initialcapacity);

It will create StringBuffer object with specified initial capacity.

ex:

```
class Test
{
	public static void main(String[] args)
	{
		StringBuffer sb=new StringBuffer(19);

		System.out.println(sb.capacity()); //19


	}
}
```

3)StringBuffer sb=new StringBuffer(String s);

It will create StringBuffer object equivalent to String.

Here capacity will be created with below formulea.

ex:

```
        capacity = s.length()+16;


ex:
--
class Test
{
        public static void main(String[] args)
        {
                StringBuffer sb=new StringBuffer("bhaskar");


                System.out.println(sb.capacity()); //s.length() + 16 =23


        }
}
```

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

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

```
        String rev="";

        StringBuffer sb=new StringBuffer(str);

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

        System.out.println(rev);

    }
}
```

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

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

        String rev="";
```

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

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

            if(str.equals(rev))
                System.out.println("It is  a palindrome string");
            else
                System.out.println("It is not a palindrome string");

        }
}
```

Q) Write a java program to remove duplicate characters from String?

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

```
        StringBuffer sb=new StringBuffer();
        str.chars().distinct().forEach(c->sb.append((char)c));
        System.out.println(sb);
    }
}
```

StringBuilder

==========

StringBuilder is exactly same as StringBuffer with following differences.

StringBuffer	StringBuilder
All the methods present in StringBuffer are synchronized.	No method present in StringBuilder is synchronized.
At a time only one thread is allowed to execute.Hence we can achieve thread safety.	Multiple threads are allowed to execute.Hence we can't achieve thread safety.

Waiting time of a thread will increase	There is no waiting time of a thread
effectively performance is low.	effecitively performance is high.
It is introduced in 1.0v.	It is introduced in 1.5v.

Note:

If our content not change frequently then it is recommanded to use String object.

If our content change frequently where thread safety is required then we need to use StringBuffer.

If our content change frequently where thread safety is not required then we need to use StringBuilder.

StringTokenizer

==========

StringTokenizer is a class which is present in java.util package.

It is used to tokenize the string irrespective of regular expression.

We can create StringTokenizer object as follow.

ex:

StringTokenizer st=new StringTokenizer(String s,RegularExpress reg);

StringTokenizer class contains following five methods.

ex:

public boolean hasMoreTokens()

public String nextToken()

public boolean hasMoreElements()

public Object nextElement()

public int countTokens()

ex:

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

```
            StringTokenizer st=new StringTokenizer("This is java class");

            System.out.println(st.countTokens());//4
        }
}
```

Note:

Default regular expression is space.

ex:2

```
import java.util.StringTokenizer;
class Test
{
        public static void main(String[] args)
        {
            StringTokenizer st=new StringTokenizer("This is java class","
");

            System.out.println(st.countTokens());//4
        }
}
```

ex:3

```
import java.util.StringTokenizer;
class Test
{
	public static void main(String[] args)
	{
		StringTokenizer st=new StringTokenizer("This is java class","
");

		while(st.hasMoreTokens())
		{
			String s=st.nextToken();
			System.out.println(s);
		}
	}
}
```

ex:

```
import java.util.StringTokenizer;
```

```
class Test
{
	public static void main(String[] args)
	{
		StringTokenizer st=new StringTokenizer("This is java class","
");


		while(st.hasMoreElements())
		{
			String s=(String)st.nextElement();
			System.out.println(s);
		}
	}
}
```

ex:

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

```
            StringTokenizer st=new StringTokenizer("9,99,999",",");

            while(st.hasMoreElements())
            {
                  String s=(String)st.nextElement();
                  System.out.println(s);
            }
      }
}
```

Assignment

==========

Q)Write a java program to find out number of uppercase letters ,lowercase letters , digits , words and spaces?

input:

This Is Java Class29

output:

uppercase letters : 4

lowercase letters : 11

digits : 2

```
        words                   : 4
        spaces                  : 3


class Test
{
        public static void main(String[] args)
        {
                String str="This Is Java Class29";

                int upper=0,lower=0,digit=0,space=0,word=1;

                char[] carr=str.toCharArray();

                for(char c:carr)
                {
                        if(c>='A' && c<='Z')
                                upper++;
                        else if(c>='a' && c<='z')
                                lower++;
                        else if(c>='0' && c<='9')
                                digit++;
```

```
                else if(c==' ')
                {
                        space++;
                        word++;
                }
            }
            System.out.println("Uppercase letters : "+upper);
            System.out.println("Lowercase letters : "+lower);
            System.out.println("Digit : "+digit);
            System.out.println("space : "+space);
            System.out.println("word  : "+word);
        }
}
```

java.io package

==========

File

=======

```
File f=new File("abc.txt");
```

File will check is there any abc.txt file already created or not.

If it is available it simply refers to that file.If it is not created then

it won't create any new file.

ex:

```
import java.io.*;
class Test
{
	public static void main(String[] args)
	{
		File f=new File("abc.txt");
		System.out.println(f.exists());//false
	}
}
```

A File object can be used to create a physical file.

ex:

```
import java.io.*;
class Test
{
	public static void main(String[] args)throws IOException
```

```
	{
		File f=new File("abc.txt");
		System.out.println(f.exists());//false

		f.createNewFile();
		System.out.println(f.exists());//true

	}
}
```

A File object can be used to create a directory also.

ex:

```
import java.io.*;
class Test
{
	public static void main(String[] args)throws IOException
	{
		File f=new File("bhaskar123");
		System.out.println(f.exists());//false

		f.mkdir();
```

```
            System.out.println(f.exists());//true


        }
}
```

Q)Write a java program to Create a "cricket123" folder and inside that folder create "abc.txt" file?

```
import java.io.*;
class Test
{
        public static void main(String[] args)throws IOException
        {
                File f1=new File("cricket123");
                f1.mkdir();

                File f2=new File("cricket123","abc.txt");
                f2.createNewFile();

                System.out.println("Please check the location");
```

```
        }
}
```

FileWriter

==========

FileWriter is used to write character oriented data into a file.

constructor

FileWriter fw=new FileWriter(String s);

FileWriter fw=new FileWriter(File f);

ex:

```
        FileWriter fw=new FileWriter("aaa.txt");
        or

        File f=new File("aaa.txt");
        FileWriter fw=new FileWriter(f);
```

If file does not exist then FileWriter will create a physical file.

Methods

---

1)write(int ch)

It will insert single character into a file.

2)write(char[] ch)

It will insert array of characters into a file.

3)write(String s)

It will insert String into a file.

4)flush()

It gives guaranttee that last character of a file is also inserted.

5)close()

It is used to close the FileWriter object.

ex:

```
import java.io.*;
class Test
{
	public static void main(String[] args)throws IOException
	{
		FileWriter fw=new FileWriter("aaa.txt");

		fw.write(98);// b
		fw.write("\n");

		char[] ch={'a','b','c'};
		fw.write(ch);
		fw.write("\n");

		fw.write("bhaskar\nsolution");
		fw.flush();
```

```
            fw.close();
            System.out.println("Please check the location");
        }
}
```

FileReader

==========

It is used to read character oriented data from a file.

constructor

```
FileReader fr=new FileReader(String s);
FileReader fr=new FileReader(File f);
```

ex:

```
        FileReader fr=new FileReader("aaa.txt");
        or
        File f=new File("aaa.txt");
        FileReader fr=new FileReader(f);
```

Methods

---

1)read()

It will read next character from a file and return unicode value.

If next character is not available then it will return -1.

2)read(char[] ch)

It will read collection of characters from a file.

3)close()

It is used to close FileReader object.

ex:1

```
import java.io.*;
class Test
{
	public static void main(String[] args)throws IOException
	{
		FileReader fr=new FileReader("aaa.txt");
```

```
            int i=fr.read();
            while(i!=-1)
            {
                System.out.print((char)i);
                i=fr.read();
            }
            fr.close();
        }
}
```

ex:2

```
import java.io.*;
class Test
{
        public static void main(String[] args)throws IOException
        {
            FileReader fr=new FileReader("aaa.txt");

            char[] carr=new char[255];
```

```
            //load the data from file to char array
            fr.read(carr);

            //reading the data from char array
            for(char c:carr)
            {
                  System.out.print(c);
            }

            fr.close();

      }
}
```

Usage of FileWriter and FileReader is not recommanded to use

==========

While inserting the data by using FileWriter ,we need to insert line seperator(\n) which is very headache for the programmer.

While reading the data by using FileReader object ,we need to read character

by character which is not convenient to the programmer.

To overcome this limitation Sun micro system introduced BufferedWriter and BufferedReader.

BufferedWriter

==========

It is used to insert character oriented data into a file.

constructor

BufferedWriter bw=new BufferedWriter(Writer w);

BufferedWriter bw=new BufferedWriter(Writer w,int buffersize);

BufferedWriter object does not communicate with files directly.

It will take the support of some writer objects.

ex:

```
FileWriter fw=new FileWriter("bbb.txt");
BufferedWriter bw=new BufferedWriter(fw);
```

or

BufferedWriter bw=new BufferedWriter(new FileWriter("bbb.txt"));

Methods

---

1)write(int ch)

It will insert single character into a file.

2)write(char[] ch)

It will insert array of characters into a file.

3)write(String s)

It will insert String into a file.

4)flush()

It gives guaranttee that last character of a file is also inserted.

5)close()

It is used to close the BufferedWriter object.

6)newLine()

It will insert new line into a file.

ex:

```
import java.io.*;
class Test
{
	public static void main(String[] args)throws IOException
	{

	BufferedWriter bw=new BufferedWriter(new
FileWriter("bbb.txt"));
		bw.write(98);//b
		bw.newLine();
```

```
            char[] ch={'a','b','c'};
            bw.write(ch);
            bw.newLine();

            bw.write("bhaskar");
            bw.newLine();

            bw.flush();
            bw.close();
            System.out.println("Please check the location");
        }
}
```

BufferedReader

==========

It is enhanced reader to read character oriented data from a file.

constructor

BufferedReader br=new BufferedReader(Reader r);

BufferedReader br=new BufferedReader(Reader r,int buffersize);

BufferedReader object can't communicate with files directly.IT will take support of some reader objects.

ex:

```
FileReader fr=new FileReader("bbb.txt");
BufferedReader br=new BufferedReader(fr);
```

or

```
BufferedReader br=new BufferedReader(new FileReader("bbb.txt"));
```

The main advantage of BufferedReader over FileReader is we can read character line by line instead of character by character.

methods

---

1)read()

It will read next character from a file and return unicode value.

If next character is not available then it will return -1.

2)read(char[] ch)

It will read collection of characters from a file.

3)close()

It is used to close BufferedReader object.

4)nextLine()

It is used to read next line from the file.If next line is not available then it will return null.

ex:

```
import java.io.*;
class Test
{
	public static void main(String[] args)throws IOException
	{
```

```
            BufferedReader br=new BufferedReader(new
FileReader("bbb.txt"));

            String line=br.readLine();
            while(line!=null)
            {
                  System.out.println(line);
                  line=br.readLine();
            }

            br.close();

      }
}
```

PrintWriter

==========

It is enhanced write to write character oriented data into a file.

constructor

PrintWriter pw=new PrintWriter(String s);

PrintWriter pw=new PrintWriter(File f);

PrintWriter pw=new PrintWriter(Writer w);

PrintWriter can communicate with files directly and it will take the support of some writer objects.

ex:

```
PrintWriter pw=new PrintWriter("ccc.txt");

or

PrintWriter pw=new PrintWriter(new File("ccc.txt"));

or

PrintWriter pw=new PrintWriter(new FileWriter("ccc.txt"));
```

The main advantage of PrintWriter over FileWriter and BufferedWriter is we can insert any type of data.

Assume if we want insert primitive values then PrintWriter is best choice.

methods

---

write(int ch)

write(char[] ch)

write(String s)

flush()

close()

writeln(int i)

writeln(float f)

writeln(double d)

writeln(String s)

writeln(char c)

writeln(boolean b)

write(int i)

write(float f)

write(double d)

write(String s)

write(char c)

write(boolean b)

ex:

```
import java.io.*;
class Test
{
	public static void main(String[] args)throws IOException
	{
		PrintWriter pw=new PrintWriter("ccc.txt");

		pw.write(100);// d
		pw.println(100);// 100
		pw.print('a');
		pw.println(true);
		pw.println("hi");
		pw.println(10.5d);
```

```
            pw.flush();
            pw.close();
            System.out.println("Please check the location");
        }
}
```

Various ways to provide input values from 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);
        }
}
```

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 occur due to syntax error.

ex:

ArithmeticException

IllegalArgumentException

FileNotFoundException

Error

Error is a problem for which we can't provide solution programmatically.

Erorr will occur due to lack of system resources.

ex:

OutOfMemory

StackOverFlow

LinkageError

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.

Whenever exception raised in our program then we must and should handle the 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

==========

It is a unwanted, unexpected event which disturbs normal flow of our program.

Exception always raised at runtime so they are also known as runtime events.

The main objective of exception handling is to provide graceful termination.

In java, exceptions are divided into two types.

1)Predefined exceptions

2)Userdefined exceptions

1)Predefined exceptions

Built-In exceptions are called predefined exceptions.

It is divided into two types.

i)Checked exceptions

ii)Unchecked exceptions

i)Checked exceptions

Exceptions which are checked by the compiler at the time of compilation is called checked exceptions.

ex:

InterruptedException

FileNotFoundException

EOFException

and etc.

ii)Unchecked exceptions

Exceptions which are checked by the JVM at the time of runtime is called runtime exceptions.

ex:

ArithmeticException

IllegalArgumentException

ClassCastException

and etc

Diagram: java31.1

If any checked exception raised in our program then we must and should handle that exception by using try and catch block.

try block

==========

It is a block which contains risky code.

It is always associate with catch block.

It is used to throw the exception to catch block.

If any exception raised in try block then try block won't be executed.

If any exception raised in the middel of the program then rest of the code won't be executed.

catch Block

==========

It is a block which contains error handling code.

A catch block always associate with try block.

A catch block is used to catch the exception which is thrown by try block.

A catch block will take exception class 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("catch-block");
		}
	}
}
```

o/p:

try-block

ex:

```
class Test
{
	public static void main(String[] args)
	{
		try
		{
			System.out.println(10/0);
```

```
            }
            catch(Exception e)
            {
                System.out.println("catch-block");
            }
        }
}
```

o/p:

catch-block

ex:

```
class Test
{
    public static void main(String[] args)
    {
        try
        {
            System.out.println("stmt1");
            System.out.println(10/0);
            System.out.println("stmt2");
        }
```

```
            catch(ArithmeticException ae)
            {
                System.out.println("catch-block");
            }
        }
}
```

o/p:

stmt1

catch-block

A try with multiple catch blocks

==========

A try block can have multiple catch blocks.

If a 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 ways to provide 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 give 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 we get any exception then try block won't be executed.

It is never recommanded to maintain cleanup code in catch block because if there is not exception in try block then catch block won't be executed.

We need a place where we can maintain cleanup code and it should execute irrespective of exception raised or not.Such block is called finally block.

syntax:

```
try
{
        -
        - //risky code
        -
}
catch(ArithmeticException ae)
{
        -
        - //error handling code
```

```
        -
    }
    finally
    {
        -
        - //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
```

ex:

```
import java.io.*;
class Test
{
```

```
public static void main(String[] args)
{
	FileWriter fw=null;

	try
	{
		fw=new FileWriter("abc.txt");
		fw.write(98);//b
		fw.write("\n");
		char[] ch={'a','b','c'};
		fw.write(ch);
		fw.write("\n");
		fw.write("bhaskar");
		fw.flush();
		System.out.println("Please check the location");
	}
	catch (IOException ioe)
	{
		ioe.printStackTrace();
	}
	finally
	{
```

```
                try
                {
                        fw.close();
                }
                catch (IOException ioe)
                {
                        ioe.printStackTrace();
                }
        }

    }
}
```

Q)How can we handle multiple exceptions in a single catch block?

ex:

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

```
	{
		try
		{
			System.out.println(10/0);
		}
		catch(ArithmeticException | ClassCastException |
IllegalArgumentException  e)
		{
			e.printStackTrace();
		}

	}
}
```

Q)What is the difference between final, finally and finalized method?

final

A 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 should execute irrespective of exception raised or not.

finalized

It is a method called by garbage collector just before destroying an object for cleanup activity.

throw statement

==========

Sometimes we will create exception object explicitly and handover to JVM manually by using throw statement.

ex:

```
throw new ArithemticException("Don't divided by zero");
```

ex:

```
class Test
{
	public static void main(String[] args)
	{
		throw new ArithmeticException("Don't divide by zeroooo");
	}
}
```

throws statement

==========

If any checked exception raised in our program then we must and should handle that exception by using try and catch block or by using throws statement.

ex:1

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

```
            try
            {
                  Thread.sleep(5000);
                  System.out.println("Welcome to Java Class");
            }
            catch (InterruptedException ie)
            {
                  ie.printStackTrace();
            }

      }
}
```

ex:2

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

                  Thread.sleep(5000);
                  System.out.println("Welcome to Java Class");
```

```
        }
}
```

2)Userdefined Exceptions

==========

Exceptions which are created by the user based on the application requirements are called userdefined exceptions or customized exceptions.

ex:

NotPracticingException

NoInterestInCourseException

NoBalanceInAccountException

TooYoungException

TooOldException

and etc.

ex:

```
import java.util.Scanner;
class TooYoungException extends RuntimeException
{
        TooYoungException(String s)
        {
```

```
            super(s);
        }
}
class TooOldException extends RuntimeException
{
        TooOldException(String s)
        {
            super(s);
        }
}
class Test
{
        public static void main(String[] args)
        {
            Scanner sc=new Scanner(System.in);
            System.out.println("Enter the age :");
            int age=sc.nextInt();

            if(age>=18)
                throw new TooOldException("U r eligible to vote");
            else
            throw new TooYoungException("U r not eligible to vote");
```

```
        }
}
```

Generics

==========

Arrays are typesafe. It means we can provide guarantee that what type of elements are present in array.

If requirement is there to store string values then it is recommanded to use String[] array.

ex:

```
        String[] sarr=new String[10];
        sarr[0]="hi";
        sarr[1]="hello";
        sarr[2]="bye";
        sarr[3]=10;  ---> invalid
```

At the time of retrieving the data from array, we don't need to perform any typecasting.

ex:

```
        String[] sarr=new String[10];
```

```
sarr[0]="hi";
sarr[1]="hello";
sarr[2]="bye";
-
-
String val=sarr[0];
```

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 because we won't get any compile time error or runtime error but some times our program get failure.

ex:

```
ArrayList al=new ArrayList();
al.add("hi");
al.add("hello");
al.add("bye");
al.add(10);
```

At the time of retrieving the data from collections, compulsary we need to perform typecasting.

ex:

```
ArrayList al=new ArrayList();
al.add("hi");
al.add("hello");
al.add("bye");
al.add(10);
-
-
String val=(String)al.get(0);
```

To overcome above limitations Sun Micro system introduced Generics concept in 1.5 version.

The main objective of generics are.

1) To make Collections as typesafe.

2) To avoid typecasting problem.

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.
Arrays are fixed in size.	Collections are growable in nature.
Performance point of view arrays are recommanded to use.	Memory point of view Collections are recommanded to use.
It can hold primitive types and object types.	It can hold only object types.
Arrays are not implemented based on data structure concept.Hence we can't expect any readymade method.For every logic we need to write the code explictly.	Collections are implemented based on data structure concept.Hence we can expect readymade methods.

java.util package

==========

Collection

==========

A Collection is an interface which is present in java.util package.

It is a root interface for entire Collection Framework.

If we want to represent group of individual objects in a single entity then we need to use Collection interface.

Collection interface contains following methods.

ex:

```
public abstract boolean add(E);
public abstract boolean remove(java.lang.Object);
public abstract boolean containsAll(java.util.Collection<?>);
public abstract boolean addAll(java.util.Collection<? extends E>);
public abstract boolean removeAll(java.util.Collection<?>);
public boolean removeIf(java.util.function.Predicate<? super E>);
```

public abstract boolean retainAll(java.util.Collection<?>);

public abstract void clear();

public abstract boolean equals(java.lang.Object);

public abstract int hashCode();

public java.util.Spliterator<E> spliterator();

public java.util.stream.Stream<E> stream();

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: java32.1

ArrayList

The underlying data structure is resizable array or growable array.

Duplicate objects are allowed.

Order is preserved.

Hetrogeneous objects are allowed.

Null insertion is possible.

It implements Serializable, Cloneable and RandomAccess interface.

If our frequent operation is a retrieval operation then ArrayList is a best choice.

ex:1

```
import java.util.*;
class Test
{
	public static void main(String[] args)
	{
		ArrayList al=new ArrayList();
		al.add("one");
```

```
            al.add("two");
            al.add("three");
            System.out.println(al);//[one,two,three]
            al.add("one");
            System.out.println(al);//[one,two,three,one]
            al.add(10);
            System.out.println(al); //[one,two,three,one,10]
            al.add(null);
            System.out.println(al);//[one,two,three,one,10,null]
        }
}
```

ex:

```
import java.util.*;
class Test
{
        public static void main(String[] args)
        {
            ArrayList<String> al=new ArrayList<String>();
            al.add("one");
            al.add("two");
```

```
            al.add("three");
            System.out.println(al);//[one,two,three]
            al.add("one");
            System.out.println(al);//[one,two,three,one]
            al.add(null);
            System.out.println(al);//[one,two,three,one,null]
        }
}
```

ex:

```
import java.util.*;
class Test
{
        public static void main(String[] args)
        {
            ArrayList<String> al=new ArrayList<String>();
            al.add(new String("one"));
            al.add(new String("two"));
            al.add(new String("three"));
            System.out.println(al);//[one,two,three]
```

```
        }
}
```

ex:

```
import java.util.*;
class Test
{
        public static void main(String[] args)
        {
                List<String> l=new ArrayList<String>();
                l.add("one");
                l.add("two");
                l.add("three");
                System.out.println(l);//[one,two,three]

                for(int i=0;i<l.size();i++)
                {
                        System.out.println(l.get(i));
                }
```

```
	}
}

ex:
----
import java.util.*;
class Test
{
	public static void main(String[] args)
	{
		List<String> l=new ArrayList<String>();
		l.add("one");
		l.add("two");
		l.add("three");
		System.out.println(l);//[one,two,three]

		System.out.println(l.isEmpty()); //false

		l.remove("two");
		System.out.println(l);//[one,three]

		l.clear();
```

```
            System.out.println(l);//[]
        }
}
```

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 data structure is doubly LinkedList.

Duplicate objects are allowed.

Order is preserved.

Hetrogeneous objects are allowed.

Null insertion is possible.

It implements Serializable, Cloneable and Deque interface.

If our frequent operation is adding and removal in the middle then LinkedList is a best choice.

LinkedList contains following methods.

ex:

addFirst()

addLast()

getFirst()

getLast()

removeFirst()

removeLast()

and etc.

ex:

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

```
            LinkedList<String> ll=new LinkedList<String>();
            ll.add("one");
            ll.add("two");
            ll.add("three");
            System.out.println(ll);//[one,two,three]
            ll.add("one");
            System.out.println(ll);//[one,two,three,one]
            ll.add(null);
            System.out.println(ll);//[one,two,three,null]

        }
}
```

ex:

```
import java.util.*;
class Test
{
        public static void main(String[] args)
        {
            LinkedList<String> ll=new LinkedList<String>();
            ll.add("one");
```

```
        ll.add("two");
        ll.add("three");
        System.out.println(ll);//[one,two,three]

        ll.addFirst("gogo");
        ll.addLast("jojo");
        System.out.println(ll);//[gogo, one, two, three, jojo]

        System.out.println(ll.getFirst()); //gogo
        System.out.println(ll.getLast()); //jojo

        ll.removeFirst();
        ll.removeLast();
        System.out.println(ll);//[one,two,three]
    }
}
```

Vector

=======

The underlying data structure is resizable array or growable array.

Duplicate objects are allowed.

Insertion order is preserved.

Hetrogeneous objects are allowed.

Null insertion is possible.

It implements Serializable, Cloneable and RandomAccess interface.

All the methods present in Vector are synchronized.Hence we can achieve thread safety.

Vector contains following methods.

ex:

addElement()

removeElementAt()

removeAllElements()

firstElement()

lastElement()

and etc.

ex:

```
import java.util.*;
class Test
{
	public static void main(String[] args)
	{
		Vector<Integer> v=new Vector<Integer>();

		System.out.println(v.capacity());//10

		for(int i=1;i<=10;i++)
		{
			v.addElement(i);
		}
		System.out.println(v);//[1,2,3,4,5,6,7,8,9,10]

		System.out.println(v.firstElement()); //1
		System.out.println(v.lastElement());  //10

		v.removeElementAt(5);
		System.out.println(v);//[1, 2, 3, 4, 5, 7, 8, 9, 10]
```

```
            v.removeAllElements();
            System.out.println(v);//[]

        }
}
```

ex:

```
import java.util.*;
class Test
{
        public static void main(String[] args)
        {
            Vector<Integer> v=new Vector<Integer>();

            System.out.println(v.capacity());//10

            for(int i=1;i<=10;i++)
            {
                v.add(i);
            }
            System.out.println(v);//[1,2,3,4,5,6,7,8,9,10]
```

```
            System.out.println(v.get(0)); //1
            System.out.println(v.get(v.size()-1));  //10

            v.remove(5);
            System.out.println(v);//[1, 2, 3, 4, 5, 7, 8, 9, 10]

            v.clear();
            System.out.println(v);//[]

        }
}
```

Stack

=======

It is a child class of Vector class.

If we depends upon LIFO(Last In First Out) order then we need to use Stack.

constructor

```
Stack  s =new Stack();
```

Methods

---

1) push(Object o)

It is used to add the elment to stack.

2) pop()

It is used to remove the element from stack.

3) peek()

It is used to return toppest element from the stack.

4) isEmpty()

It is used to check stack is empty or not.

5) Search(Object o)

It will return offset value if element is found otherwise it will return -1.

ex:

```
import java.util.*;
class Test
{
	public static void main(String[] args)
	{
		Stack<String> s=new Stack<String>();
		s.push("A");
		s.push("B");
		s.push("C");
		System.out.println(s);//[A,B,C]

		s.pop();
		System.out.println(s);//[A,B]

		System.out.println(s.peek());//B
```

```
            System.out.println(s.isEmpty());//false

            System.out.println(s.search("Z"));//-1

            System.out.println(s.search("A"));//2

        }
}
```

Diagram: java33.1

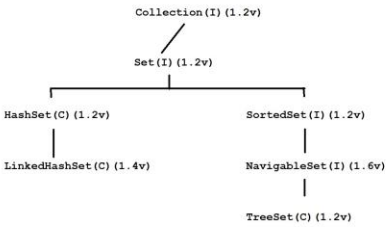
Set

=====

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.

Diagram: java33.2



HashSet

========

The underlying data structure is Hashtable.

Duplicate objects are not allowed.

Insertion order is not preserved because it will hash code of an object.

Hetrogeneous objects are allowed.

Null insertion is possible.

It implements Serializable and Cloneable interface.

ex:

```
import java.util.*;
class Test
{
	public static void main(String[] args)
	{
		HashSet hs=new HashSet();
		hs.add("one");
		hs.add("nine");
		hs.add("five");
		System.out.println(hs);//[nine, one, five]
		hs.add("one");
		System.out.println(hs);//[nine, one, five]
		hs.add(10);
		System.out.println(hs);//[nine, one, 10, five]
		hs.add(null);
		System.out.println(hs);//[null, nine, one, 10, five]
	}
}
```

ex:

```
import java.util.*;
class Test
{
	public static void main(String[] args)
	{
		HashSet<String> hs=new HashSet<String>();
		hs.add("one");
		hs.add("nine");
		hs.add("five");
		System.out.println(hs);//[nine, one, five]
		hs.add("one");
		System.out.println(hs);//[nine, one, five]
		hs.add(null);
		System.out.println(hs);//[null, nine, one, five]
	}
}
```

LinkedHashSet

==========

It is a child class of HashSet class.

LinkedHashSet is exactly same as HashSet class with following differences.

HashSet	LinkedHashSet
The underlying data structure is Hashtable.	The underlying data structure is Hashtable and LinkedList.
Insertion order is not preserved.	Insertion order is preserved.
It is introduced in 1.2v.	It is introduced in 1.4v.

ex:

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

```
        LinkedHashSet<String> lhs=new LinkedHashSet<String>();
        lhs.add("one");
        lhs.add("nine");
        lhs.add("five");
        System.out.println(lhs);//[one,nine,five]
        lhs.add("one");
        System.out.println(lhs);//[one,nine,five]
        lhs.add(null);
        System.out.println(lhs);//[one,nine,five,null]
    }
}
```

TreeSet

========

The underlying data structure is BALANCED TREE.

Duplicate objects are not allowed.

Insertion order is not preserved because it will take sorting order of an object.

Hetrogeneous objects are not allowed otherwise we will get ClassCastException.

Null insertion is not possible otherwise we will get NullPointerException.

It implements NavigableSet, Serializable and Cloneable interface.

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);
		ts.add(3);
		System.out.println(ts);//[1,3,5,7,10]
		ts.add(1);
		System.out.println(ts);//[1, 3, 5, 7, 10]
```

```
            //ts.add(null);
            //System.out.println(ts);//R.E NullPointerException

            //ts.add("hi");
            //System.out.println(ts);// R.E .ClassCastException
        }
}
```

Q)What is the difference between Comparable and Comparator interface?

Comparable

It is an interface which is present in java.lang package.

It contains only one method i.e compareTo() method.

ex:

```
obj1.compareTo(obj2)
```

It will return -ve if obj1 comes before obj2.

It will return +ve if obj1 comes after obj2.

It will return 0 if both objects are same.

ex:

```
class Test
{
	public static void main(String[] args)
	{
		System.out.println("A".compareTo("Z")); //-25

		System.out.println("Z".compareTo("A")); // 25

		System.out.println("K".compareTo("K")); // 0
	}
}
```

Note:

If we depends upon default natural sorting order then we need to use Comparable interface.

Comparator

It is an interface which is present in java.util package.

It contains following two methods i.e compare() and equals() method.

ex:

public int compare(Object obj1,Object obj2)

It will return +ve if obj1 comes before obj2.

It will return -ve if obj1 comes after obj2.

It will return 0 if both objects are same.

public boolean equals(Object obj)

Implementation of compare() is mandatory.

Implementation of equals() method is optional because it will be available in our program through inheritance.

Note:

If we depends upon customized sorting order then we need to use Comparator interface.

ex:

```
import java.util.*;
class Test
{
	public static void main(String[] args)
	{
		TreeSet<Integer> ts=new TreeSet<Integer>(new MyComparator());
		ts.add(10);
		ts.add(5);
		ts.add(1);
		ts.add(3);
		System.out.println(ts);//[10,5,3,1]
```

```
        }
}
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:

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

```
{
	public static void main(String[] args)
	{
		TreeSet<Integer> ts=new TreeSet<Integer>(new
MyComparator());
		ts.add(10);
		ts.add(5);
		ts.add(1);
		ts.add(3);
		System.out.println(ts);//[1,3,5,10]
	}
}
class MyComparator implements Comparator
{
	public int compare(Object obj1,Object obj2)
	{
		Integer i1=(Integer)obj1;
		Integer i2=(Integer)obj2;
		if(i1<i2)
			return -1;
		else if(i1>i2)
			return 1;
```

```
                else
                        return 0;
        }
}
```

Diagram: java33.3

classname	datastructure	duplicates	insertion order	hetrogeneous obj	null insertion
HashSet	Hashtable	Not allowed	Not preserved	Allowed	possible
LinkedHashSet	Hashtable LinkedList	Not allowed	Preserved	Allowed	possible
TreeSet	Balanced Tree	Not allowed	Sorting order	Not Allowed	Not possible

Map

=====

It is not a child interface of Collection interface.

If we want to represent group of individual objects in key,value pair then we need to use Map interface.

A key can't be duplicated but value can be duplicated.

Both key and value must be objects.

Each key and value pair is known as one entry.

Diagram: java34.1

HashMap

=======

The underlying data structure is Hashtable.

Duplicate keys are not allowed but value can be duplicated.

Insertion order is not preserved because it will take hash code of the key.

Hetrogeneous 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("nine","jose");
		hm.put("six","nelson");
		hm.put("four","Alex");
		System.out.println(hm);//{nine=jose, six=nelson, four=Alex, one=raja}
		hm.put("one","rani");
		System.out.println(hm);//{nine=jose, six=nelson, four=Alex, one=rani}
		hm.put(null,null);
		System.out.println(hm);//{null=null, nine=jose, six=nelson, four=Alex, one=rani}
		hm.put(1,100);
		System.out.println(hm);//{null=null, nine=jose, 1=100, six=nelson, four=Alex, one=rani}
	}
```

```
}

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("nine","jose");
		hm.put("six","nelson");
		hm.put("four","Alex");

		Set s=hm.keySet();
		System.out.println(s);//[nine, six, four, one]

		Collection c=hm.values();
		System.out.println(c);//[jose, nelson, Alex, raja]

		Set s1=hm.entrySet();
```

```
        System.out.println(s1);//[nine=jose, six=nelson, four=Alex,
one=raja]


    }
}
```

LinkedHashMap

==========

It is a child class of HashMap class.

LinkedHashMap is exactly same as HashMap class with following differences.

HashMap	LinkedHashMap
The underlying data structure is Hashtable.	The underlying data structure is Hashtable and LinkedList.
Insertion order is not preserved.	Insertion order is preserved.
It is introduced in 1.2v.	It is introduced in 1.4v.

ex:

```
import java.util.*;
class Test
{
	public static void main(String[] args)
	{
		LinkedHashMap<String,String> lhm=new
LinkedHashMap<String,String>();
		lhm.put("one","raja");
		lhm.put("nine","jose");
		lhm.put("six","nelson");
		lhm.put("four","Alex");

	System.out.println(lhm);//{one=raja,nine=jose,six=nelson,four=Al
ex}
	}
}
```

TreeMap

========

The underlying data structure is RED BLACK TREE.

Duplicate keys are not allowed but values can be duplicated.

Insertion order is not preserved because it will take sorting order of the key.

If we depend upon default natural sorting order then keys can be homogeneous and Comparable.

If we depend upon customized sorting order then keys can 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(3,"three");
            System.out.println(tm);//{1=one,3=three,5=five,10=ten}
            tm.put(4,null);
            System.out.println(tm);//{1=one, 3=three, 4=null, 5=five,
10=ten}
            tm.put(null,"six");
            System.out.println(tm);//R.E NullPointerException
        }
}
```

Hashtable

==========

The underlying data structure is Hashtable.

Duplicate keys are not allowed but values can be duplicated.

Insertion order is not preserved because it will descending order of key.

Hetrogeneous objects are allowed for both key and value.

Key and value both 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(3,"three");
		System.out.println(ht);//{10=ten, 5=five, 3=three, 1=one}

		ht.put(1,"raja");
		System.out.println(ht);//{10=ten, 5=five, 3=three, 1=raja}

		//ht.put(4,null);
		//System.out.println(ht);//R.E NullPointerException
```

```
            ht.put(null,"four");
            System.out.println(ht);//R.E NullPointerException

        }
}
```

Types of Cursors

==========

Cursor is used to read objects one by one from Collections.

We have three types of cursors in java.

1) Enumeration

2) Iterator

3) ListIterator

1) Enumeration

It is used to read objects one by one from legal Collection objects.

We can create Enumeration object as follow.

ex:

```
Enumeration e=v.elements();
```

Enumeration interface contains following two methods.

ex:

```
public boolean void hasMoreElements()
public Object next nextElement()
```

ex:

```
import java.util.*;
class Test
{
    public static void main(String[] args)
    {
        Vector v=new Vector();
        for(int i=1;i<=10;i++)
        {
            v.add(i);
        }
```

```
            System.out.println(v);//[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

            Enumeration e=v.elements();
            while(e.hasMoreElements())
            {
                  Integer i=(Integer)e.nextElement();
                  System.out.println(i);
            }
      }
}
```

Limitations with Enumeration

Using Enumeration we can read objects one by one , only from legacy Collection objects.Hence it is not a universal cursor.

Using Enumeration interface 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 object one by one from any Collection objects.Hence it is a universal cursor.

Using iterator interface we can perform read and remove operations.

We can create Iterator object as follow.

ex:

```
Iterator itr=al.iterator();
```

Iterator interface contains following three methods.

ex:

```
public boolean hasNext()
public Object next()
public void remove()
```

ex:

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

```
    {
        ArrayList al=new ArrayList();
        for(int i=1;i<=10;i++)
        {
            al.add(i);
        }
        System.out.println(al);//[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

        Iterator itr=al.iterator();
        while(itr.hasNext())
        {
            Integer i=(Integer)itr.next();
            if(i%2==0)
                itr.remove();
            else
                System.out.println(i);
        }

        System.out.println(al);//1 3 5 7 9
    }
}
```

Limitations with Iterator

> Using Enumeration and Iterator interface we can read objects only in forward direction but not in backward direction.Hence they are not bi-directional cursors.

> Using Iterator interface we can perform read and remove operation but not adding and replacement of new objects.

>To overcome this limitation sun micro system introduced ListIterator.

3)ListIterator

ListIterator interface is used to read objects one by one ,only from List Collection objects.

Using ListIterator object we can perform read, remove , adding and replacement of new objects.

We can create ListIterator object as follow.

ex:

```
ListIterator litr=al.listIterator();
```

ListIterator interface contains following 9 methods.

ex:

```
public boolean hasNext()
public Object next()
public boolean hasPrevious()
public Object previous()
public void remove();
public void nextIndex()
public void previousIndex()
public void set(Object o)
public void add(Object o)
```

ex:

```
import java.util.*;
class Test
{
    public static void main(String[] args)
    {
        ArrayList al=new ArrayList();
        al.add("bala");
        al.add("nag");
```

```
            al.add("chiru");
            al.add("venki");
            System.out.println(al);//[bala,nag,chiru,venki]

            ListIterator litr=al.listIterator();
            while(litr.hasNext())
            {
                  String s=(String)litr.next();
                  System.out.println(s);
            }
      }
}
```

ex:

```
import java.util.*;
class Test
{
      public static void main(String[] args)
      {
            ArrayList al=new ArrayList();
            al.add("bala");
```

```
            al.add("nag");
            al.add("chiru");
            al.add("venki");
            System.out.println(al);//[bala,nag,chiru,venki]

            ListIterator litr=al.listIterator();
            while(litr.hasNext())
            {
                  String s=(String)litr.next();
                  if(s.equals("bala"))
                  {
                        litr.remove();
                  }
            }

            System.out.println(al);//[nag,chiru,venki]
      }
}
```

ex:

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

```
class Test
{
	public static void main(String[] args)
	{
		ArrayList al=new ArrayList();
		al.add("bala");
		al.add("nag");
		al.add("chiru");
		al.add("venki");
		System.out.println(al);//[bala,nag,chiru,venki]

		ListIterator litr=al.listIterator();
		while(litr.hasNext())
		{
			String s=(String)litr.next();
			if(s.equals("nag"))
			{
				litr.add("chetaniya");
			}
		}

		System.out.println(al);//[bala, nag, chetaniya, chiru, venki]
```

```
        }
}
```

ex:

```
import java.util.*;
class Test
{
        public static void main(String[] args)
        {
                ArrayList al=new ArrayList();
                al.add("bala");
                al.add("nag");
                al.add("chiru");
                al.add("venki");
                System.out.println(al);//[bala,nag,chiru,venki]

                ListIterator litr=al.listIterator();
                while(litr.hasNext())
                {
                        String s=(String)litr.next();
```

```
                if(s.equals("nag"))
                {
                        litr.set("chetaniya");
                }
        }

        System.out.println(al);//[bala, chetaniya, chiru, venki]
    }
}
```

Interview Questions

==========

Q)Write a java program to find out number of words present in a string?

input:

This is is java java class

output:

This=1,is=2,java=2,class=1

ex:

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

		LinkedHashMap<String,Integer> lhm=new
LinkedHashMap<String,Integer>();

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

		//for each loop
		for(String s:sarr)
		{
			if(lhm.get(s)!=null)
			{
				lhm.put(s,lhm.get(s)+1);
			}
			else
```

```
                {
                        lhm.put(s,1);
                }
        }

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

Q)Write a java program to find out number of alphabets present in a string?

input:

java

output:

j=1,a=2,v=1

ex:

```
import java.util.*;
class Test
{
```

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

		LinkedHashMap<Character,Integer> lhm=new
LinkedHashMap<Character,Integer>();

		char[] carr=str.toCharArray();

		//for each loop
		for(char c:carr)
		{
			if(lhm.get(c)!=null)
			{
				lhm.put(c,lhm.get(c)+1);
			}
			else
			{
				lhm.put(c,1);
			}
		}
```

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

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

input:

{([])}

output:

It is a balanced string

ex:

```
import java.util.*;
class Test
{
        public static void main(String[] args)
        {
                String str="{([])}";
```

```
        //caller method
        if(isBalanced(str))
                System.out.println("It is balanced string");
        else
                System.out.println("It is not balanced string");
}
//callie method
public static boolean isBalanced(String str)
{
        Stack<Character> stack=new Stack<Character>();

        char[] carr=str.toCharArray();

        //for each loop
        for(char c:carr)  // ] ) }
        {
                if(c=='{' || c=='(' || c=='['
                {
                        stack.push(c);
                }
                else if(c=='}' && !stack.isEmpty() && stack.peek()=='{')
                {
```

```
                stack.pop();
            }
            else if(c==')' && !stack.isEmpty() && stack.peek()=='(')
            {
                stack.pop();
            }
            else if(c==']' && !stack.isEmpty() && stack.peek()=='[')
            {
                stack.pop();
            }
            else
            {
                return false;
            }

        }

        return stack.isEmpty();
    }
}
```

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

input:

4 5 7 1 3 2 2 5 7 9 1

output:

distinct elements : 4 5 7 1 3 2 9

duplicate elements : 2 5 7 1

ex:

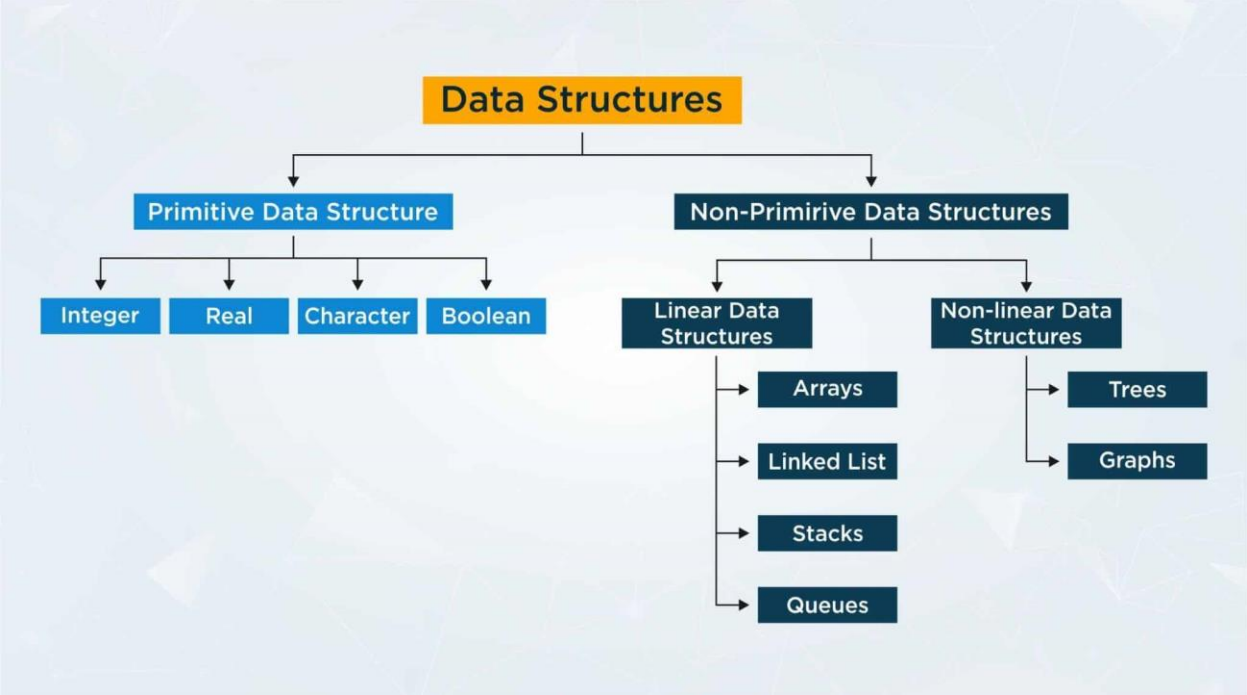
```
import java.util.*;
class Test
{
	public static void main(String[] args)
	{
		int[] arr={4,5,7,1,3,2,2,5,7,9,1};

		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);
      }
}
```

Q)Types of data structures in java?

We have two types of data structures in java.

Diagram: java35.1



Multithreading

==========

Q)What is the difference between Thread and Process?

Thread

It is a leight weight sub process.

We can run multiple threads concurently.

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 concurently.

One process can't communicate with another process.

ex:

typing the notes in notepad is one process

downloading a file from internet is one process

taking a class by using zoom metting is one processs.

Multitasking

==========

Executing several task simultenously such concept is called multitasking.

We have two types of multitasking.

1)Thread based multitasking

Executing several task simultenously where each task is a same part of a program.

It is best suitable for programmatic level.

2)Process based multitasking

Executing several task simultenously where each task is a independent process.

It is best suitable for OS level.

Multithreading

==========

Executing several threads simultenously such concept is called multithreading.

In multithreading only 10% of work should be done by a programmer and 90% of work will be done by JAVA API.

The main important application area of multithreading area.

1) To implements multimedia graphics

2) To develop video games

3) To develop animations

In how many ways we can create a thread in java

==========

There are two ways to create a thread in java.

1) By extending Thread class

2) By implementing Runnable interface

1) By extending Thread class

```
class MyThread extends Thread
{
	//work of a thread
	public void run()
	{
```

```
            for(int i=1;i<=5;i++)
            {
                System.out.println("Child-Thread");
            }
        }
}
class Test
{
        public static void main(String[] args)
        {
            //instantiate a thread
            MyThread t=new MyThread();

            //start a thread
            t.start();

            for(int i=1;i<=5;i++)
            {
                System.out.println("Parent-Thread");
            }

        }
```

}

case1: Thread Schedular

If multiple threads are waiting for execution which thread should execute will decided by thread schedular.

What algorithm,mechanism or behaviour used by thread schedular is depends upon JVM vendor.

Hence we can't expect any execution order of exact output in multithreading.

case2: difference between t.start() and t.run() method

If we invoke t.start() method then a new thread will be created which is responsible to execute run() method automatically.

ex:

```
class MyThread extends Thread
{
        //work of a thread
```

```
	public void run()
	{
		for(int i=1;i<=5;i++)
		{
			System.out.println("Child-Thread");
		}
	}
}
class Test
{
	public static void main(String[] args)
	{
		//instantiate a thread
		MyThread t=new MyThread();

		//start a thread
		t.start();

		for(int i=1;i<=5;i++)
		{
			System.out.println("Parent-Thread");
		}
```

```
    }
}
```

If we invoke t.run() method then no new thread will be created but run() method will execute just like normal method.

ex:

```
class MyThread extends Thread
{
    //work of a thread
    public void run()
    {
        for(int i=1;i<=5;i++)
        {
            System.out.println("Child-Thread");
        }
    }
}
class Test
{
    public static void main(String[] args)
```

```
	{
		//instantiate a thread
		MyThread t=new MyThread();

		t.run();

		for(int i=1;i<=5;i++)
		{
			System.out.println("Parent-Thread");
		}

	}
}
```

case 3: if 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 empty implementation.

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 always execute run() method with no argument method only.

ex:

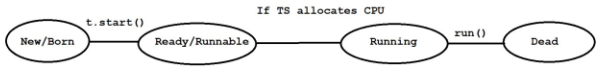
```
class MyThread extends Thread
{
	public void run(int i)
	{
		System.out.println("int-arg method");
	}
	public void run()
	{
		System.out.println("0-arg method");
	}
}
class Test
{
	public static void main(String[] args)
```

```
        {
                //instantiate a thread
                MyThread t=new MyThread();

                //start a thread
                t.start();

                for(int i=1;i<=5;i++)
                {
                        System.out.println("Parent-Thread");
                }

        }
}
```

case5: Life cycle of a thread

Diagram: java35.2

```
MyThread t=new MyThread(); //instantiate a thread
t.start(); // start a thread
```



1)Once if we create a thread our thread will be in new or born state.

2)Once if we call t.start() method then our thread goes to ready/runnable state.

3)If thread schedular allocates to CPU then our threads enters to running state.

4)Once the run() method execution is completed then out thread goes to dead state.

2) By implementing Runnable interface

```
class MyRunnable implements Runnable
{
	public void run()
	{
		for(int i=1;i<=5;i++)
		{
			System.out.println("Child-Thread");
		}
	}
}
class Test
{
	public static void main(String[] args)
	{
		MyRunnable r=new MyRunnable();
		Thread t=new Thread(r); // r is a targatable interface
		t.start();
		for(int i=1;i<=5;i++)
		{
			System.out.println("Parent-Thread");
```

```
            }
        }
}
```

Setting and getting name of the thread

==========

In java ,every thread has a name explicitly provided by the programmer or automatically

generated by JVM.

We have following methods to set and get name of a thread.

ex:

```
public final void setName(String name)
public final String getName()
```

ex:

```
class MyThread extends Thread
{
}
class Test
{
    public static void main(String[] args)
```

```
	{
		System.out.println(Thread.currentThread().getName());
//main

		MyThread t=new MyThread();
		System.out.println(t.getName());//Thread-0

		Thread.currentThread().setName("parent-thread");
		System.out.println(Thread.currentThread().getName()); //
parent-thread

		t.setName("child-thread");
		System.out.println(t.getName());//child-thread
	}
}
```

Thread priority

==========

In java, every thread has a name explicitly provided by the programmer and

automatically generated by JVM.

The valid range of thread priority is 1 to 10. Where 1 is a least priority and 10 is a

highest priority.

If we take more then 10 priority then we will get runtime exception called

IllegalArgumentException.

We have following standards constants to represent Thread priority.

ex:

Thread.MAX_PRIORITY - 10

Thread.NORM_PRIORITY - 5

Thread.MIN_PRIORITY - 1

We have done such constants like LOW_PRIORITY and HIG_PRIORITY.

The 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 folowing 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
        }
}
```

Various ways to prevent a thread from execution

==========

There are three ways to prevent(stop) a thread from execution.

1) yield()

2) join()

3) sleep()

1) yield()

It will pause current execution thread and gives the change to other threads having same

priority.

If there is not waiting thread or low priority thread then same thread will continue its

execution.

If multiple waiting threads having same priority then we can't expect any execution order.

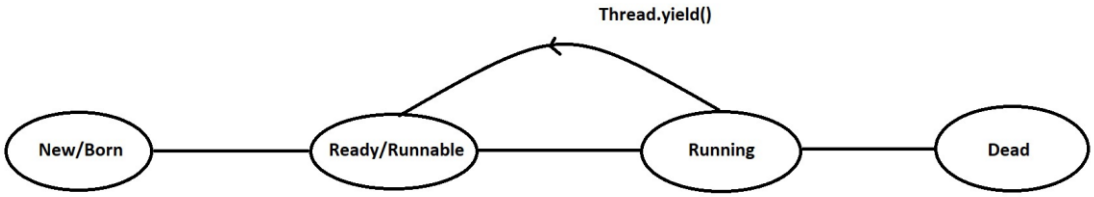
The thread which is yielded , when it will get a chance for execution is depends upon mercy

of thread schedular.

ex:

```
public static native void yield()
```

Diagram:java36.1



ex:

```
class MyThread extends Thread
{
	public void run()
	{
		for(int i=1;i<=5;i++)
		{
			System.out.println("child-thread");
		}
	}
}
class Test
{
```

```
	public static void main(String[] args)
	{
		MyThread t=new MyThread();
		t.start();
		for(int i=1;i<=5;i++)
		{
			Thread.currentThread().yield();
			System.out.println("parent-thread");
		}
	}
}
```

2)join()

If 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.

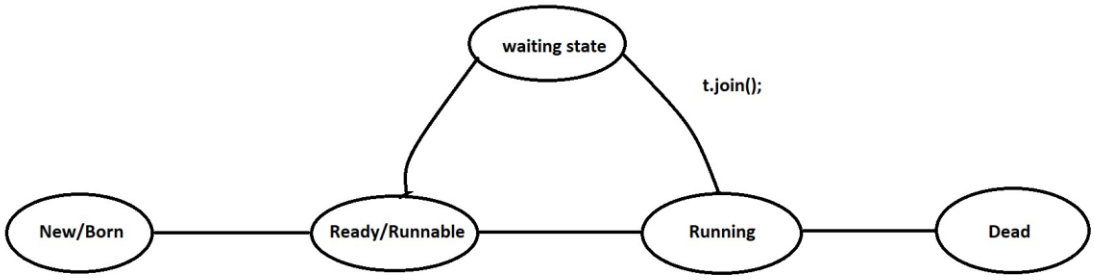
ex:

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: java36.2




```
class MyThread extends Thread
{
	public void run()
	{
		for(int i=1;i<=5;i++)
		{
```

```
                System.out.println("child-thread");
            }
        }
}
class Test
{
        public static void main(String[] args)throws InterruptedException
        {
            MyThread t=new MyThread();
            t.start();
            t.join();
            for(int i=1;i<=5;i++)
            {
                System.out.println("parent-thread");
            }
        }
}
```

3)sleep()

If thread don't want to perform any operation on perticular amount of time then we need to

sleep() method.

A sleep() method will throw one checked exception called InterruptedException so we must

and should handle that exception by using try and catch block or by using throws statement.

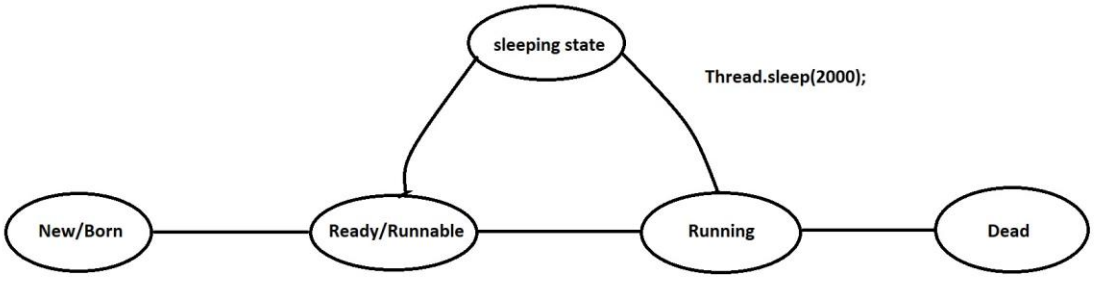
ex:

public static native void sleep()throws InterruptedException

public static native void sleep(long ms)throws InterruptedException

public static native void sleep(long ms,int ns)throws InterruptedException

Diagram: java36.3



ex:

```
class MyThread extends Thread
{
	public void run()
	{
		for(int i=1;i<=5;i++)
		{
			System.out.println("child-thread");
			try
			{
				Thread.sleep(2000);
			}
			catch(InterruptedException ie)
			{
				ie.printStackTrace();
			}
		}
	}
}
class Test
```

```
{
	public static void main(String[] args)
	{
		MyThread t=new MyThread();
		t.start();
		for(int i=1;i<=5;i++)
		{
			System.out.println("parent-thread");
		}
	}
}
```

Daemon Thread

==========

It 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 terminate automatically.

There are many daemon threads are running internally like Garbage Collector, finalizer and etc.

We can start a daemon thread by using 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

==========

• Synchronized is the keyword applicable for methods and blocks but not for classes and

variables.

• If a method or block declared as the synchronized then at a time only one Thread is allow

to execute that method or block on the given object.

• The main advantage of synchronized keyword is we can resolve date inconsistency problems.

• But the main disadvantage of synchronized keyword is it increases waiting time of the

Thread and effects performance of the system.

• Hence if there is no specific requirement then never recommended to use synchronized

keyword.

• Internally synchronization concept is implemented by using lock concept.

• Every object in java has a unique lock. Whenever we are using synchronized keyword then

only lock concept will come into the picture.

• If a Thread wants to execute any synchronized method on the given object 1st it has to get

the lock of that object. Once a Thread got the lock of that object then it's allow to execute

any synchronized method on that object. If the synchronized method execution completes

then automatically Thread releases lock.

• While a Thread executing any synchronized method the remaining Threads are not allowed

execute any synchronized method on that object simultaneously. But remaining Threads

are allowed to execute any non-synchronized method simultaneously. [lock concept is

implemented based on object but not based on method].

ex:

```
class Table
{
	synchronized void printTable(int n)
	{
		for(int i=1;i<=5;i++)
		{
			System.out.println(n*i);
			try
			{
				Thread.sleep(2000);
			}
			catch (InterruptedException ie)
			{
				ie.printStackTrace();
```

```
                }
            }
        }
}
class MyThread1 extends Thread
{
        Table  t;
        MyThread1(Table t)
        {
            this.t=t;
        }
        public void run()
        {
            t.printTable(5);
        }
}
class MyThread2 extends Thread
{
        Table  t;
        MyThread2(Table t)
        {
            this.t=t;
```

```
    }
    public void run()
    {
        t.printTable(10);
    }
}
class Test
{
    public static void main(String[] args)
    {
        Table obj=new Table();
        MyThread1 t1=new MyThread1(obj);
        MyThread2 t2=new MyThread2(obj);
        t1.start();
        t2.start();
    }
}
```

Synchronized block:

==========

• If very few lines of the code required synchronization then it's never recommended to

declare entire method as synchronized we have to enclose those few lines of the code

with in synchronized block.

• The main advantage of synchronized block over synchronized method is it reduces

waiting time of Thread and improves performance of the system.

ex:

```
class Table
{
	void printTable(int n)
	{
		synchronized(this)
		{
		for(int i=1;i<=5;i++)
		{
			System.out.println(n*i);
			try
			{
				Thread.sleep(2000);
			}
			catch (InterruptedException ie)
			{
```

```
                    ie.printStackTrace();
                }
            }
            }//block
        }
}
class MyThread1 extends Thread
{
        Table  t;
        MyThread1(Table t)
        {
            this.t=t;
        }
        public void run()
        {
            t.printTable(5);
        }
}
class MyThread2 extends Thread
{
        Table  t;
        MyThread2(Table t)
```

```
        {
                this.t=t;
        }
        public void run()
        {
                t.printTable(10);
        }
}
class Test
{
        public static void main(String[] args)
        {
                Table obj=new Table();
                MyThread1 t1=new MyThread1(obj);
                MyThread2 t2=new MyThread2(obj);
                t1.start();
                t2.start();
        }
}
```

static synchronization

==========

Every class in java has a unique lock. If a Thread wants to execute a static synchronized

method then it required class level lock.

ex:

```
class Table
{
	static synchronized void printTable(int n)
	{

		for(int i=1;i<=5;i++)
		{
			System.out.println(n*i);
			try
			{
				Thread.sleep(2000);
			}
			catch (InterruptedException ie)
			{
				ie.printStackTrace();
			}
```

```
            }
        }
}
class MyThread1 extends Thread
{
        public void run()
        {
            Table.printTable(5);
        }
}
class MyThread2 extends Thread
{
        public void run()
        {
            Table.printTable(10);
        }
}
class Test
{
        public static void main(String[] args)
        {
            MyThread1 t1=new MyThread1();
```

```
        MyThread2 t2=new MyThread2();
        t1.start();
        t2.start();
    }
}
```

Inter-Thread communication

==========

ex:

```
class MyThread extends Thread
{
    int total=0;

    public void run()
    {
        synchronized(this)
        {
            System.out.println("Child started calculation");
            for(int i=1;i<=10;i++)
            {
```

```
                    total+=i;
                }
                System.out.println("Child 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 method waiting for
notification");
                t.wait();
                System.out.println("Main method got notification");
                System.out.println(t.total);
            }
        }
```

```
}
```

DeadLock In Java

==========

ex:

```
class Test
{
	public static void main(String[] args)throws InterruptedException
	{
		final String res1="hi";
		final String res2="bye";

		//anonymous inner class
		Thread t1=new Thread()
		{
			public void run()
			{
				synchronized(res1)
				{
```

```
                        System.out.println("Thread1: Locking
Resource1");
                        synchronized(res2)
                        {
                                System.out.println("Thread1:Locking
Resource2");
                        }
                }
        }
};

//anonymous inner class
Thread t2=new Thread()
{
        public void run()
        {
                synchronized(res2)
                {
                        System.out.println("Thread2: Locking
Resource2");
                        synchronized(res1)
                        {
```

```
                                        System.out.println("Thread2:Locking
Resource1");
                                    }
                                }
                            }
                        };

                        t1.start();
                        t2.start();
                    }
}
```

Disadvantages of Multithreading

==========

We have following disadvantages of multithreading.

1) Thread starvation

2) DeadLock

Java 8 Features

==========

1) java.time package

2) Functional interface

3) Lamda Expression

4) default methods in interface

5) static methods in interface

6) Stream API

7) forEach() method

8) Method reference(::)

and etc.

2) Functional interface

==========

Functional interface introduced in java 8.

An interface which contains only one abstract method is called functional interface.

ex:

Runnable ---> run()

Comparable ---> compareTo()

ActionListener --> actionPerformed()

It can have any number of default methods and static methods.

Functional interface is also known as SAM or Single Abstract Method interface.

Functional interface is used to achieve functional programming.

ex:

a=f1()

{}

f1(f2(){})

{}

@FunctionalInterface annotation is used to declare functional interface and it is optional.

ex:

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

```
            a.m1();
        }
}

ex:
----
@FunctionalInterface
interface A
{
        public abstract void m1();
}

class Test
{
        public static void main(String[] args)
        {
                A a=new A()
                {
                        public void m1()
                        {
                                System.out.println("From M1 Method");
                        }
```

```
            };
            a.m1();
        }
}
```

Lamda Expression

==========

Lamda expression introduced in java 8.

Lamda expression is used to concise the code.

Lamda expression consider as method but not a class or object.

We can use lamda expression when we have functional interface.

Lamda expression is used to achieve functional programming.

Lamda expression does not support modifier,returntype and name of the method.

ex:

java method

```
----------
        public void m1()
        {
                System.out.println("Hello");
        }

    lamda expression
    ------------
        ()->
        {
                System.out.println("Hello");
        };

ex:
---
@FunctionalInterface
interface A
{
    public abstract void m1();
}

class Test
```

```
{
	public static void main(String[] args)
	{
		A a=()->
			{
				System.out.println("M1 Method");
			};
		a.m1();

	}
}
```

ex:

```
@FunctionalInterface
interface A
{
	public abstract void m1(int i,int j);
}

class Test
```

```
{
	public static void main(String[] args)
	{
		A a=(int i,int j)->
			{
				System.out.println(i+j);
			};
		a.m1(10,20);

	}
}
```

ex:

```
@FunctionalInterface
interface A
{
	public abstract String m1();
}

class Test
{
```

```
	public static void main(String[] args)
	{
		A a=()->
			{
				return "Hello World";
			};
		System.out.println(a.m1());

	}
}
```

default methods in interface

==========

Default methods in interface introduced in java 8.

To declare default methods we need to use default keyword.

Default method is a non-abstract method.

Default methods we can override.

ex:

```
        default void m1()
        {
                -
                - //code to be execute
                -
        }
```

ex:

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

        //default method
        default void m2()
        {
                System.out.println("default method");
        }
}
class B implements A
```

```
{
	public void m1()
	{
		System.out.println("M1 Method");
	}
}
class Test
{
	public static void main(String[] args)
	{
		A a=new B();
		a.m1();
		a.m2();
	}
}
```

ex:

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

```
	//default method
	default void m2()
	{
		System.out.println("default method");
	}
}
class B implements A
{
	public void m1()
	{
		System.out.println("M1 Method");
	}
	public void m2()
	{
		System.out.println("override default method");
	}
}
class Test
{
	public static void main(String[] args)
	{
```

```
        A a=new B();
        a.m1();
        a.m2();
    }
}
```

We can achive multiple inheritance by using default methods of an interface.

ex:

```
interface Right
{
    default void m1()
    {
        System.out.println("Right-M1 Method");
    }
}
interface Left
{
    default void m1()
    {
```

```
            System.out.println("Left-M1 Method");
        }
}
class Middle implements Right,Left
{
        public void m1()
        {
            System.out.println("Middle-M1 method");
        }
}
class Test
{
        public static void main(String[] args)
        {
            Middle m=new Middle();
            m.m1();
        }
}
```

ex:

```
interface Right
```

```
{
	default void m1()
	{
		System.out.println("Right-M1 Method");
	}
}
interface Left
{
	default void m1()
	{
		System.out.println("Left-M1 Method");
	}
}
class Middle implements Right,Left
{
	public void m1()
	{
		Right.super.m1();
	}
}
class Test
{
```

```
	public static void main(String[] args)
	{
		Middle m=new Middle();
		m.m1();
	}
}
```

ex:

```
interface Right
{
	default void m1()
	{
		System.out.println("Right-M1 Method");
	}
}
interface Left
{
	default void m1()
	{
		System.out.println("Left-M1 Method");
	}
```

```
}
class Middle implements Right,Left
{
	public void m1()
	{
		Left.super.m1();
	}
}
class Test
{
	public static void main(String[] args)
	{
		Middle m=new Middle();
		m.m1();
	}
}
```

static methods in interface

==========

Static methods in interface introduced in java 8.

To declare static method we need to use static keyword.

A static method is a non-abstract method.

It can't be override.

ex:

```
        static void m1()
        {
                -
                - //code to be execute
                -
        }
```

ex:

```
interface A
{
        static void m1()
        {
                System.out.println("M1 method");
        }
```

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

Stream API

==========

Stream API introduced in java 8.

A Stream is an interface which is present in java.util.stream package.

Stream API is used to perform bulk operations on Collections.

ex:

--

```
import java.util.*;
import java.util.stream.*;
class Test
```

```
{
	public static void main(String[] args)
	{
		List<Integer> list=Arrays.asList(4,7,1,3,9,6,2);

		List<Integer> even=list.stream().filter(i-
>i%2==0).collect(Collectors.toList());

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

ex:

```
import java.util.*;
import java.util.stream.*;
class Test
{
	public static void main(String[] args)
	{
		List<Integer> list=Arrays.asList(4,7,1,3,9,6,2);
```

```
            List<Integer> odd=list.stream().filter(i-
>i%2==1).collect(Collectors.toList());


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

ex:

```
import java.util.*;
import java.util.stream.*;
class Test
{
        public static void main(String[] args)
        {
            List<Integer> list=Arrays.asList(4,7,1,3,9,6,2);


            List<Integer>
l=list.stream().sorted().collect(Collectors.toList());


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

ex:

```
import java.util.*;
import java.util.stream.*;
class Test
{
	public static void main(String[] args)
	{
		List<Integer> list=Arrays.asList(4,7,1,3,9,6,2);

		List<Integer>
l=list.stream().sorted(Comparator.reverseOrder()).collect(Collectors.toL
ist());

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

ex:

```
import java.util.*;
import java.util.stream.*;
```

```
class Test
{
	public static void main(String[] args)
	{
		List<Integer> list=Arrays.asList(44,70,51,39,79,66,42);

		List<Integer> l=list.stream().map(i->i+10).collect(Collectors.toList());

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

ex:

```
import java.util.*;
import java.util.stream.*;
class Test
{
	public static void main(String[] args)
	{
		List<Integer> list=Arrays.asList(44,70,51,13,79,66,22);
```

```
		List<Integer> failed=list.stream().filter(i->i<35).collect(Collectors.toList());


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

ex:

```
import java.util.*;
import java.util.stream.*;
class Test
{
	public static void main(String[] args)
	{
		List<Integer> list=Arrays.asList(44,70,51,13,79,66,22);
		long failed=list.stream().filter(i->i<35).count();
		System.out.println(failed);
	}
}
```

forEach() method

==========

A forEach() method introduced in java 8.

It is used to iterate the elements one by one.

ex:

```
import java.util.*;
import java.util.stream.*;
class Test
{
	public static void main(String[] args)
	{
		List<Integer> list=Arrays.asList(44,70,51,13,79,66,22);

		list.forEach(System.out::println);

	}
}
```

MHR interview Question

==========

Q) Write a java program to sort employees by id using java 8 stream api?

```
import java.util.*;
import java.util.stream.*;
class Employee
```

```
{
        private int empId;
        private String empName;
        private double empSal;
        //parameterized constructor
        Employee(int empId,String empName,double empSal)
        {
                this.empId=empId;
                this.empName=empName;
                this.empSal=empSal;
        }
        //getter methods
        public int getEmpId()
        {
                return empId;
        }
        public String getEmpName()
        {
                return empName;
        }
        public double getEmpSal()
        {
```

```
        return empSal;
    }
}
class Test
{
    public static void main(String[] args)
    {
        List<Employee> list=new ArrayList<Employee>();
        list.add(new Employee(401,"Alan",10000));
        list.add(new Employee(101,"Jose",20000));
        list.add(new Employee(201,"Mark",30000));
        list.add(new Employee(301,"Alex",40000));
        List<Employee>
newList=list.stream().sorted(Comparator.comparingInt(Employee::getE
mpId)).collect(Collectors.toList());
        newList.forEach(employee ->
System.out.println(employee.getEmpId() +"
"+employee.getEmpName() +" "+employee.getEmpSal()));
    }
}
```

ex:

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

```
import java.util.stream.*;
class Employee
{
	private int empId;
	private String empName;
	private double empSal;
	//parameterized constructor
	Employee(int empId,String empName,double empSal)
	{
		this.empId=empId;
		this.empName=empName;
		this.empSal=empSal;
	}
	//getter methods
	public int getEmpId()
	{
		return empId;
	}
	public String getEmpName()
	{
		return empName;
	}
```

```
	public double getEmpSal()
	{
		return empSal;
	}
}
class Test
{
	public static void main(String[] args)
	{
			List<Employee> list=new ArrayList<Employee>();
			list.add(new Employee(401,"Alan",10000));
			list.add(new Employee(101,"Jose",20000));
			list.add(new Employee(201,"Mark",30000));
			list.add(new Employee(301,"Alex",40000));
			List<Employee>
newList=list.stream().sorted(Comparator.comparingDouble(Employee::
getEmpSal)).collect(Collectors.toList());
			newList.forEach(employee ->
System.out.println(employee.getEmpId() +"
"+employee.getEmpName() +" "+employee.getEmpSal()));
	}
}
```