# JAVA FULL STACK DEVELOPER

## Introduction

Full Stack Java Development

(Full Stack Developer)

|

-------------------------------------------------------------------------------------

| |

Frontend Technologies Backend Technologies

(Frontend Developer) (BackEnd Developer)

> HTML5 > Core Java

> CSS3 > Advanced Java

> JavaScript > JDBC

> Bootstrap5 > Servlets

> ReactJS > JSP

> Oracle Database

> SQL

> PL/SQL

> Frameworks

>Spring Boot

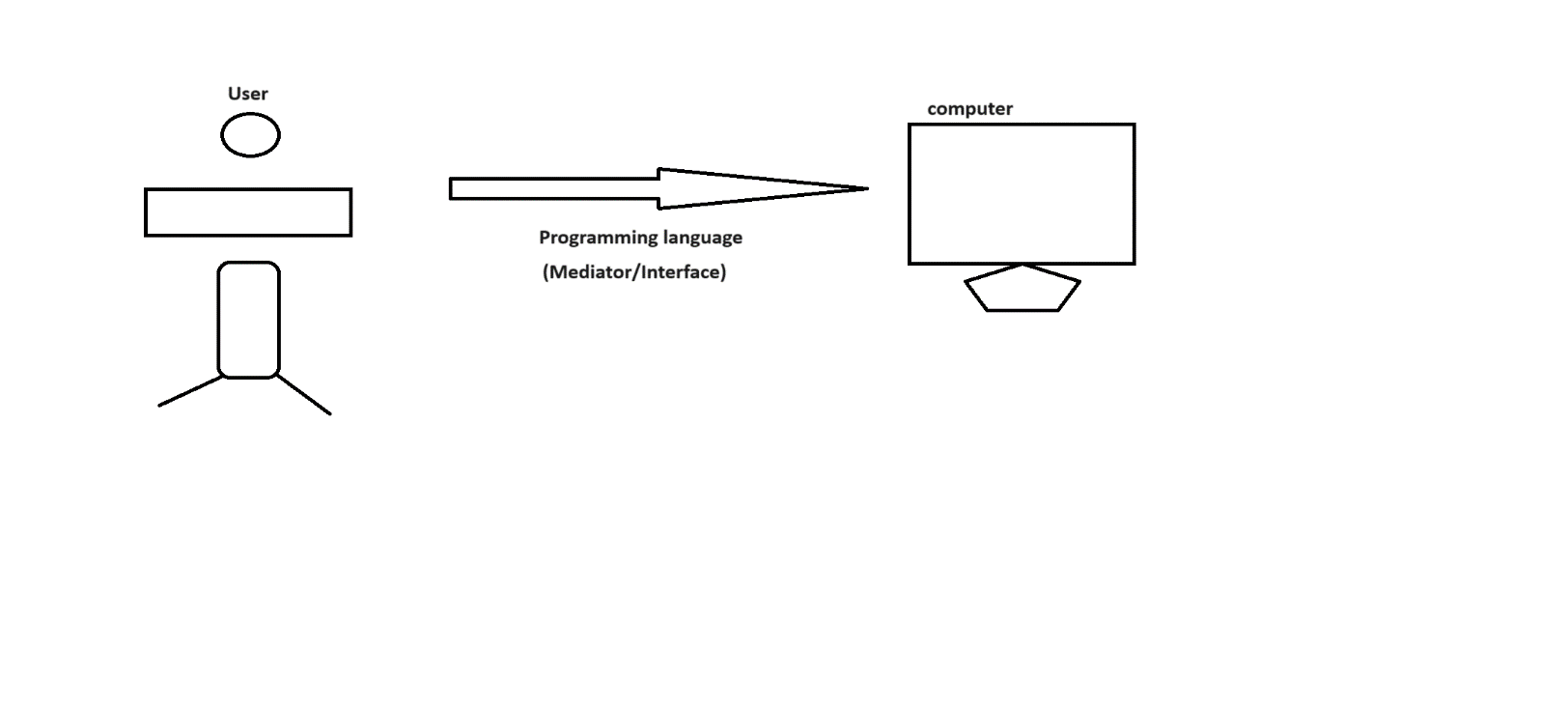
> Microservices

# 1.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.

Java Software --> JDK software ---> Sun Micro System --> Oracle Corporation

1995 --> James Gosling

**Interview Questions**

**Q) What is Java?**

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

**ex:1**

Java Code

for(int i=1;i<=10;i++)

System.out.print(i);

Python Code

for i in range(1,11):

print(i)

**ex:2**

Java Code

int count=1;

while(count<=10){

System.out.println(count);

count++;}

python Code

count=1;

while count<=10:

print(count)

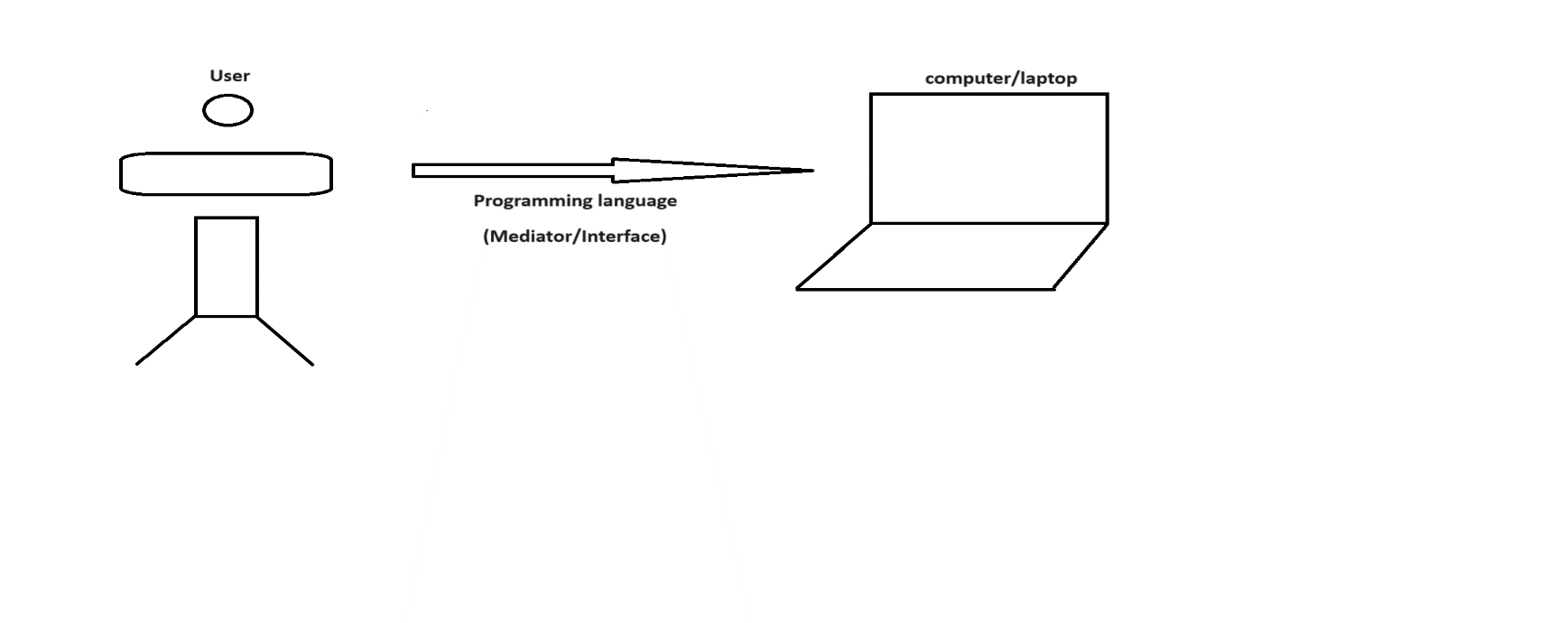
count+1;

**Programming Language**

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

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

**Diagram: introduction2.1**

****Programming language divided into two types.

1. Low level language
2. High level language
3. **Low level language**

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

**ex:**

Machine Language

Assembly Language

**Machine Language**

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

It is also known as binary language.

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

**Advantages:**

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

**Disadvantages:**

* It is a burdun on a programmer to remember dozen's of binary code.
* 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 our mathematical programming code.

It is also known as symbolic language.

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

**We have three translators.**

1. Assembler
2. Compiler
3. Interpreter

**1) Assembler**

It is used to convert 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.

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

ex:

C++, Java, .Net, Python and etc.

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

**compiler**

It will compile and execute our program at a time.

**interpreter**

It will execute our program line by line procedure.

**Advantages:**

* 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 translators.
* It is not efficient when compare to low level language.

**C program**

#include<stdio.h>

#include<conio.h>

void main()

{

clrscr();

printf("Hello World");

getch();

}

**C++ program**

#include<iostream.h>

#include<conio.h>

void main()

{

clrscr();

cout<<"Hello World";

getch();

}

**Java program**

class Test

{

public static void main(String[] args)

{

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

}

}

**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 lang.

It is a interpreted language. It is a compiled language.

It is a dynamically typed language. It is a statically typed language.

It contains PVM. It contains JVM.

Performance is low. Performance is high.

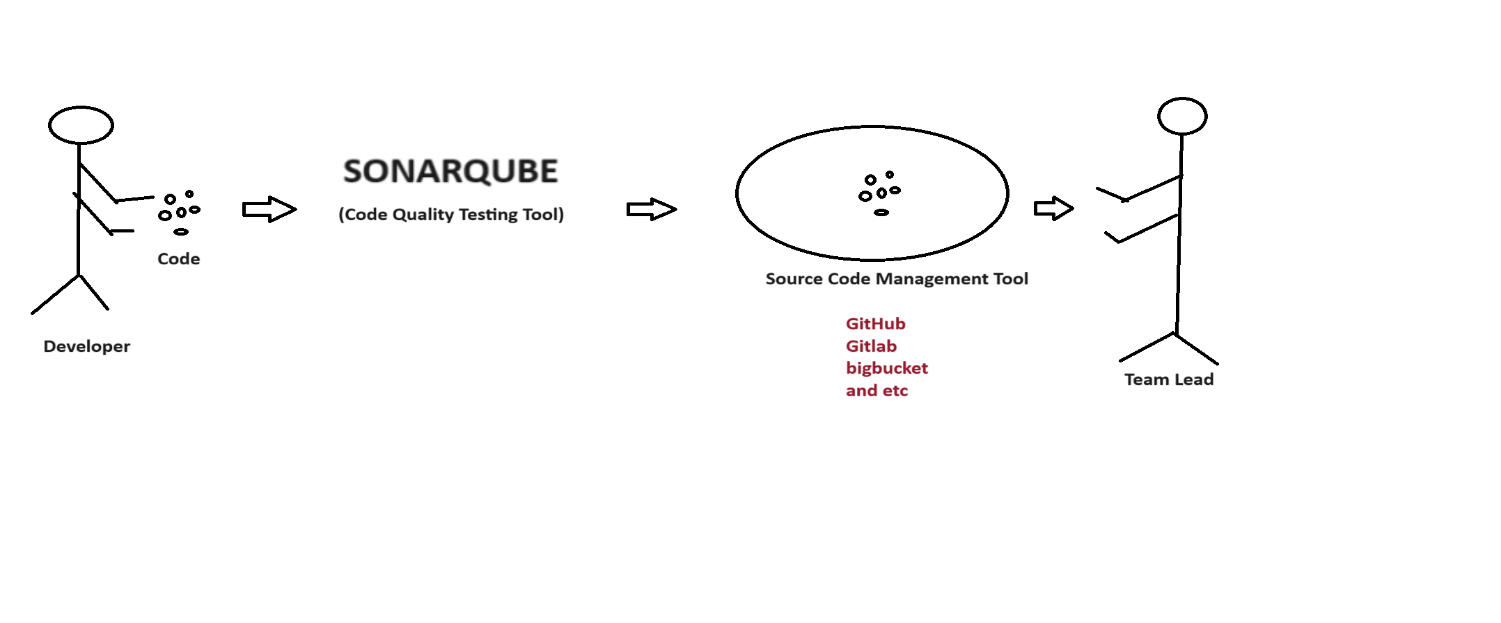
Less Security. Highly secured.

It contains less code. It contains more code.

**Comments in java**

* Comments are created for documentation purpose.
* Comments are used to improve readability of our code.
* It is highly recommanded to use comments in our regular programming.

**Diagram: introduction3.1**

****

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

In java, we have two types of comments.

**1) Single line comment**

It is used to comment a single line.

ex:

// comment here

**2) Multiple line comment**

It is used to comment multiple lines.

ex:

/\*

-

- comment here

-

\*/

**Program**

//class declaration

class Test

{

//main method

public static void main(String[] args)

{

//variable declaration

int i=10;

//output stmt

System.out.println(i);

}

}

**Escape Characters or Escape Sequences**

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

Escape characters starts with back slash (\) followed by a character.

ex:

\n

Mostly escape characters are placed inside output statement.

ex:

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

We have following list of escape characters in java.

1. \n (new line)
2. \t (horizontal tab)
3. \b (back space)
4. \r (carriage return)
5. \f (form feeding)
6. \\ (back slash)
7. \" (double quote)
8. \' (single quote) and etc.

**1) \n (new line)**

class Salam

{

public static void main(String[] args)

{

System.out.println("IHUB\nTALENT");

}

}

**o/p:**

IHUB

TALENT

**2) \t (horizontal tab)**

class Trivedula

{

public static void main(String[] args)

{

System.out.println("IHUB\tTALENT");

}

}

**o/p:**

IHUB TALENT

**3) \b (back space)**

class Sai

{

public static void main(String[] args)

{

System.out.println("I\bHUBTALENT");

}

}

**o/p:**

HUBTALENT

**ex:**

class Koteshwar

{

public static void main(String[] args)

{

System.out.println("IHUB\b\b\bTALENT");

}

}

**o/p:** ITALENT

**4) \r (carriage return)**

class Jagadeesh

{

public static void main(String[] args)

{

System.out.println("IHUB\rTALENT");

}

}

**o/p:**

TALENT

**ex:**

---

class Shivani

{

public static void main(String[] args)

{

System.out.println("TALENT\rIHUB");

}

}

**o/p:**

IHUBNT

**6) \\ (back slash)**

class Lokesh

{

public static void main(String[] args)

{

System.out.println("IHUB\\TALENT");

}

}

**o/p:**

IHUB\TALENT

**7) \" (double quote)**

class Shaheda

{

public static void main(String[] args)

{

System.out.println("I love \"java\" programming");

}

}

**o/p:**

I love "java" programming

**8) \' (single quote)**

class Laxmi

{

public static void main(String[] args)

{

System.out.println("I Love 'java' programming");

System.out.println("I Love \'java\' programming");

}

}

**o/p:**

I Love 'java' programming

I Love 'java' programming

**C program**

Q) Write a c program to print %d ?

void main()

{

clrscr();

printf("%d"); //0

getch();

}

**or**

void main()

{

clrscr();

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

getch();

}

Alladi Cloud solutions

**Q) What is the output of below snippet?**

class Example

{

public static void main(String[] args)

{

System.out.print("\nlt");

System.out.print("\bpi");

System.out.print("\rha");

}

}

**o/p:**

hai

# 2.Modules In Java

We have three modules in java.

JAVA

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

JSE/J2SE JEE/J2EE JME/J2ME

(Java Standard Edition) (Java Enterprise Edition) (Java Micro Edition)

> Standalone App > Distributed App > Mobile App

> Desktop App > Enterprise App

> Two-tier App > ERP App

> N-Tier App

**> Standalone App**

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

ex:

class Test

{

public static void main(String[] args)

{

-

- //code to be execute

-

}

}

**> Desktop App**

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

ex:

Control Panel

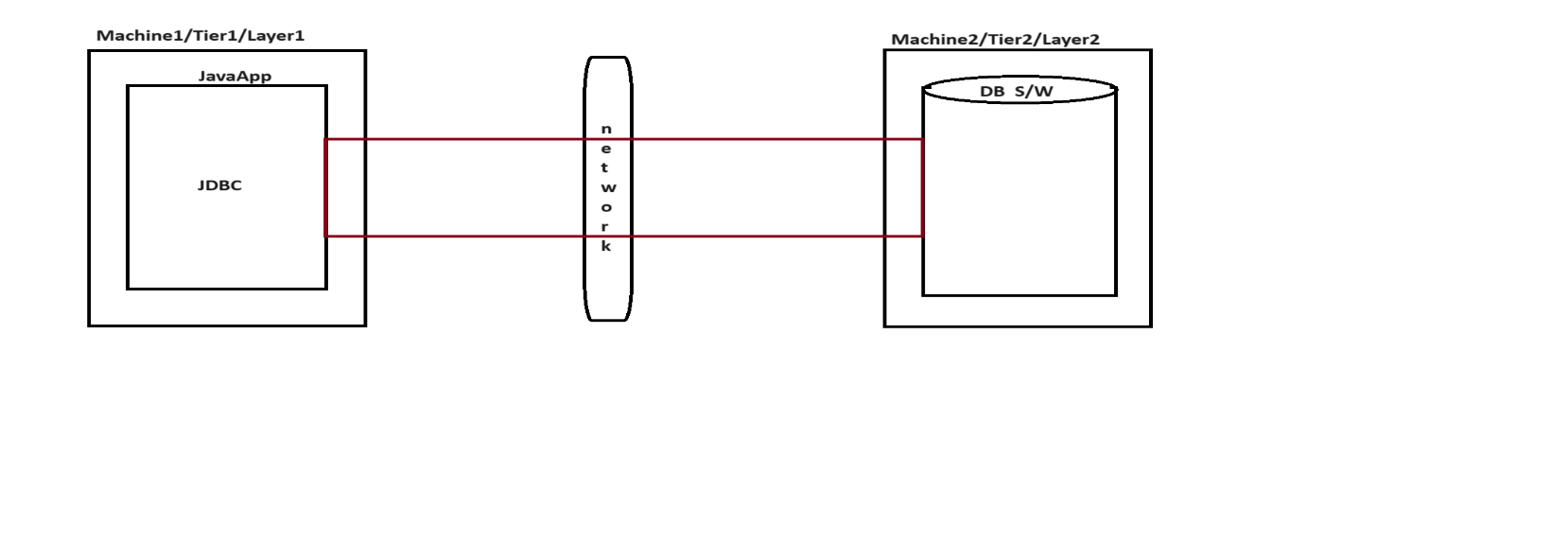
Recycle Bin

VLC Media Player

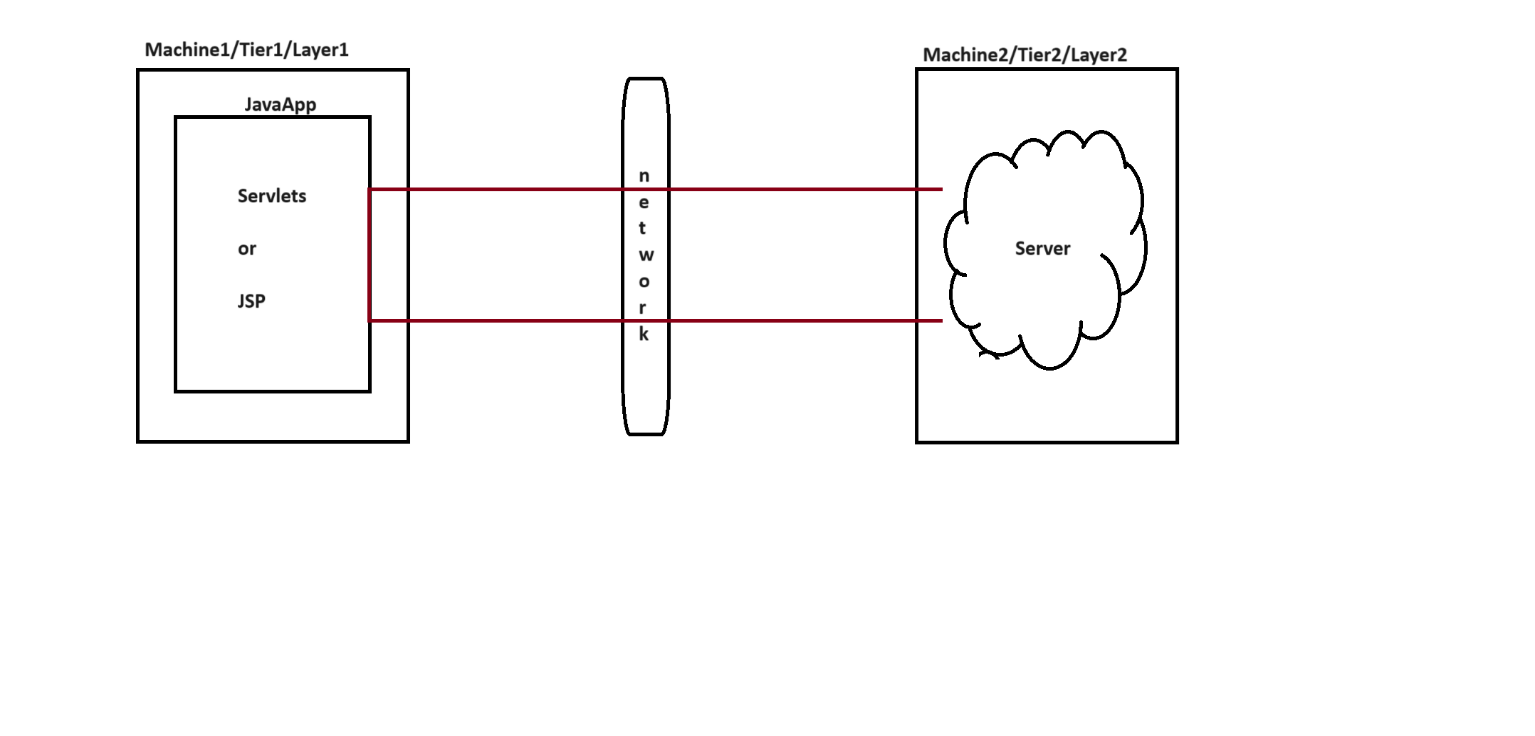
**> Two-tier App**

Having more then one tier is called two tier application.

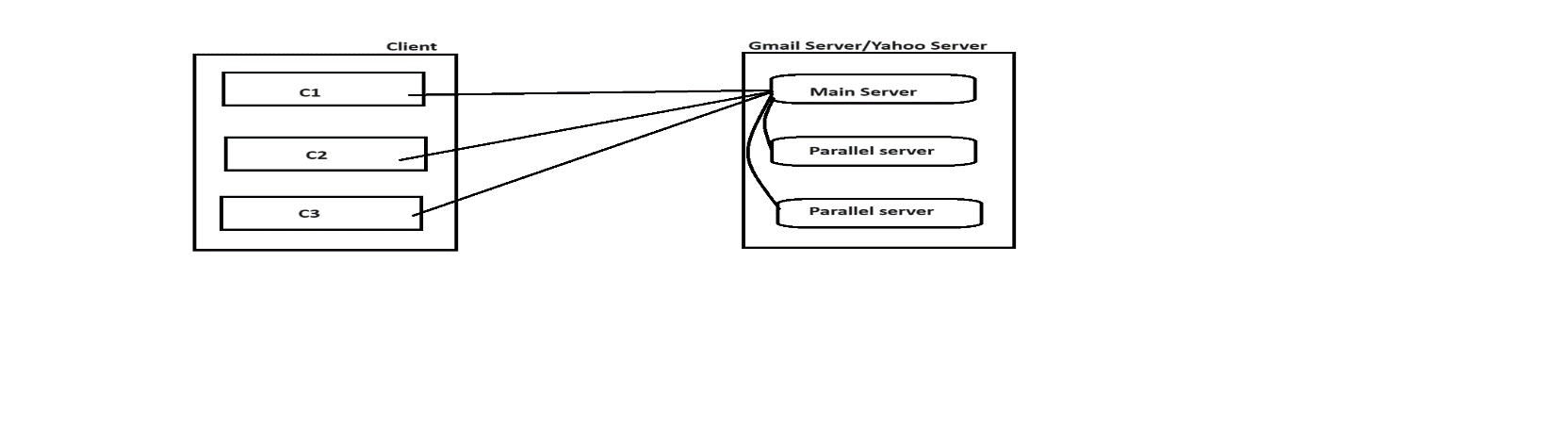
**Diagram: class1.1**



**Diagram: class1.2**

****

**>DistributedApp**

****

**Diagram:class1.3**

In client-server architecture, If multiple clients sending the request to the main server then main server will distribute the request to it's parallel servers to reduce the burdun of main server is called distributed application.

**> Enterprise App**

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

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

**ex:**

Facebook

Online shopping websites

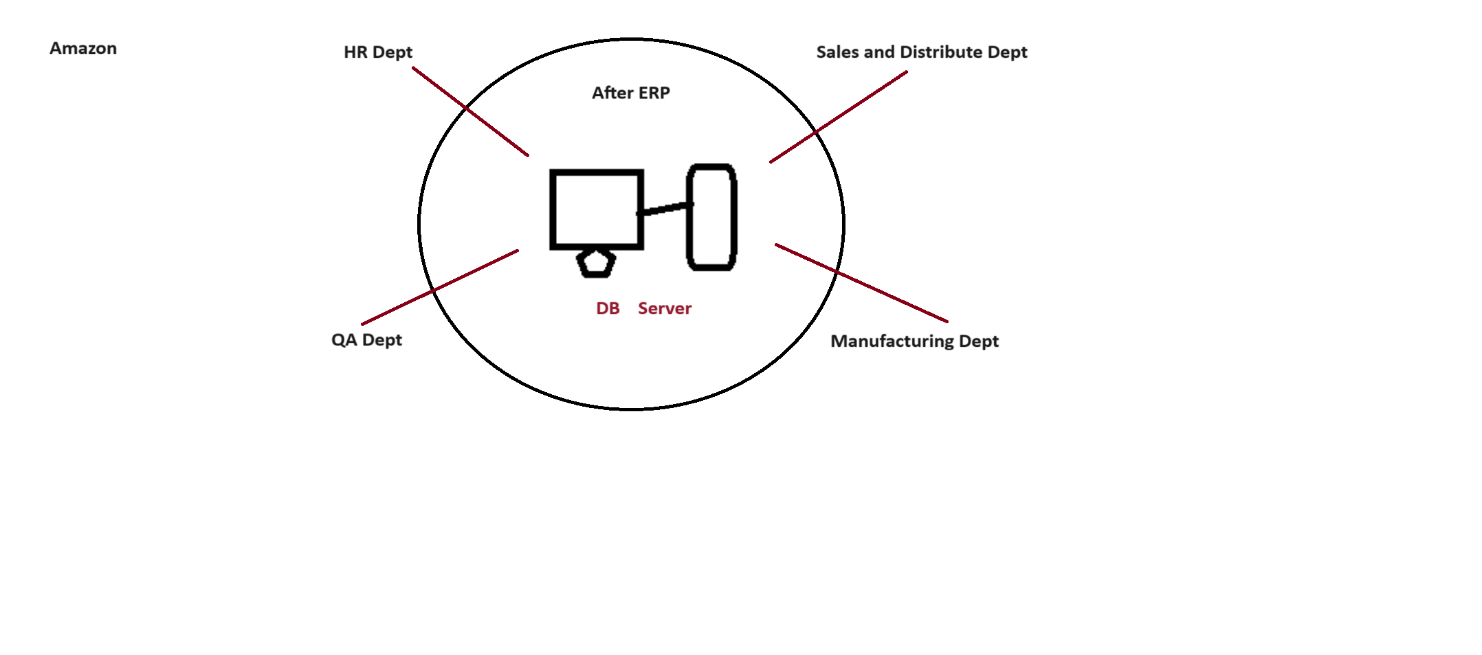
and etc

**> ERP App**

ERP stands for Enterprise Resource Planning.

ERP is used to maintain the data in enterprise.

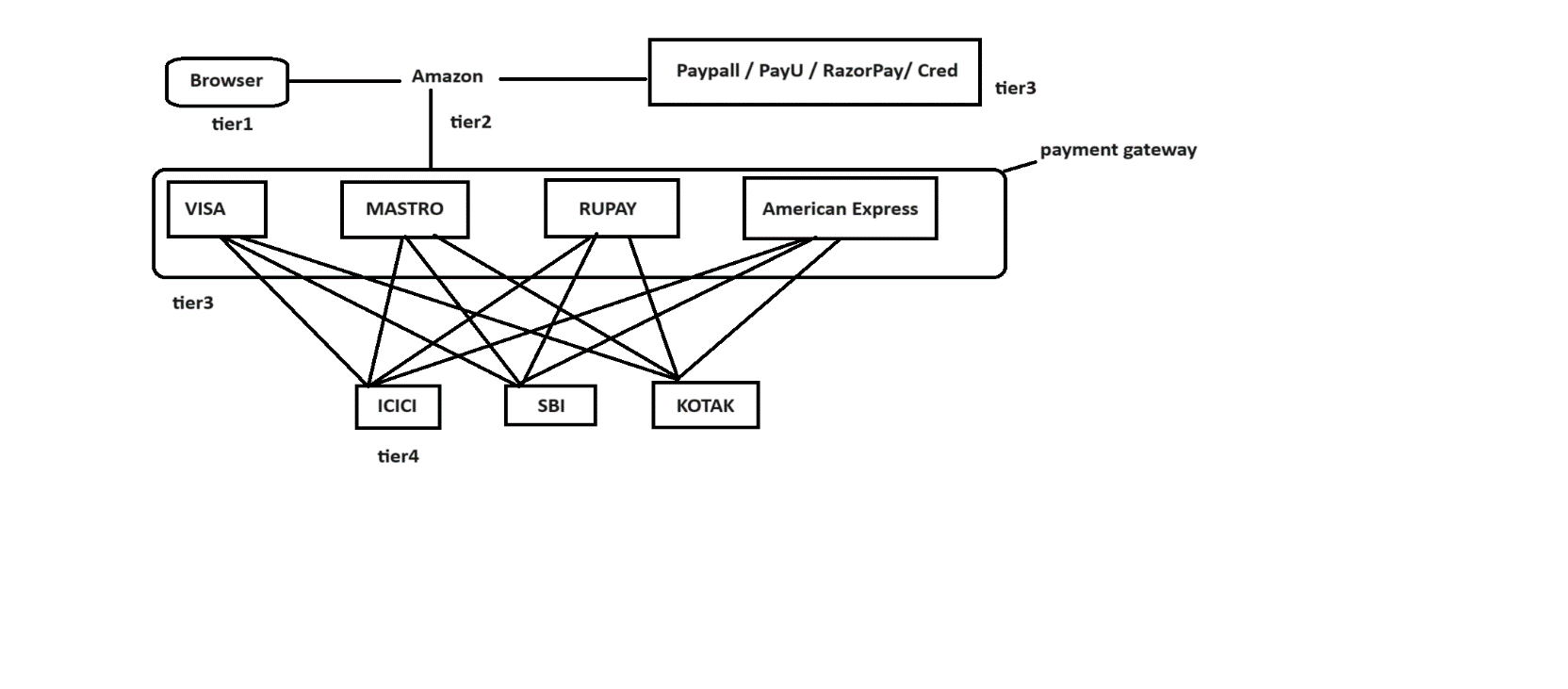
**Diagram: class1.4**

****

**> N-Tier App**

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

**Diagram: class1.5**

****

**> Mobile App**

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

**ex:**

GooglePay

PhonePay

PubG

and etc.

# 3.Naming Conventions in java

In java, uppercase letters will consider as different and lowercase letters will consider as different that 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:**

1. classes
2. interfaces
3. variables
4. methods
5. keywords
6. packages
7. and
8. constants

**1)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

File DemoApp

FileWriter ExampleApp

BufferedReader TestApp

ArrayList ReverseNumber

and etc. and etc.

**2)interfaces**

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

ex:

**predefined interfaces** **userdefined interfaces**

Runnable ITest

Serializable IDemoApp

Iterator IExampleApp

ListIterator IReverseNumber

Enumeration and etc.

and etc.

**3)variables**

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

ex:

**predefined variables** **userdefined variables**

out i

in x

err empId

length studName

and etc. deptNo

and etc.

**4) methods**

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

ex:

**predefined methods** **userdefined methods**

getClass() getDetails()

hashCode() calculatBillAmt()

toString() getInfo()

setName() setStudentInfo()

getPriority() fetchDetails()

and etc. and etc.

**5)keywords**

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

ex:

**predefined keywords**

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

**6)packages**

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

ex:

**predefined packages** **userdefined packages**

java.lang (default pkg) ihub

java.io com.ihub.www

java.util google

java.util.stream com.google.www

java.text and etc.

java.time

java.sql

javax.servlet

and etc.

**7)constants**

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

ex:

**predefined constants**  **userdefined constants**

MAX\_PRIORITY N=10;

MIN\_PRIORITY LIMIT=10;

NORM\_PRIORITY

MAX\_VALUE

MIN\_VALUE

and etc.

**Assignment**

1) class : QualityThought

2) interface : IQualityThought

3) variable : qualityThought

4) method : qualityThought()

5) package : com.qualitythought.www

6) constant : QUALITY\_THOUGHT

**Interview Questions:**

**Q) Which package is a default package in java?**

java.lang package

**Q) What is package?**

A package is a collection of classes and interfaces.

**Q) What is Java?**

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

**Q) Which component is responsible to destroy objects in java?**

Garbage Collector

**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();

**HOW TO DOWNLODE 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 installed successfully.

**step2:**

Copy java "lib" directory from "JAVA\_HOME" folder.

ex:

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

**step3:**

Paste java "lib" directory inside environmental variables.

ex:

right click to my pc --> properties --> advanced system settings -->

environmental variables -->

user variables --> click to new button

variable name : CLASSPATH

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

system variables --> click to new button

variable name : path

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

**step4:**

Check the environmental setup done perfectly or not.

ex: cmd> javap

cmd> java -version

***Steps to develop first java application***

**step1:**

Make sure JDK 1.8 installed successfully.

**step2:**

Make sure environmental setup done perfectly.

**step3:**

create a "javaprog" folder in 'E' drive.

**step4:**

Open the notepad and develop simple Hello world program.

ex:

class Test

{

public static void main(String[] args)

{

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

}

}

**step5:**

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

**step6:**

Open the command prompt from javaprog location.

**step7:**

Compile the program by using below command.

ex:

javac Test.java

|

filename

**step8:**

Execute the program by using below command.

ex:

java Test

|

classname

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

**JDK**

JDK stands for Java Development Kit.

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

**JRE**

JRE stands for Java Runtime Environment.

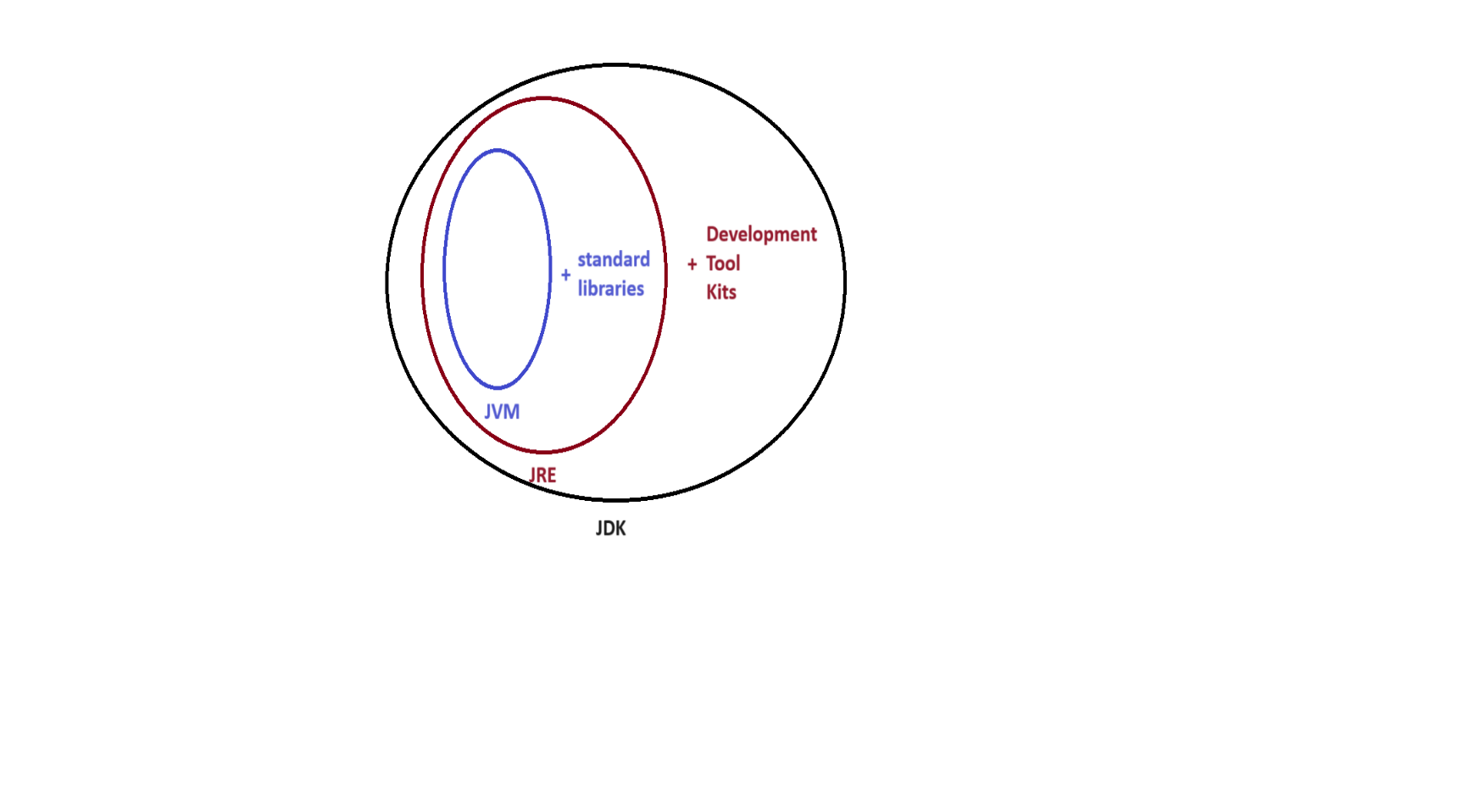
JRE provides very good environment to run java applications only.

**JVM**

JVM stands for Java Virtual Machine.

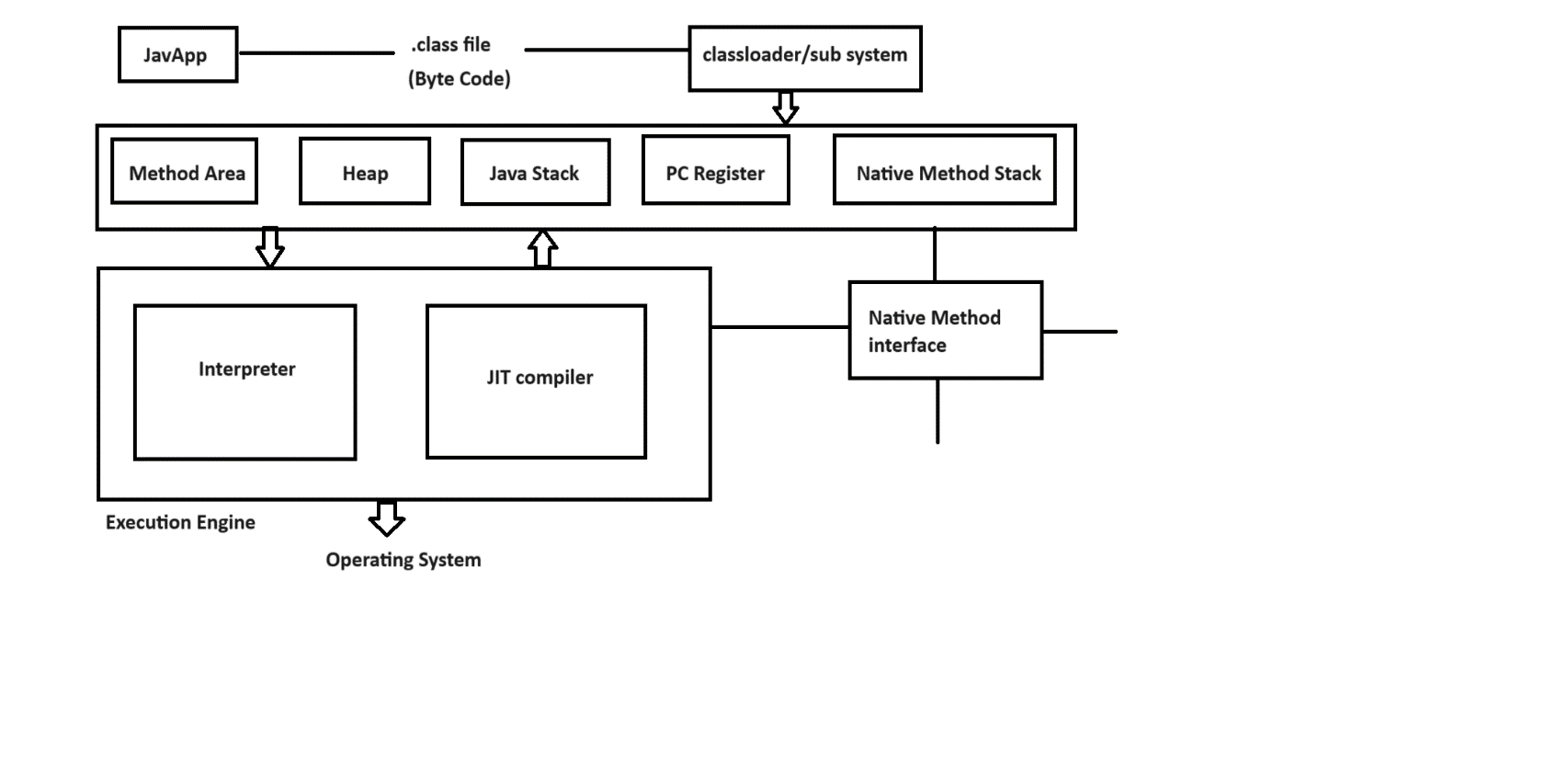
It is an interpreter which is used to execute our program line by line procedure.

**Diagram: class3.1**

****

# 4.Internal Architecture of JVM

**Diagram: class4.1**

****

Java program 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/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 then it will refuse the execution.If they are proper then it will allocate the memory.

We have five types of memories.

**1) Method area**

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

**2) Heap**

Our object creations will store in heap area.

**Note:**

Whenever jvm loads byte code instructions from .class file. It will create method area and heap area automatically.

**3) Java Stack**

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

**4) PC Register**

IT is a program counter register which is used to track 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. But we can't execute native method directly.We required a program called Native method interface.

**Execution Engine**

Executiong engine contains interpreter and jit compiler.

Whenever jvm loads byte code instructions from .class file , it will uses interpreter and jit compile 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) A .class file contains what code \_\_\_ ?**

byte code instructions

**Q) How many memories are there in java?**

We have five memories in java.

1) Method area

2) Heap

3) Java Stack

4) PC Register

5) Native method stack

**Q) What is JIT compiler?**

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

**Q) What is native method in java?**

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

**Q) How many classloaders are there in java?**

We have three predefined classloaders in java.

**1) Bootstrap classloader**

It is used to load rt.jar file.

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

**2) Extension classloader**

It is used to load all the jar files present in ext folder.

ex:

C:\Program Files\Java\jre1.8.0\_181\lib\ext

**3) Application/System classloader**

It is used to load the .class file from CLASSPATH.

# 5.History of java

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

James Gosling, Mike Sheradin and Patrick Naughton were there to develop the project.They have met in a place called Aspan/Colarado to start the work with Graphic System. James Gosling decided to use C and C++ languages to develop the project. But the problem what they have faced is C and C++ languages are system dependent. Then james Gosling decided why don't we 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 a island of an indonasia where first coffee of seed was produced.During the development of project they were consuming lot of coffee's.Hence the symbol of java is a cup of cofee with saucer.

**Interview Questions**

**Q) Who is the creator of java?**

James Gosling

**Q) In which year java was developed?**

In 1995.

**Q) Java originally known as \_\_?**

OAK

**Q) Is java platform dependent or independent?**

platform independent

**Q) Is JVM platform dependent or independent?**

platform dependent

# 6.Identifiers

A name in java is called identifier.

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

ex:

class Test

{

public static void main(String[] args)

{

int x = 10;

System.out.println(x);

}

}

Here Test, main , args and x are identifiers.

**Rules to declare an identifiers**

**Rule1:**

Identifier will accept following characters.

ex:

A-Z

a-z

0-9

\_

$

**Rule2:**

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

ex:

int emp#id; //invalid

int emp\_id; //valid

int emp$al; //valid

**Rule3:**

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

ex:

int a1234; //valid

int \_abcd; //valid

int $alary; //valid

int 1abcd; //invalid

**Rule4:**

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

ex:

int if; //invalid

int else; //invalid

int for; //invalid

**Rule5:**

There is no length limit for an identifier but it is not recommanded to take

more then 15 characters.

**Rule6:**

Every identifier is a case sensitive.

ex:

int number;

int NUMBER;

int NumBer;

**Rule7:**

We can take predefined classes and interfaces as an identifier name but it is not

good programming practice.

ex:

int String=10; //valid

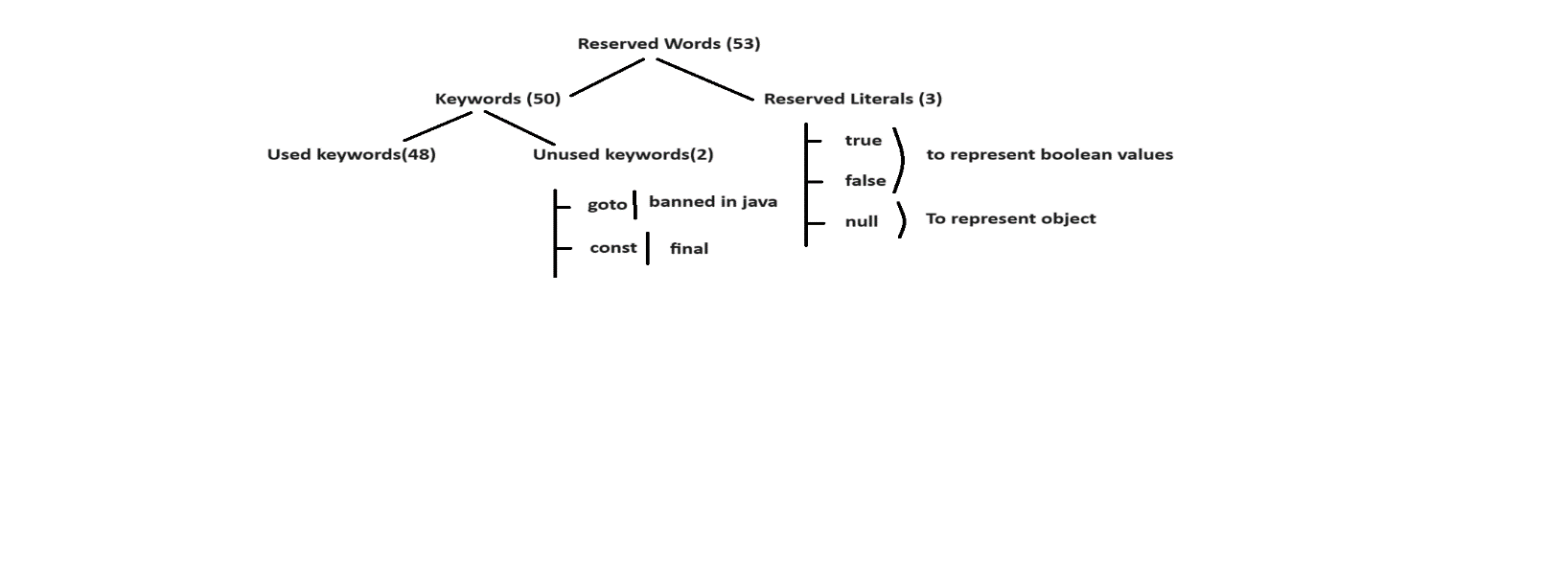
int Runnable=20; //valid

# 7.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 write under lowercase letters only.
* In java, reserved words are divided into two types.

**Diagram: class5.1**

****

**Used keywords with respect to class**

* package
* import
* enum
* interface
* class
* 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 return type**

* void

**Used keywords with respect to modifiers**

* default
* public
* private
* protected
* final
* static
* abstract
* synchronized
* strictfp
* transient
* volatile
* native

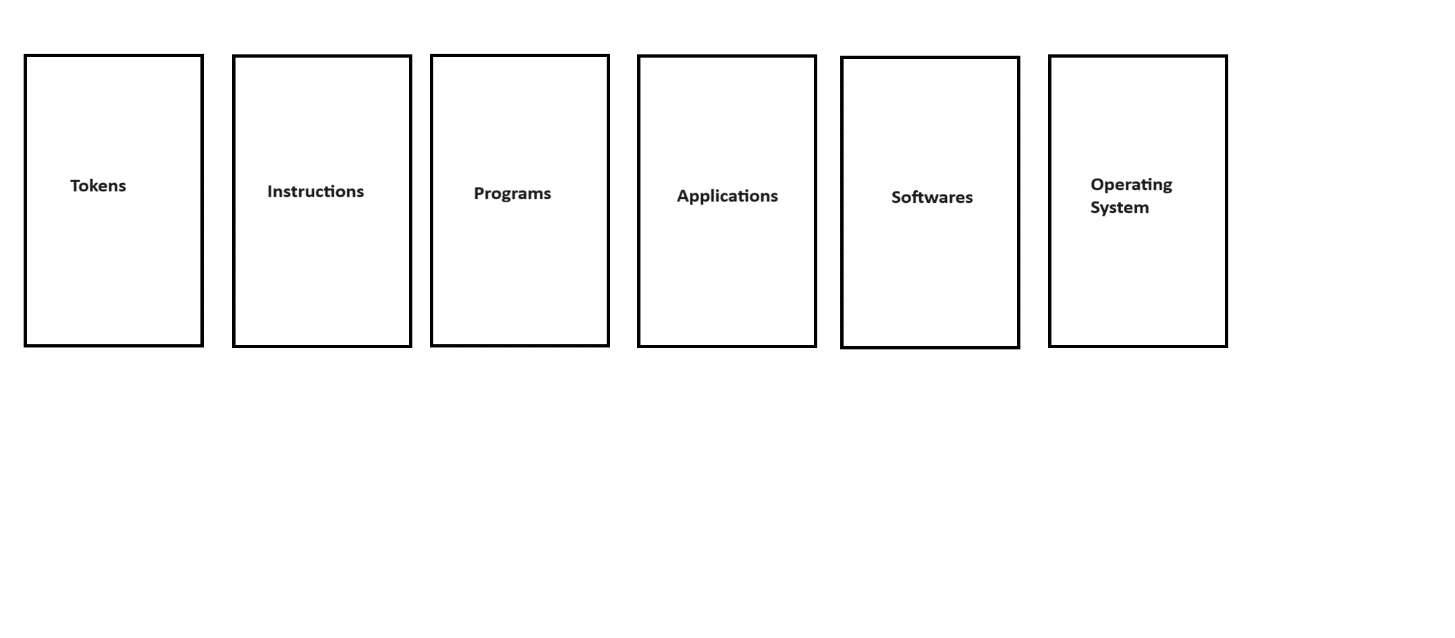
**Used keywords with respect to flow control**

* if
* else
* for
* do
* while
* switch
* case
* break
* continue

**Used keywords with respect to exception handling**

* try
* catch
* throw
* throws
* finally
* assert

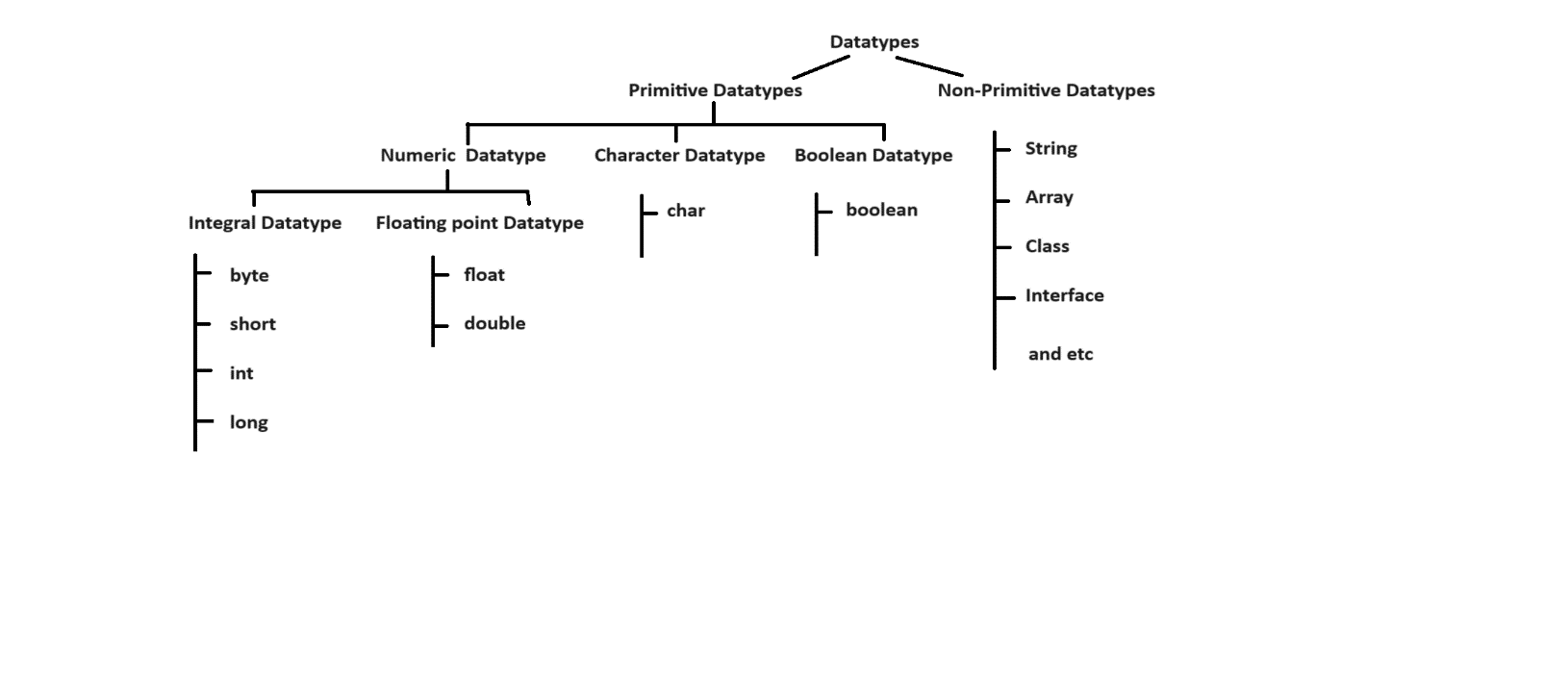
**Diagram: class5.2**



# 8.Datatypes

* Datatype describes what type of value we want to accept in a variable.
* Datatype also tells how much memory has to be created for a variable.
* In java, datatypes are divided into two types.

**Diagram: class6.1**

****

**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="hi";

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

**short**

It is rarely used datatype in java.

Size : 2 bytes (16 bits)

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

**ex:**

1) byte b=10;

short s=b;

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

2) short s=10.5;

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

3) short s="hello";

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

**int**

It is mostly used datatype in java.

Size : 4 bytes (32 bits)

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

**ex:**

1) int i=10.5;

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

2) int i="true";

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=10.5;

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

2) long l="hi";

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

3) long l=true;

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

4) long l='A';

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

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

byte range : -128 to 127

**ex:**

class Test

{

public static void main(String[] args)

{

System.out.println(Byte.MIN\_VALUE);

System.out.println(Byte.MAX\_VALUE);

}

}

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

short range : -32768 to 32767

ex:

class Test

{

public static void main(String[] args)

{

System.out.println(Short.MIN\_VALUE);

System.out.println(Short.MAX\_VALUE);

}

}

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

short range : -2147483648 to 2147483647

class Test

{

public static void main(String[] args)

{

System.out.println(Integer.MIN\_VALUE);

System.out.println(Integer.MAX\_VALUE);

}

}

**Q) int vs Integer memory?**

Integer wrapper class will take more memory then int datatype.

class Test

{

public static void main(String[] args)

{

int i=10;

System.out.println(i);

Integer j=20;

System.out.println(j);

}

}

**float** **double**

If we need 4 to 6 decimal point of accuracy If we need 14 to 16 decimal point

then we need to use float. 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 To declare a double value we need

with 'f' or 'F'. to suffix with 'd' or 'D'.

ex: ex:

10.56f 10.56d

**ex:**

1) float f=10.5f;

System.out.println(f); // 10.5

2) float f=10;

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

3) float f='a';

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

4) float f=true;

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

5) float f="chaitanya";

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

**ex:**

1) double d=10.5d;

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

2) double d=10;

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

3) double d='a';

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

4) double d=true;

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

5) double d="chaitanya";

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

**boolean**

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

Size : (Not Applicable) (1-bit)

Range : (Not Applicable)

**ex:**

1) boolean b="true";

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

2) boolean b=TRUE;

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

3) boolean b=true;

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

**char**

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

Size : 2 bytes (16 bits)

Range : 0 to 65535

ex:

1) char ch='a';

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

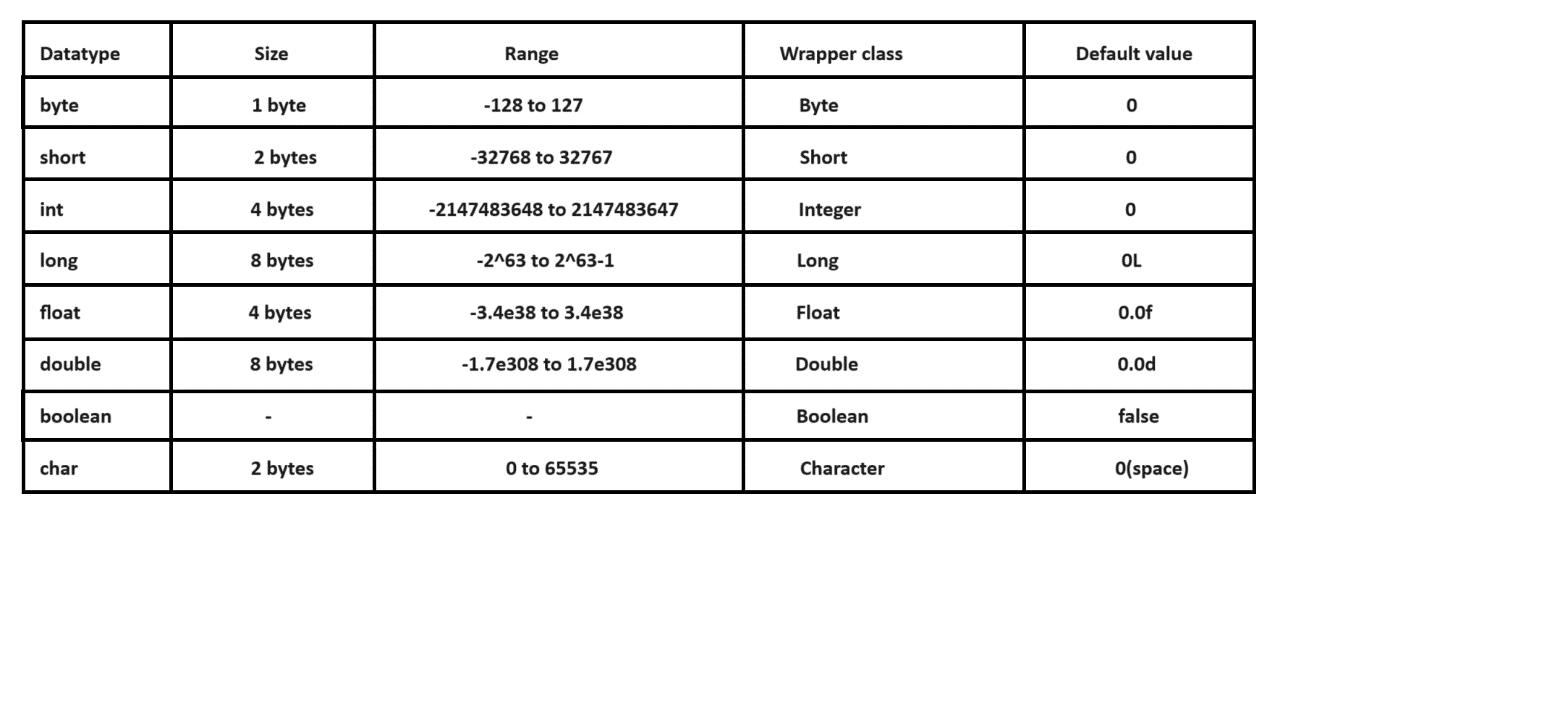
2) char ch="A";

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

3) char ch=65;

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

**Diagram: class7.1**



**Project**

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

A project is a collection of modules.

We have following list of modules.

ex:

registration module

login module

report generation module

admin module

customer module

payment module

and etc.

Every project contains two domains.

**1) Technical domain**

Using which technology we developed our project.

ex:

Java

**2) Functional domain**

It describes state of a project.

ex:

Healthcare domain

Banking domain

ERP domain

insaurance domain

and etc.

# 9.Types of variables

* A name which is given to a memory location is called variable.
* Purpose of variable is used to store the data.
* In java, we have two types of variables.

**1) Primitive variables**

It is used to represent primitive values.

**2) Reference variables**

It is used to represent object reference.

ex:

Student s=new Student

|

reference variable

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

1. Instance variable / Non-Static variable
2. Static variable / Global variable
3. Local variable / Temperory variable / Automatic variable

## 1) Instance variable

* A value of a variable which is varied(changes) from object to object is called instance variable.
* Instance variable will be created at the time object creation and it will destroy at the time of object destruction.Hence scope of instance variable is same as scope of an object.
* Instance variable will store in heap area as a part of an object.
* Instance variable must and should declare immediately after the class but not inside methods,blocks and constructors.
* Instance variable we can access directly from instance area but we can't access directly from static area.
* To access instance variable from static area we need to create object reference.

**ex:1**

class Test

{

//instance variable

int i=10;

public static void main(String[] args)

{

System.out.println(i);

}

}

o/p:

C.T.E : non-static variable i cannot be referenced from a static context

**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 values.

**ex:3**

class Test

{

//instance variable

boolean b;

public static void main(String[] args)

{

Test t=new Test();

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

}

}

**ex:4**

class Test

{

//instance variable

int i=10;

public static void main(String[] args)

{

Test t1=new Test();

Test t2=new Test();

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

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

t1.i=20;

System.out.println(t1.i);//20

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

}

}

**ex:5**

class Test

{

public static void main(String[] args)

{

Test t=new Test();

t.m1();

}

//non-static method

public void m1()

{

System.out.println("instance method");

}

}

**Interview Question**

**Q) Jack and John are living in a small town.Both are studing in a same school in a same class.One day while going to school they saw one poor person(beggar). Jack and John both they decided to help that poor guy. Jack gave Rs. 100 from his pocket and John gave Rs. 200 from his side.Write a java console to display how much money they gave to beggar?**

ex:

class Test

{

//instance variables

int a=100;

int b=200;

public static void main(String[] args)

{

Test t=new Test();

t.sum();

}

//non-static method

public void sum()

{

int c=a+b;

System.out.println(c);

}

}

**Q) Jack and John are living in a small town.Both are studing in a same school in a same class.One day while going to exam Jack gave 10 rupees to helpless guy.Due his good deed he secured good marks in the subject.He told the same story to John how he secured highest marks in that subject.No John also decided to give double then Jack contribution to helpless guy.Now a java console to calculate john amount?**

class Test

{

//instance variable

int money=10;

public static void main(String[] args)

{

Test t=new Test();

t.find();

}

//non-static method

public void find()

{

int total=money\*2;

System.out.println(total);

}

}

## 2) Static variable

* A value of a variable which is not varied from object to object is called static variable.
* Static variable will be created at the time of classloading and it will destroy at the time of
* class unloading.Hence scope of static variable is same as scope of a .class file.
* Static variable will store in method area.
* Static variable must and should declare immediately after the class using static keyword but not inside methods,blocks and constructors.
* Static variable we can access directly from instance area and static area.
* Static variable can access by using object reference and class name.

**ex:1**

class Test

{

//static variable

static int i=10;

public static void main(String[] args)

{

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

Test t=new Test();

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

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

}

}

**Note:**

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

**ex:2**

class Test

{

//static variable

static String s;

public static void main(String[] args)

{

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

}

}

**ex:3**

class Test

{

//static variable

static int i=10;

public static void main(String[] args)

{

Test t1=new Test();

Test t2=new Test();

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

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

t1.i=20;

System.out.println(t1.i);//20

System.out.println(t2.i);//20

}

}

**ex:4**

class Test

{

public static void main(String[] args)

{

m1();

Test t=new Test();

t.m1();

Test.m1();

}

//static method

public static void m1()

{

System.out.println("static-method");

}

}

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

class Test

{

//static variable

static int a=10;

static int b=20;

public static void main(String[] args)

{

sum();

}

//static method

public static void sum()

{

int c=a+b;

System.out.println(c);

}

}

## 3) Local variable

* To meet temperory requirements, programmer will declare some variables inside methods, blocks and constructors such type of variables are called local variables.
* Local variable will be created as a part of execution block and it will destroy when execution block is executed.Hence scope of local variable is same as scope of execution block where it is declared.
* Local variable will store in java stack.

**ex:1**

class Test

{

public static void main(String[] args)

{

//local variable

int i=10;

System.out.println(i);

}

}

**Note:**

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

**ex:2**

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 one modifier i.e **final**.

**ex:3**

class Test

{

public static void main(String[] args)

{

//local variable

final int i=10;

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

}

}

**Interview Questions**

**Q) What is Java?**

Java is a versatile programming language which is used to develop various types of applications.

**Q) What are the features of Java?**

We have following important features in java.

1) Simple

2) Object oriented

3) High level

4) Platform independent

5) Portable

6) Highly Secured

7) Architecture Neutral

8) Robust

9) Multithreaded

10) Dynamic

11) Distributed

and etc.

# 10.Main method

* Our program contains main method or not.Either it is propery declare or not. It is not a responsibility of a compiler to check.It is a liability of a JVM to check for main method.
* If JVM won't find main method then it will throw one runtime error called main method not found.
* JVM looks for main method with following signature.

ex:

public static void main(String[] args)

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

**Q) Can you explain main method in java?**

**public**

JVM wants to call this method from anywhere.

**static**

JVM wants to call this method without using object reference.

**void**

Main method does not return anything to JVM

**main**

It is an identifier given to a main method.

**String[] args**

It is a command line argument.

We can perform following changes in main.

1) Order of modifiers is not important.Incase of public static we can take static public also.

ex: static public void main(String[] args)

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

ex:

public static void main(String[] args)

public static void main(String []args)

public static void main(String args[])

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

ex:

public static void main(String... args)

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

ex:

public static void main(String[] ihub)

5) Main method will accept following modifiers.

ex:

synchronized

strictfp

final

# 11.Command Line Argument

* Arguments which are passing through command prompt such type of arguments are called command line arguments.
* In command line arguments we need to ask 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]);

}

}

**Interview Questions**

**Q) How many modules are there in java?**

We have three modules in java.

1) JSE/J2SE

2) JEE/J2EE

3) JME/J2ME

**Q) What is Literal?**

It is a synthetic representation of boolean, character, numeric, or string data.

A value which we 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

**Q) What is Java ?**

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

**or**

It is a versatile programming language which is used to develop various types of applications.

# 12.System.out.println()

* It is a output statement in java.
* Whenever we want to display variables , data or userdefined messages then we need to use output statement.

syntax:

static variable

|

System.out.println();

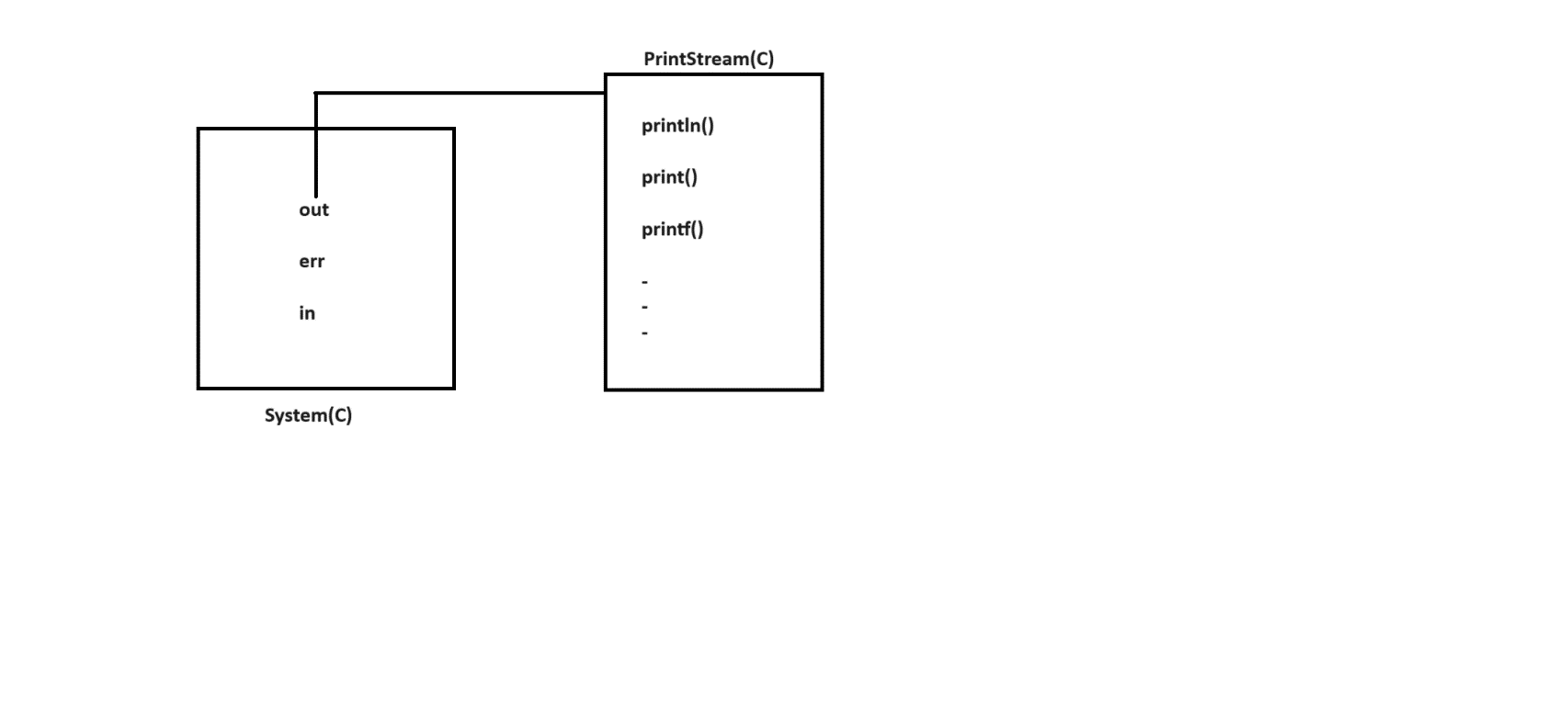
| |

predefined predefined method

final

class

**Diagram: class11.1**



ex:

class Test

{

public static void main(String[] args)

{

System.out.println("stmt1");

System.out.print("stmt2");

System.out.printf("stmt3");

}

}

**Various ways to display the data**

1)

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

2)

int i=10;

System.out.println(i);

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

3)

int i=10,j=20;

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

System.out.println(i+" and "+j);

4)

int i=10,j=20,k=30;

System.out.println(i+" "+j+" "+k);

**Q) What is the difference between System.out.println() and System.err.println() ?**

**System.out.println()**

It is used to redirect our output to console.

ex:

class Test

{

public static void main(String[] args)

{

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

}

}

o/p:

javac Test.java

java Test

**System.err.println()**

* It is used to redirect our output to console as well as to a physical file.

ex:

class Test

{

public static void main(String[] args)

{

System.err.println("Hello World");

}

}

o/p:

javac Test.java

java Test 2>abc.txt

# 13.Fully Qualified Name

* Full qualified name means we will declare a class or interface along with package name.
* Full qualified name is used to improve the readability of our code.

ex:

class Test

{

public static void main(String[] args)

{

java.util.Date d=new java.util.Date();

System.out.println(d);

}

}

**To see the list of methods present in Date class we need to javap tool.**

**ex:**

**cmd> javap java.util.Date**

ex:

class Test

{

public static void main(String[] args)

{

java.util.Date date=new java.util.Date();

int h=date.getHours();

int m=date.getMinutes();

int s=date.getSeconds();

System.out.println(h+":"+m+":"+s);

}

}

ex:

class Test

{

public static void main(String[] args)

{

java.util.Date date=new java.util.Date();

int d=date.getDate();

int m=date.getMonth()+1;

date.setYear(2024);

int y=date.getYear();

System.out.println(d+"/"+m+"/"+y);

}

}

# 13.Import statements

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

In java, import statements are divided into three types.

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:1**

import java.util.Date;

class Test

{

public static void main(String[] args)

{

Date d=new Date();

System.out.println(d);

}

}

Whenever we want to display and date and time we need to use java.time package.

A java.time package introduced in java 8.

**ex:2**

import java.time.LocalDate;

import java.time.LocalTime;

class Test

{

public static void main(String[] args)

{

LocalDate date=LocalDate.now();

System.out.println(date);

LocalTime time=LocalTime.now();

System.out.println(time);

}

}

## 2) Implicit class import

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

ex:

import java.time.\*;

class Test

{

public static void main(String[] args)

{

LocalDate date=LocalDate.now();

System.out.println(date);

LocalTime time=LocalTime.now();

System.out.println(time);

}

}

## 3) Static import

* Using static import we can call/access static members directly.
* Often use of static import makes our program complex and unreadable.

**ex:**

import static java.lang.System.\*;

class Test

{

public static void main(String[] args)

{

out.println("stmt1");

out.println("stmt2");

out.println("stmt3");

}

}

**ex:**

import static java.lang.System.\*;

class Test

{

public static void main(String[] args)

{

out.println("stmt1");

exit(0);

out.println("stmt2");

}

}

**Basic Java Programs**

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

import java.util.Scanner;

class Example1

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter the First Number :");

int a=sc.nextInt();

System.out.println("Enter the Second Number :");

int b=sc.nextInt();

//logic

int c=a+b;

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

}

}

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

import java.util.Scanner;

class Example2

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter the First Number :");

int a=sc.nextInt();

System.out.println("Enter the Second Number :");

int b=sc.nextInt();

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

}

}

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

import java.util.Scanner;

class Example3

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter the number :");

int n=sc.nextInt();

//logic

int square=n\*n;

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

}

}

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

import java.util.Scanner;

class Example4

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter the number :");

int n=sc.nextInt();

//logic

int cube=n\*n\*n;

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

}

}

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

import java.util.Scanner;

class Example5

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter the radius :");

int r=sc.nextInt();

//logic

float area=3.14f\*r\*r;

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

}

}

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

import java.util.Scanner;

class Example6

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter the radius :");

int r=sc.nextInt();

//logic

float perimeter=2\*3.14f\*r;

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

}

}

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

import java.util.Scanner;

class Example7

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter the first number :");

int a=sc.nextInt();//10

System.out.println("Enter the second number :");

int b=sc.nextInt();//20

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

//logic

int temp=a;

a=b;

b=temp;

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

}

}

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

import java.util.Scanner;

class Example8

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter the first number :");

int a=sc.nextInt();//10

System.out.println("Enter the second number :");

int b=sc.nextInt();//20

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

//logic

a=a+b;

b=a-b;

a=a-b;

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

}

}

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

ex:

import java.util.Scanner;

class Example9

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter the CGPA :");

float cgpa=sc.nextFloat();

//logic

float percentage=cgpa\*9.5f;

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

}

}

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

import java.util.Scanner;

class Example10

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter the salary :");

int salary=sc.nextInt();

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

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

}

}

**Assignment**

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

class Marks

{

public static void main(String[] args)

{

int m1=60,m2=49,m3=48,m4=58,m5=68,m6=84;

int total=m1+m2+m3+m4+m5+m6;

System.out.println("Sum of total marks="+total);

float avg=(float)total/6;

System.out.println("Average of marks="+avg);

}

}

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

import java.util.Scanner;

class Rectangle

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter Length of Rectangle:");

int l=sc.nextInt();

System.out.println("Enter the Breath of Rectangle:");

int b=sc.nextInt();

//logic

float area=(float)l\*b;

System.out.println("Area of Rectangle="+area);

}

}

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

import java.util.Scanner;

class Rectangle

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter Breath of Triangle:");

int b=sc.nextInt();

System.out.println("Enter the Length of Triangle:");

int h=sc.nextInt();

//logic

float area=(float)1/2\*b\*h;

System.out.println("Area of Triangle ="+area);

}

}

**Editplus Editor**

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

# 14.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 in a bigger variable then we need to use implicit typecasting.
* A compiler is responsible to perform implicit typecasting.
* There is no possibility to loss the information.
* It is also known as Widening or Upcasting.
* We can perform implicit typecasting as follow.

ex:

byte --> short

-->

int --> long --> float --> double

-->

char

**ex:1**

class Test

{

public static void main(String[] args)

{

byte b=10;

int i=b;

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

}

}

**ex:2**

class Test

{

public static void main(String[] args)

{

char ch='a';

int i=ch;

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

}

}

**ex:3**

class Test

{

public static void main(String[] args)

{

int i=65;

float f=i;

System.out.println(f);// 65.0

}

}

**2) Explicit typecasting**

* If we want to store bigger value into a smaller variable then we need to use explicit typecasting.
* A programmer is responsible to perform explicit typecasting.
* There is a possibility to loss the information.
* It is also known as Narrowing or Downcasting.
* We can perform explicit typecasting as follow.

ex:

byte <-- short

<--

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

<--

char

**ex:1**

class Test

{

public static void main(String[] args)

{

float f=10.56f;

int i=(int)f;

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

}

}

**ex:2**

class Test

{

public static void main(String[] args)

{

int i=65;

char ch=(char)i;

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

}

}

**ex:3**

class Test

{

public static void main(String[] args)

{

int i=130;

byte b=(byte)i;

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

}

}

## Types of blocks in java

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

In java, we have three types of blocks.

1) Instance block

2) Static block

3) Local block

**1) Instance block**

* A instance block is used to initialize the instance variables.
* A instance block must and should declare immediately after the class but not inside methods and constructors.
* A instance block will execute when we create an object.

syntax:

//instance block

{

-

- //set of statements

-

}

**ex:1**

class Test

{

//instance block

{

System.out.println("instance-block");

}

public static void main(String[] args)

{

System.out.println("main-method");

}

}

o/p:

main-method

**ex:2**

class Test

{

//instance block

{

System.out.println("instance-block");

}

public static void main(String[] args)

{

System.out.println("main-method");

Test t=new Test();

}

}

o/p:

main-method

instance-block

**ex:3**

class Test

{

//instance block

{

System.out.println("instance-block");

}

public static void main(String[] args)

{

Test t1=new Test();

System.out.println("main-method");

Test t2=new Test();

}

}

o/p:

instance-block

main-method

instance-block

**ex:4**

class Test

{

//instance variable

int i;

//instance block

{

i=100;

}

public static void main(String[] args)

{

Test t=new Test();

System.out.println(t.i);

}

}

**2) Static block**

* A static block is used to initialize the values to static variables.
* A static block must and should declare immediately after the class using static keyword but not inside methods and constructors.
* A static block will be executed at the time of classloading.

**ex:1**

class Test

{

//static block

static

{

System.out.println("static-block");

}

public static void main(String[] args)

{

System.out.println("main-method");

}

}

o/p:

static-block

main-method

**ex:2**

class Test

{

//instance block

{

System.out.println("instance-block");

}

//static block

static

{

System.out.println("static-block");

}

public static void main(String[] args)

{

Test t=new Test();

System.out.println("main-method");

}

}

o/p:

static-block

instance-block

main-method

**ex:3**

class Test

{

//static variable

static int i;

//static block

static

{

i=200;

}

public static void main(String[] args)

{

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

}

}

**3) Local block**

* A local block is used to initialize the values to local variables.
* A local block must and should declare inside methods and constructors.
* A local block will execute just like normal statement.

syntax:

//local block

{

-

- //set of statements

-

}

**ex:1**

class Test

{

public static void main(String[] args)

{

System.out.println("stmt1");

//local block

{

System.out.println("stmt2");

}

System.out.println("stmt3");

}

}

**ex:2**

class Test

{

public static void main(String[] args)

{

//local variables

int i;

//local block

{

i=300;

}

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

}

}

**Interview Question**

**Q) Can we execute java program without main method?**

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

**ex:**

class Test

{

//static block

static

{

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

System.exit(0);

}

}

**Q) Is java purely object oriented or not?**

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

# 15.Operators

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

ex:

c = a + b;

Here a,b and c are operands.

Here = and + are operators.

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

We have following list of operators in java.

1) Assignment operators

2) Ternary operators / Conditional operators

3) Logical operators

4) Bitwise operators

5) Arithmetic operators

6) Relational operators

7) Shift operators

8) Unary operators

**1) Assignment operators**

class Test

{

public static void main(String[] args)

{

int i=10;

i=20;

i=30;

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

}

}

Note:

Reinitialization is possible in java.

**ex:**

class Test

{

public static void main(String[] args)

{

final int i=10;

i=20;

i=30;

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

}

}

Note:

final variables we can't modify

**ex:**

class Test

{

public static void main(String[] args)

{

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

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

}

}

**ex:**

class Test

{

//global variable

static int i=10;

public static void main(String[] args)

{

//local variable

int i=20;

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

}

}

Note:

Here priority goes to local variable

**ex:**

class Test

{

public static void main(String[] args)

{

int i=10;

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

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

}

}

**ex:**

class Test

{

public static void main(String[] args)

{

int i=10;

i\*=5+3; // i = i \* 8

System.out.println(i); // 80

}

}

**ex:**

class Test

{

public static void main(String[] args)

{

int i=10;

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

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

}

}

**ex:**

class Test

{

public static void main(String[] args)

{

int i=10;

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

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

}

}

**ex:**

class Test

{

public static void main(String[] args)

{

int i=10;

i%=2; // i = i % 2;

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

}

}

**ex:**

class Test

{

public static void main(String[] args)

{

int i=10;

i%=20; // i = i % 20;

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

}

}

**ex:**

class Test

{

public static void main(String[] args)

{

int i=10;

i-=5;

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

}

}

**ex:**

class Test

{

public static void main(String[] args)

{

int i=j=5;

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

}

}

**ex:**

class Test

{

public static void main(String[] args)

{

System.out.println((-10 + 5)); // -5

}

}

**2) Ternary operators**

syntax:

(condition)?value1:value2;

**ex:**

class Test

{

public static void main(String[] args)

{

String str=(true)?"Hi":"Bye";

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

}

}

**ex:**

class Test

{

public static void main(String[] args)

{

String str=(false)?"Hi":"Bye";

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

}

}

**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 using ternary operator?**

import java.util.Scanner;

class Test

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter the first number :");

int a=sc.nextInt();

System.out.println("Enter the second number :");

int b=sc.nextInt();

//logic

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

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

}

}

**Q) Write a java program to find out greatest of three numbers using ternary operator?**

import java.util.Scanner;

class Test

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter the first number :");

int a=sc.nextInt();

System.out.println("Enter the second number :");

int b=sc.nextInt();

System.out.println("Enter the third number :");

int c=sc.nextInt();

//logic

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

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

}

}

**3) Logical operators**

**i) logical AND operator (&&)**

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

***Truth table***

T T = T

T F = F

F T = F

F F = F

**ex:**

class Test

{

public static void main(String[] args)

{

boolean b1= true && true;

System.out.println(b1); //true

boolean b2= true && false;

System.out.println(b2); //false

boolean b3= false && true;

System.out.println(b3); //false

boolean b4= false && false;

System.out.println(b4); //false

}

}

**ex:**

class Test

{

public static void main(String[] args)

{

boolean b=((5>2) && (10<15))?true:false;

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

}

}

**ex:**

class Test

{

public static void main(String[] args)

{

boolean b=((5>2) && (10<5))?true:false;

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

}

}

**i) logical OR operator (||)**

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

***Truth table***

T T = T

T F = T

F T = T

F F = F

**ex:**

class Test

{

public static void main(String[] args)

{

boolean b1=true || true;

System.out.println(b1);//true

boolean b2=true || false;

System.out.println(b2); //true

boolean b3=false || true;

System.out.println(b3);//true

boolean b4=false || false;

System.out.println(b4);//false

}

}

**ex:**

class Test

{

public static void main(String[] args)

{

boolean b=((5>20) || (6<4))?true:false;

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

}

}

**ex:**

class Test

{

public static void main(String[] args)

{

boolean b=((5>20) || (6<40))?true:false;

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

}

}

**ex:**

class Test

{

public static void main(String[] args)

{

boolean b=((5>2) && (6<10) || false)?true:false;

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

}

}

**iii) logical NOT operator (!)**

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

}

}

**How to convert decimal to binary**

Decimal No : 10

Binary No : 1010

2|10

--- 0

2|5

--- 1

2|2

--- 0 ^

1 |

-----------------

1 0 1 0

**How to convert binary to decimal**

Binary No : 1010

Decimal No : 10

1 0 1 0

<----

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

0 + 2 + 0 + 8

10

**4) Bitwise operators**

**Bitwise AND operator (&&)**

Bitwise AND operator deals with binary numbers.

**Truth table**

T T = T

T F = F

F T = F

F F = F

**ex:**

class Test

{

public static void main(String[] args)

{

int a=10,b=15;

int c = a & b;

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

}

}

/\*

10 - 1010

15 - 1111

----------

& - 1010 <---

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

0+2+0+8 =10

\*/

ex:

class Test

{

public static void main(String[] args)

{

int a=10,b=5;

int c = a & b;

System.out.println(c); //0

}

}

/\*

10 - 1010

5 - 0101

----------

& - 0000

\*/

**Bitwise OR operator (|)**

Bitwise OR operator deals with binary numbers.

**Truth table**

T T = T

T F = T

F T = T

F F = F

**ex:**

class Test

{

public static void main(String[] args)

{

int a=10,b=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 opertor 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=5;

int c = a ^ b;

System.out.println(c); //15

}

}

/\*

10 - 1010

5 - 0101

----------

^ - 1111

\*/

**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

\*/

**Bitwise Not operator (~)**

**ex:**

class Test

{

public static void main(String[] args)

{

int i=~10;

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

}

}

**ex:**

class Test

{

public static void main(String[] args)

{

int i=~23;

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

}

}

**ex:**

class Test

{

public static void main(String[] args)

{

int i=~98;

System.out.println(i); //-99

}

}

**ex:**

class Test

{

public static void main(String[] args)

{

int i=~(-10);

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

}

}

**Q) Write a java program to find out swapping of two numbers using bitwise operator?**

import java.util.Scanner;

class Test

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter the first number :");

int a=sc.nextInt();

System.out.println("Enter the second number :");

int b=sc.nextInt();

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

a = a ^ b;

b = a ^ b;

a = a ^ b;

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

}

}

**5) Arithmetic operators**

% - modules

/ - division

\* - multiplication

+ - addition

- - subtraction

**ex:**

class Test

{

public static void main(String[] args)

{

int i=6+7%2+8/4+7\*5+9/10+5-20;

System.out.println(i); //29

}

}

/\*

6 + 7%2 + 8/4 + 7\*5 + 9/10 + 5 - 20

6 + 1 + 2 + 35 + 0 + 5 - 20

49 - 20

29

\*/

**6) Relational operators**

class Test

{

public static void main(String[] args)

{

System.out.println(5>2);// true

System.out.println(5<2);// false

System.out.println(5>=2);// true

System.out.println(5<=5);// 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

}

}

**7) Shift operators**

**Right Shift operator (>>)**

10 >> 1 = 10/2

10 >> 2 = 10/4

10 >> 3 = 10/8

10 >> 4 = 10/16

-

**ex:**

class Test

{

public static void main(String[] args)

{

int i= 10 >> 3; // 10 / 8(2\*2\*2)

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

}

}

**ex:**

class Test

{

public static void main(String[] args)

{

int i= 20 >> 2; // 20 / 4

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

}

}

**ex:**

class Test

{

public static void main(String[] args)

{

int i= 10 >> 7; //10 / 128

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

}

}

**Left Shift operator (<<)**

10 << 1 = 10\*2

10 << 2 = 10\*4

10 << 3 = 10\*8

10 << 4 = 10\*16

**ex:**

class Test

{

public static void main(String[] args)

{

int i= 10 << 3; //10 \* 8 (2\*2\*2)

System.out.println(i); // 80

}

}

**ex:**

class Test

{

public static void main(String[] args)

{

int i= 20 << 4; // 20 \* 16

System.out.println(i); // 320

}

}

**8.Unary operators**

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

We have two type of increment operators.

**1) post-increment**

ex:

i++;

**2) Pre-increment**

ex:

++i;

We have two types of decrement operators.

**1) Post-decrement**

ex:

i--;

**2) pre-decrement**

ex:

--i;

**POST increment/decrement operators**

**Rule1:** First Take

**Rule2:** Then Change

**ex:**

class Test

{

public static void main(String[] args)

{

int i=10;

i++;

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

}

}

**ex:**

class Test

{

public static void main(String[] args)

{

int i=10;

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

}

}

**ex:**

class Test

{

public static void main(String[] args)

{

int i=10;

int j=i++;

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

}

}

**ex:**

class Test

{

public static void main(String[] args)

{

int i=10;

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

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

}

}

**ex:**

class Test

{

public static void main(String[] args)

{

int i=10;

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

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

}

}

**Pre increment/decrement operators**

**Rule1:** First Change

**Rule2:** Then Take

**ex:**

class Test

{

public static void main(String[] args)

{

int i=10;

++i;

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

}

}

**ex:**

class Test

{

public static void main(String[] args)

{

int i=10;

System.out.println(++i);

}

}

ex:

class Test

{

public static void main(String[] args)

{

int i=10;

int j = ++i;

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

}

}

**ex:**

class Test

{

public static void main(String[] args)

{

int i=10;

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

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

}

}

**ex:**

class Test

{

public static void main(String[] args)

{

int i=10;

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

}

}

**ex:**

class Test

{

public static void main(String[] args)

{

int i=10;

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

} }

**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;

++(++i);

System.out.println(i);//

}

}

**ex:**

class Test

{

public static void main(String[] args)

{

byte b=127;

b++;

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

}

}

# 16.Control Statements

* Control statement enables the programmer to control flow of the program.
* Control statement allows us to make decisions, to jump from one section of code to another section and to execute the code repeatedly.
* In java, we have four types of control statements.

1) Decision Making statement

2) Selection statement

3) Iteration statement

4) Jump statement

## 1) Decision Making statement

* It is used to declare the conditions in our program.
* Decision making statement is possible by using following ways.

i) if stmt

ii) if else stmt

iii) if else if ladder

iv) nested if stmt

**i) if stmt**

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

syntax:

if(condition)

{

-

- //code to be execute

-

}

**ex:**

class Test

{

public static void main(String[] args)

{

System.out.println("stmt1");

if(5>2)

{

System.out.println("stmt2");

}

System.out.println("stmt3");

}

}

o/p:

stmt1

stmt2

stmt3

**ex:**

class Test

{

public static void main(String[] args)

{

System.out.println("stmt1");

if(!(5>2))

{

System.out.println("stmt2");

}

System.out.println("stmt3");

}

}

o/p:

stmt1

stmt3

**ex:**

class Test

{

public static void main(String[] args)

{

if((5>2) && (6<2))

System.out.println("stmt1");

System.out.println("stmt2");

System.out.println("stmt3");

}

}

o/p:

stmt2

stmt3

**ex:**

class Test

{

public static void main(String[] args)

{

if((5>2) || (6<2))

System.out.println("stmt1");

System.out.println("stmt2");

System.out.println("stmt3");

}

}

o/p:

stmt1

stmt2

stmt3

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

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(10 != 20)

{

System.out.println("stmt2");

}

else

{

System.out.println("stmt3");

}

System.out.println("stmt4");

}

}

o/p:

stmt1

stmt2

stmt4

**ex:**

class Test

{

public static void main(String[] args)

{

System.out.println("stmt1");

if(10 != 10)

{

System.out.println("stmt2");

}

else

{

System.out.println("stmt3");

}

System.out.println("stmt4");

}

}

o/p:

stmt1

stmt3

stmt4

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

import java.util.Scanner;

class Test

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter the age :");

int age=sc.nextInt();

if(age>=18)

System.out.print("U r eligible to vote");

else

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

}

}

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

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.print("It is a even number");

else

System.out.println("It is a 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)

System.out.print("It is odd number");

else

System.out.println("It is not odd number");

}

}

**Q) Write a java program to check given number is +ve or -ve ?**

import java.util.Scanner;

class Test

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter the number :");

int n=sc.nextInt();

if(n==0)

{

System.out.println("It is not a positive or negative number");

System.exit(0);

}

if(n>0)

System.out.print("It is a positive number");

else

System.out.println("It is a 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 Leap year :");

int year=sc.nextInt();

if(((year%4==0) && (year%100!=0)) || year%400==0)

System.out.print("It is a leap year");

else

System.out.println("It is not a leap year");

}

}

**iii) if else if ladder**

It will execute the source code based on multiple conditions.

syntax:

if(cond1)

{

- //code to be execute if cond1 is true

}

else if(cond2)

{

- //code to be execute if cond2 is true

}

else if(cond3)

{

- //code to be execute if 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 a police number");

else if(option==103)

System.out.println("It is a enquiry number");

else if(option==108)

System.out.println("It is a emergency number");

else

System.out.println("Invalid option");

}

}

**Q) Write a java program to find out given alphabet is a upper case letter, lower case letter , digit or a special symbol?**

import java.util.Scanner;

class Test

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter the alphabet :");

char ch=sc.next().charAt(0);

if(ch>='A' && ch<='Z')

System.out.println("It is uppercase letter");

else if(ch>='a' && ch<='z')

System.out.println("It is lowercase letter");

else if(ch>='0' && ch<='9')

System.out.println("It is a digit");

else

System.out.println("It is a special symbol");

}

}

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

import java.util.Scanner;

class Test

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter the alphabet :");

char ch=sc.next().charAt(0);

if(ch=='a' || ch=='A')

System.out.println("It is a vowel");

else if(ch=='e' || ch=='E')

System.out.println("It is a vowel");

else if(ch=='i' || ch=='I')

System.out.println("It is a vowel");

else if(ch=='o' || ch=='O')

System.out.println("It is a vowel");

else if (ch=='u' || ch=='U')

System.out.println("It is a vowel");

else

System.out.println("It is not a vowel");

}

}

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

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

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

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

iv) if average is less then 35 then failed.

**ex:**

class Test

{

public static void main(String[] args)

{

int m1=89,m2=45,m3=55,m4=62,m5=59,m6=74;

int total=m1+m2+m3+m4+m5+m6;

float avg=(float)total/6;

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

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

if(avg>=70)

System.out.println("Grade : A grade");

else if(avg>=50)

System.out.println("Grade : B grade");

else if(avg>=35)

System.out.println("Grade : C grade");

else

System.out.println("Grade : Failed");

}

}

**Assignment**

**Q) Write a java program to check given alphabet is a vowel or not using if else stmt?**

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=='e' || ch=='i' || ch=='o' || ch=='u')

System.out.println("It is a vowel");

else

System.out.println("It is not a vowel");

}

}

**iv)Nested if stmt**

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

syntax:

if(condition)

{

if(condition)

{

-

- //code to be execute

-

}

}

**ex:**

class Test

{

public static void main(String[] args)

{

System.out.println("stmt1");

if(!(5>20))

{

System.out.println("stmt2");

if(true)

{

System.out.println("stmt3");

}

System.out.println("stmt4");

}

System.out.println("stmt5");

}

}

o/p:

stmt1

stmt2

stmt3

stmt4

stmt5

**ex:**

class Test

{

public static void main(String[] args)

{

System.out.println("stmt1");

if(!(5>20))

{

System.out.println("stmt2");

if(false)

{

System.out.println("stmt3");

}

System.out.println("stmt4");

}

System.out.println("stmt5");

}

}

o/p:

stmt1

stmt2

stmt4

stmt5

ex:

class Test

{

public static void main(String[] args)

{

System.out.println("stmt1");

if(!(5>2))

{

System.out.println("stmt2");

if(true)

{

System.out.println("stmt3");

}

System.out.println("stmt4");

}

System.out.println("stmt5");

}

}

o/p:

stmt1

stmt5

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

{

case value1: //code to be execute

break stmt

case value2: //code to be execute

break stmt

-

default: //code to be execute if all cases are false

}

**ex:**

import java.util.Scanner;

class Test

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter the option :");

int option=sc.nextInt();

switch(option)

{

case 100: System.out.println("It is police number");

break;

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

break;

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

break;

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

}

}

}

Declaration of break statement is optional.If we won't defined 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");

}

}

}

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

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

import java.util.Scanner;

class Test

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter the alphabet :");

char ch=sc.next().charAt(0);

switch(ch)

{

case 'a': System.out.println("It is a vowel"); break;

case 'e': System.out.println("It is a vowel"); break;

case 'i': System.out.println("It is a vowel"); break;

case 'o': System.out.println("It is a vowel"); break;

case 'u': System.out.println("It is a vowel"); break;

default: System.out.println("It is a consonent");

}

}

}

**ex:**

import java.util.Scanner;

class Test

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter the string :");

String str=sc.next();

switch(str)

{

case "one": System.out.println("January"); break;

case "two": System.out.println("February"); break;

case "three": System.out.println("March"); break;

case "four": System.out.println("April"); break;

case "five": System.out.println("May"); break;

default: System.out.println("Coming Soon...");

}

}

}

## 3) Iteration statement

* Iteration statement is used to execute the code repeately.
* Iteration statement is possible by using LOOPS.
* We have four types of loops in java.

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);

}

}

**ex:**

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 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 perform sum of 10 natural numbers?**

class Test

{

public static void main(String[] args)

{

int i=1,sum=0;

do

{

sum=sum+i;

i++;

}

while (i<=10);

System.out.println(sum);

}

}

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

input:

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

output:

120

**ex:**

import java.util.Scanner;

class Test

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter the number :");

int n=sc.nextInt();

int i=n,fact=1;

do

{

fact=fact\*i;

i--;

}

while (i>=1);

System.out.println(fact);

}

}

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

import java.util.Scanner;

class Test

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter the number :");

int n=sc.nextInt();

int i=1;

do

{

System.out.println(n+" \* "+i+" = "+n\*i);

i++;

}

while (i<=10);

}

}

**ii) while loop**

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

syntax:

while(condition)

{

-

- //code to be execute

-

}

**ex:**

class Test

{

public static void main(String[] args)

{

int i=11;

while(i<=10)

{

System.out.print(i+" "); //nothing

i++;

}

}

}

**ex:**

class Test

{

public static void main(String[] args)

{

int i=1;

while(i<=10)

{

System.out.print(i+" "); //infinite 1

}

}

}

**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+" ");

i++;

}

}

}

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

class Test

{

public static void main(String[] args)

{

int i=1,sum=0;

while(i<=10)

{

sum=sum+i;

i++;

}

System.out.println(sum);

}

}

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

import java.util.Scanner;

class Test

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter the number :");

int n=sc.nextInt();

int i=n,fact=1;

while(i>=1)

{

fact=fact\*i;

i--;

}

System.out.println(fact);

}

}

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

import java.util.Scanner;

class Test

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter the number :");

int n=sc.nextInt();

int i=1;

while(i<=10)

{

System.out.println(n+" \* "+i+" = "+n\*i);

i++;

}

}

}

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

Input:

123

output:

6

ex:

import java.util.Scanner;

class Test

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter the number :");

int n=sc.nextInt();

int rem,sum=0;

while(n>0)

{

rem=n%10;

sum=sum+rem;

n=n/10;

}

System.out.println(sum);

}

}

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

input:

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

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(); //153

int temp=n;

int rem,sum=0;

while(n>0)

{

rem=n%10;

sum=sum+rem\*rem\*rem;

n=n/10;

}

if(temp==sum)

System.out.println("It is armstrong number");

else

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

}

}

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

Input:

123

Output:

321

ex:

import java.util.Scanner;

class Test

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter the number :");

int n=sc.nextInt();

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 a palindrome number

**ex:**

import java.util.Scanner;

class Test

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter the number :");

int n=sc.nextInt();

int temp=n;

int rem,rev=0;

while(n>0)

{

rem=n%10;

rev=rev\*10+rem;

n=n/10;

}

if(temp==rev)

System.out.println("It is a palindrome number");

else

System.out.println("It is not a palindrome number");

}

}

**iii) for loop**

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

syntax:

for(initialization;condition;Increment/Decrement)

{

-

- //code to be execute

-

}

**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(int i=1;i<=10;i++)

{

if(i%2==0)

{

System.out.print(i+" "); // 2 4 6 8 10

}

}

}

}

**ex:**

class Test

{

public static void main(String[] args)

{

int cnt=0;

for(int i=1;i<=10;i++)

{

if(i%2!=0)

{

cnt++;

}

}

System.out.println(cnt);//5

}

}

**ex:**

class Test

{

public static void main(String[] args)

{

for(int i=1;i<=20;i++)

{

if(i%2==0)

{

System.out.print(i+" "); //2 6 10 14 18

i=i+2;

}

}

}

}

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

prime numbers :

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

**ex:**

import java.util.Scanner;

class Test

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter the number :");

int n=sc.nextInt();

boolean flag=true;

for(int i=2;i<=n/2;i++)

{

if(n%i==0)

{

flag=false;

break;

}

}

if(flag==true)

System.out.println("It is prime number");

else

System.out.println("It is not prime number");

}

}

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

class Test

{

public static void main(String[] args)

{

for(int n=2;n<=100;n++)

{

boolean flag=true;

for(int i=2;i<=n/2;i++)

{

if(n%i==0)

{

flag=false;

break;

}

}

if(flag==true)

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

}

}

}

**Q) Write a java program to 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=sum+i;

}

}

if(n==sum)

System.out.println("It is a perfect number");

else

System.out.println("It is not a perfect number");

}

}

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

fibonacci series : 0 1 1 2 3 5 8

ex:

import java.util.Scanner;

class Test

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter the number :");

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

}

}

}

**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

class Test

{

public static void main(String[] args)

{

//rows

for(char i='A';i<='D';i++)

{

//cols

for(char j='A';j<='D';j++)

{

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

}

//new line

System.out.println();

}

}

}

6)

D D D D

C C C C

B B B B

A A A A

class Test

{

public static void main(String[] args)

{

//rows

for(char i='D';i>='A';i--)

{

//cols

for(char j='A';j<='D';j++)

{

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

}

//new line

System.out.println();

}

}

}

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

Input:

12 18

Output:

6

class Test

{

public static void main(String[] args)

{

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

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

{

if((a%i==0) && (b%i==0))

{

gcd=i;

}

}

System.out.println("GCD of two numbers is ="+gcd);

}

}

**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.

**LOOP Patterns continuation**

7)

\* \* \* \*

\* \*

\* \*

\* \* \* \*

class Test

{

public static void main(String[] args)

{

//rows

for(int i=1;i<=4;i++)

{

for(int j=1;j<=4;j++)

{

if(i==1||i==4||j==1||j==4)

System.out.print("\* ");

else

System.out.print(" ");

}

//new line

System.out.println();

}

}

}

8)

\* - - -

- \* - -

- - \* -

- - - \*

class Test

{

public static void main(String[] args)

{

//rows

for(int i=1;i<=4;i++)

{

for(int j=1;j<=4;j++)

{

if(i==j)

System.out.print("\* ");

else

System.out.print("- ");

}

//new line

System.out.println();

}

}

}

9)

\* - - - \*

- \* - \* -

- - \* - -

- \* - \* -

\* - - - \*

class Test

{

public static void main(String[] args)

{

//rows

for(int i=1;i<=5;i++)

{

for(int j=1;j<=5;j++)

{

if(i==j || i+j==6)

System.out.print("\* ");

else

System.out.print("- ");

}

//new line

System.out.println();

}

}

}

**Left Side 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++)

{

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--)

{

for(int j=1;j<=i;j++)

{

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

}

//new line

System.out.println();

}

}

}

3)

**1**

**1 2**

**1 2 3**

**1 2 3 4**

ex:

class Test

{

public static void main(String[] args)

{

//rows

for(int i=1;i<=4;i++)

{

for(int j=1;j<=i;j++)

{

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

}

//new line

System.out.println();

}

}

}

4)

\*

\* \*

\* \* \*

\* \* \* \*

\* \* \*

\* \*

\*

class Test

{

public static void main(String[] args)

{

//ascending

//rows

for(int i=1;i<=4;i++)

{

for(int j=1;j<=i;j++)

{

System.out.print("\* ");

}

//new line

System.out.println();

}

//descending

//rows

for(int i=3;i>=1;i--)

{

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**

class Test

{

public static void main(String[] args)

{

int k=1;

//rows

for(int i=1;i<=4;i++)

{

for(int j=1;j<=i;j++)

{

if(k<=9)

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

else

System.out.print("0 ");

}

//new line

System.out.println();

}

}

}

**6) even number loop pattern?**

**2**

**4 6**

**8 10 12**

**14 16 18 20**

class Test

{

public static void main(String[] args)

{

int k=2;

//rows

for(int i=1;i<=4;i++)

{

for(int j=1;j<=i;j++)

{

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

k+=2;

}

//new line

System.out.println();

}

}

}

**7) odd number loop pattern?**

**1**

**3 5**

**7 9 11**

**13 15 17 19**

class Test

{

public static void main(String[] args)

{

int k=1;

//rows

for(int i=1;i<=4;i++)

{

for(int j=1;j<=i;j++)

{

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

k+=2;

}

//new line

System.out.println();

}

}

}

**8) prime number loop pattern?**

2

3 5

7 11 13

17 19 23 29

class Test

{

public static void main(String[] args)

{

int n=2;

//rows

for(int i=1;i<=4;i++)

{

//cols

for(int j=1;j<=i;j++)

{

while(true)

{

boolean flag=true;

for(int k=2;k<=n/2;k++)

{

if(n%k==0)

{

flag=false;

break;

}

}

if(flag==true)

{

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

break;

}

else

{

n++;

}

}

n++;

}

//new line

System.out.println();

}

}

}

9)

**1**

**2 1**

**1 2 3**

**4 3 2 1**

class Test

{

public static void main(String[] args)

{

for(int i=1;i<=4;i++)

{

if(i%2!=0)

{

for(int j=1;j<=i;j++)

{

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

}

//new line

System.out.println();

}

else

{

for(int j=i;j>=1;j--)

{

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

}

//new line

System.out.println();

}

}

}

}

**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++)

{

//spaces

for(int j=4;j>i;j--)

{

System.out.print(" ");

}

//right side elements

for(int j=1;j<=i;j++)

{

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

}

//new line

System.out.println();

}

}

}

2)

**4 4 4 4**

**3 3 3**

**2 2**

**1**

class Test

{

public static void main(String[] args)

{

//rows

for(int i=4;i>=1;i--)

{

//spaces

for(int j=4;j>i;j--)

{

System.out.print(" ");

}

//right side elements

for(int j=1;j<=i;j++)

{

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

}

//new line

System.out.println();

}

}

}

3)

\*

\* \*

\* \* \*

\* \* \* \*

\* \* \*

\* \*

\*

class Test

{

public static void main(String[] args)

{

//ascending

//rows

for(int i=1;i<=4;i++)

{

//space

for(int j=4;j>i;j--)

{

System.out.print(" ");

}

//right side elements

for(int j=1;j<=i;j++)

{

System.out.print("\* ");

}

//new line

System.out.println();

}

//descending

//rows

for(int i=3;i>=1;i--)

{

//space

for(int j=4;j>i;j--)

{

System.out.print(" ");

}

//right side elements

for(int j=1;j<=i;j++)

{

System.out.print("\* ");

}

//new line

System.out.println();

}

}

}

**Assignment**

1)

**0 0 0 0**

**1 1 1 1**

**2 2 2 2**

**3 3 3 3**

class Test

{

public static void main(String[] args)

{

for (int i=0;i<=3;i++)

{

for (int j=0;j<=3;j++)

{

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

}

System.out.println();

}

}

}

2)

**1 1 1**

**1 0 1**

**1 1 1**

class Test

{

public static void main(String[] args)

{

for (int i=1;i<=3;i++)

{

for (int j=1;j<=3;j++)

{

if ((i==2 || i==2) && (i+j==4))

System.out.print("0 ");

else

System.out.print("1 ");

}

System.out.println();

}

}

}

**Pyramid loop patterns**

1)

**1**

**1 2 1**

**1 2 3 2 1**

**1 2 3 4 3 2 1**

class Test

{

public static void main(String[] args)

{

//rows

for(int i=1;i<=4;i++)

{

//space

for(int j=4;j>i;j--)

{

System.out.print(" ");

}

//left 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**

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)

\*

\* \* \*

\* \* \* \* \*

\* \* \* \* \* \* \*

\* \* \* \* \*

\* \* \*

\*

class Test

{

public static void main(String[] args)

{

//ascending

//rows

for(int i=1;i<=4;i++)

{

//space

for(int j=4;j>i;j--)

{

System.out.print(" ");

}

//left side elements

for(int j=1;j<=i;j++)

{

System.out.print("\* ");

}

//right side elements

for(int j=i-1;j>=1;j--)

{

System.out.print("\* ");

}

//new line

System.out.println();

}

//descending

//rows

for(int i=3;i>=1;i--)

{

//space

for(int j=4;j>i;j--)

{

System.out.print(" ");

}

//left side elements

for(int j=1;j<=i;j++)

{

System.out.print("\* ");

}

//right side elements

for(int j=i-1;j>=1;j--)

{

System.out.print("\* ");

}

//new line

System.out.println();

}

}

}

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

1 1

1 2 2 1

1 2 3 3 2 1

1 2 3 4 4 3 2 1

class Test

{

public static void main(String[] args)

{

int rows=4;

//rows

for(int i=1;i<=rows;i++)

{

//left side elements

for(int j=1;j<=i;j++)

{

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

}

//space

for(int j=1;j<=(rows-i)\*2;j++)

{

System.out.print(" ");

}

//right side elements

for(int j=i;j>=1;j--)

{

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

}

//new line

System.out.println();

}

}

}

**Q) Write a java program to display pascal triangle?**

1

1 1

1 2 1

1 3 3 1

1 4 6 4 1

**ex:**

class Test

{

public static void main(String[] args)

{

//rows

for(int i=0;i<5;i++)

{

//spaces

for(int j=1;j<5-i;j++)

{

System.out.print(" ");

}

int number=1;

for(int k=0;k<=i;k++)

{

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

number = number \* (i-k)/(k+1);

}

//new line

System.out.println();

}

}

}

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

\*

\*

\* \* \* \* \*

\*

\*

class Test

{

public static void main(String[] args)

{

//rows

for(int i=1;i<=5;i++)

{

//cols

for(int j=1;j<=5;j++)

{

if(i==3 || j==3)

System.out.print("\* ");

else

System.out.print(" ");

}

//new line

System.out.println();

}

}

}

## 4) Jump Statement

* It is used to jump from one section of code to another section.
* We have two types of jump statements.

i) break stmt

ii) continue stmt

**i) break stmt**

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

For conditional statement we can use if condition.

syntax:

break stmt;

**ex:1**

class Test

{

public static void main(String[] args)

{

System.out.println("stmt1");

break;

System.out.println("stmt2");

}

}

o/p:

C.T.E : break outside switch or loop

**ex:2**

class Test

{

public static void main(String[] args)

{

System.out.println("stmt1");

if(true)

{

break;

}

System.out.println("stmt2");

}

}

o/p:

C.T.E : break outside switch or loop

**ex:3**

class Test

{

public static void main(String[] args)

{

for(int i=1;i<=10;i++)

{

if(i==5)

{

break;

}

System.out.print(i+" "); // 1 2 3 4

}

}

}

**ii) continue stmt**

* It is used to continue the execution of loops.
* For conditional statements we can use if condition.

syntax:

continue stmt

ex:1

class Test

{

public static void main(String[] args)

{

System.out.println("stmt1");

continue;

System.out.println("stmt2");

}

}

o/p:

C.T.E : continue outside of loop

**ex:2**

class Test

{

public static void main(String[] args)

{

System.out.println("stmt1");

if(true)

{

continue;

}

System.out.println("stmt2");

}

}

o/p:

C.T.E : continue outside of loop

**ex:3**

class Test

{

public static void main(String[] args)

{

for(int i=1;i<=10;i++)

{

if(i==5)

{

continue;

}

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

}

}

}

**ORSCHAPS company**

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

Input: Enter total weight of items(kgs) : 11

Enter shipping destination : domestic

<=5kg --> Rs. 500

>5kg --> per kg --> Rs. 100

Enter total weight of items(kgs) : 11 (1000 + 1200 + 500)

Enter shipping destination : international

<=5kg --> Rs. 1000

>5kg --> per kg --> Rs. 200

>10kg --> 500

import java.util.Scanner;

class Test

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter the total weight of item (kg) :");

int weight=sc.nextInt();

System.out.println("Enter the destination location :");

String destination=sc.next(); // domestic

if(destination.equals("domestic"))

{

if(weight<=5)

System.out.println(500);

else

System.out.println(500 + (weight-5) \* 100);

}

else

{

if(weight<=5)

System.out.println(1000);

else if(weight<=10)

System.out.println(1000 + (weight-5) \* 200);

else if(weight>10)

System.out.println(1000 + (weight-5) \* 200 + 500);

}

}

}

# 17.Arrays

* Array is a collection of homogeneous data elements.
* The main advantages of arrays are

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

ex:

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

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

The main disadvantages of arrays are

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

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

In java, arrays are classified into three types.

1) Single Dimensional Array

2) Double Dimensional Array

3) Multi Dimensional Array

**Array Declaration**

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

Arrays

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

Single Dimensional Array Double Dimensional Array Multi Dimensional Array

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

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

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

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

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

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

int[] arr[][];

int[] []arr[];

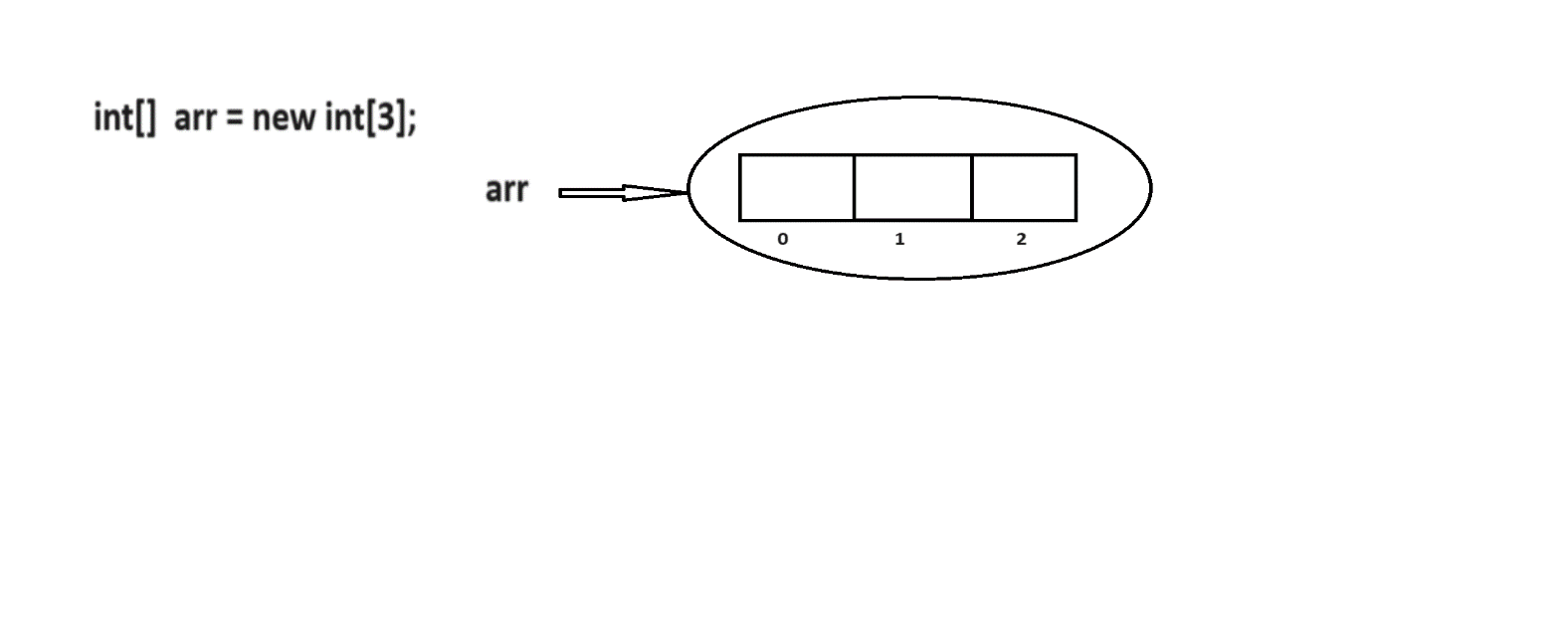
int [][]arr[];

int []arr[][];

**Array Creation**

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

**Diagram: class21.1**



**Rules to construct an array**

**Rule1:**

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

ex:

int[] arr=new int[3];

int[] arr=new int[]; // C.T.E Array diminsion 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 otherwiser we will get runtime exception called NegativeArraySizeException.

ex:

int[] arr=new int[-3];// R.E NegativeArraySizeException

**Rule4:**

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

If we take other datatype 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]; // invalid

**Rule5:**

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

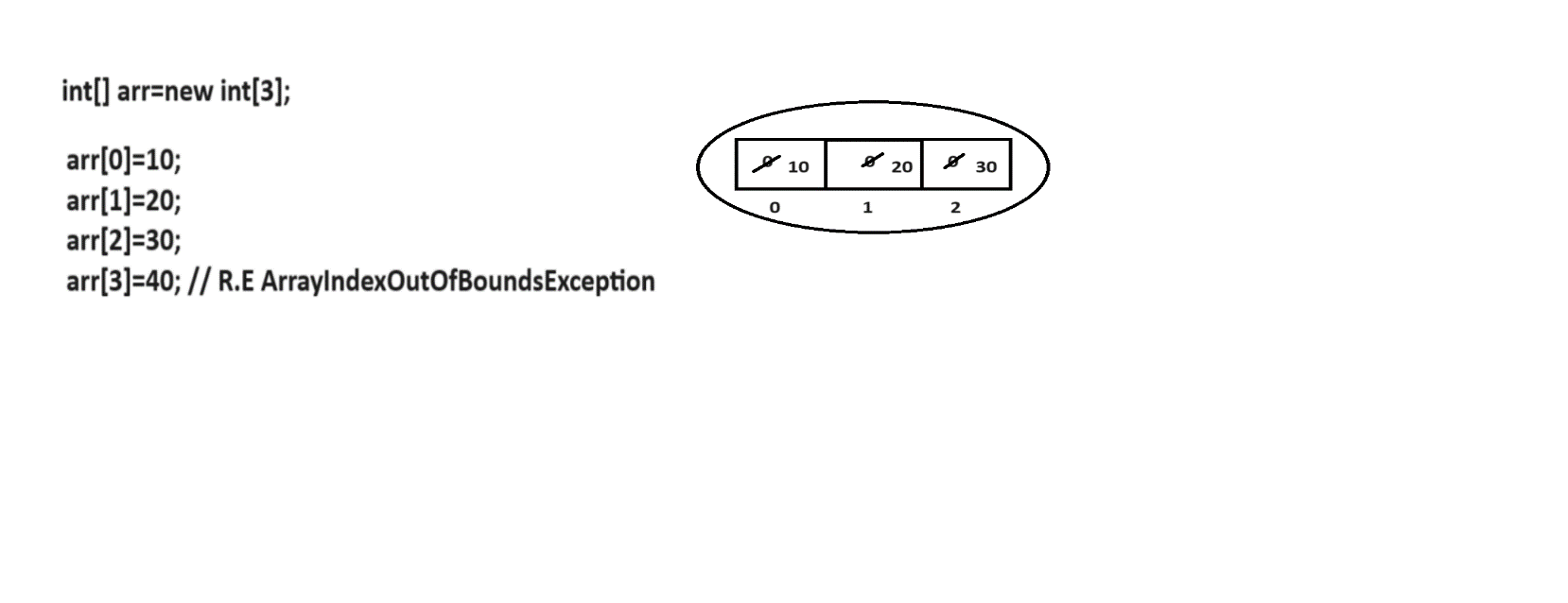
ex:

int[] arr=new int[2147483647];

**Array Initialization**

* Once if we create an array , every array element will be initialized with default values.
* If we are not happy with default values then we can change with customized values.

**Diagram: class22.1**



**Array Declaration , Creation and Initialization using single line**

int[] arr;

arr=new int[3];

arr[0]=10;

arr[1]=20;

arr[2]=30; ==> int[] arr={10,20,30};

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

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

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

**length**

A length if a 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[3];

System.out.println(arr.length); //3

}

}

**length()**

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

It will return number of characters present in String.

ex:

class Test

{

public static void main(String[] args)

{

String str="bhaskar";

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

}

}

## 1.Single Dimensional Array programs

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

import java.util.Scanner;

class Test

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter the array size :");

int size=sc.nextInt();//4

int[] arr=new int[size];

//inserting elements

for(int i=0;i<arr.length;i++)

{

System.out.println("Enter the element :");

arr[i]=sc.nextInt();

}

//display elements

for(int i=0;i<arr.length;i++)

{

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

}

}

}

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

Input:

5 9 2 7 6

Output:

5 9 2 7 6

**approach1**

class Test

{

public static void main(String[] args)

{

int[] arr={5,9,2,7,6};

//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={5,9,2,7,6};

//display elements

//for each loop

for(int i:arr)

{

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

}

}

}

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

Input:

5 9 2 7 6

Output:

6 7 2 9 5

**ex:**

class Test

{

public static void main(String[] args)

{

int[] arr={5,9,2,7,6};

//reading reverse

for(int i=arr.length-1;i>=0;i--)

{

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

}

}

}

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

Input:

5 9 2 7 6

Output:

29

class Test

{

public static void main(String[] args)

{

int[] arr={5,9,2,7,6};

int sum=0;

//for each loop

for(int i:arr)

{

sum+=i;

}

System.out.println(sum);

}

}

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

input:

3 4 7 9 2 6 1

Output:

4 2 6

class Test

{

public static void main(String[] args)

{

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

//for each loop

for(int i:arr)

{

if(i%2==0)

{

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

}

}

}

}

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

input:

3 4 7 9 2 6 1

Output:

3 7 9 1

**ex:**

class Test

{

public static void main(String[] args)

{

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

//for each loop

for(int i:arr)

{

if(i%2!=0)

{

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

}

}

}

}

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

input:

3 4 7 9 2 6 1

Output:

Even elements : 3

Odd elements : 4

**ex:**

class Test

{

public static void main(String[] args)

{

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

int even=0;

int odd=0;

//for each loop

for(int i:arr)

{

if(i%2==0)

{

even++;

}

else

{

odd++;

}

}

System.out.println("Even Elements :"+even);

System.out.println("Odd Elements :"+odd);

}

}

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

input:

5 9 3 11 15 7 14

output:

5 3 11 7

**ex:**

class Test

{

public static void main(String[] args)

{

int[] arr={5,9,3,11,15,7,14};

//for each loop

for(int n:arr)

{

boolean flag=true;

for(int i=2;i<=n/2;i++)

{

if(n%i==0)

{

flag=false;

break;

}

}

if(flag==true)

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

}

}

}

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

input:

5 9 3 11 15 7 14

output:

3 5 7 9 11 14 15

**ex:**

import java.util.Arrays;

class Test

{

public static void main(String[] args)

{

int[] arr={5,9,3,11,15,7,14};

Arrays.sort(arr);

//for each loop

for(int i:arr)

{

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

}

}

}

**Q) Write a java program to display array elements in sorting order without using sort() method?**

input:

5 9 3 11 15 7 14

output:

3 5 7 9 11 14 15

ex:

class Test

{

public static void main(String[] args)

{

int[] arr={5,9,3,11,15,7,14};

//ascending logic

for(int i=0;i<arr.length;i++)

{

for(int j=0;j<arr.length;j++)

{

if(arr[i]<arr[j])

{

int temp=arr[i];

arr[i]=arr[j];

arr[j]=temp;

}

}

}

//display elements

for(int i:arr)

{

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

}

}

}

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

input:

5 9 3 11 15 7 14

output:

15 14 11 9 7 5 3

**ex:**

import java.util.Arrays;

class Test

{

public static void main(String[] args)

{

int[] arr={5,9,3,11,15,7,14};

Arrays.sort(arr); // 3 5 7 9 11 14 15

for(int i=arr.length-1;i>=0;i--)

{

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

}

}

}

**Q) Write a java program to display array elements in descending order without using sort() method?**

input:

5 9 3 11 15 7 14

output:

15 14 11 9 7 5 3

**ex:**

class Test

{

public static void main(String[] args)

{

int[] arr={5,9,3,11,15,7,14};

//descending order

for(int i=0;i<arr.length;i++)

{

for(int j=0;j<arr.length;j++)

{

if(arr[i]>arr[j])

{

int temp=arr[i];

arr[i]=arr[j];

arr[j]=temp;

}

}

}

//display the elements

for(int i:arr)

{

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

}

}

}

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

Input:

6 8 3 4 9 2 1

Output:

ex:

class Test

{

public static void main(String[] args)

{

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

int big=arr[0];

for(int i:arr)

{

if(i>big)

{

big=i;

}

}

System.out.println(big);

}

}

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

Input:

7 2 4 9 1 6 3

Output:

1

class Test

{

public static void main(String[] args)

{

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

int small=arr[0];

for(int i:arr)

{

if(i<small)

{

small=i;

}

}

System.out.println(small);

}

}

**Q) Write a java program to display three highest elements from given array?**

Input:

7 2 4 9 1 6 3

Output:

9 7 6

ex:

class Test

{

public static void main(String[] args)

{

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

int first=Integer.MIN\_VALUE;

int second=Integer.MIN\_VALUE;

int third=Integer.MIN\_VALUE;

//for each loop

for(int i:arr)

{

if(i>first)

{

third=second;

second=first;

first=i;

}

else if(i>second)

{

third=second;

second=i;

}

else if(i>third)

{

third=i;

}

}

System.out.println(first+" "+second+" "+third);

}

}

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

Input:

3 6 1 2 3 9 4 4 7 6 10

Output:

3 6 4

ex:

class Test

{

public static void main(String[] args)

{

int[] arr={3,6,1,2,3,9,4,4,7,6,10};

//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:

3 6 1 2 3 9 4 4 7 6 10

Output:

1 2 9 7 10

ex:

class Test

{

public static void main(String[] args)

{

int[] arr={3,6,1,2,3,9,4,4,7,6,10};

//unique elements

for(int i=0;i<arr.length;i++)

{

int cnt=0;

for(int j=0;j<arr.length;j++)

{

if(arr[i]==arr[j])

{

cnt++;

}

}

if(cnt==1)

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

}

}

}

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

input:

5 1 2 4 2 9 7 2 2 6 6 2

output:

2 is repeating for 5 times

ex:

class Test

{

public static void main(String[] args)

{

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

int maxCount=0;

int element=0;

for(int i=0;i<arr.length;i++)

{

int cnt=0;

for(int j=0;j<arr.length;j++)

{

if(arr[i]==arr[j])

{

cnt++;

}

}

if(cnt>maxCount)

{

maxCount=cnt;

element=arr[i];

}

}

System.out.println(element+" is repeating for "+maxCount+" times");

}

}

**Q) Write a java program to perform sum of two array elements and display them in third array?**

input:

1 6 2 9 4

8 2 4 1 3

output:

9 8 6 10 7

**ex:**

class Test

{

public static void main(String[] args)

{

int[] arr1={1,6,2,9,4};

int[] arr2={8,2,4,1,3};

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

for(int i=0;i<arr1.length && i<arr2.length;i++)

{

resArr[i]=arr1[i]+arr2[i];

}

//display

for(int i:resArr)

{

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

}

}

}

**Q) Write a java program to display 10 fibonacci numbers?**

output:

0 1 1 2 3 5 8 13 21 34

ex:

class Test

{

public static void main(String[] args)

{

int n=10;

int[] fibonacci=new int[n];

fibonacci[0]=0;

fibonacci[1]=1;

System.out.print(fibonacci[0]+" "+fibonacci[1]+" ");

for(int i=2;i<fibonacci.length;i++)

{

fibonacci[i]= fibonacci[i-1]+fibonacci[i-2];

System.out.print(fibonacci[i]+" ");

}

}

}

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

Input:

5 2 1 3 4

9 7 8 6 10

output:

1 2 3 4 5 6 7 8 9 10

ex:

import java.util.Arrays;

class Test

{

public static void main(String[] args)

{

int[] arr1={5,2,1,3,4};

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

int size1=arr1.length; // 5

int size2=arr2.length; // 5

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

int j=0;

for(int i=size1;i<arr1.length;i++)

{

arr1[i]=arr2[j++];

}

//sorting

Arrays.sort(arr1);

//display

for(int i:arr1)

{

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

}

}

}

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

input:

arr = 6 4 2 3 9 2 7 2 1

element = 2

output:

6 4 3 9 2 7 2 1

ex:

class Test

{

public static void main(String[] args)

{

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

int element = 2;

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

int j=0,cnt=0;

for(int i=0;i<arr.length;i++)

{

if(arr[i]==element && cnt==0)

{

cnt=1;

continue;

}

resArr[j++]=arr[i];

}

//display

for(int i:resArr)

{

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

}

}

}

**Interview Question**

**Q) Write a java program to display perfect square root from 1 to 100?**

output:

1 4 9 16 25 36 49 64 81 100

ex:

class Test

{

public static void main(String[] args)

{

for(int i=1;i<=10;i++)

{

int square=i\*i;

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

}

}

}

**Q) Given an integer array.find the contiguous subarray which has the larget sum 6?**

Input:

nums = [-2,1,-3,4,-1,2,1,-5,4]

Output:

6

Explaination : [4,-1,2,1] has the largest continguous subarray.

ex:

class Test

{

public static void main(String[] args)

{

int[] nums={-2,1,-3,4,-1,2,1,-5,4}

int maxCount=nums[0];

int sum=nums[0];

for(int i=1;i<nums.length;i++)

{

if(sum>0)

{

sum+=nums[i];

}

else

{

sum=nums[i];

}

if(sum>maxCount)

{

maxCount=sum;

}

}

System.out.println(maxCount);

}

**Q) Write a java program to display pair elements equals to given sum?**

input:

arr = 5 2 3 8 9 1 4 6

sum = 10

output:

2 8

9 1

4 6

ex:

class Test

{

public static void main(String[] args)

{

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

int sum=10;

for(int i=0;i<arr.length;i++)

{

for(int j=i+1;j<arr.length;j++)

{

if(arr[i]+arr[j]==sum)

{

System.out.println(arr[i]+" "+arr[j]);

}

}

}

}

}

**Q) Write a java program to display triplet of elements equals to given sum?**

input:

arr = 5 2 3 8 9 1 4 6

sum = 10

output:

5 2 3

5 1 4

3 1 6

class Test

{

public static void main(String[] args)

{

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

int sum=10;

for(int i=0;i<arr.length;i++)

{

for(int j=i+1;j<arr.length;j++)

{

for(int k=j+1;k<arr.length;k++)

{

if(arr[i]+arr[j]+arr[k]==sum)

{

System.out.println(arr[i]+" "+arr[j]+" "+arr[k]);

}

}

}

}

}

}

## 2.Double Dimensional Array

* Double dimensional array is a combination of rows and columns.
* Double dimensional array is implemented based on array of arrays approach but not in matrix form.
* The main objective of double dimensional array is memory utilization.
* We can declare double dimensional array as follow.

ex:

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

Here we can store 9 elements.

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

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

import java.util.Scanner;

class Test

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter the rows :");

int rows=sc.nextInt(); //3

System.out.println("Enter the cols :");

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

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

//insert elements

for(int i=0;i<rows;i++)

{

for(int j=0;j<cols;j++)

{

System.out.println("Enter the element :");

arr[i][j]=sc.nextInt();

}

}

//display elements

for(int i=0;i<rows;i++)

{

for(int j=0;j<cols;j++)

{

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

}

//new line

System.out.println();

}

}

}

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

Input: 1 2 3

4 5 6

7 8 9

output:

1 4 9

16 25 36

49 64 81

class Test

{

public static void main(String[] args)

{

int[][] arr={

{1,2,3},

{4,5,6},

{7,8,9}

};

for(int i=0;i<arr.length;i++)

{

for(int j=0;j<arr[0].length;j++)

{

System.out.print(arr[i][j] \* arr[i][j]+" ");

}

//new line

System.out.println();

}

}

}

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

class Test

{

public static void main(String[] args)

{

int[][] arr={

{1,2,3},

{4,5,6},

{7,8,9}

};

int sum=0;

for(int i=0;i<arr.length;i++)

{

for(int j=0;j<arr[0].length;j++)

{

if(i==j)

{

sum+=arr[i][j];

}

}

}

System.out.println("sum of diagonal elements is ="+sum);

}

}

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

class Test

{

public static void main(String[] args)

{

int[][] arr={

{1,2,3},

{4,5,6},

{7,8,9}

};

int sum=0;

for(int i=0;i<arr.length;i++)

{

for(int j=0;j<arr[0].length;j++)

{

if(i<j)

{

sum+=arr[i][j];

}

}

}

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

}

}

**Q) Write a java program to perform lower triangle elements?**

class Test

{

public static void main(String[] args)

{

int[][] arr={

{1,2,3},

{4,5,6},

{7,8,9}

};

int sum=0;

for(int i=0;i<arr.length;i++)

{

for(int j=0;j<arr[0].length;j++)

{

if(i>j)

{

sum+=arr[i][j];

}

}

}

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

}

}

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

input:

1 2 3

4 5 6

7 8 9

output:

1 2 3 6 9 8 7 4 5

ex:

class Test

{

public static void main(String[] args)

{

int[][] 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;

}

for(int i=left;i<=right;i++)

{

System.out.print(matrix[top][i]+" ");

}

top++;

if(top>bottom)

{

break;

}

for(int i=top;i<=bottom;i++)

{

System.out.print(matrix[i][right]+" ");

}

right--;

if(left>right)

{

break;

}

for(int i=right;i>=left;i--)

{

System.out.print(matrix[bottom][i]+" ");

}

bottom--;

if(top>bottom)

{

break;

}

for(int i=bottom;i>=top;i--)

{

System.out.print(matrix[i][left]+" ");

}

left++;

}

}

}

**Anonymous Array**

* Sometimes we will declare an array without name such type of nameless array is called anonymous array.
* The main objective of anonymous array is just for instance use.
* We can declare anonymous array as follow.

ex:

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

new int[][]{{1,2,3},{4,5,6}};

ex:

class Test

{

public static void main(String[] args)

{

//calling

sum(new int[]{10,20,30});

}

//static method

public static void sum(int[] arr)

{

int sum=0;

for(int i:arr)

{

sum+=i;

}

System.out.println(sum);//60

}

}

ex:

class Test

{

public static void main(String[] args)

{

//calling

System.out.println(sum(new int[]{10,20,30,40}));

}

//static method

public static int sum(int[] arr)

{

int sum=0;

for(int i:arr)

{

sum+=i;

}

return sum;

}

}

# 18.Various ways to write the methods in java

There are four ways to declare the methods in java.

1) No returntype with No argument method

2) No returntype with Argument method

3) With returntype with No argument method

4) With returntype with Argument method

**1) No returntype with No argument method**

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

**Q) Write a java program to 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();

}

//static method

//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();

int c=a+b;

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

}

}

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

import java.util.Scanner;

class Test

{

public static void main(String[] args)

{

//caller method

factorial();

}

//static method

//callie method

public static void factorial()

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter the number :");

int n=sc.nextInt();

int fact=1;

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

{

fact\*=i;

}

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

}

}

**2) No returntype with Argument method**

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

Number of arguments depends upon number of inputs.

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

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);

}

//static method

//callie method

public static void sum(int a,int b)

{

int c=a+b;

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

}

}

**Q) Write a java program to display prime elements between two numbers using no returntype with argument method?**

import java.util.Scanner;

class Test

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter the first number :");

int a=sc.nextInt(); //10

System.out.println("Enter the second number :");

int b=sc.nextInt(); //20

//caller method

prime(a,b);

}

//static method

//callie method

public static void prime(int a,int b)

{

for(int n=a;n<=b;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+" ");

}

}

}

**3) With returntype with No argument method**

* Returntype is completely depends upon output datatype.

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

import java.util.Scanner;

class Test

{

public static void main(String[] args)

{

//caller method

int k=sum();

System.out.println("sum of two numbers is ="+k);

}

//static method

//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 perform area of a circle using with returntype with no argument method?**

import java.util.Scanner;

class Test

{

public static void main(String[] args)

{

//caller method

float k=circle();

System.out.println("Area of a circle is ="+k);

}

//static method

//callie method

public static float circle()

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter the radius :");

int r=sc.nextInt();

float area=3.14f\*r\*r;

return area;

}

}

**4) With returntype with Argument method**

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

}

//static method

//callie method

public static int sum(int a,int b)

{

int c=a+b;

return c;

}

}

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

**approach1**

import java.util.Scanner;

class Test

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter the number :");

int n=sc.nextInt();

//caller method

System.out.println(find(n));

}

//static method

//callie method

public static String find(int n)

{

if(n%2==0)

return "It is even number";

else

return "IT is odd number";

}

}

**approach2**

import java.util.Scanner;

class Test

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter the number :");

int n=sc.nextInt();

//caller method

if(find(n))

System.out.println("It is even number");

else

System.out.println("It is odd number");

}

//static method

//callie method

public static boolean find(int n)

{

if(n%2==0)

return true;

else

return false;

}

}

**Assignment**

**Q) Write a java program to check given number is palindrome or not? Using five types?**

**Q)Write a java program to determine the smallest number of coins needed to total**

**86 rupees. Use the denominations provided in the array {1,2,5,10}?**

Output:

1 coin(s) of 1 rupee(s)

1 coin(s) of 5 rupee(s)

8 coin(s) of 10 rupee(s)

ex:

import java.util.Scanner;

class Test

{

public static void main(String[] args)

{

int[] denominations={1,2,5,10};

int amount=86;

//caller method

int[] result=findMinimumCoins(denominations,amount);

for(int i=0;i<result.length;i++)

{

if(result[i]>0)

{

System.out.println(result[i]+" coin(s) of "+ denominations[i] +" rupee(s)");

}

}

}

//static method

//callie method

public static int[] findMinimumCoins(int[] denominations,int amount)

{

int[] coinsCount=new int[denominations.length];

for(int i=denominations.length-1;i>=0;i--)

{

coinsCount[i]=amount/denominations[i];

amount%=denominations[i];

}

return coinsCount;

}

}

# 19.Recursion

* A method which call itself for many number of times is called recursion.
* Recursion is similar to loopings.
* It means 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+" ");

display(i+1);

}

}

}

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

import java.util.Scanner;

class Test

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter the first number :");

int a=sc.nextInt();

System.out.println("Enter the second number :");

int b=sc.nextInt();

//caller method

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

}

//callie method

public static int sum(int a,int b)

{

if(a==0)

return b;

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

}

}

**Q) Write a java program to display factorial of a given number using recursion?**

import java.util.Scanner;

class Test

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter the number :");

int n=sc.nextInt();

//caller method

System.out.println("Factorial of a given number is ="+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 display N-th element of a fibonacci series?**

fibonacci series : 0 1 1 2 3 5 8

Input:

4

Output:

2

**ex:**

import java.util.Scanner;

class Test

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter the number :");

int n=sc.nextInt();

//caller method

System.out.println(fibonacci(n));// 4

}

//callie method

public static int fibonacci(int n)

{

if(n==0 || n==1)

return 0;

if(n==2)

return 1;

return fibonacci(n-1)+fibonacci(n-2);

}

}

# 20.OOPS

* OOPS stands for Object Oriented Programming System/Structure.
* The purpose of OOPS is to deal with real world entities using programming language.
* We have following list of OOPS features.

1) class

2) object

3) Abstraction

4) Encapsulation

5) Inheritance

6) Polymorhism

**object oriented technology**

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

## 1)class

* A class is a blue print of an object.
* A class is a collection of variables and methods.
* A class is a logical entity.

We can declare a class as follow.

**syntax:**

optional

|

modifier class class\_name <extends> Parent\_classname

<implements> Interface\_name

{

-

- //variables

- // methods

-

}

**A class will accept following modifiers.**

**ex:**

default

public

private

protected

**Realtime example**

To construct a building we required a design and that design is a class.

**Q) What is 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

{

-

- //variables and methods

-

}

**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

{

-

- //variables and methods

-

}

**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

## 2)object

* It is a outcome of a blue print.
* It is a instance of a class.
* Here instance means allocating memory for our data members.
* It is a physical entity.
* It is a collection of properties and behaviours.

We can create object as follow.

ex:

operator

|

Test t = new Test();

| | |

classname reference variable constructor

* Memory space will be allocated when we create an object.
* It is possible to create more then on object in a single class.

**Realtime example**

Dog (object)

|

-----------------------------------------

| |

properties behaviours

> Height > Eating

> Age > Sleeping

> Color > Barking

> Weight > Running

> Width and etc.

and etc.

**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@Hexadecimalno

System.out.println(t2.toString());

System.out.println(t3.toString());

}

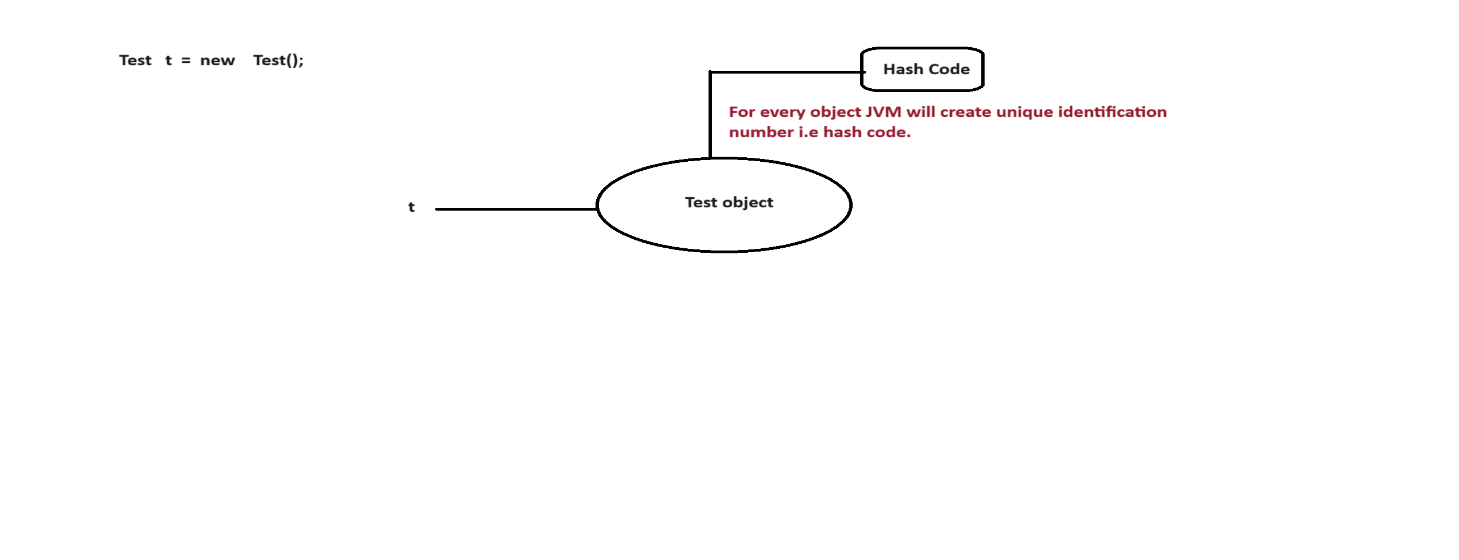
}

**Q) What is hash code in java?**

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

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

**Diagram: class27.1**

****

**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) Which class consider as parent class in java?**

* Object class consider as parent class in java.
* Object class present in java.lang package.
* Object class contains following methods.

ex:

cmd>javap java.lang.Object

ex:

hashCode()

toString()

notify()

notifyAll()

wait()

getClass()

and etc.

**Data Hiding**

* Data hiding is used to provide security to the program by hiding object data.
* Using private modifier we can implements data hiding concept.
* Data hiding is used to prevent unauthorized data access.

**ex:**

class Account

{

//instance variable

private double balance=20000;

}

class Customer

{

public static void main(String[] args)

{

Account a=new Account();

System.out.println(a.balance);

}

}

C.T.E : balance has private access in Account

## 3)Abstraction

* Hiding internal implementation and highlighting the set of services is called abstraction.
* Using abstract classes and interfaces we can implements abstraction.
* The main objective of abstraction is to hide the code.

**Realtime example**

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

**The main advantages of abstraction are**

1) It gives security because it will hide internal implementation from the outsider.

2) Enhancement becomes more easy because without effecting enduser they can perform any changes in our internal system.

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

4) It improves maintainability of an application.

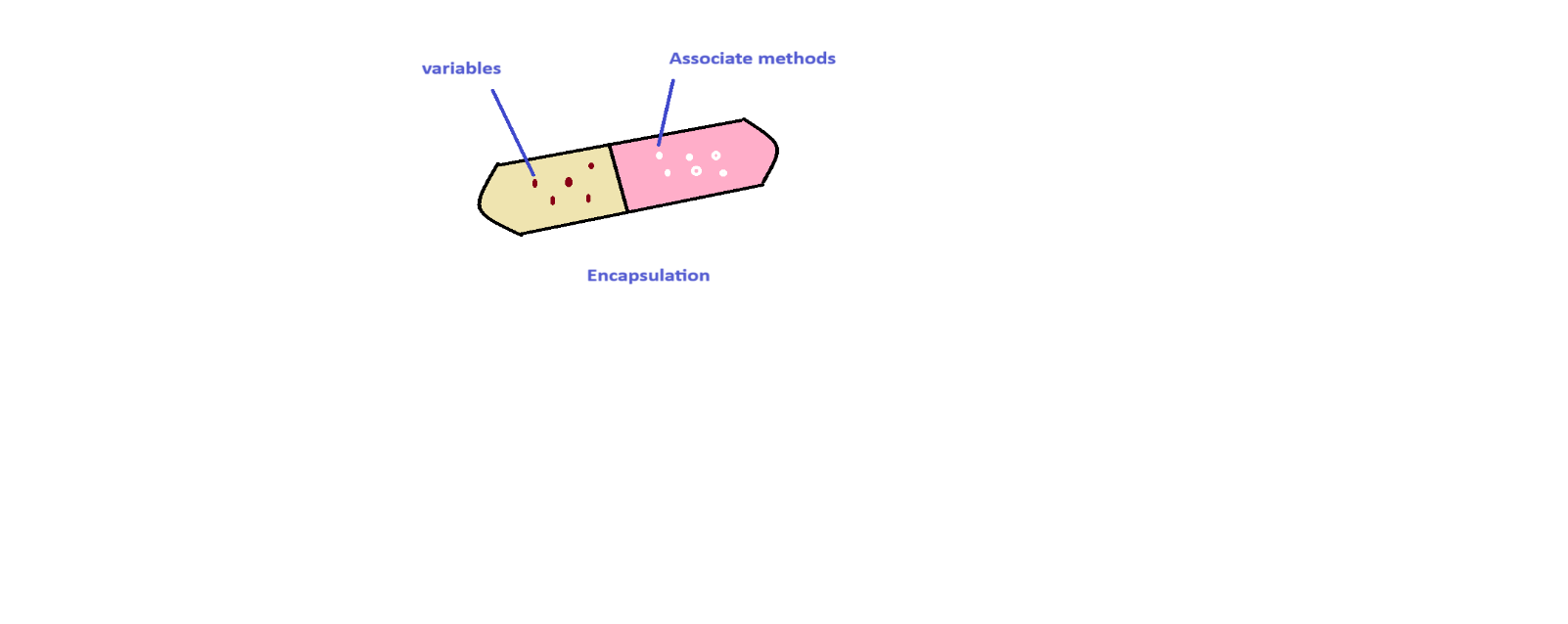
## 4)Encapsulation

* The process of encapsulating or grouping variables and it's associate methods in a single entity is called encapsulation.
* A class is said to be encapsulated class if it supports data hiding and abstraction.
* The main object of encapsulation is to protect the data.

**Realtime example**

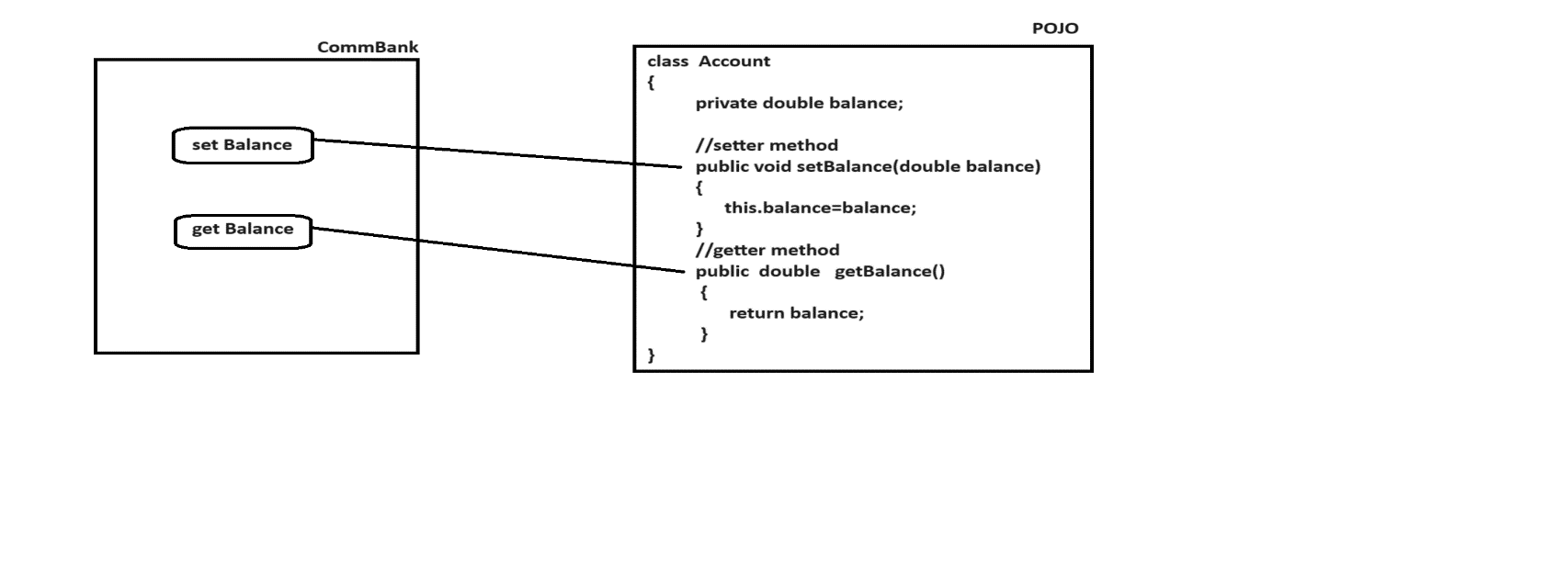
The best example of encapsulation is capsule.

**Diagram: class28.1**



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

**Diagram: class28.2**



**The main advantages of encapsulation are**

1) It gives security.

2) Enhancement becomes more easy.

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

4) It improves maintainability of an application.

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

**ex:**

class Student

{

private int studId;

private String studName;

private double studFee;

//setter methods

public void setStudId(int studId)

{

this.studId=studId;

}

public void setStudName(String studName)

{

this.studName=studName;

}

public void setStudFee(double studFee)

{

this.studFee=studFee;

}

//getter methods

public int getStudId()

{

return studId;

}

public String getStudName()

{

return studName;

}

public double getStudFee()

{

return studFee;

}

}

class Test

{

public static void main(String[] args)

{

Student s=new Student();

s.setStudId(101);

s.setStudName("Alan");

s.setStudFee(1000d);

System.out.println("Student Id :"+s.getStudId());

System.out.println("Student Name :"+s.getStudName());

System.out.println("Student Fee :"+s.getStudFee());

}

}

**Q) What is tightly encapsulated class?**

A class is said to be tightly encapsulated class if and only if all variables of that class must be declare as private and here we don't need to check these variables having setter and getter method or not.

ex:

class A

{

int i=10;

}

It is not a tighly encapsulated class.

ex:

class A

{

private int i=10;

}

It is a tighly encapsulated class.

ex:

class A

{

int i=10;

}

class B extends A

{

private int j=20;

}

If no parent is tightly encapsulted then no child is tighly encapsulated class.

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

**POJO class**

POJO stands for Plain Old Java Object.

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

1) All variables must be private

2) All variables must have setter and getter method.

**Java Bean class**

A class is said to be Java bean class if it supports following four properties.

1) A class should be public.

2) A class should have atleast zero argument constructor.

3) All variables must be private

4) All variables must have setter and getter method

**Note:**

Every Java bean class is a POJO class.But every POJO class is not a java bean class.

**Is-A Relationship**

Is-A relationship is also known as inheritance.

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

The main objective of Is-A relationship is to achieve reusability.

ex:

class Parent

{

public void m1()

{

System.out.println("Parent-M1 Method");

}

}

class Child extends Parent

{

public void m2()

{

System.out.println("Child-M2 Method");

}

}

class Test

{

public static void main(String[] args)

{

Parent p=new Parent();

p.m1();

Child c=new Child();

c.m1();

c.m2();

Parent p1=new Child();

p1.m1();

Child c1=new Parent(); // invalid

}

}

**Conclusions**

Whatever our parent having properties it comes to child.But whatever our child is having properties it never goes back to parent.

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

## 5)Inheritance

* Inheritance is a mechanism we will derived a class in the presence of existing class.
* Inheritance is a mechanism where one class will inherit the properties of another class.
* The main objective of inheritance is to achieve reusability.
* Using extends keyword we can implements inheritance.

We have five types of inheritance in java.

1) Single Level inheritance

2) Multi Level inheritance

3) Multiple inheritance

4) Hierarchical inheritance

5) Hybrid inheritance

**1) Single Level inheritance**

If a class is derived from one base class is called single level inheritance.

ex:

A (Parent/Super/Base class)

|

|

|

B (Child/Sub/Derived class )

ex:

class A

{

public void m1()

{

System.out.println("M1-Method");

}

}

class B extends A

{

public void m2()

{

System.out.println("M2-Method");

}

}

class Test

{

public static void main(String[] args)

{

A a=new A();

a.m1();

B b=new B();

b.m1();

b.m2();

}

}

**2) Multi Level inheritance**

If we derived a class in the presence of one base class and that class is derived from another base class is called multi level inheritance.

ex:

A

|

|

B

|

|

C

ex:

class A

{

public void m1()

{

System.out.println("M1-Method");

}

}

class B extends A

{

public void m2()

{

System.out.println("M2-Method");

}

}

class C extends B

{

public void m3()

{

System.out.println("M3-Method");

}

}

class Test

{

public static void main(String[] args)

{

A a=new A();

a.m1();

B b=new B();

b.m1();

b.m2();

C c=new C();

c.m1();

c.m2();

c.m3();

}

}

**3) Multiple inheritance**

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

ex:

class A {}

class B {}

class C extends A,B --> invalid

{

}

But interface can extends more then one inheritance so we can achieve multiple inheritance 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: Diag:

class A Object

{ |

} |

A

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

ex: Diag:

class A Object

{ |

} |

class B extends A A

{ |

|

} B

Java does not support cyclic inheritence.

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 by using one base class is called hierarchical inheritance.

ex:

A

|

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

B C

ex:

class A

{

public void m1()

{

System.out.println("A-M1 Method");

}

}

class B extends A

{

public void m2()

{

System.out.println("B-M2 Method");

}

}

class C extends A

{

public void m3()

{

System.out.println("C-M3 Method");

}

}

class Test

{

public static void main(String[] args)

{

A a=new A();

a.m1()

B b=new B();

b.m1();

b.m2()

C c=new C();

c.m1();

c.m3();

}

}

**5)Hybrid inheritance**

* Hybrid inheritance is a combination of more then one inheritance.
* Java does not support hybrid inheritance

ex:

A

|

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

B C

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

|

D

**Has-A relationship**

* Has-A relationship is also known as composition and aggregation.
* There is no specific keyword to implements Has-A relationship but mostly we will use new operator.
* The main objective of Has-A relationship is to provide reusability.
* Has-A relationship will increase dependency between two components.

ex:

class Ihub

{

public String courseName()

{

return "Full Stack Java with AWS";

}

public double courseFee()

{

return 30000d;

}

public String trainerName()

{

return "Niyaz Sir";

}

}

class Usha

{

public void getCourseDetails()

{

Ihub i=new Ihub();

System.out.println("Course Name :"+i.courseName());

System.out.println("Course Fee :"+i.courseFee());

System.out.println("Trainer Name :"+i.trainerName());

}

}

class Student

{

public static void main(String[] args)

{

Usha u=new Usha();

u.getCourseDetails();

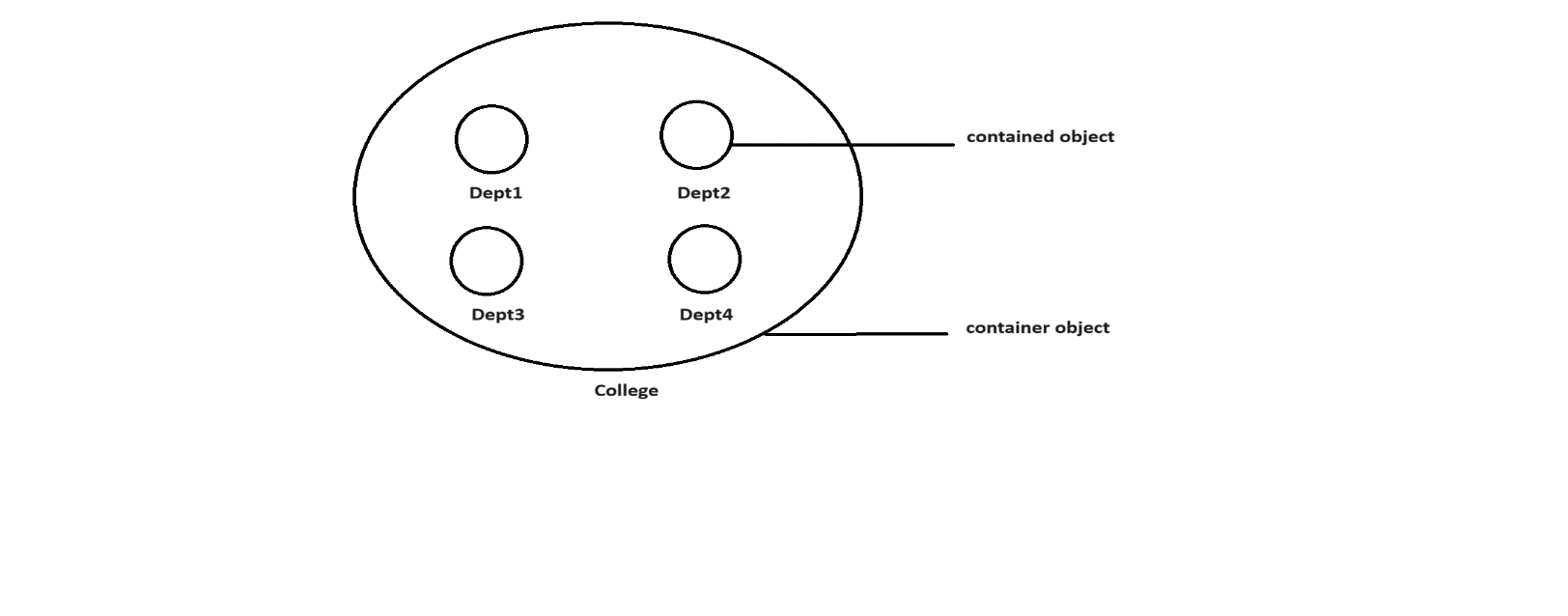
}

}

**composition**

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

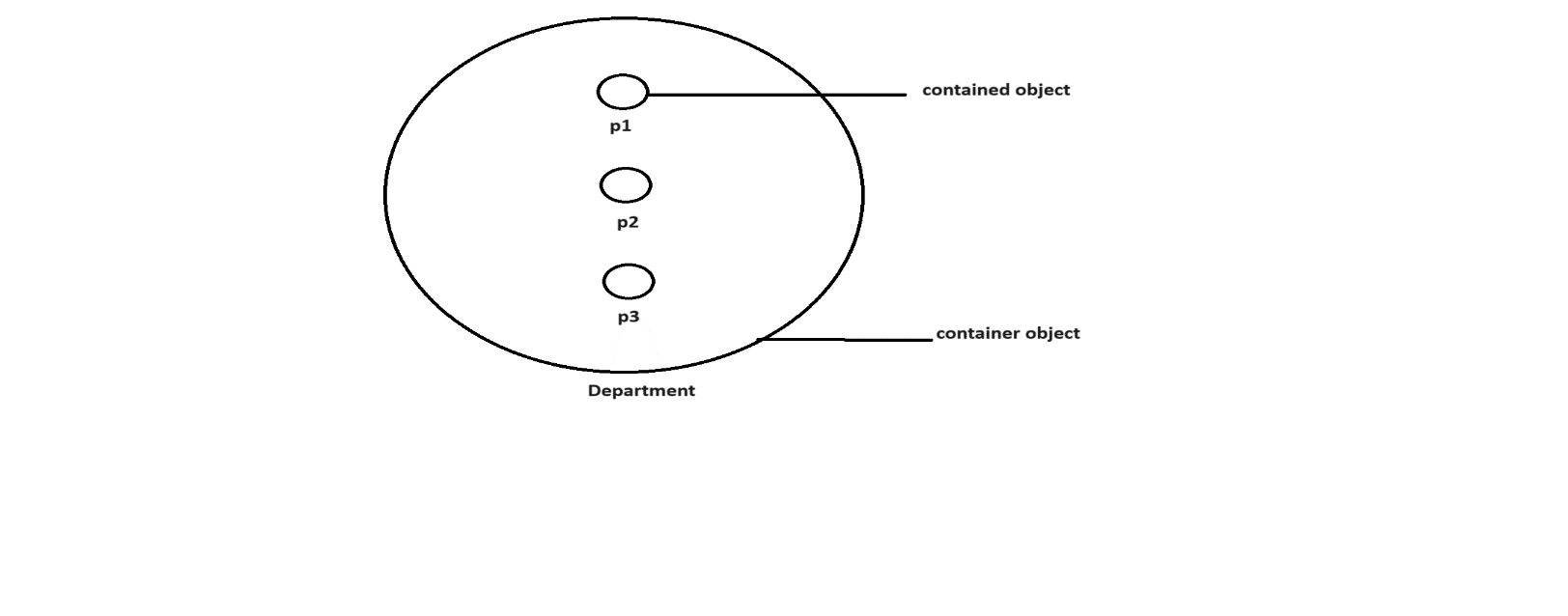
**Diagram: class29.1**



**aggregation**

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

**Diagram: class29.2**



**Method overloading**

* Having same method name with different parameters in a single class is called method overloading.
* All the methods present in a class are called overloaded methods.
* Method overloading will reduce complexity of the programming.

ex:

class MeeSeva

{

//overloaded methods

public void search(int voterId)

{

System.out.println("Details Found via voterId");

}

public void search(String houseNo)

{

System.out.println("Details Found via houseNo");

}

public void search(long aadharNo)

{

System.out.println("Details Found via aadharNo");

}

}

class Test

{

public static void main(String[] args)

{

MeeSeva ms=new MeeSeva();

ms.search(101);

ms.search("1-4-614/3");

ms.search(101L);

}

}

**Method overriding**

* Having same method name with same parameters in two different classes.
* Methods which are present in a parent class are called overridden methods.
* Methods which are present in a child class are called overriding methods.

ex:

class Parent

{

public void property()

{

System.out.println("Cash+Gold+Land");

}

//overridden methods

public void marry()

{

System.out.println("Laxmi");

}

}

class Child extends Parent

{

//overriding methods

public void marry()

{

System.out.println("Rashmika");

}

}

class Test

{

public static void main(String[] args)

{

Parent p=new Parent();

p.property(); //cash+gold+land

p.marry(); // Laxmi

Child c=new Child();

c.property(); // cash+gold+land

c.marry(); // Rashmika

Parent p1=new Child();

p1.property(); // cash+gold+land

p1.marry(); // Rashmika

}

}

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

ex:

class Parent

{

public void property()

{

System.out.println("Cash+Gold+Land");

}

//overridden methods

public final void marry()

{

System.out.println("Laxmi");

}

}

class Child extends Parent

{

//overriding methods

public void marry()

{

System.out.println("Rashmika");

}

}

class Test

{

public static void main(String[] args)

{

Parent p=new Parent();

p.property(); //cash+gold+land

p.marry(); // Laxmi

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 All the methods present in method hiding must be non-static. must be static.

Method resolution will taken care by a JVM Method resolution will taken care by

based on runtime object. acompiler based on reference type.

It is also known as runtime polymorphism, It is also known as compile time polymorphism, dynamic polymorhism or static polymorphism or early binding. late binding.

ex:

class Parent

{

public static void property()

{

System.out.println("Cash+Gold+Land");

}

public static void marry()

{

System.out.println("Laxmi");

}

}

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(); // Laxmi

Child c=new Child();

c.property(); // cash+gold+land

c.marry(); // Rashmika

Parent p1=new Child();

p1.property(); // cash+gold+land

p1.marry(); // Laxmi

}

}

**Q) Can we overload main method in java?**

Yes, we can overload main method in java but JVM always execute main method with String[] parameter only.

ex:

class Test

{

public static void main(int[] iargs)

{

System.out.println("int[] parameter");

}

public static void main(String[] args)

{

System.out.println("String[] parameter");

}

}

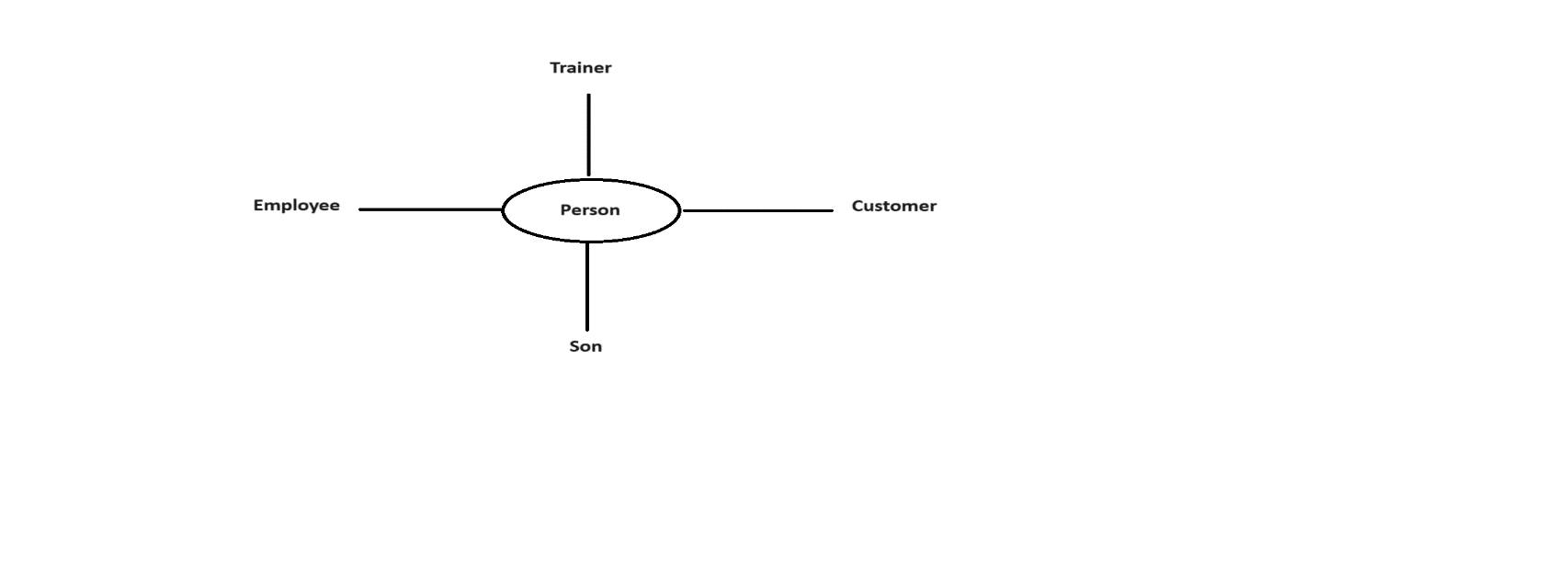
**Q) Can we override main method in java?**

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

## 6)Polymorphism

* Poly means many and morphism means forms.
* The ability to represent in different forms is called polymorphism.

**Diagram: class30.1**



* The main objective of polymorphism is to provide flexibility.

In java, polymorphism is divided into two types.

1) Compile time polymorphism / Static polymorphism / Early Binding

2) Runtime polymorphism / Dynamic polymorphism / Late Binding

**1) Compile time polymorphism**

A polymorphism which exhibits at compile time is called compile time polymorphism.

ex:

Method overloading

Method hiding

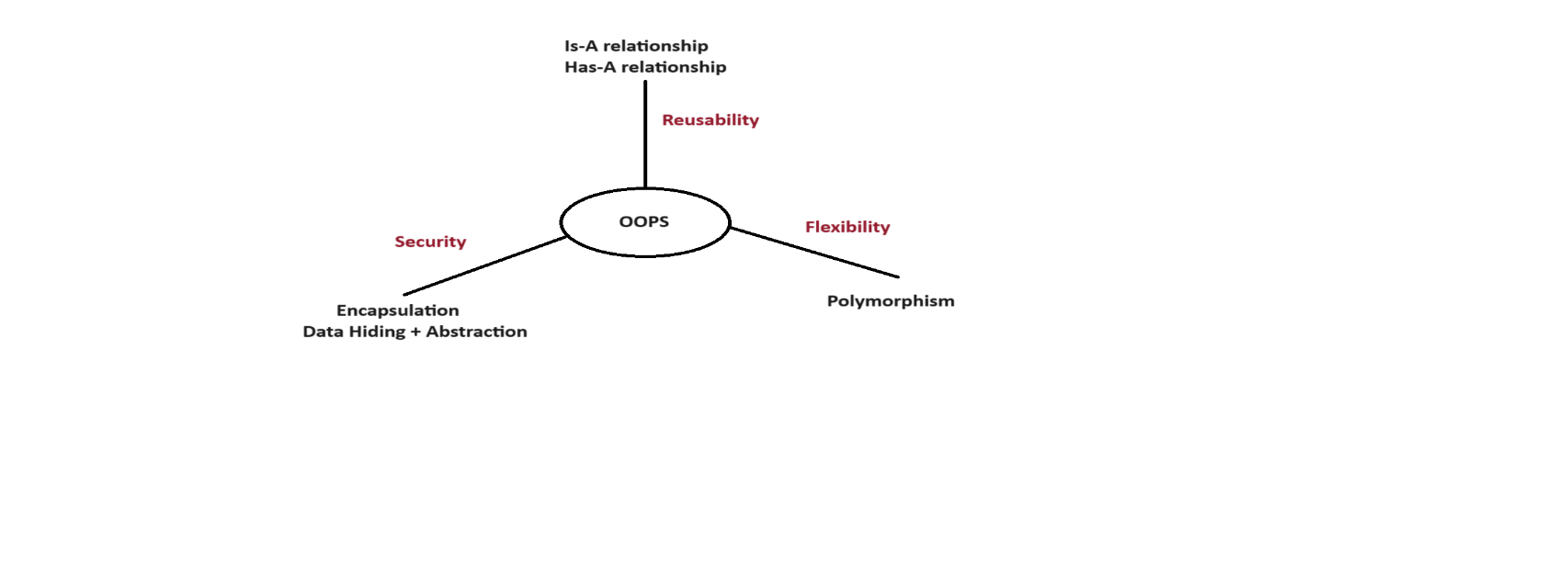
**2) Runtime polymorphism**

A polymorphism which exhibits at runtime is called runtime polymorphism.

ex:

Method overriding

**Summary Diagram: class30.2**



# 21.Constructor

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

ex:

Test t=new Test();

* Having same name as class name is called constructor.
* A constructor will execute when we create an object.
* A constructor does not allow any returntype.
* A constructor will accept following modifiers.

ex:

default

public

private

protected

In java, constructors are divided into two types.

1) Userdefined constructor

2) Default constructor

**1) Userdefined constructor**

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

It is divided into two types.

i) Zero-Argument constructor

ii) Parameterized constructor

**i) Zero-Argument constructor**

Suppose if we are not passing any argument to userdefined constructor is called zero-argument constructor.

ex:

class Test

{

Test()

{

System.out.println("0-arg const");

}

public static void main(String[] args)

{

System.out.println("Main-Method");

}

}

o/p:

Main-Method

ex:

class Test

{

private Test()

{

System.out.println("0-arg const");

}

public static void main(String[] args)

{

System.out.println("Main-Method");

Test t=new Test();

}

}

o/p:

Main-Method

0-arg const

ex:

class Test

{

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();

}

}

**ii) Parameterized constructor**

Suppose if we pass atleast one argument to userdefined constructor then that constructor is called parameterized constructor.

ex:

class Employee

{

//instance variables

//current class variables

private int empId;

private String empName;

private double empSal;

public Employee(int empId,String empName,double empSal)

{

this.empId=empId;

this.empName=empName;

this.empSal=empSal;

}

public void getEmployeeDetails()

{

System.out.println("Employee Id :"+empId);

System.out.println("Employee Name :"+empName);

System.out.println("Employee Salary :"+empSal);

}

}

class Test

{

public static void main(String[] args)

{

Employee e=new Employee(101,"Alan",1000d);

e.getEmployeeDetails();

}

}

**2) Default constructor**

* It is a compiler generated constructor for every java program where we are not defining atleast zero argument constructor.
* To see the default constructor we need to use below command.

ex:

javap -c Test

ex:

class Test

{

public static void main(String[] args)

{

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

}

}

o/p:

javac Test.java

javap -c Test

**Constructor overloading**

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

ex:

class A

{

A()

{

System.out.println("0-arg const");

}

A(int i)

{

System.out.println("int-arg const");

}

A(double d)

{

System.out.println("double-arg const");

}

}

class Test

{

public static void main(String[] args)

{

A a1=new A();

A a2=new A(10);

A a3=new A(10.5d);

}

}

## this keyword

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

We can utilize this keyword in following ways.

i) To refer current class variables

ii) To refer current class methods

iii) To refer current class constructors

**i) To refer current class variables**

class A

{

int i=10;

int j=20;

A(int i,int j)

{

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

System.out.println(this.i+" "+this.j); // 10 20

}

}

class Test

{

public static void main(String[] args)

{

A a=new A(100,200);

}

}

**ii) To refer current class methods**

class A

{

public void m1()

{

System.out.println("M1-Method");

this.m2();

}

public void m2()

{

System.out.println("M2-Method");

}

}

class Test

{

public static void main(String[] args)

{

A a=new A();

a.m1();

}

}

**iii) 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.

i) To refer super class variables

ii) To refer super class methods

iii) To refer super class constructors

**i) To refer super class variables**

class A

{

int i=10;

int j=20;

}

class B extends A

{

int i=100;

int j=200;

B(int i,int j)

{

System.out.println(i+" "+j); // 1000 2000

System.out.println(super.i+" "+super.j); //10 20

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

}

}

class Test

{

public static void main(String[] args)

{

B b=new B(1000,2000);

}

}

**ii) To refer super class methods**

class A

{

public void m1()

{

System.out.println("M1-Method");

}

}

class B extends A

{

public void m2()

{

super.m1();

System.out.println("M2-Method");

}

}

class Test

{

public static void main(String[] args)

{

B b=new B();

b.m2();

}

}

**iii) To refer super class constructors**

class A

{

A()

{

System.out.println("A-constructor");

}

}

class B extends A

{

B()

{

super(); //optional

System.out.println("B-constructor");

}

}

class Test

{

public static void main(String[] args)

{

B b=new B();

}

}

## Interface

* Interface is a collection of abstract methods, default methods and static methods.
* Abstract methods are incomplete methods because they ends with semicolon and don't have any body.

ex:

void m1();

* It is not possible to create object for interfaces.
* To write the implementation of abstract methods an interface we will use implementation class.
* It is possible to create object for implementation class but it contains method with body.
* By default every abstract method is a public and abstract.

ex:

public abstract void m1();

Interface contains only constants i.e public static final.

syntax:

interface <interface\_name>

{

-

- // abstract methods

- // default methods

- // static methods

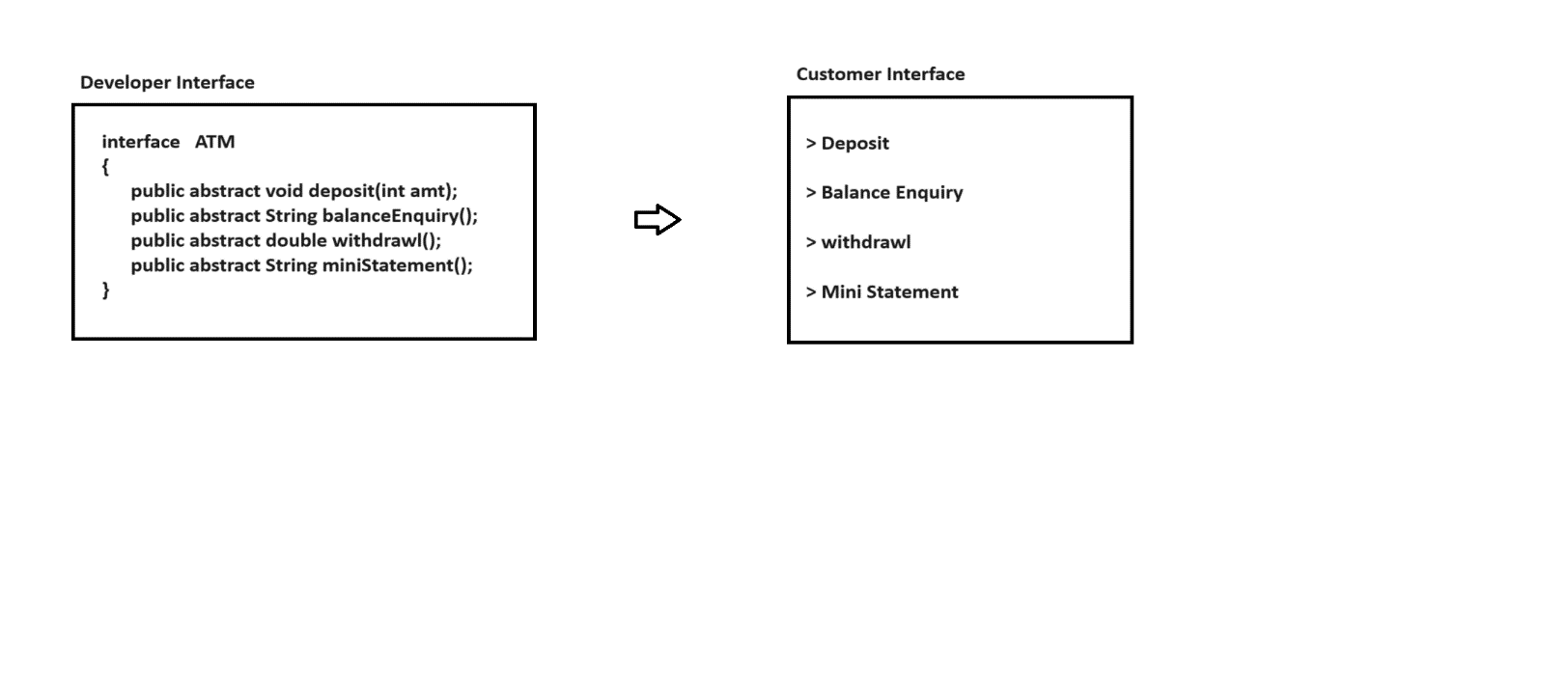
- // constants

-

}

If we know service requirement specification then we need to use interface.

**Diagram: class31.1**



ex:

interface A

{

void m1();

}

class B implements A

{

public void m1()

{

System.out.println("M1-Method");

}

}

class Test

{

public static void main(String[] args)

{

A a=new B();

a.m1();

}

}

ex:

interface A

{

void m1();

}

class Test

{

public static void main(String[] args)

{

A a=new A()

{

public void m1()

{

System.out.println("From M1 Method");

}

};

a.m1();

}

}

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

ex:

interface A

{

public abstract void see();

public void show();

abstract void view();

void display();

}

class B implements A

{

public void see()

{

System.out.println("See Method");

}

public void show()

{

System.out.println("Show Method");

}

public void view()

{

System.out.println("View Method");

}

public void display()

{

System.out.println("Display Method");

}

}

class Test

{

public static void main(String[] args)

{

A a=new B();

a.see();

a.show();

a.view();

a.display();

}

}

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

ex:

interface A

{

public void m1();

}

interface B

{

public void m2();

}

interface C extends A,B

{

public void m3();

}

class D implements C

{

public void m1()

{

System.out.println("M1-Method");

}

public void m2()

{

System.out.println("M2-Method");

}

public void m3()

{

System.out.println("M3-Method");

}

}

class Test

{

public static void main(String[] args)

{

C c=new D();

c.m1();

c.m2();

c.m3();

}

}

A class can implements more then one interface.

ex:

interface Father

{

float HT=6.2f;

void height();

}

interface Mother

{

float HT=5.8f;

void height();

}

class Child implements Father,Mother

{

public void height()

{

float height=(Father.HT+Mother.HT)/2;

System.out.println("Child Height is ="+height);

}

}

class Test

{

public static void main(String[] args)

{

Child c=new Child();

c.height();

}

}

## Marker interface

* An interface which does not have any methods and constants is called marker interface.
* In general, Empty interface is called marker interface.
* Using marker interface we will get some ability to do.

We have following list of marker interfaces.

ex:

Serializable

Remote

Cloneable

and etc.

ex:

import java.io.Serializable;

class Item implements Serializable

{

private int itemId;

private String itemName;

private double itemPrice;

//setter methods

public void setItemId(int itemId)

{

this.itemId=itemId;

}

public void setItemName(String itemName)

{

this.itemName=itemName;

}

public void setItemPrice(double itemPrice)

{

this.itemPrice=itemPrice;

}

//getter methods

public int getItemId()

{

return itemId;

}

public String getItemName()

{

return itemName;

}

public double getItemPrice()

{

return itemPrice;

}

}

**Q) Why can't we create object for interfaces?**

In Java, we cannot create objects of interfaces directly because they are abstract by nature.

## Abstract class

* Abstract class is a collection of zero or more abstract methods and concrete methods.
* A "abstract" keyword we can apply for methods and classes but not for variables.
* It is not possible to create object for abstract class.
* To write the implemenation of abstract methods an abstract class we will use sub classes.
* By default every abstract method is a public and abstract.
* Abstract class contains only instance variables.

syntax:

abstract class <class\_name>

{

-

- //abstract methods

- //concrete methods

- //instance variables

-

}

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

ex:

abstract class Plan

{

//instance variable

protected double rate;

//abstract method

public abstract void getRate();

//concrete method

public void calculateBillAmount(int units)

{

System.out.println("Total Units :"+units);

System.out.println("Total Bill :"+ units\*rate);

}

}

class DomesticPlan extends Plan

{

public void getRate()

{

rate=2.5d;

}

}

class CommercialPlan extends Plan

{

public void getRate()

{

rate=5.0d;

}

}

class Test

{

public static void main(String[] args)

{

DomesticPlan dp=new DomesticPlan();

dp.getRate();

dp.calculateBillAmount(250);

CommercialPlan cp=new CommercialPlan();

cp.getRate();

cp.calculateBillAmount(250);

}

}

**Q) Why we can't create object for abstract class?**

We cannot create an object for an abstract class in Java because it is a incomplete class.

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

**Interface** **Abstract class**

To declare interface we will use interface To declare abstract class we will use abstract keyword. keyword.

It is a collection of abstract methods, It is a collection of abstract methods and default methods and static methods. concrete methods.

We can achieve multiple inheritance. We can't achieve multiple inheritance.

It does not allow constructor. It allows constructor.

It does not allow blocks. It allows blocks.

To write the implementation for abstract To write the implementation for abstract methods methods we will use implementation class. we will use sub class.

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

# 22.API

* API stands for Application Programming Interface.
* API is a collection of packages.
* It is a base for the programmers to develop software applications.

In java, API's are divided into three types.

**1) Predefined API**

Built-In API is called predefined API.

ex:

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

**2) Userdefined API**

API which is created by the user is called userdefined API.

ex:

http://localhost:9090/customer/fetch

**3) Third party API**

API which is given by third party vendors.

ex:

https://restcountries.com/v3.1/all

# 23.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.

A package is also known as folder or a directory.

In java, packages are divided into two types.

1) Predefined packages

2) Userdefined packages

## 1) Predefined packages

Built-In packages are called predefined packages.

ex:

java.lang

java.io

java.time

java.text

java.util

java.util.stream

java.sql

javax.servlet

and etc.

## 2) Userdefined packages

* Packages which are created by the user based on the application requirement are called userdefined packages.
* It is highly recommanded to declare package name in the reverse order of url.

syntax:

package <package\_name>;

**ex:**

package com.ihub.www;

import java.util.Calendar;

class Test

{

public static void main(String[] args)

{

Calendar c=Calendar.getInstance();

//convert time to 24 hours

int h=c.get(Calendar.HOUR\_OF\_DAY);

if(h<12)

System.out.println("Good Morning");

else if(h<16)

System.out.println("Good Afternoon");

else if(h<20)

System.out.println("Good Evening");

else

System.out.println("Good Night");

}

}

We can compile above program by using below command.

ex:

current directory

|

javac -d . Test.java

|

destination folder

We can run above program by using below command.

ex:

java com.ihub.www.Test

|

package name

# 24.Wrapper classes

The main objective of wrapper classes are

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

2) To define several utility methods.

ex:

**primitive type** **Wrapper class**

byte Byte

short Short

int Integer

long Long

float Float

double Double

boolean Boolean

char Character

**constructor**

For every wrapper class we can create two objects.One will take corresponding primitive as an argument and another will take corresponding String as an argument.

ex:

**wrapper class**  **contructor**

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);

System.out.println(i1);

Integer i2=new Integer("20");

System.out.println(i2);

}

}

ex:2

class Test

{

public static void main(String[] args)

{

Boolean b1=new Boolean(true);

System.out.println(b1);

Boolean b2=new Boolean("false");

System.out.println(b2);

}

}

ex:3

class Test

{

public static void main(String[] args)

{

Character c=new Character('a');

System.out.println(c);

}

}

## Utility Methods

**1) valueOf()**

* It is similar to constructor.
* It is used to convert primitive type to wrapper object.

ex:

class Test

{

public static void main(String[] args)

{

Integer i1=Integer.valueOf(10);

System.out.println(i1); //10

Integer i2=Integer.valueOf("20");

System.out.println(i2);//20

}

}

**2) xxxValue()**

It is used to convert primitive type to wrapper object.

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 is used to convert string type to primitive type.

ex

class Test

{

public static void main(String[] args)

{

String str="20";

int i=Integer.parseInt(str);

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

long l=Long.parseLong(str);

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

float f=Float.parseFloat(str);

System.out.println(f); //20.0

}

}

**4) toString()**

It is used to convert wrapper object to string type.

ex:

class Test

{

public static void main(String[] args)

{

Integer i=new Integer(10)

String s=i.toString();

System.out.println(s);

}

}

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

Input:

1010

0101

Output:

1111

ex:

import java.util.Scanner;

class Test

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter the binary1 :");

String binary1=sc.next(); //1010

System.out.println("Enter the binary2 :");

String binary2=sc.next(); //0101

//convert binary to decimal

int a=Integer.parseInt(binary1,2);

int b=Integer.parseInt(binary2,2);

int c=a+b;

//converting decimal to binary

String result=Integer.toBinaryString(c);

System.out.println(result);

}

}

# 25.Enum

* Enum concept introduced in 1.5v.
* Enum is a group of named constants.
* Using enum we can create our own datatype called enumerated datatype.
* When compare to old language enum , java enum is more powerful.

syntax:

enum <type\_name>

{

value1,value2,...,valueN

}

ex:

enum Months

{

JAN,FEB,MAR

}

**Internal implementation of enum**

* Every enum internally consider as class concept and it is extended with java.lang.Enum class.
* Every enum constant is a reference variable of enum type.

ex:

enum Months public final class Months extends java.lang.Enum

{ {

JAN,FEB,MAR ==> public static final Months JAN=new Months();

} public static final Months FEB=new Months();

public static final Months MAR=new Months();

}

**Declaration and Usage of enum**

enum Months

{

JAN,FEB,MAR

}

class Test

{

public static void main(String[] args)

{

Months m=Months.JAN;

System.out.println(m); // JAN

}

}

A switch case allowed enum .

**ex:**

enum Months

{

JAN,FEB,MAR

}

class Test

{

public static void main(String[] args)

{

Months m=Months.FEB;

switch(m)

{

case JAN: System.out.println("January"); break;

case FEB: System.out.println("Febrary"); break;

case MAR: System.out.println("March"); break;

}

}

}

**java.lang.Enum class**

The power to enum will be inherited from java.lang.Enum class.

It contains following two methods.

**1) values()**

It will return group of constants from enum.

**2) ordinal()**

It will return ordinal number.

ex:

enum Week

{

MON,TUE,WED,THU,FRI,SAT,SUN

}

class Test

{

public static void main(String[] args)

{

Week[] w=Week.values();

for(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 variable, method and constructor.

ex:

enum Cloth

{

SILK,COTTON,KHADI;

Cloth()

{

System.out.println("constructor");

}

}

class Test

{

public static void main(String[] args)

{

Cloth c=Cloth.SILK;

}

}

ex:

enum Cloth

{

SILK,COTTON,KHADI;

static int i=10;

public static void main(String[] args)

{

System.out.println(i);

}

}

# 26.Singleton class

* A class which allows us to create only one object is called singleton class.
* It is one of the design pattern which allows us to create only one object for the class.
* If we call any method using class name and that method returns same class object is called singleton class.

ex:

Calendar

LocalDate

LocalTime

and etc.

To create a singleton class we required 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());

}

}

# 27.Inner classes

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

ex:

class Outer\_class

{

class Inner\_class

{

-

- //code to be declare

-

}

}

* Inner classes introduced as a part of event handling to remove GUI bugs.
* But due to powerful features and benefits of inner classes, programmers started to use inner classes in regular programming.
* According to Java 8, Inner class does not allow static members.

**Accessing inner class data from static area of outer class**

class Outer

{

class Inner

{

public void m1()

{

System.out.println("M1-Method");

}

}

public static void main(String[] args)

{

Outer.Inner i=new Outer().new Inner();

i.m1();

}

}

Note:

If we compile above program then two .class files will be generated i.e

Outer.class and Outer$Inner.class.

ex:

class Outer

{

class Inner

{

public void m1()

{

System.out.println("M1-Method");

}

}

public static void main(String[] args)

{

new Outer().new Inner().m1();

}

}

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

class Outer

{

class Inner

{

public void m1()

{

System.out.println("M1-Method");

}

}

public void m2()

{

Inner i=new Inner();

i.m1();

}

public static void main(String[] args)

{

Outer o=new Outer();

o.m2();

}

}

# 28.Types of objects

We have two types of objects in java.

1) Immutable object

2) Mutable object

## 1) Immutable object

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

ex:

String and Wrapper classes

## 2) Mutable object

After object creation if we perform any changes then all the changes will reflect to single object

such type of object is called mutable object.

ex:

StringBuffer and StringBuilder

# 29.String

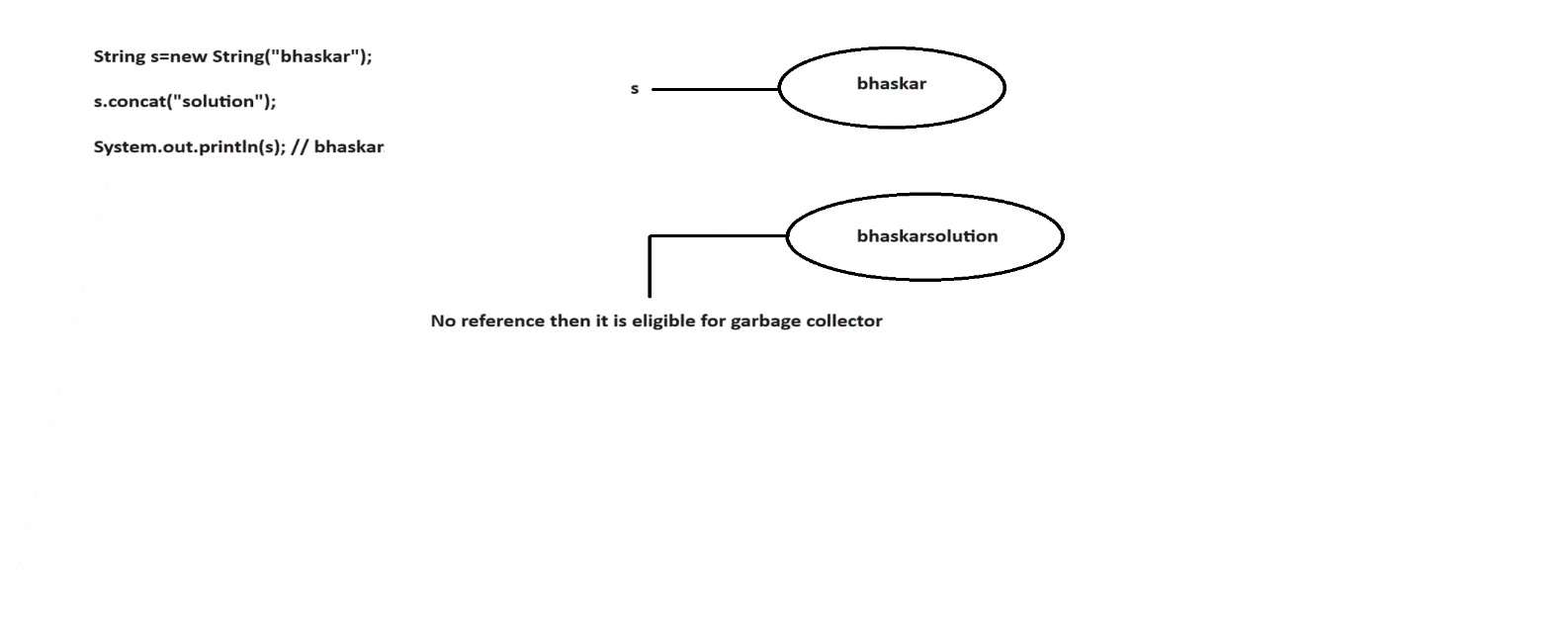
* It is a collection of characters which is enclosed in a double quotation.
* A String is a immutable object.

**case1:**

Once if we create a String object we can't perform any changes.For every change a new object

will be created such behaviour is called immutability of an object.

**Diagram: class34.1**



**case2:**

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

* It is a comparision operator which returns boolean value.
* It is used to reference comparision or address comparision.

ex:

class Test

{

public static void main(String[] args)

{

String s1=new String("bhaskar");

String s2=new String("bhaskar");

System.out.println(s1==s2); //false

}

}

**.equals()**

* It is a method present in String class which returns boolean value.
* It is used for content comparision.
* It is a case sensitive.

ex:

class Test

{

public static void main(String[] args)

{

String s1=new String("bhaskar");

String s2=new String("bhaskar");

System.out.println(s1.equals(s2)); // true

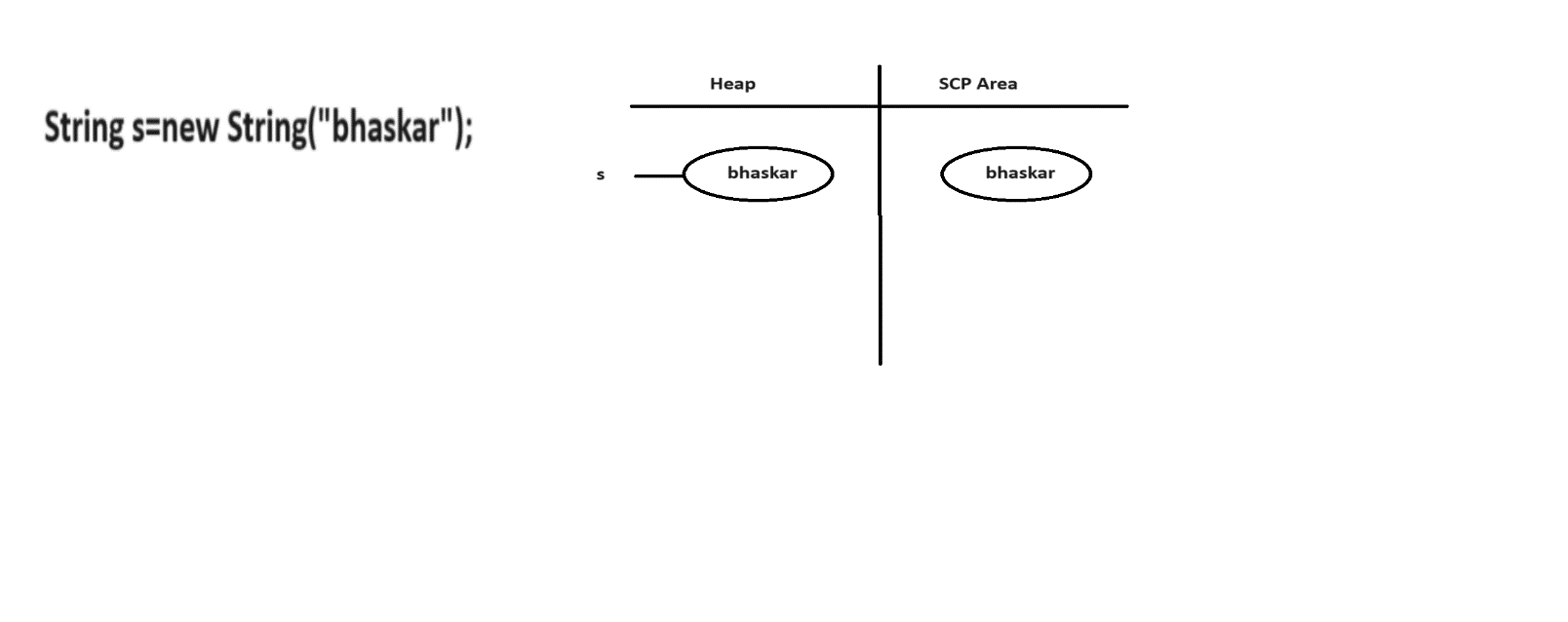
}

}

**case3:**

Whenever we create String object.Two objects will be created internally.One is on heap and another is on SCP (String Constant Pool) area.But 's' points to heap area only.

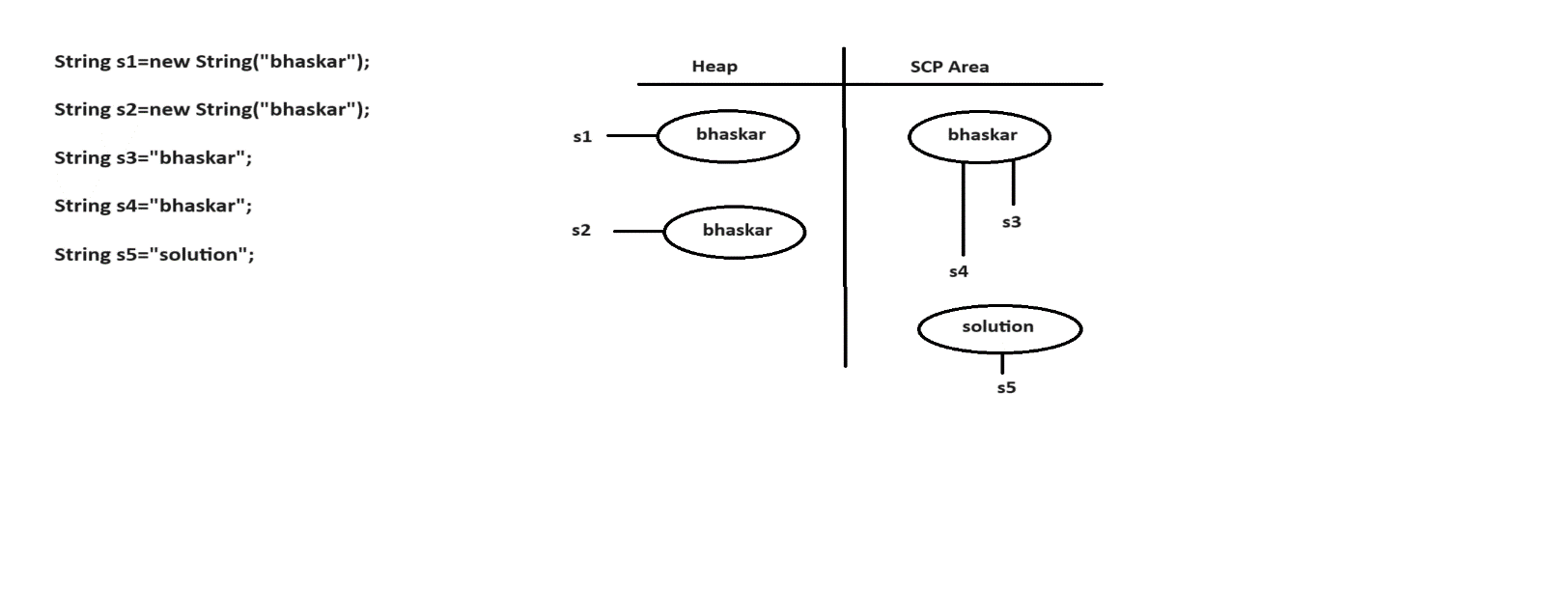
**Diagram: class34.2**



Object creations in SCP area is always optional. JVM will check is there any object is created with same content or not.If it is created then JVM simply refers to that object.If it is not created then JVM will create a new object.Hence there is no chance of having duplicates objects in SCP area.

* SCP area objects do not have any reference even though garbage collector can't access them.
* SCP area objects will destroy automatically whenever JVM shutdowns or terminated.

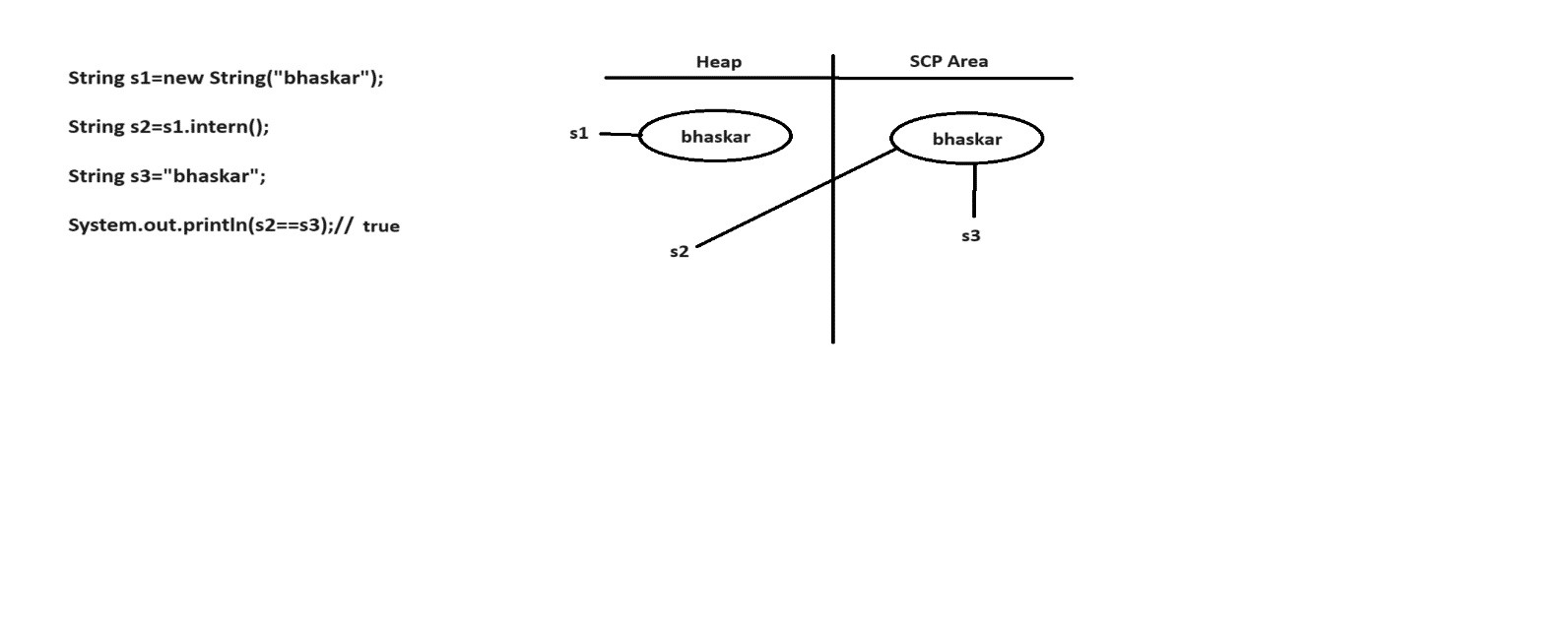
**Diagram: class34.3**



**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: class34.4**



## String important methods

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

Input:

Hello

output:

5

ex:

class Test

{

public static void main(String[] args)

{

String str="Hello";

int len=str.length();

System.out.println(len);

}

}

**Q) Write a java program to convert uppercase string to lowercase string?**

input:

RAJA@GMAIL.COM

output:

raja@gmail.com

class Test

{

public static void main(String[] args)

{

String str="RAJA@GMAIL.COM";

str=str.toLowerCase();

System.out.println(str);

}

}

**Q) Write a java program to convert lowercase string to uppercase string?**

input:

ihub

output:

IHUB

ex

class Test

{

public static void main(String[] args)

{

String str="ihub";

str=str.toUpperCase();

System.out.println(str);

}

}

**Q) Write a java program to check both strings are equal or not?**

input:

bhaskar

bhaskar

output:

Both are equals

ex:

class Test

{

public static void main(String[] args)

{

String str1="bhaskar";

String str2="bhaskar";

if(str1.equals(str2))

System.out.println("Both are equals");

else

System.out.println("Both are not equals");

}

}

Q) Write a java program to check both strings are equal or not?

input:

bhaskar

BHASKAR

output:

Both are equals

ex:

class Test

{

public static void main(String[] args)

{

String str1="bhaskar";

String str2="BHASKAR";

if(str1.equalsIgnoreCase(str2))

System.out.println("Both are equals");

else

System.out.println("Both are not equals");

}

}

**Q) Write a java program to return a character based on given index?**

input:

str = bhaskar

index = 3

output:

s

ex:

class Test

{

public static void main(String[] args)

{

String str="bhaskar";

int index=3;

char ch=str.charAt(index);

System.out.println(ch);

}

}

**Q) Write a java program to count number of vowels present in string?**

input:

umbrella

output:

3

ex:

class Test

{

public static void main(String[] args)

{

String str="umbrella";

int cnt=0;

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

{

if(str.charAt(i)=='a' || str.charAt(i)=='e' ||str.charAt(i)=='i' || str.charAt(i)=='o' || str.charAt(i)=='u')

{

cnt++;

}

}

System.out.println(cnt);

}

}

**Q) Write a java program to display number of upper case letters, lowercase letters, digits,words and spaces in a given string?**

input:

This Is Java Class34

output:

uppercase letters : 4

lowercase lettesr : 11

Digits : 2

words : 4

spaces : 3

ex:

class Test

{

public static void main(String[] args)

{

String str="This Is Java Class34";

int upper=0,lower=0,digit=0,space=0,word=1;

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

{

if(str.charAt(i)>='A' && str.charAt(i)<='Z')

{

upper++;

}

else if(str.charAt(i)>='a' && str.charAt(i)<='z')

{

lower++;

}

else if(str.charAt(i)>='0' && str.charAt(i)<='9')

{

digit++;

}

else if(str.charAt(i)==' ')

{

space++;

word++;

}

}

System.out.println("uppercase letters :"+upper);

System.out.println("lowercase letters :"+lower);

System.out.println("Digits :"+digit);

System.out.println("Words :"+word);

System.out.println("Spaces :"+space);

}

}

**Q) Write a java program to return first index of a given character?**

input:

str = bhaskar

ch = 'a'

output:

2

ex:

class Test

{

public static void main(String[] args)

{

String str="bhaskar";

char ch='a';

int index=str.indexOf(ch);

System.out.println(index);

}

}

**Q) Write a java program to return last index of a given character?**

input:

str = bhaskar

ch = 'a'

output:

5

ex:

class Test

{

public static void main(String[] args)

{

String str="bhaskar";

char ch='a';

int index=str.lastIndexOf(ch);

System.out.println(index);

}

}

**Q) Write a java program to concatinate two strings?**

input:

ihub

talent

output:

ihubtalent

ex:

class Test

{

public static void main(String[] args)

{

String str1="ihub";

String str2="talent";

String result=str1.concat(str2);

System.out.println(result);

}

}

**Q) Write a java program to remove the spaces from given string?**

input:

I hub Tale nt

output:

IhubTalent

ex:

class Test

{

public static void main(String[] args)

{

String str="I hub Tale nt";

str=str.replaceAll("\\s","");

System.out.println(str);

}

}

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

input:

I@hub\_Ta$le#nt

output:

IhubTalent

ex:

class Test

{

public static void main(String[] args)

{

String str="I@hub\_Ta$le#nt";

str=str.replaceAll("[^A-Za-z0-9]","");

System.out.println(str);

}

}

**Q) Write a java program to display the string in a given format?**

input:

ihub27

talent25

output:

ihubtalent52

ex:

class Test

{

public static void main(String[] args)

{

String str1="ihub27";

String str2="talent25";

String word1=str1.replaceAll("[^A-Za-z]","");

int num1=Integer.parseInt(str1.replaceAll("[^0-9]",""));

String word2=str2.replaceAll("[^A-Za-z]","");

int num2=Integer.parseInt(str2.replaceAll("[^0-9]",""));

String word=word1+word2;

int num=num1+num2;

System.out.println(word+num);

}

}

**Q) Write a java program to return a sub string from a given string?**

input:

str = bhaskar

index = 4

output:

kar

ex:

class Test

{

public static void main(String[] args)

{

String str="bhaskar";

int index=4;

String result=str.substring(index);

System.out.println(result);

}

}

**Q) Write a java program to return sub string of a given string?**

input:

ihubtalentinjava

output:

talent

ex:

class Test

{

public static void main(String[] args)

{

String str="ihubtalentinjava";

String result=str.substring(4,10);

System.out.println(result);

}

}

**Q) Write a java program to insert a given string based on the index?**

input:

str = ihubtalent

index = 4

word = for

output:

ihubfortalent

ex:

class Test

{

public static void main(String[] args)

{

String str="ihubtalent";

int index =4;

String word="for";

String str1=str.substring(0,index);

String str2=str.substring(index,str.length());

String result=str1+word+str2;

System.out.println(result);

}

}

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

input:

hello

output:

olleh

ex:

class Test

{

public static void main(String[] args)

{

String str="hello";

char[] carr=str.toCharArray(); // h e l l o

String rev="";

for(int i=carr.length-1;i>=0;i--)

{

rev+=carr[i];

}

System.out.println(rev);

}

}

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

input:

racar

output:

It is a palindrome string

ex:

class Test

{

public static void main(String[] args)

{

String str="racar";

char[] carr=str.toCharArray(); // r a c a r

String rev="";

for(int i=carr.length-1;i>=0;i--)

{

rev+=carr[i];

}

if(str.equals(rev))

System.out.println("It is a palindrome string");

else

System.out.println("It is not a palindrome string");

}

}

Q) Write a java program to display reverse of a sentence?

input:

This is java class

output:

class java is This

ex:

class Test

{

public static void main(String[] args)

{

String str="This is java class";

String[] sarr=str.split(" "); // This is java class

String rev="";

for(int i=sarr.length-1;i>=0;i--)

{

rev+=sarr[i]+" ";

}

System.out.println(rev);

}

}

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

input:

This is java class

output:

sihT si avaj ssalc

ex:

class Test

{

public static void main(String[] args)

{

String str="This is java class";

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

String rev="";

//for each loop

for(String s:sarr)

{

char[] carr=s.toCharArray(); // T h i s

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 the string starting with upper case letter?**

input:

This is Java class For students

output:

This Java For

ex:

class Test

{

public static void main(String[] args)

{

String str="This is Java class For students";

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

String rev="";

for(String s:sarr)

{

if(s.charAt(0)>='A' && s.charAt(0)<='Z')

{

rev+=s+" ";

}

}

System.out.println(rev);

}

}

**Q) Write a java program to the string present in odd position ?**

input:

This is java class for students

output:

is class students

ex:

class Test

{

public static void main(String[] args)

{

String str="This is Java class For students";

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

for(int i=0;i<sarr.length;i++)

{

if(i%2!=0)

{

System.out.print(sarr[i]+" ");

}

}

}

}

**Assignment**

**Q) Write a java program to display palindrome strings?**

input:

racar is madam for dad

output:

racar madam dad

import java.util.Scanner;

class Test

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter the Sentence:");

String str=sc.nextLine();

String[] sr=str.split(" ");

for(String s:sr)

{

String rev="";

char[] ch=s.toCharArray();

for(int i=ch.length-1;i>=0;i--)

{

rev+=ch[i];

}

if (s.equals(rev))

{

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

}

}

}

}

**Q) Write a java program to display the given string in a below format?**

input:

A1B2C3D4

output:

ABBCCCDDDD

ex

class Test

{

public static void main(String[] args)

{

String str="A1B2C3D4";

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

{

if(Character.isAlphabetic(str.charAt(i)))

{

System.out.print(str.charAt(i));

}

else

{

int k=Character.getNumericValue(str.charAt(i));

for(int j=1;j<k;j++)

{

System.out.print(str.charAt(i-1));

}

}

}

}

}

**Q) Write a java program to display plaindrome strings**

input:

racar is madam for da

output:

racar madam dad

ex:

class Test

{

public static void main(String[] args)

{

String str="racar is madam for dad"

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

//for each loop

for(String s:sarr)

{

char[] carr=s.toCharArray(); // r a c a r

String rev="";

for(int i=carr.length-1;i>=0;i--)

{

rev+=carr[i];

if(s.equals(rev))

{

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

}

}

}

**Q) Write a java program to display the string in a given format**

input:

XYZ

output:

XY

XZ

YX

YZ

ZX

ZY

ex:

class Test

{

public static void main(String[] args)

{

String str="XYZ";

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

{

for(int j=0;j<str.length();j++)

{

if(i!=j)

{

System.out.println(str.charAt(i)+""+str.charAt(j));

}

}

}

}

}

**Q) Write a java program to check given string is Anagram or not**

input:

str1= "silent";

str2= "listen";

output:

It is a Anagram string

**Explaination**

silent : e i l n s t

listen : e i l n s t

ex:

import java.util.Arrays;

class Test

{

public static void main(String[] args)

{

String str1= "silent";

String str2= "listen";

//convert string to char array

char[] carr1=str1.toCharArray();

char[] carr2=str2.toCharArray();

//sort the array

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

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

boolean flag=true;

for(int i=0;i<carr1.length && i<carr2.length; i++)

{

if(carr1[i] != carr2[i])

{

flag=false;

break;

}

}

if(flag==true)

System.out.println("It is a Anagram string");

else

System.out.println("It is not a Anagram string");

}

}

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

input:

ABC

output:

ABC

ACB

BAC

BCA

CBA

CAB

ex:

class Test

{

public static void main(String[] args)

{

String str="ABC";

//caller method

permutation(str.toCharArray(),0);

//callie method

public static void permutation(char[] arr,int fi)

{

if(fi==arr.length-1)

{

System.out.println(arr);

return;

}

for(int i=fi;i<arr.length;i++)

{

swapping(arr,fi,i);

permutation(arr,fi+1);

swapping(arr,fi,i);

}

}

//callie method

public static void swapping(char[] arr,int fi,int i)

{

char temp=arr[fi];

arr[fi]=arr[i];

arr[i]=temp;

}

}

## StringBuffer

* If our content change frequently then it is never recommanded to go with String.
* To overcome this limitation Sun Micro System introduced StringBuffer.
* In StringBuffer, All the required changes will be done in a single object.
* It is mutable object.

**constructors**

**1) StringBuffer sb=new StringBuffer();**

* It will create empty StringBuffer object with default initial capacity of 16.
* If we 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 which is equivalent to String.
* Here capacity will be created with below formulea.

ex:

capacity = s.length() + 16.

ex:

class Test

{

public static void main(String[] args)

{

StringBuffer sb=new StringBuffer("bhaskar");

System.out.println(sb.capacity()); //7+16=23

}

}

**Q) Write a java program to display reverse of a string?**

input:

hello

output:

olleh

ex:

class Test

{

public static void main(String[] args)

{

String str="hello";

StringBuffer sb=new StringBuffer(str);

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

System.out.println(rev);

}

}

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

input:

racar

output:

It is a palindrome string

ex:

class Test

{

public static void main(String[] args)

{

String str="racar";

StringBuffer sb=new StringBuffer(str);

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

if(str.equals(rev))

System.out.println("It is a palindrome string");

else

System.out.println("It is not a palindrome string");

}

}

**Q) Write a java program to display the string in a given format?**

input:

ABBCCCDDDD

output:

A1B2C3D4

ex:

class Test

{

public static void main(String[] args)

{

String str="ABBCCCDDDD";

StringBuffer sb=new StringBuffer();

int count=1;

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

{

if(i<str.length()-1 && str.charAt(i) == str.charAt(i+1))

{

count++;

}

else

{

sb.append(str.charAt(i)).append(count);

count=1;

}

}

System.out.println(sb.toString());

}

}

**Q) Write a java program to multiply two arrays?**

input:

arr1 = 4 6 1

arr2 = 1 5

output:

6915 (461\*15)

ex:

class Test

{

public static void main(String[] args)

{

int[] arr1={4,6,1};

int[] arr2={1,5};

**//caller method**

String str1=arrayToString(arr1);

String str2=arrayToString(arr2);

int a=Integer.parseInt(str1);

int b=Integer.parseInt(str2);

System.out.println(a\*b);

}

**//callie method**

public static String arrayToString(int[] arr)

{

StringBuffer sb=new StringBuffer();

for(int i:arr)

{

sb.append(i);

}

return sb.toString();

}

}

## StringBuilder

* StringBuilder is exactly same as StringBuffer with following differences.

**StringBuffer** **StringBuilder**

Methods present in StringBuffer are No method present in StringBuilder is synchronized. synchronized.

At a time only one thread is allowed to Multiple threads are allowed to access

access StringBuffer.Hence we can achieve StringBuilder object.Hence we can't

thread safety. Achieve thread safety.

Waiting time of a thread will increase There is no waiting threads effectively performance is low. effectively performance is high.

It is introduced in 1.0v. It is introduced in 1.5v.

Note:

* If our content not change fequently then we need to use String.
* If our content change frequently where thread safety is required then we need to use StringBuffer.
* If our content change frequently where thread safety is not required then we need to use StringBuilder.

## StringTokenizer

* StringTokenizer is a class which is present in java.util package.
* It is used to tokenize the string irrespective of regular expression.

We can create StringTokenizer object as follow.

ex:

StringTokenizer st=new StringTokenizer(String s,RegularExpression regex);

StringTokenizer class contains following five methods.

ex:

public boolean hasMoreTokens()

public String nextToken()

public boolean hasMoreElements()

public Object nextElement()

public int countTokens()

ex:1

import java.util.\*;

class Test

{

public static void main(String[] args)

{

StringTokenizer st=new StringTokenizer("This is java class");

System.out.println(st.countTokens());

}

}

Note:

Default regular expression is space.

**ex:2**

import java.util.\*;

class Test

{

public static void main(String[] args)

{

StringTokenizer st=new StringTokenizer("This is java class"," ");

System.out.println(st.countTokens());

}

}

ex:3

import java.util.\*;

class Test

{

public static void main(String[] args)

{

StringTokenizer st=new StringTokenizer("This is java class"," ");

while(st.hasMoreTokens())

{

String s=st.nextToken();

System.out.println(s);

}

}

}

ex:4

import java.util.\*;

class Test

{

public static void main(String[] args)

{

StringTokenizer st=new StringTokenizer("This is java class"," ");

while(st.hasMoreElements())

{

String s=(String)st.nextElement();

System.out.println(s);

}

}

}

ex:5

import java.util.\*;

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);

}

}

}

## Math

* A Math class present in java.lang package.
* It is used to perform mathematical calculations.
* A Math class contains following methods.

ex:

max()

min()

ceil()

floor()

power() and etc.

**ex:**

class Test

{

public static void main(String[] args)

{

int max=Math.max(10,20);

System.out.println(max);

}

}

ex:

class Test

{

public static void main(String[] args)

{

int min=Math.min(10,20);

System.out.println(min);

}

}

ex:

class Test

{

public static void main(String[] args)

{

int result=(int)Math.pow(5,3);

System.out.println(result);

}

}

ex:

class Test

{

public static void main(String[] args)

{

int result=(int)Math.sqrt(25);

System.out.println(result);

}

}

# 30.Exception Handling

**Q) What is the difference between Exception and Error?**

**Exception**

* Exception is a problem for which we can provide solution programmatically.
* Exception will raise due to syntax errors.

ex:

ArithmeticException

FileNotFoundException

IllegalArgumentException

and etc.

**Error**

* Error is a problem for which we can't provide solution programmatically.
* Error will raise due to lack of system resources.

ex:

OutOfMemoryError

LinkageError

StackOverFlowError

and etc.

As a part of java application development it is a responsibility of a programmer to provide smooth termination for every java program.

**We have two types of terminations.**

**1) Smooth termination / Graceful termination**

During the program execution suppose if we are not getting any interruption in the middle of the program such type of termination is called smooth termination.

ex:

class Test

{

public static void main(String[] args)

{

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

}

}

**2) Abnormal termination**

During the program execution suppose if we are getting any interruption in the middle of the program such type of termination is called abnormal termination.

ex:

class Test

{

public static void main(String[] args)

{

System.out.println(10/0);

}

}

If any exception raised in our program , we must and should handle that exception otherwise our program will terminate abnormally.

Here exception will display name of the exception, description of the exception and line number of the exception.

## Exception

* It is a unwanted, unexpected event which disturbs normal flow a program.
* Exceptions always raised at runtime so they are also known as runtime events.
* The main objective of exception handling is to provide graceful termination.

In java, exceptions are divided into two types.

1) Predefined exceptions

2) Userdefined exceptions

## 1) Predefined exceptions

Built-In exceptions are called predefined exceptions.

They are classified into two types.

**i) Checked exceptions**

Exceptions which are checked by the compiler at the time of compilation are called checked exceptions.

ex:

InterruptedException

EOFException

FileNotFoundException

and etc.

**ii) Unchecked exceptions**

Exceptions which are checked by the JVM at the time of runtime are called unchecked exceptions.ex:

ex:

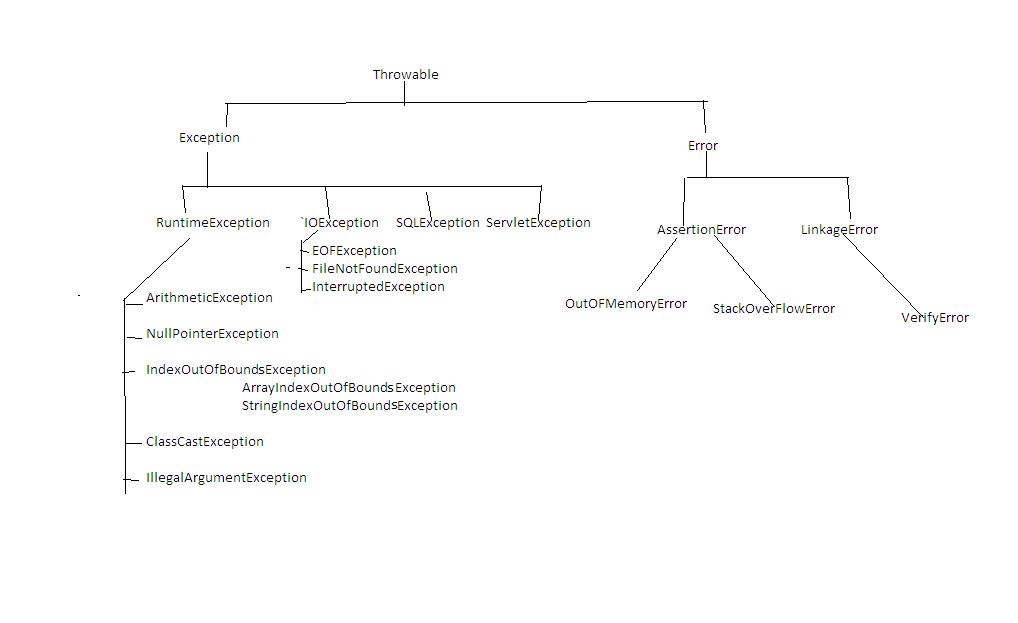
ArithmeticException

IllegalArgumentException

ClassCastException

and etc.

**Diagram: class38.1**



If any checked exception raise in our program we must and should handle that exception by using try and catch block.

## try block

* It is a block which contains risky code.
* A try block associate with catch block.
* A try block is used to throw the exception to catch block.
* If any exception raise in try block then it won't be executed.

## catch block

* It is a block which contains error handling code.
* A catch block always associate with try block.
* A catch block is used to catch the exception which is thrown by try block.
* If there is no exception in try block then catch block won't be executed.
* A catch block will take exception name as a parameter and that name must match with exception class name.

syntax:

try

{

-

- // Risky Code

-

}

catch(ArithmeticException ae)

{

-

- // Error Handling Code

-

}

ex:

class Test

{

public static void main(String[] args)

{

try

{

System.out.println("try-block");

}

catch (ArithmeticException ae)

{

System.out.println("catch-block");

}

}

}

o/p:

try-block

**ex:2**

class Test

{

public static void main(String[] args)

{

try

{

System.out.println(10/0);

}

catch (ArithmeticException ae)

{

System.out.println("catch-block");

}

}

}

o/p:

catch-block

ex:3

class Test

{

public static void main(String[] args)

{

try

{

System.out.println("stmt1");

System.out.println(10/0);

System.out.println("stmt2");

}

catch (ArithmeticException ae)

{

System.out.println("catch-block");

}

}

}

o/p:

stmt1

catch-block

**ex:4**

class Test

{

public static void main(String[] args)

{

try

{

return 10;

}

catch (ArithmeticException ae)

{

return 20;

}

}

}

**ex:**

class Test

{

public static void main(String[] args)

{

System.out.println(m1());

}

public static int m1()

{

try

{

return 10;

}

catch (ArithmeticException ae)

{

return 20;

}

}

}

o/p:

10

## A try with multiple catch blocks

* A try block can have multiple catch blocks.
* If a try block contains multiple catch blocks then order of catch blocks are very important it should be from child to parent but not from parent to child.

ex:

class Test

{

public static void main(String[] args)

{

try

{

System.out.println(10/0);

}

catch(ArithmeticException ae)

{

System.out.println("From AE");

}

catch(RuntimeException re)

{

System.out.println("From RE");

}

catch(Exception e)

{

System.out.println("From E");

}

}

}

## Various methods to display exception details

Throwable class defines following three methods to display exception details.

**1) printStackTrace()**

It will display name of the exception, description of the exception and line number of the exception.

**2) toString()**

It will display name of the exception and description of the exception.

**3) getMessage()**

It will display description of the exception.

ex:

class Test

{

public static void main(String[] args)

{

try

{

System.out.println(10/0);

}

catch(Exception e)

{

e.printStackTrace();

System.out.println("================");

System.out.println(e.toString());

System.out.println("================");

System.out.println(e.getMessage());

}

}

}

**Q) How can we handle multiple exceptions in a single catch block?**

class Test

{

public static void main(String[] args)

{

try

{

System.out.println(10/0);

}

catch(ArithmeticException | IllegalArgumentException | ClassCastException e)

{

e.printStackTrace();

}

}

}

## finally block

* It is never recommanded to maintain cleanup code in try block because if any exception raise then try block won't be executed.
* It is never recommanded to maintain cleanup code in catch block because if there is no exception in try block then catch block won't be executed.
* But we need a place where we can maintain cleanup code and it should execute irrespective of exception raise or not. such type of block is called finally block.

syntax:

try

{

-

- // Risky Code

-

}

catch(Exception e)

{

-

- //Error handling code

-

}

finally

{

-

- //Cleanup Code

-

}

ex:

class Test

{

public static void main(String[] args)

{

try

{

System.out.println("try-block");

}

catch(Exception e)

{

e.printStackTrace();

}

finally

{

System.out.println("finally-block");

}

}

}

o/p:

try-block

finally-block

**ex:**

class Test

{

public static void main(String[] args)

{

try

{

System.out.println(10/0);

}

catch(Exception e)

{

e.printStackTrace();

}

finally

{

System.out.println("finally-block");

}

}

}

o/p:

java.lang.ArithmeticException: / by zero

at Test.main(Test.java:7)

finally-block

* A try with finally combination is valid in java.

ex:

class Test

{

public static void main(String[] args)

{

try

{

System.out.println("try-block");

}

finally

{

System.out.println("finally-block");

}

}

}

o/p:

try-block

finally-block

**Q) What is the difference between final, finally and finalize method ?**

**final**

* It is a modifier which is applicable for variables, methods and classes.
* If we declare any variable as final then reassignment of that variable is not possible.
* If we declare any method as final then overriding of 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 raise or not.

**finalize**

It is a method called by garbage collector just before destroying an object for cleanup acitivity.

## throw statement

* Sometimes we will create exception object explicitly and handover to JVM manually by using throw statement.

ex:

class Test

{

public static void main(String[] args)

{

throw new ArithmeticException("don't divide by zerooo");

}

}

**throws Statement**

* If any checked exception raised in our program so we must and should handle that exception by using try and catch block or by using throws statement.

ex:

class Test

{

public static void main(String[] args)

{

try

{

Thread.sleep(4000);

System.out.println("Welcome to Java");

}

catch (InterruptedException ie)

{

ie.printStackTrace();

}

}

}

**ex:**

class Test

{

public static void main(String[] args)throws InterruptedException

{

Thread.sleep(5000);

System.out.println("Welcome to Java");

}

}

## 2) Userdefined exceptions

Exceptions which are created by the user based on the application requirements are called userdefined exceptions or customized exceptions.

ex:

StudentsNotPracticingException

NoInterestInJobException

ComingToClassForACException

TooYoungException

TooOldException

and etc.

**ex:**

import java.util.Scanner;

class TooYoungException extends RuntimeException

{

TooYoungException(String s)

{

super(s);

}

}

class TooOldException extends RuntimeException

{

TooOldException(String s)

{

super(s);

}

}

class Test

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter the Age :");

int age=sc.nextInt();

if(age<18)

throw new TooYoungException("U r not eligible to vote");

else

throw new TooOldException("U r eligible to vote");

}

}

**Accenture Question**

**Here the given integer is 34, and the square its digits are.**

**. 3^2 = 9**

**. 4^2 = 16**

**On concatenating the squares, we get 916. Therefore , 916 is return as the output.**

ex:

class Test

{

public static void main(String[] args)

{

int n=34;

int result=digitSquare(n);

System.out.println(result);

}

//callie method

public static int digitSquare(int n) // s = 34

{

StringBuffer sb=new StringBuffer();

String s=Integer.toString(n);

for(int i=0;i<s.length();i++)

{

int digit=Character.getNumericValue(s.charAt(i));

int square=digit\*digit;

sb.append(square);

}

return Integer.parseInt(sb.toString());

}

}

# 31.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");

}

}

ex:2

import java.io.\*;

class Test

{

public static void main(String[] args)

{

FileWriter fw=null;

try

{

fw=new FileWriter("aaa.txt");

fw.write(98); // b

fw.write("\n");

char[] ch={'a','b','c'};

fw.write(ch);

fw.write("\n");

fw.write("bhaskar\nsolution");

fw.flush();

System.out.println("Please check the location");

}

catch (IOException ioe)

{

ioe.printStackTrace();

}

finally

{

try

{

fw.close();

}

catch (IOException ioe)

{

ioe.printStackTrace();

}

}

}

}

**Try with Resources**

* The try-with-resources statement is a try statement that declares one or more resources.
* The try -with-resources statement ensures that each resource is closed at the end of the statement.

ex:

import java.io.\*;

class Test

{

public static void main(String[] args)

{

try(FileWriter fw=new FileWriter("a1.txt");)

{

fw.write(98); // b

fw.write("\n");

char[] ch={'a','b','c'};

fw.write(ch);

fw.write("\n");

fw.write("bhaskar\nsolution");

fw.flush();

System.out.println("Please check the location");

}

catch (IOException ioe)

{

ioe.printStackTrace();

}

}

}

## 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 characterby 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

We have following ways to provide input values from keyboard.

1) Command Line Argument

2) BufferedReader class

3) Console 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) BufferedReader class**

* BufferedReader class present in java.io package.
* BufferedReader class will take the support of InputStreamReader object as a parameter which is embedded with System.in.

ex:

BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

* BufferedReader class contains readLine() method to read the inputs from console.

ex:

import java.io.\*;

class Test

{

public static void main(String[] args)throws IOException

{

BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

System.out.println("Enter the Name :");

String name=br.readLine();

System.out.println("Welcome :"+name);

}

}

**3)Console class**

* Console class present in java.io package.
* We can create Console class object as follow.

ex:

Console c=System.console();

* Console class contains readLine() method to read inputs from console.

ex:

import java.io.\*;

class Test

{

public static void main(String[] args)throws IOException

{

Console c=System.console();

System.out.println("Enter the Name :");

String name=c.readLine();

System.out.println("Welcome :"+name);

}

}

**4) Scanner class**

* Scanner class present in java.util package.
* We can create Scanner class object as follow.

ex:

Scanner sc=new Scanner(System.in);

* Scanner class contains nextXxx() to read inputs from console.
* Here nextXxx() method means nextInt(), nextFloat(), nextDouble() and etc.

ex:

import java.util.\*;

class Test

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter the no :");

int no=sc.nextInt();

System.out.println("Enter the Name :");

String name=sc.next();

System.out.println("Enter the fee :");

double fee=sc.nextDouble();

System.out.println(no+" "+name+" "+fee);

}

}

## Generics

* 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 go with String[] array.

ex:

String[] sarr=new String[10];

sarr[0]="hi";

sarr[1]="bye";

sarr[2]=10; //invalid

* At the time of retrieving the data from array we don't need to perform typecasting.

ex:

String[] sarr=new String[10];

sarr[0]="hi";

sarr[1]="bye";

-

-

String val1=sarr[0];

* Collections are not typesafe.We can't provide gurantee that what type of elements are present in Collections.
* If requirement is there to store String values then it is never recommnaded to go with ArrayList class.here we won't get any compile time error or runtime error but some times our program will get failure.

ex:

ArrayList al=new ArrayList();

al.add("hi");

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("bye");

al.add(10);

-

-

String val1=(String)al.get(0);

To overcome this limitation Sun Micro System introduced Generics concept in 1.5v.

The main objective of Generics are

1) The make Collections as typesafe.

2) To avoid typecasting problem.

**java.util package**

Q) What is the difference between Arrays and Collections ?

**Arrays Collections**

It is a collection of homogeneous data It is a collection of homogeneous and elements. hetrogeneous data elements.

Arrays are fixed in size. Collections are growable in nature.

Performance point of view arrays are Memory point of view Collections are

recommanded to use. recommanded to use.

Arrays not implemented based on Collections are implemented based on data

data structure concept. structure concept.Hence we can expect ready

Hence we can't expect any ready made made methods.

methods.

It can hold primitive types and object types. It can hold only object types.

**Collection Framework**

It defines several classes and interfaces to represent group of individual objects in a single entity.

# 32.Collection

* It is an interface which is present in java.util package.
* It is a root interface for entire Collection Framework.
* If we want to represent group of individual objects in a single entity then we need to use Collection interface.
* Collection interface contains following methods which are available for entire Collection objects.

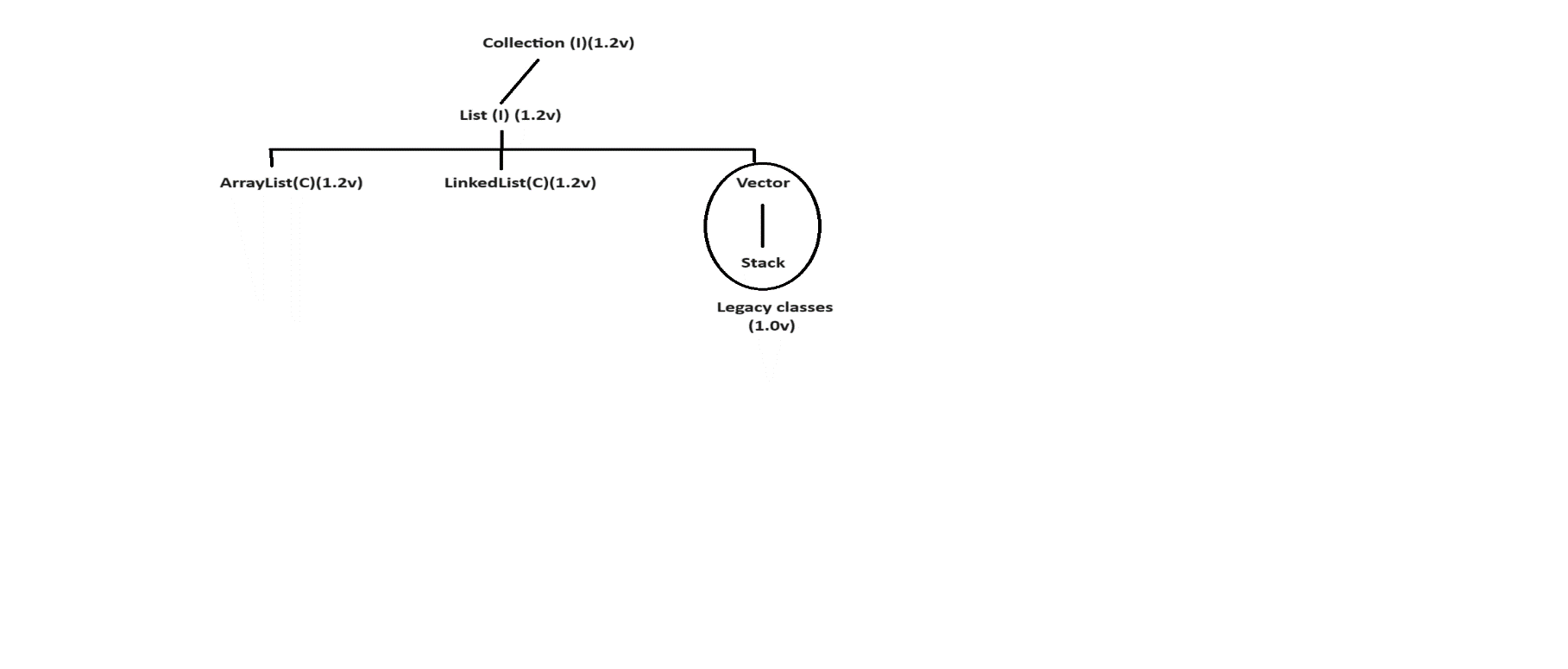
ex:

cmd> javap java.util.Collection

## 1.List

* It is a child interface of Collection interface.
* If we want to represent group of individual objects in a single entity where duplicate objects are allowed and order is preserved.

**Diagram: class40.1**



### ArrayList

* 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 interfaces.
* 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("one");

al.add("two");

al.add("three");

System.out.println(al.isEmpty());//false

al.add(1,"raja");

System.out.println(al);//[one, raja, two, three]

al.remove("raja");

System.out.println(al);//[one, two, three]

al.clear();

System.out.println(al); //[]

}

}

ex:

import java.util.\*;

class Test

{

public static void main(String[] args)

{

ArrayList<String> al=new ArrayList<String>();

al.add("one");

al.add("two");

al.add("three");

for(int i=0;i<al.size();i++)

{

String s=al.get(i);

System.out.println(s);

}

}

}

ex:

import java.util.\*;

class Test

{

public static void main(String[] args)

{

List<String> list=new ArrayList<String>();

list.add("one");

list.add("two");

list.add("three");

System.out.println(list);

}

}

ex:

import java.util.\*;

class Test

{

public static void main(String[] args)

{

List<Integer> list=Arrays.asList(7,3,1,9,5,4);

System.out.println(list);

}

}

import java.util.\*;

class Test

{

public static void main(String[] args)

{

ArrayList<String> java=new ArrayList<String>();

java.add("html");

java.add("css");

java.add("bootstrap");

java.add("reactjs");

java.add("java");

java.add("frameworks");

System.out.println(java); //[html,css,bootstrap,reactjs,java,frameworks]

ArrayList<String> dotnet=new ArrayList<String>();

dotnet.add("html");

dotnet.add("css");

dotnet.add("bootstrap");

dotnet.add("angularjs");

dotnet.add("dotnet");

dotnet.add("frameworks");

System.out.println(dotnet); //[html,css,bootstrap,angularjs,dotnet,frameworks]

java.retainAll(dotnet);

System.out.println(java); //[html,css,bootstrap,frameworks]

}

}

### 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 a adding and removing in the middle then we need to use LinkedList.

LinkedList contains following methods.

ex:

public E getFirst();

public E getLast();

public E removeFirst();

public E removeLast();

public void addFirst(E);

public void addLast(E);

ex:

import java.util.\*;

class Test

{

public static void main(String[] args)

{

LinkedList ll=new LinkedList();

ll.add("one");

ll.add("two");

ll.add("three");

System.out.println(ll);//[one,two,three]

ll.add("one");

System.out.println(ll);//[one,two,three,one]

ll.add(10);

System.out.println(ll); //[one,two,three,one,10]

ll.add(null);

System.out.println(ll);//[one,two,three,one,10,null]

}

}

ex:

import java.util.\*;

class Test

{

public static void main(String[] args)

{

LinkedList<String> ll1=new LinkedList<String>();

ll1.add("one");

ll1.add("two");

ll1.add("three");

System.out.println(ll1);//[one,two,three]

LinkedList<String> ll2=new LinkedList<String>();

ll2.add("raja");

System.out.println(ll2);//[raja]

ll2.addAll(ll1);

System.out.println(ll2);//[raja,one,two,three]

System.out.println(ll2.containsAll(ll1)); //true

ll2.removeAll(ll1);

System.out.println(ll2); //[raja]

}

}

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.
* All methods present in Vector are synchronized.Hence we can achieve thread safety.

Vector class contains following methods.

ex:

addElement()

removeElement()

firstElement()

lastElement()

removeAllElements()

and etc.

ex:

import java.util.\*;

class Test

{

public static void main(String[] args)

{

Vector<Integer> v=new Vector<Integer>();

System.out.println(v.capacity());

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.removeElement(5);

System.out.println(v);//[1,2,3,4,6,7,8,9,10]

v.removeAllElements();

System.out.println(v);//[]

}

}

ex:

import java.util.\*;

class Test

{

public static void main(String[] args)

{

Vector<Integer> v=new Vector<Integer>();

System.out.println(v.capacity());

for(int i=1;i<=10;i++)

{

v.add(i);

}

System.out.println(v);//[1,2,3,4,5,6,7,8,9,10]

System.out.println(v.get(0)); //1

System.out.println(v.get(v.size()-1)); //10

v.remove(4);

System.out.println(v);//[1,2,3,4,6,7,8,9,10]

v.clear();

System.out.println(v);//[]

}

}

### Stack

* It is a child class of Vector class.
* If we depends upon Last In First Out (LIFO) order then we need to use Stack.

**constructor**

Stack s=new Stack();

**methods**

1) push()

It is used to push the element in a Stack.

2) pop()

It is used to remove the element from stack.

3) peek()

It will return toppest element from stack.

4) isEmpty()

It is used to check stack is empty or not.

5) search()

It will return offet 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.add("A");

s.add("B");

s.add("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

}

}

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

import java.util.\*;

class Test

{

public static void main(String[] args)

{

List<String> list=Arrays.asList("apple","ball","cat","dog");

System.out.println(list);//[apple,ball,cat,dog]

Collections.reverse(list);

System.out.println(list); //[dot,cat,ball,apple]

}

}

**Q) Write a java program to display the list in sorting order?**

input:

ball dog elephant apple cat

output:

apple ball cat dog elephant

import java.util.\*;

class Test

{

public static void main(String[] args)

{

List<String> list=Arrays.asList("ball","dog","elephant","apple","cat");

System.out.println(list);//[ball,dog,elephant,apple,cat]

Collections.sort(list);

System.out.println(list); //[apple, ball, cat, dog, elephant]

}

}

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

input:

[{()}]

output:

It is balanced string

ex:

import java.util.\*;

class Test

{

public static void main(String[] args)

{

String str="[{()}]";

if(isBalanced(str.toCharArray()))

System.out.println("It is balanced string");

else

System.out.println("It is not balanced string");

}

public static boolean isBalanced(char[] carr)

{

Stack<Character> s=new Stack<Character>();

for(char ch:carr)

{

if(ch=='[' || ch=='{' || ch=='(')

{

s.push(ch);

}

else if(ch==']' && !s.isEmpty() && s.peek()=='[')

{

s.pop();

}

else if(ch=='}' && !s.isEmpty() && s.peek()=='{')

{

s.pop();

}

else if(ch==')' && !s.isEmpty() && s.peek()=='(')

{

s.pop();

}

else

{

return false;

}

}

return s.isEmpty();

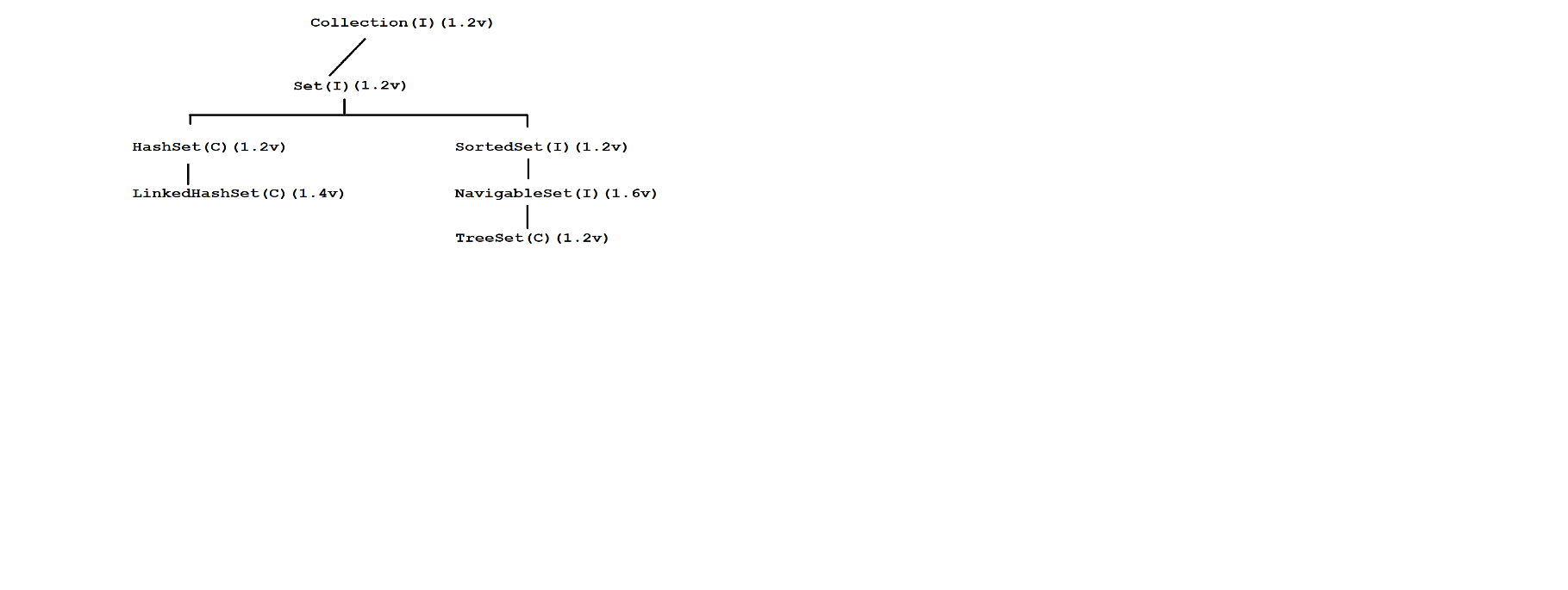
}

}

## 2.Set

* It is a child interface of Collection interface.
* If we want to represent group of individual objects in a single entity where duplicate objects are not allowed and order is not preserved then we need to use Set interface.

**Diagram: class42.1**



### HashSet

* The underlying data structure is Hashtable.
* Duplicate objects are not allowed.
* Insertion order is not preserved because it will take hash code of an object.
* Hetrogeneous objects are allowed.
* Null insertion is possible.

ex:

import java.util.\*;

class Test

{

public static void main(String[] args)

{

HashSet hs=new HashSet();

hs.add("nine");

hs.add("four");

hs.add("one");

hs.add("six");

System.out.println(hs);//[nine, six, four, one]

hs.add("nine");

System.out.println(hs);//[nine, six, four, one]

hs.add(10);

System.out.println(hs);//[nine, six, four, one, 10]

hs.add(null);

System.out.println(hs);//[null, nine, six, four, one, 10]

}

}

### LinkedHashSet

It is a child class of HashSet class.

LinkedHashSet is exactly same as HashSet class with following differences.

**HashSet**  **LinkedHashSet**

The underlying data structure is Hashtable. The underlying data structure is Hashtable and LinkedList.

Insertion order is not preserved. Insertion order is preserved.

It is introduced in 1.2v. It is introduced in 1.4v.

ex:

import java.util.\*;

class Test

{

public static void main(String[] args)

{

LinkedHashSet lhs=new LinkedHashSet();

lhs.add("nine");

lhs.add("four");

lhs.add("one");

lhs.add("six");

System.out.println(lhs);//[nine, four, one, six]

lhs.add("nine");

System.out.println(lhs);//[nine, four, one, six]

lhs.add(10);

System.out.println(lhs);//[nine, four, one, six, 10]

lhs.add(null);

System.out.println(lhs);//[nine, four, one, six, 10, null]

}

}

### TreeSet

* The underlying data structure is Balanced Tree.
* Duplicate objects are not allowed.
* Insertion order is not preserved because it will take sorting order of an object.
* Hetrogeneous objects are not allowed.
* If we try to insert hetrogeneous objects then we will get ClassCastException
* For empty TreeSet, if are trying to insert null then we will get NullPointerException.
* For non-empty TreeSet, if we are trying to insert null then we will get NullPointerException.

ex:

import java.util.\*;

class Test

{

public static void main(String[] args)

{

TreeSet ts=new TreeSet();

ts.add(10);

ts.add(1);

ts.add(5);

ts.add(3);

System.out.println(ts); //[1, 3, 5, 10]

ts.add(10);

System.out.println(ts); //[1,3,5,10]

//ts.add("hi");

//System.out.println(ts);//R.E ClassCastException'

//ts.add(null);

//System.out.println(ts);// R.E NullPointerException

}

}

**Q) What is the difference between Comparable and Comparator interface?**

**Comparable**

* Comparable interface present in java.lang package.
* It contains following 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.

If depends upon default natural sorting order then we need to use Comparable interface.

ex:

class Test

{

public static void main(String[] args)

{

System.out.println("A".compareTo("Z")); // -25

System.out.println("Z".compareTo("A")); // 25

System.out.println("K".compareTo("K")); // 0

}

}

**Comparator**

* Comparator interface present in java.util package.
* It contains following two methods i.e compare() method 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.

If we depend upon customized sorting order then we need to use Comparator interface.

Implementation of equals() method is optional because it is avaiable by default through inheritance.

ex:

import java.util.\*;

class Test

{

public static void main(String[] args)

{

TreeSet<Integer> ts=new TreeSet<Integer>(new MyComparator());

ts.add(10);

ts.add(1);

ts.add(5);

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(1);

ts.add(5);

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;

}

}

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

input:

1 2 2 3 3 3 4 4 4 4

output:

1 2 3 4

ex:

import java.util.\*;

class Test

{

public static void main(String[] args)

{

int[] arr={1,2,2,3,3,3,4,4,4,4};

Set<Integer> set=new LinkedHashSet<Integer>();

for(int i:arr)

{

set.add(i);

}

set.forEach(element-> System.out.print(element+" "));

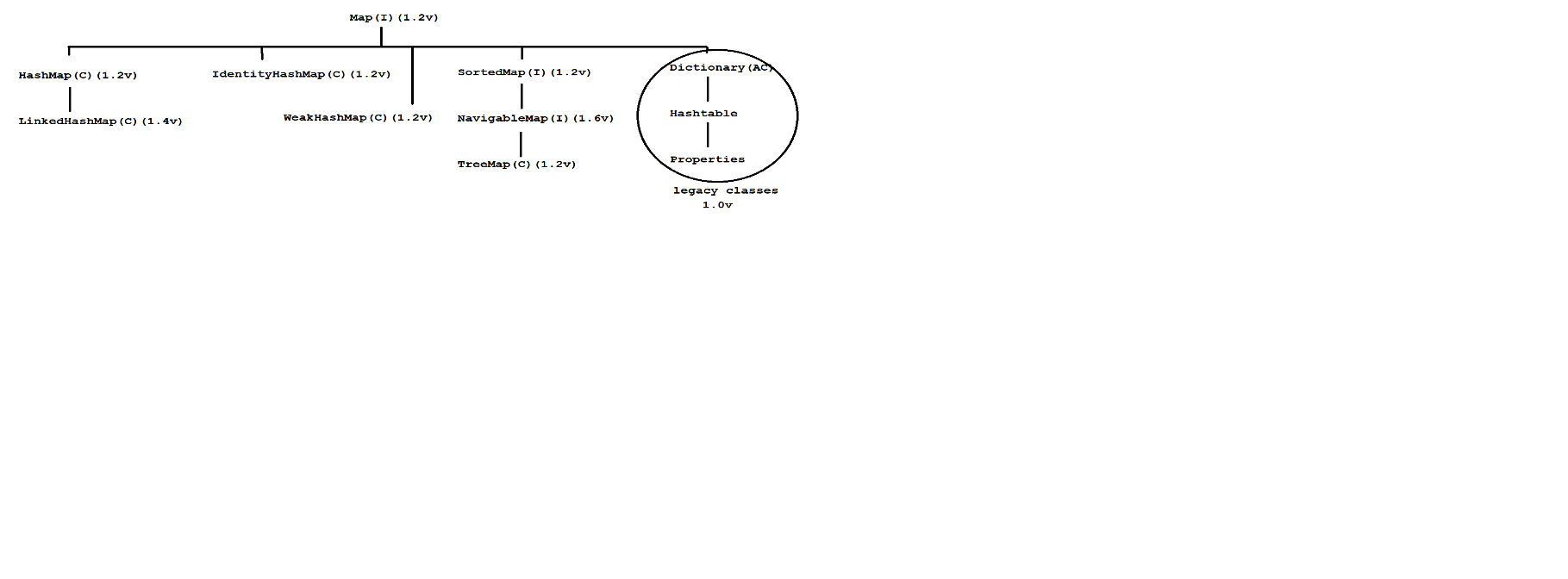
}

}

## 3.Map

* It is not a child interface of Collection interface.
* If we want to represent individual objects in a key and value pair then we need to use Map interface.
* Key can't be duplicate but value can be duplicate.
* Key and value both must be objects.
* Each key and value pair is called one-entry.

**Diagram: class43.1**



### HashMap

* The underlying data structure is Hashtable.
* Duplicate keys are not allowed but values can be duplicate.
* Insertion order is not preserved because it will take hashcode of the key.
* Hetrogeneous objects are allowed for both key and value.
* Null insertion is possible for both key and value.

ex:

import java.util.\*;

class Test

{

public static void main(String[] args)

{

HashMap hm=new HashMap();

hm.put("nine","alan");

hm.put("one","jose");

hm.put("four","lisa");

hm.put("six","nancy");

System.out.println(hm); // {nine=alan, six=nancy, four=lisa, one=jose}

hm.put("one","gogo");

System.out.println(hm); // {nine=alan, six=nancy, four=lisa, one=gogo}

hm.put(1,10);

System.out.println(hm); // {nine=alan, 1=10, six=nancy, four=lisa, one=gogo}

hm.put(null,null);

System.out.println(hm); // {null=null, nine=alan, 1=10, six=nancy, four=lisa, one=gogo}

}

}

ex:

import java.util.\*;

class Test

{

public static void main(String[] args)

{

HashMap<String,String> hm=new HashMap<String,String>();

hm.put("nine","alan");

hm.put("one","jose");

hm.put("four","lisa");

hm.put("six","nancy");

Set s=hm.keySet();

System.out.println(s);//[nine, six, four, one]

Collection c=hm.values();

System.out.println(c);//[alan, nancy, lisa, jose]

Set s1=hm.entrySet();

System.out.println(s1);//[nine=alan, six=nancy, four=lisa, one=jose]

}

}

### LinkedHashMap

It is a child class of HashMap class.

If is 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 1.5v.

ex:

import java.util.\*;

class Test

{

public static void main(String[] args)

{

LinkedHashMap lhm=new LinkedHashMap();

lhm.put("nine","alan");

lhm.put("one","jose");

lhm.put("four","lisa");

lhm.put("six","nancy");

System.out.println(lhm);//{nine=alan, one=jose, four=lisa, six=nancy}

lhm.put("one","gogo");

System.out.println(lhm);//{nine=alan, one=gogo, four=lisa, six=nancy}

lhm.put(1,10);

System.out.println(lhm);//{nine=alan, one=gogo, four=lisa, six=nancy, 1=10}

lhm.put(null,null);

System.out.println(lhm);//{nine=alan, one=gogo, four=lisa, six=nancy, 1=10, null=null}

}

}

### TreeMap

* The underlying data structur is RED BLACK TREE.
* Duplicate keys are not allowed but values can be duplicate.
* Insertion order is not preserved because it will take sorting order of key.
* If we depends upon default natural sorting order then key can be homogeneous and Comparable.
* If we depends upon customized sorting order then key 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(7,"seven");

tm.put(5,"five");

System.out.println(tm);//{1=one, 5=five, 7=seven, 10=ten}

tm.put(1,"hundred");

System.out.println(tm);//{1=hundred, 5=five, 7=seven, 10=ten}

tm.put(4,null);

System.out.println(tm);//{1=hundred, 4=null, 5=five, 7=seven, 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 duplicate.
* Insertion order is not preserved because it will display descending order of key.
* Hetrogenous objects are allowed for both key and value.
* Null insertion is not possible for both key and value.

ex:

import java.util.\*;

class Test

{

public static void main(String[] args)

{

Hashtable<Integer,String> ht=new Hashtable<Integer,String>();

ht.put(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,"gogo");

System.out.println(ht); // {10=ten, 5=five, 3=three, 1=gogo}

//ht.put(4,null);

//System.out.println(ht); // R.E NullPointerException

//ht.put(null,"four");

//System.out.println(ht); //R.E NullPointerException

}

}

**Interview Questions**

**Q) Write a java program to compare two dates ?**

ex:

import java.time.\*;

class Test

{

public static void main(String[] args)

{

LocalDate date1=LocalDate.now();

LocalDate date2=LocalDate.now();

if((date1.compareTo(date2))<0)

System.out.println("Date1 comes before Date2");

else if((date1.compareTo(date2))>0)

System.out.println("Date1 comes after Date2");

else

System.out.println("Both are same");

}

}

ex:

import java.time.\*;

class Test

{

public static void main(String[] args)

{

LocalDate date1=LocalDate.now(); // 10-05-2024

LocalDate date2=LocalDate.of(2024,8,15);

if((date1.compareTo(date2))<0)

System.out.println("Date1 comes before Date2");

else if((date1.compareTo(date2))>0)

System.out.println("Date1 comes after Date2");

else

System.out.println("Both are same");

}

}

**Q) Write a java program to display number of words present in a string?**

input:

This is is java java class

output:

This=1 , is=2, java=2 , class=1

import java.util.\*;

class Test

{

public static void main(String[] args)

{

String str="This is is java java class";

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

Map<String,Integer> map=new LinkedHashMap<String,Integer>();

for(String s:sarr)

{

if(map.get(s)!=null)

{

map.put(s,map.get(s)+1);

}

else

{

map.put(s,1);

}

}

System.out.println(map);

}

}

**Q) Write a java program to display number of characters present in a string?**

input:

java

output:

j=1,a=2,v=1

ex:

import java.util.\*;

class Test

{

public static void main(String[] args)

{

String str="java";

char[] carr=str.toCharArray();

Map<Character,Integer> map=new LinkedHashMap<Character,Integer>();

for(char c:carr)

{

if(map.get(c)!=null)

{

map.put(c,map.get(c)+1);

}

else

{

map.put(c,1);

}

}

System.out.println(map);

}

}

## Types of Cursors in Java

Cursor is used to read objects one by one from Collections.

We have three types of cursors.

1) Enumeration

2) Iterator

3) ListIterator

### 1) Enumeration

* It is used to read objects one by one from legacy Collection objects.
* We can create Enumeration object as follow.

ex:

Enumeration e=v.elements();

* Enumeration interface contains following two methods.

ex:

public boolean hasMoreElements()

public Object nextElement()

ex:

import java.util.\*;

class Test

{

public static void main(String[] args)

{

Vector v=new Vector();

for(int i=1;i<=10;i++)

{

v.add(i);

}

Enumeration e=v.elements();

while(e.hasMoreElements())

{

Integer i=(Integer)e.nextElement();

System.out.println(i);

}

}

}

**Limitations with Enumeration**

* Using Enumeration interface we can read objects one by one from legacy Collection objects.Hence it is not a universal cursor.
* It will perform read operation but not remove operation.
* To overcome this limitations Sun Micro System introduced Iterator.

### 2) Iterator

* Iterator is used to read read objects one by one from any Collection objects.Hence it is a unviversal cursor.
* Using Iterator interface we can perform read and remove operation.
* 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);

}

Iterator itr=al.iterator();

while(itr.hasNext())

{

Integer i=(Integer)itr.next();

if(i%2==0)

System.out.println(i);

else

itr.remove();

}

System.out.println(al); //[2,4,6,8,10s]

}

}

**Limitations with Iterator**

* Using Enumeration and Iterator we can read objects one by one 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 is used to read objects one by one from List Collection objects.
* ListIterator interface can perform read, remove, adding and replacement of new objects.
* We can create ListIterator object as follow.

ex:

ListIterator litr=al.listIterator();

ListIterator interface contains following 9 methods.

ex:

public boolean hasNext()

public Object next();

public boolean hasPrevious()

public Object previous();

public void remove();

public int previousIndex()

public int nextIndex()

public void set()

public void add()

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("bala"))

{

litr.set("allu");

}

}

System.out.println(al);//[allu,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("bala"))

{

litr.add("ram");

}

}

System.out.println(al);//[bala,ram,nag,chiru,venki]

}

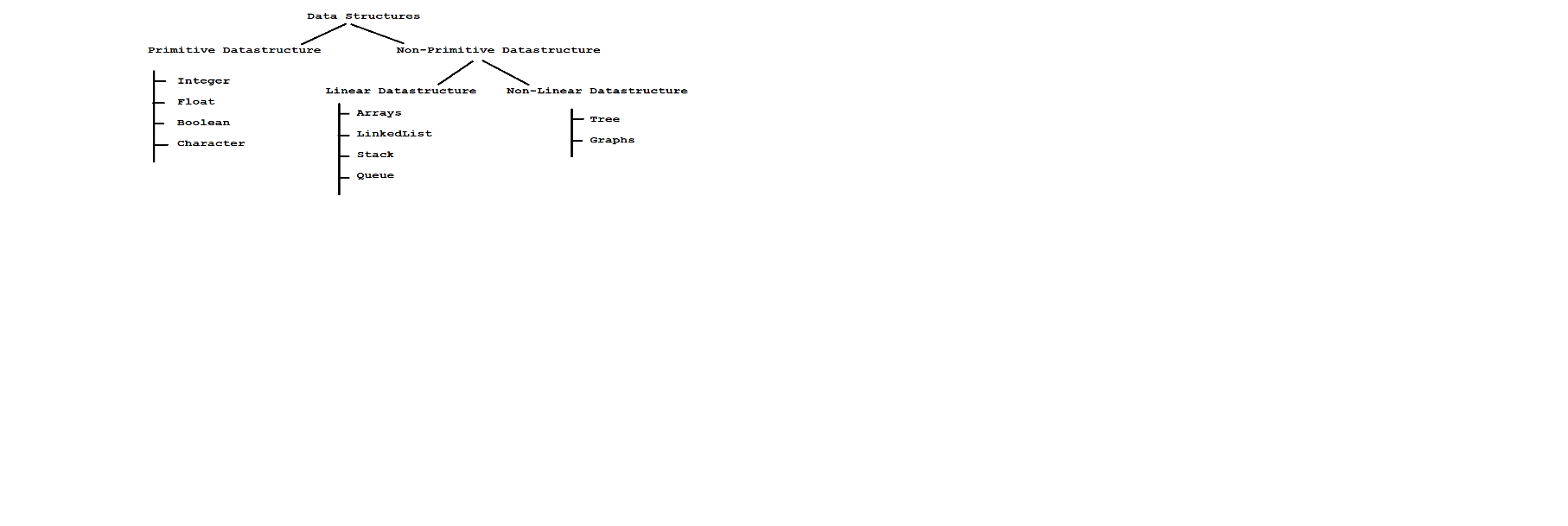
}

**Interview Questions**

**Q) Types of Data structures in Java?**

We have two types of data structures in java.

**Diagram: class44.1**



**Q) How to display the objects from List?**

import java.util.\*;

class Test

{

public static void main(String[] args)

{

List<Integer> list=Arrays.asList(3,7,1,2,9,5);

list.forEach(element -> System.out.println(element+" "));

}

}

**Q) How to display the objects from Map?**

import java.util.\*;

class Test

{

public static void main(String[] args)

{

Map<Integer,String> map=new LinkedHashMap<Integer,String>();

map.put(1,"java");

map.put(2,"dotnet");

map.put(3,"python");

map.forEach((key,value)-> System.out.println(key+" "+value));

}

}

**Q) Write a java program to store Employee object in Collection?**

import java.util.\*;

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)

{

ArrayList<Employee> al=new ArrayList<Employee>();

al.add(new Employee(101,"Alan",1000d));

al.add(new Employee(102,"Kelvin",2000d));

al.add(new Employee(103,"Jose",3000d));

al.forEach(element -> System.out.println(element.getEmpId()+" "+element.getEmpName()+" "+element.getEmpSal()));

}

}

**Q) Give me any example on Abstraction?**

abstract class Vehicle

{

public abstract void run();

}

class Bike extends Vehicle

{

public void run()

{

System.out.println("Bike runs with 180 speed");

}

}

class Test

{

public static void main(String[] args)

{

Bike b=new Bike();

b.run();

}

}

ex:

import java.util.\*;

class Test

{

public static void main(String[] args)

{

Scanner sc=new Scanner(System.in);

System.out.println("Enter the No :");

int no=sc.nextInt();

// consumes the dangling newline character

sc.nextLine();

System.out.println("Enter the Name :");

String name=sc.nextLine();

System.out.println("Enter the Salary :");

double sal=sc.nextDouble();

System.out.println(no+" "+name+" "+sal);

}

}

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

Input:

1 1 2 2 2 3 3 3

output:

3

ex:

import java.util.\*;

class Test

{

public static void main(String[] args)

{

int[] arr={1,1,2,2,2,3,3,3};

//caller method

System.out.println(luckyNumber(arr));

}

public static int luckyNumber(int[] arr)

{

Map<Integer,Integer> map=new LinkedHashMap<Integer,Integer>();

for(int i=0;i<arr.length;i++)

{

if(map.containsKey(arr[i]))

{

map.put(arr[i],map.get(arr[i])+1);

}

else

{

map.put(arr[i],1);

}

}

int x=0;

int max=-1;

//{1=2,2=3,3=3}

for(Map.Entry<Integer,Integer> entry: map.entrySet())

{

if(entry.getKey() == entry.getValue())

{

x=entry.getValue();

max=Math.max(x,max);

}

}

return max;

}

}

## MultiTasking

Executing several task simultenously such concept is called multitasking.

We have two types of multitasking.

1) Process Based multitasking

2) Thread Based multitasking

### 1) Process Based multitasking

Executing several task simultenously where each task is a independent process.

It is best suitable for OS Level.

### 2) Thread Based multitasking

Executing several task simultenously where each task is a same part of a program.

It is best suitable for programmatic level.

# 33.Multithreading

**Q) What is the difference between Thread and Process?**

**Thread**

* A thread is a leight weight sub process.
* We can run multiple threads concurently.
* One thread can communicate with another thread.

ex:

a class is one thread

a block is one thread

a constructor is one thread

**Process**

* A process is a collection of threads.
* We can run multiple process concurently.
* One process can't communicate with another process.

ex:

taking a class using zoom meeting is one process

typing the notes in editor is one process

downloading a file from internet is one process

## MutiThreading

* Executing several threads simultenously such concept is called multithreading.
* In multithreading, 10% of work should be done by a programmer and 90% of work will be done by JAVA API.

The main important application area of multithreading are.

1) To implements multimedia graphics.

2) To develop video games.

3) To develop animations.

## Ways to create a thread in java

There are two ways to create a thread in java.

1) By extending a Thread class

2) By implementing a Runnable interface

### 1) By extending a Thread class

class MyThread extends Thread

{

public void run()

{

for(int i=1;i<=5;i++)

{

System.out.println("Child-Thread");

}

}

}

class Test

{

public static void main(String[] args)

{

//instantiate a thread

MyThread t=new MyThread();

//start a thread

t.start();

for(int i=1;i<=5;i++)

{

System.out.println("Parent-Thread");

}

}

}

**case1: Thread Schedular**

* If multiple threads are waiting for execution which thread has to be execute will decided by thread schedular.
* What algorithm,behaviour and mechanism used by thread schedular is depends upon JVM vendor.
* Hence we can't expect any execution order or exact output in multithreading.

**case2: Difference between t.start() and t.run() method**

* If we invoke t.start() method then new thread will be created which is responsible to execute run() method automatically.

ex:

class MyThread extends Thread

{

public void run()

{

for(int i=1;i<=5;i++)

{

System.out.println("Child-Thread");

}

}

}

class Test

{

public static void main(String[] args)

{

//instantiate a thread

MyThread t=new MyThread();

//new 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

{

public void run()

{

for(int i=1;i<=5;i++)

{

System.out.println("Child-Thread");

}

}

}

class Test

{

public static void main(String[] args)

{

//instantiate a thread

MyThread t=new MyThread();

//no new thread

t.run();

for(int i=1;i<=5;i++)

{

System.out.println("Parent-Thread");

}

}

}

**case3: If we won't override run() method**

* If we won't override run() method then t.start() method will execute Thread class run() method automatically.
* But Thread class run() method is empty implementation.Hence we won't get any output from child thread.

ex:

class MyThread extends Thread

{

}

class Test

{

public static void main(String[] args)

{

//instantiate a thread

MyThread t=new MyThread();

//new thread

t.start();

for(int i=1;i<=5;i++)

{

System.out.println("Parent-Thread");

}

}

}

**case4: If we overload run() method**

* If we overload run() method then Thread class start() method always execute run() method with no parameters only.

ex:

class MyThread extends Thread

{

public void run()

{

System.out.println("0-arg method");

}

public void run(int i)

{

System.out.println("int-arg method");

}

}

class Test

{

public static void main(String[] args)

{

//instantiate a thread

MyThread t=new MyThread();

//new thread

t.start();

for(int i=1;i<=5;i++)

{

System.out.println("Parent-Thread");

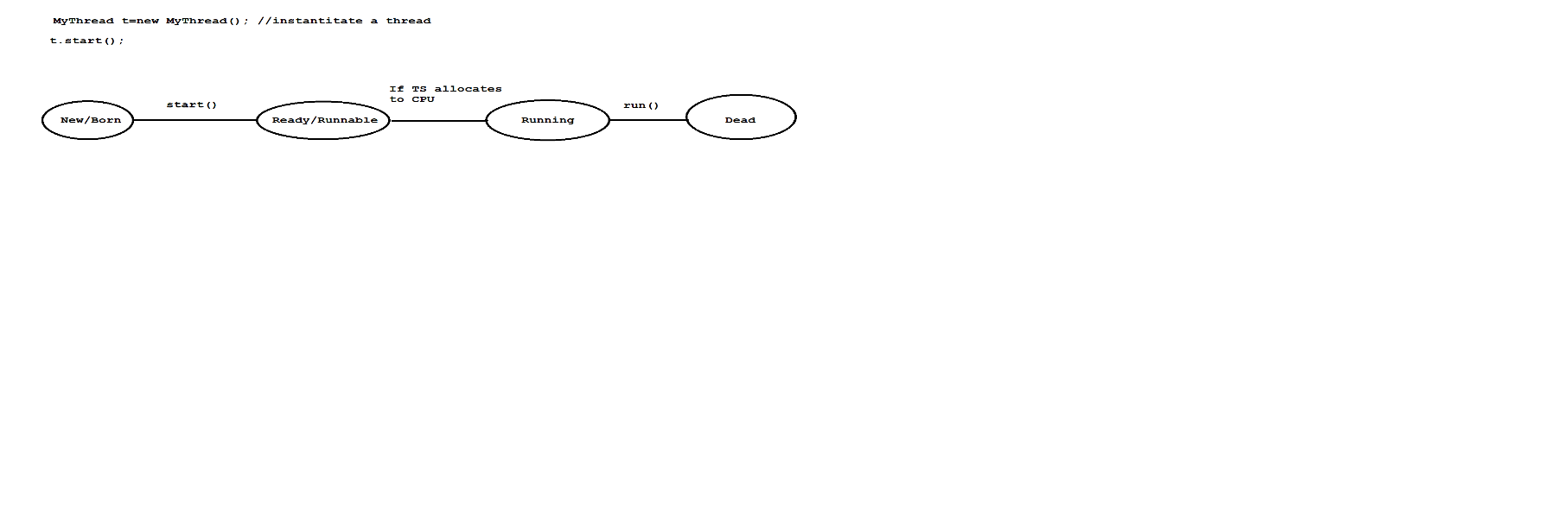
}

}

}

**case 5: Life cycle of a thread**

**Diagram: class45.1**



* Once if we create a thread object then our thread will be in new or born state.
* Once if we call t.start() method then our threads will be in ready or runnable state.
* If thread schedular allocates to CPU then our thread goes to running state.
* Once the run() method execution is completed then our thread enters to dead state.

### 2) By implementing a 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

//start a thread

t.start();

for(int i=1;i<=5;i++)

{

System.out.println("Parent-Thread");

}

}

}

**Setting and Getting Name of a thread**

In java, every thread has a name explicitly provided by the programmer or automatically generated by JVM.

We have following methods to set and get name of a thread.

ex:

public final void setName(String name);

public final String getName();

ex:

class MyThread extends Thread

{

}

class Test

{

public static void main(String[] args)

{

System.out.println(Thread.currentThread().getName());//main

MyThread t=new MyThread();

System.out.println(t.getName());//Thread-0

Thread.currentThread().setName("Parent-Thread");

System.out.println(Thread.currentThread().getName());//Parent-Thread

t.setName("Child-Thread");

System.out.println(t.getName());//Child-Thread

}

}

## Thread Priority

* In java, every thread has a priority explicitly provided by the programmer or automatically generated by JVM.
* The valid range of thread priority is 1 to 10. Where 1 is a least priority and 10 is a highest priority.
* If we take more then 10 priority then we will get IllegalArgumentException.
* A Thread defines following standard constants as thread priorties.

ex:

Thread.MAX\_PRIORITY - 10

Thread.NORM\_PRIORITY - 5

Thread.MIN\_PRIORITY - 1

* We don't have such constants like LOW\_PRIORITY and HIGH\_PRIORITY.
* Thread Schedular uses thread priority while allocating to CPU.
* A thread which is having highest priority will be executed first.
* If multiple threads having same priority then we can't expect any execution order.
* We have following methods to set and get thread priority.

ex:

public final void setPriority(int priority)

public final int getPriority()

ex:

class MyThread extends Thread

{

}

class Test

{

public static void main(String[] args)

{

System.out.println(Thread.currentThread().getPriority());//5

MyThread t=new MyThread();

System.out.println(t.getPriority());//5

Thread.currentThread().setPriority(9);

System.out.println(Thread.currentThread().getPriority());//9

t.setPriority(4);

System.out.println(t.getPriority());//4

t.setPriority(11);//R.E IllegalArgumentException

}

}

## Daemon Thread

* It is a service provider thread which provides services to user threads.
* Life of daemon thread is depends upon user threads. When user threads died then deamon thread will terminate automatically.
* There are many daemon threads are running internally.

ex:

Garbage Collector

Finalizer

and etc.

* To start a daemon thread we need to use setDaemon(true) method.
* To check thread is a daemon or not we will 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");

}

}

}

## Ways to prevent a thread from execution

There are three ways to prevent(stop) a thread from execution.

1) yield()

2) join()

3) sleep()

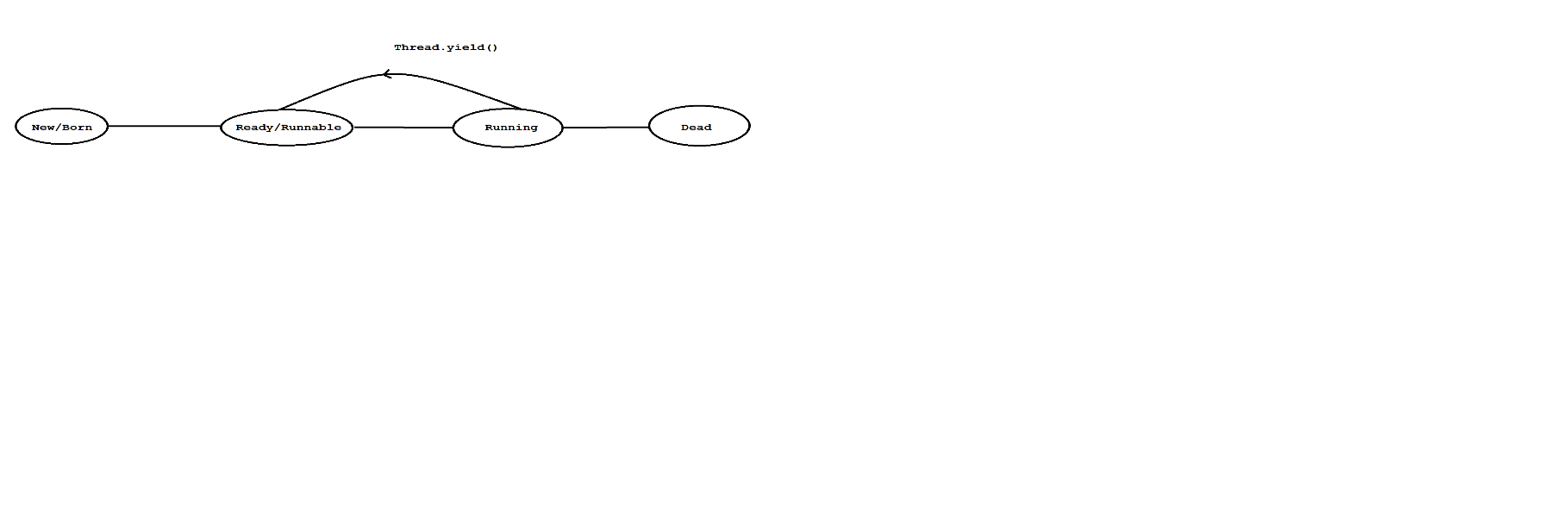
### 1) yield()

* It will pause the current execution thread and gives the chance to other threads having same priority.
* If multiple threads having same priority then we can't expect any execution order.
* If there is no waiting threads or low priority threads then same thread will continue it's execution.
* The thread 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: class46.1**



ex:

class MyThread extends Thread

{

public void run()

{

for(int i=1;i<=5;i++)

{

Thread.currentThread().yield();

System.out.println("Child-Thread");

}

}

}

class Test

{

public static void main(String[] args)

{

MyThread t=new MyThread();

t.start();

for(int i=1;i<=5;i++)

{

System.out.println("Parent-Thread");

}

}

}

### 2) join()

* If thread wants to wait untill the completion of some other thread then we need to use join() method.
* A join() method throws one checked exception so we must and should handle that exception by using try and catch block or by using throws statement.

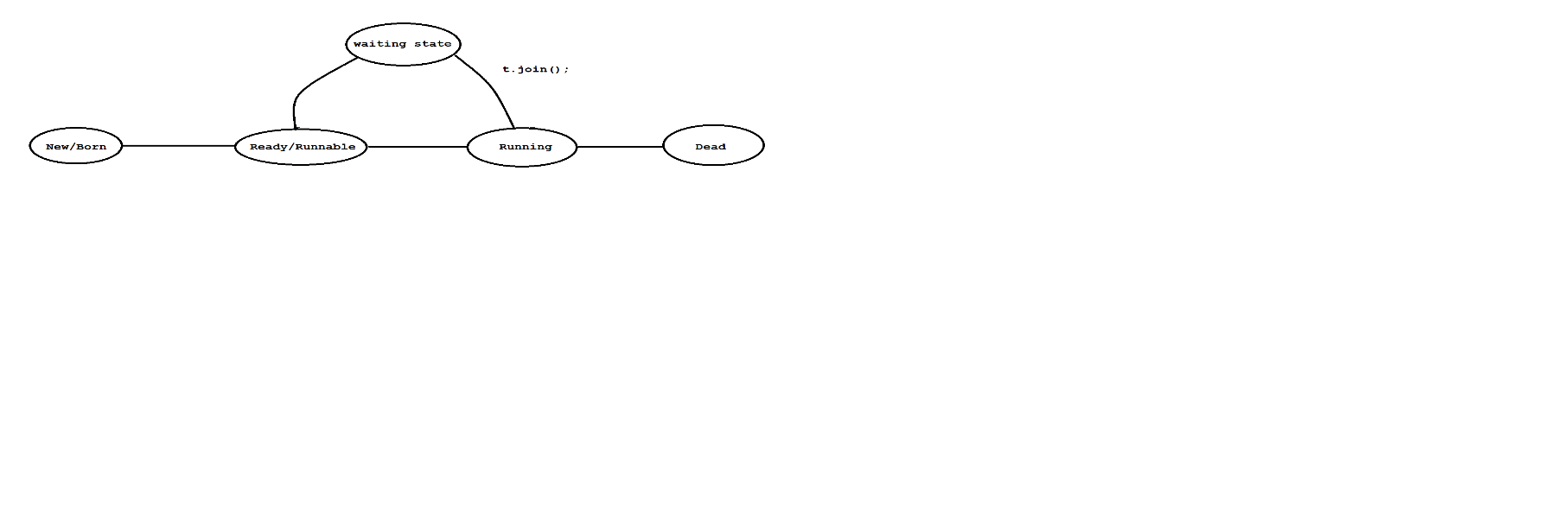
ex:

public final void join()throws InterruptedException

public final void join(long ms)throws InterruptedException

public final void join(long ms,int ns)throws InterruptedException

**Diagram: class46.2**



ex:

class MyThread extends Thread

{

public void run()

{

for(int i=1;i<=5;i++)

{

System.out.println("Child-Thread");

}

}

}

class Test

{

public static void main(String[] args)throws InterruptedException

{

MyThread t=new MyThread();

t.start();

t.join();

for(int i=1;i<=5;i++)

{

System.out.println("Parent-Thread");

}

}

}

### 3) sleep()

* If a thread don't want to perform any operation on perticular amount of time then we need to use sleep() method.
* A sleep() method throws one checked exception so we must and should handle that exception by using try and catch block or by using throws statement.

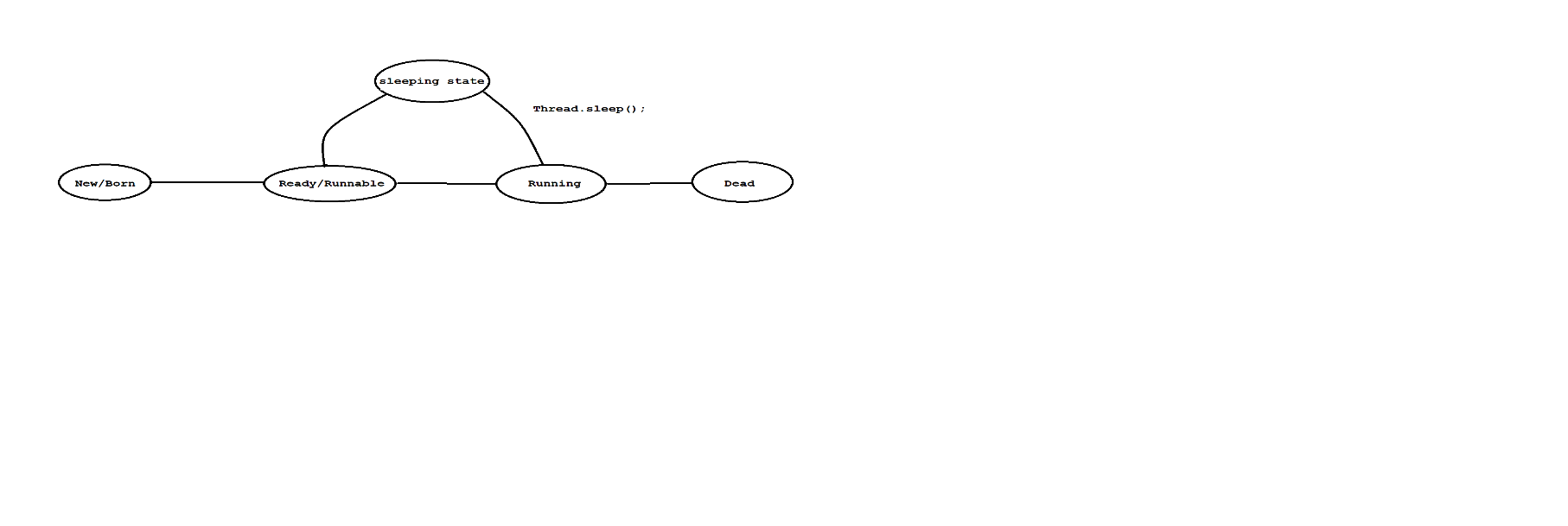
ex:

public static native void sleep()throws InterruptedException

public static native void sleep(long ms)throws InterruptedException

public static native void sleep(long ms,int ns)throws InterruptedException

**Diagram: class46.3**



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");

}

}

}

## Problem without synchronization

If there is no synchronization then we will face following problems.

1) Data inconsistency

2) Thread interference

ex:

class Table

{

void printTable(int n)

{

for(int i=1;i<=5;i++)

{

System.out.println(n\*i);

try

{

Thread.sleep(2000);

}

catch (InterruptedException ie)

{

ie.printStackTrace();

}

}

}

}

class MyThread1 extends Thread

{

Table t;

MyThread1(Table t)

{

this.t=t;

}

public void run()

{

t.printTable(5);

}

}

class MyThread2 extends Thread

{

Table t;

MyThread2(Table t)

{

this.t=t;

}

public void run()

{

t.printTable(10);

}

}

class Test

{

public static void main(String[] args)

{

Table obj=new Table();

MyThread1 t1=new MyThread1(obj);

MyThread2 t2=new MyThread2(obj);

t1.start();

t2.start();

}

}

## synchronization

* A synchronized keyword is applicable for methods and blocks.
* A synchronization is allowed one thread to execute given object.Hence we achieve thread safety.
* The main advantage of synchronization is we solve data inconsistence problem.
* The main disadvantage of synchronization is ,it will increase waiting time of a thread which reduce the performance of the system.
* If there is no specific requirement then it is never recommanded to use synchronization concept.
* synchronization internally uses lock mechanism.
* Whenever a thread wants to access object , first it has to acquire lock of an object and thread will release the lock when it completes it's task.
* When a thread wants to execute synchronized method.It automatically gets the lock of an object.
* When one thread is executing synchronized method then other threads are not allowed to execute other synchronized methods in a same object concurently.But other threads are allowed to execute non-synchronized method concurently.

ex:

class Table

{

synchronized void printTable(int n)

{

for(int i=1;i<=5;i++)

{

System.out.println(n\*i);

try

{

Thread.sleep(2000);

}

catch (InterruptedException ie)

{

ie.printStackTrace();

}

}

}

}

class MyThread1 extends Thread

{

Table t;

MyThread1(Table t)

{

this.t=t;

}

public void run()

{

t.printTable(5);

}

}

class MyThread2 extends Thread

{

Table t;

MyThread2(Table t)

{

this.t=t;

}

public void run()

{

t.printTable(10);

}

}

class Test

{

public static void main(String[] args)

{

Table obj=new Table();

MyThread1 t1=new MyThread1(obj);

MyThread2 t2=new MyThread2(obj);

t1.start();

t2.start();

}

}

### synchronized block

If we want to perform synchronization on specific resource of a program then we need to use synchronization.

ex:

If we have 100 lines of code and if we want to perform synchronization only for

10 lines then we need to use synchronized block.

If we keep all the logic in synchronized block then it will act as a synchronized method.

ex:

class Table

{

void printTable(int n)

{

synchronized(this)

{

for(int i=1;i<=5;i++)

{

System.out.println(n\*i);

try

{

Thread.sleep(2000);

}

catch (InterruptedException ie)

{

ie.printStackTrace();

}

}

}//sync

}

}

class MyThread1 extends Thread

{

Table t;

MyThread1(Table t)

{

this.t=t;

}

public void run()

{

t.printTable(5);

}

}

class MyThread2 extends Thread

{

Table t;

MyThread2(Table t)

{

this.t=t;

}

public void run()

{

t.printTable(10);

}

}

class Test

{

public static void main(String[] args)

{

Table obj=new Table();

MyThread1 t1=new MyThread1(obj);

MyThread2 t2=new MyThread2(obj);

t1.start();

t2.start();

}

}

## 3)Static synchronization

In static synchronization the lock will be on class but not on object.

If we declare any static method as synchronized then it is called static synchronization method.

ex:

class Table

{

static synchronized void printTable(int n)

{

for(int i=1;i<=5;i++)

{

System.out.println(n\*i);

try

{

Thread.sleep(2000);

}

catch (InterruptedException ie)

{

ie.printStackTrace();

}

}

}

}

class MyThread1 extends Thread

{

public void run()

{

Table.printTable(5);

}

}

class MyThread2 extends Thread

{

public void run()

{

Table.printTable(10);

}

}

class Test

{

public static void main(String[] args)

{

MyThread1 t1=new MyThread1();

MyThread2 t2=new MyThread2();

t1.start();

t2.start();

}

}

## Inter-Thread Communication

* Two threads can communicate with one another by using wait(),notify() and notifyAll() method.
* The Thread which is expecting updations it has to wait() method and the thread which is performing updations it has to call notify() method.
* wait(),notify() and notifyAll() method present in Object class but not in Thread class.
* To call wait(),notify() and notifyAll() method our current thread must be in a synchronized area otherwise we will get IllegalMonitorStateException.
* Once a thread calls wait() method on a given object ,1st it will release the lock of that object immediately and entered into waiting state.
* Once a thread calls notify() and notifyAll() method on a given object.It will release the lock of that object but not immediately.
* Except wait(),notify() and notifyAll() method ,there is no such concept where lock release can happen.

ex:

class MyThread extends Thread

{

int total=0;

public void run()

{

synchronized(this)

{

System.out.println("Child Thread started calculation");

for(int i=1;i<=10;i++)

{

total=total+i;

}

System.out.println("Child thread giving notification");

this.notify();

}

}

}

class Test

{

public static void main(String[] args)throws InterruptedException

{

MyThread t=new MyThread();

t.start();

synchronized(t)

{

System.out.println("Main Thread waiting for updating");

t.wait();

System.out.println("Main -Thread got notification ");

System.out.println(t.total);

}

}

}

**DeadLock in java**

* DeadLock will occur in a suitation when one thread is waiting to access

object lock which is acquired by another thread and that thread is waiting

to access object lock which is acquired by first thread.

* Here both the threads are waiting release the thread but no body will

release such situation is called DeadLock.

ex:

class Test

{

public static void main(String[] args)

{

final String res1="hi";

final String res2="bye";

Thread t1=new Thread()

{

public void run()

{

synchronized(res1)

{

System.out.println("Thread1: Locking Resource 1");

synchronized(res2)

{

System.out.println("Thread1: Locking Resource2");

}

}

}

};

Thread t2=new Thread()

{

public void run()

{

synchronized(res2)

{

System.out.println("Thread2: Locking Resource 2");

synchronized(res1)

{

System.out.println("Thread1: Locking Resource 1");

}

}

}

};

t1.start();

t2.start();

}

}

## Drawbacks of multithreading

1)DeadLock

2)Thread Starvation

# 34.Java 8 Features

We have following important features in Java 8.

1) java.time package

2) Functional interface

3) Lamda Expression

4) Default methods in interface

5) Static methods in interface

6) Stream API

7) forEach() method

8) Method Reference(::)

and etc

## Functional interface

An interface that contains only one abstract method is called functional interface.

ex:

Runnable ---> run()

Comparable ---> compareTo()

ActionListener ---> actionPerformed()

and etc.

It can have any number of default methods and static methods.

It is also known as SAM interface or Single Abstract Method interface

It 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 Shape

{

public abstract void circle();

}

class ShapeImpl implements Shape

{

public void circle()

{

System.out.println("circle-method");

}

}

class Test

{

public static void main(String[] args)

{

Shape s=new ShapeImpl();

s.circle();

}

}

ex:

@FunctionalInterface

interface Shape

{

public abstract void circle();

}

class Test

{

public static void main(String[] args)

{

Shape s=new Shape()

{

public void circle()

{

System.out.println("From circle method");

}

};

s.circle();

}

}

## Lamda Expression

* Lamda expression introduced in Java 8.
* We can use lamda expression when we have functional interface.
* Lamda expression is used to concise the code.
* Lamda expression consider as method not a class.
* The main objective of lamda expression is used to achieve functional programming.
* Lamda expression does not allow name, returntype and modifier.

ex:

**Java method**

public void m1()

{

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

}

**Lamda expression**

()->

{

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

};

**ex:**

@FunctionalInterface

interface Shape

{

public abstract void circle();

}

class Test

{

public static void main(String[] args)

{

Shape s=()->

{

System.out.println("Circle Method");

};

s.circle();

}

}

**ex:**

@FunctionalInterface

interface Shape

{

public abstract void circle(int r);

}

class Test

{

public static void main(String[] args)

{

Shape s=(int r)->

{

System.out.println("Circle Method "+r);

};

s.circle(50);

}

}

**ex:**

@FunctionalInterface

interface Shape

{

public abstract int circle(int r);

}

class Test

{

public static void main(String[] args)

{

Shape s=(int r)->

{

return r;

};

System.out.println(s.circle(50));

}

}

## Default methods in interface

* If any method is declared by using "default" keyword then it is called default method.
* It is a non-abstract method.
* It can be override.
* It is introduced in Java 8.

ex:

interface Shape

{

public abstract void circle();

default void square()

{

System.out.println("square method");

}

}

class ShapeImpl implements Shape

{

public void circle()

{

System.out.println("circle-method");

}

}

class Test

{

public static void main(String[] args)

{

Shape s=new ShapeImpl();

s.circle();

s.square();

}

}

ex:

----

interface Shape

{

public abstract void circle();

default void square()

{

System.out.println("square method");

}

}

class ShapeImpl implements Shape

{

public void circle()

{

System.out.println("circle-method");

}

public void square()

{

System.out.println("Override square method");

}

}

class Test

{

public static void main(String[] args)

{

Shape s=new ShapeImpl();

s.circle();

s.square();

}

}

## static methods in interface

* If any method is declared by using "static" keyword then it is called static method.
* It is a non-abstract method.
* It can't be override.
* It is introduced in Java 8.

**ex:**

interface Shape

{

static void square()

{

System.out.println("square method");

}

}

class Test

{

public static void main(String[] args)

{

Shape.square();

}

}

## How can we achieve multiple inheritance in java

Using default methods of an interface we can achieve multiple inheritance in java.

**ex:**

interface Right

{

default void m1()

{

System.out.println("Right-M1 Method");

}

}

interface Left

{

default void m1()

{

System.out.println("Left-M1 Method");

}

}

class Middle implements Right,Left

{

public void m1()

{

System.out.println("Middle-M1 Method");

}

}

class Test

{

public static void main(String[] args)

{

Middle m=new Middle();

m.m1();

}

}

**ex:**

interface Right

{

default void m1()

{

System.out.println("Right-M1 Method");

}

}

interface Left

{

default void m1()

{

System.out.println("Left-M1 Method");

}

}

class Middle implements Right,Left

{

public void m1()

{

Right.super.m1();

}

}

class Test

{

public static void main(String[] args)

{

Middle m=new Middle();

m.m1();

}

}

ex:

interface Right

{

default void m1()

{

System.out.println("Right-M1 Method");

}

}

interface Left

{

default void m1()

{

System.out.println("Left-M1 Method");

}

}

class Middle implements Right,Left

{

public void m1()

{

Left.super.m1();

}

}

class Test

{

public static void main(String[] args)

{

Middle m=new Middle();

m.m1();

}

}

ex:

interface Right

{

default void m1()

{

System.out.println("Right-M1 Method");

}

}

interface Left

{

default void m1()

{

System.out.println("Left-M1 Method");

}

}

class Middle implements Right,Left

{

public void m1()

{

Right.super.m1();

Left.super.m1();

}

}

class Test

{

public static void main(String[] args)

{

Middle m=new Middle();

m.m1();

}

}

## Stream API

* Stream API introduced in Java 8.
* It is used to process the objects from Collections.
* A Stream is an interface which is present in java.util.stream package.
* It is used to perform bulk operations on Collections.
* We can create Stream object as follow.

ex:

Stream s=new Stream();

**ex:**

import java.util.\*;

import java.util.stream.\*;

class Test

{

public static void main(String[] args)

{

List<Integer> list=Arrays.asList(7,3,2,1,9,5,4);

//even numbers

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(7,3,2,1,9,5,4);

//odd numbers

List<Integer> odd = list.stream().filter(i->i%2!=0).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(7,3,2,1,9,5,4);

long count = list.stream().filter(i->i%2!=0).count();

System.out.println(count);

}

}

**ex:**

import java.util.\*;

import java.util.stream.\*;

class Test

{

public static void main(String[] args)

{

List<Integer> list=Arrays.asList(7,3,2,1,9,5,4);

List<Integer> newList= list.stream().map(i->i+10).collect(Collectors.toList());

System.out.println(newList);

}

}

**ex:**

import java.util.\*;

import java.util.stream.\*;

class Test

{

public static void main(String[] args)

{

List<Integer> list=Arrays.asList(7,3,2,1,9,5,4);

List<Integer> newList= list.stream().sorted().collect(Collectors.toList());

System.out.println(newList);

}

}

**ex:**

import java.util.\*;

import java.util.stream.\*;

class Test

{

public static void main(String[] args)

{

List<Integer> list=Arrays.asList(7,3,2,1,9,5,4);List<Integer> newList= list.stream().sorted(Comparator.reverseOrder()).collect(Collectors.toList());

System.out.println(newList);

}

}

**ex:**

import java.util.\*;

import java.util.stream.\*;

class Test

{

public static void main(String[] args)

{

List<Integer> list=Arrays.asList(7,3,2,1,9,5,4);

long min= list.stream().min((i1,i2)->i1.compareTo(i2)).get();

System.out.println(min);

}

}

**ex:**

import java.util.\*;

import java.util.stream.\*;

class Test

{

public static void main(String[] args)

{

List<Integer> list=Arrays.asList(7,3,2,1,9,5,4);

long max= list.stream().max((i1,i2)->i1.compareTo(i2)).get();

System.out.println(max);

}

}

## forEach() method

* A forEach() method introduced in Java 8.
* It is used to iterate the objects from Collections.

ex:

import java.util.\*;

import java.util.stream.\*;

class Test

{

public static void main(String[] args)

{

List<Integer> list=Arrays.asList(7,3,2,1,9,5,4);

list.forEach(element -> System.out.print(element+" "));

}

}

**ex:**

import java.util.\*;

import java.util.stream.\*;

class Test

{

public static void main(String[] args)

{

Map<Integer,Integer> map=new LinkedHashMap<Integer,Integer>();

map.put(1,100);

map.put(3,300);

map.put(2,200);

map.forEach((key,value)-> System.out.println(key+" = "+value));

}

}

## Method Reference

* It is introduced in Java 8.
* Method references are special type of lamda expressions.
* It is often used to create a simple lamda expression which referencing to existing methods.

ex:

import java.util.\*;

import java.util.stream.\*;

class Test

{

public static void main(String[] args)

{

List<Integer> list=Arrays.asList(7,3,2,1,9,5,4);

list.forEach(System.out::println);

}

}

ex:

import java.util.\*;

import java.util.stream.\*;

class Test

{

public static void main(String[] args)

{

Map<Integer,Integer> map=new LinkedHashMap<Integer,Integer>();

map.put(1,100);

map.put(3,300);

map.put(2,200);

map.forEach(Test::entrySet);

}

private static void entrySet(Integer key,Integer value)

{

System.out.println(key+" = "+value);

}

}

**Interview Question**

**Q) Write a java program to display employees records in sorting order of employee id?**

import java.util.\*;

import java.util.stream.\*;

class Employee

{

private int empId;

private String empName;

private double empSal;

Employee(int empId,String empName,double empSal)

{

this.empId=empId;

this.empName=empName;

this.empSal=empSal;

}

public int getEmpId()

{

return empId;

}

public String getEmpName()

{

return empName;

}

public double getEmpSal()

{

return empSal;

}

}

class Test

{

public static void main(String[] args)

{

List<Employee> list=new ArrayList<Employee>();

list.add(new Employee(103,"Alan",3000));

list.add(new Employee(101,"Jose",1000));

list.add(new Employee(104,"Nelson",4000));

list.add(new Employee(102,"Lara",2000));

List<Employee> newList=list.stream().sorted(Comparator.comparingInt(Employee::getEmpId)).collect(Collectors.toList());

newList.forEach(employee -> System.out.println(employee.getEmpId()+" "+employee.getEmpName()+" "+employee.getEmpSal()));

}

}

Q) Write a java program to display employees records in sorting order of employee id?

import java.util.\*;

import java.util.stream.\*;

class Employee

{

private int empId;

private String empName;

private double empSal;

Employee(int empId,String empName,double empSal)

{

this.empId=empId;

this.empName=empName;

this.empSal=empSal;

}

public int getEmpId()

{

return empId;

}

public String getEmpName()

{

return empName;

}

public double getEmpSal()

{

return empSal;

}

}

class Test

{

public static void main(String[] args)

{

List<Employee> list=new ArrayList<Employee>();

list.add(new Employee(103,"Alan",3000));

list.add(new Employee(101,"Jose",1000));

list.add(new Employee(104,"Nelson",4000));

list.add(new Employee(102,"Lara",2000));

List<Employee> newList=list.stream().sorted(Comparator.comparing(Employee::getEmpName)).collect(Collectors.toList());

newList.forEach(employee -> System.out.println(employee.getEmpId()+" "+employee.getEmpName()+" "+employee.getEmpSal()));

}

}