Skeleton of JAVA program:-

First.java

```
import java.lang.*;
class First {        → for calling method without
                              using object
    public static void main (String args[]) {
        System.out.println("Hello world");
    }    ↳      ↳        ↳
       class  object    method
```

```
C:\> javac First.java
          ↓
     First.class
C:\> java First.
```

Reading from keyboard.

```
import java.lang.*
import java.util.*
class keybRead {
      public static void main (String Args[]){
        Scanner s= new Scanner (System.in )
                                   keyb
        int a,b,c;
        System.out.println("Enter 2 nos");
        a= s.nextInt();
        b= s.nextInt ()
        c = a+b;
        S.O.P ("Sum is" + c );
      }
}
```

util
V5

nextInt()
nextFloat()
nextDouble()
next()
nextline()
nextByte()
nextShort()
nextLong()
nextBoolean()

return True/false { → hasNextInt()
→ hasNextFloat()

Type	Size	Range	Default.	
Byte	1	-128 to 127	0	
Short	2	-32768 to -32767	0	
int	4	-2147483648 to 2147483647	0	
long	8	—	0	
float	4	±3.4E-38 to ±3.4E+38	0.0f	
double	8	±1.7E-308 to ±1.7E+308	0.0d.	
char	2	0 to 65535	\u0000	unicode.
Boolean	?	true/false	false	

6-7 significant

0b1010

012

0XA

10

~5 = -6

0000 0101

1111 1010

0000 0101

+1

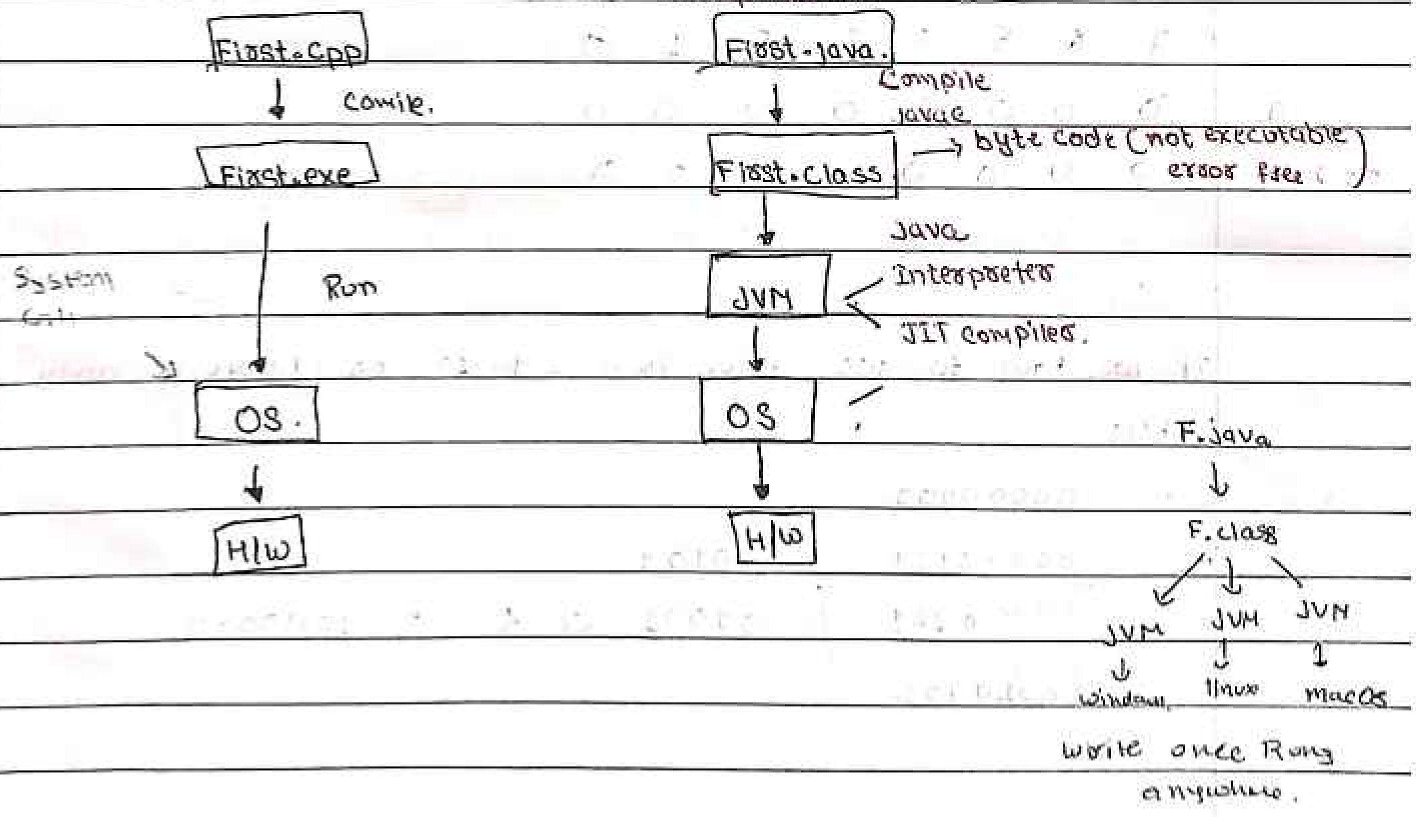
Features:-

1. Simple
2. Secure
3. Portable
4. Object-Oriented
5. Robust.
6. Multithreaded
7. Architecture-Neutral.
8. Interpreted.
9. High-Performance
10. Distributed.
11. Dynamic.

Interpreter	Compiler
Translates program one statement at a time.	Scans the entire program and translates it as a whole into machine code.
Interpreters usually take less amount of time to analyze the Source Code. However, the overall execution time is comparatively slower than compilers.	compilers usually take a large amount of time to analyze the source code. However, the overall execution time is comparatively faster than interpreters.
No Object Code is generated, hence are memory efficient.	Generates Object Code which further requires linking, hence requires more memory.
JavaScript, Python, Ruby uses interpreter.	C, C++, Java uses compilers.

* Java can be considered both a complier & interpreted language because its Source code is first complied into a binary byte-code. This byte-code runs on the Java Virtual Machine (JVM), which is usually a software-based interpreter.

Platform Independent.



Operators and Expression

Increment / Decrement

Post++ , Post--

++pre, --pre

Arithmetic

*, /, % → cannot work on float

+, -

Bitwise

&, |, ~, ^, <<, >>, >>> ^ XOR (if same then 0)

<< ↓ Left shift, >> ↓ Right shift

Left shift make value 2ti

Rightshift make value h

Relational

<, <=, >, >=, ==, !=

Logical

&&, ||, !

Merging and Masking

byte a = 0;

	7	6	5	4	3	2	1	0
a	0	0	0	0	0	0	0	0
b=8 →	0	0	0	0	1	0	0	0
	0	0	0	0	1	0	0	0

OR → merging [to mask the bit one]

and masking [to check the bit is one]

If we have to store value from 1 to 10 as it require only 4 bits.

4 bits Nibble

Ex:- 00000000

00000101 | 0101

00000101 | 00001001 << 4 = 10010000

10010101

Swapping

a = 9 b = 12

a = a ^ b = 5

~~b~~b = a ^ b = 9

a = a ^ b = 12

a ^ b - result will not be greater than maximum number.

Widening Narrowing

implicit 1. Size explicit

upcasting 2. compatability downcasting

byte → Short ↘

int → long → float → double

char ↗

Printing

Build in class ↗ System.out.println(" ")

↑ object ↑ methods

1. System.out.print(

2. System.out.println

3. System.out.printf(

4. System.out.format

Format specifier :-

% [argument_index$] [flags] [width] [.precision] conversion

argument index - 1$, 2$, 3$....

flag '-', '+', '0', ',', '('

conversion -

char c

int d, o, x

float f, e, g

String s.

String is Immutable.

STRINGS

String str = "Java"

int String = str.length()

int length()

String toLowerCase()

String toUpperCase()

to remove head & tail blank space. → String trim()

String substring(int begin)

String substring(int begin, int end)

String replace(char old, char New)

boolean startsWith(String s)

boolean endsWith(String s)

char charAt(int index)

int indexOf(String s).

int lastIndexOf(String s)

boolean equals(String s)

boolean equalsIgnoreCase(String s)

int compareTo(String s)

String valueOf(int i).

Reguler Expression.

		Regular Expression	Description
		.	Any Character
\d	Digits	[abc]	Exactly given letters.
\D	Not digits	[abc][vz]	Either first or second set.
\s	Space	[^abc]	Except abc
\S	Not space	[a-z 1-7]	a-z or 1-7
\w	Alphabet or digit	A|B	A or B.
\W	Not	X Z.	Exactly XZ

```
PSVM(S A){
String str1="f";
Sout(str1.matches("."));   → true
 }
```

Quantifiers

Regular Expression	Description
*	0 or more time
+	One or more
?	0 or 1 time
{X}	X times
{X, Y}	Between X and Y time.

Q Find if the email id is on gmail
Find username and domain name form email

```
String str = "programmer @gmail.com";
        i = indexOf ("@")
   user name = substring (0, i);
     domain = substring(i+1, str.length());           | int j=domain.indexOf(".");
       domain. StartsWith("gmail");                   | name=domain.substring(0,j
                                                        name.equals("gmai");
```

Q Find if a Number is Binary or Not.
Find is a Number is Hexa-Decimal or not.
Find is the data in Date format (dd/mm/yyyy)

```
int   b = 10110001;
String str = b+" "; valueof(b);
S.O.P (str.matches ("[01]*"));                 ("[0-9A-F]+");
  string d = " 01/12/2000";
       (" [0-3][0-9]/[01][0-9]/[0-9]{4} ")
```

Q Remove special characters from a string
Remove extraspaces from string. → str.replaceAll ("\\s+", " ");
Find number of words in a string?

Soln

```
          replaceAll ("[^a-zA-Z 0-9]", "");
                  str = str.replaceAll ("\\s+", " ").trim();
               String words[] = str.split ("\\s");
                 words.length();
```

Conditional statement.

Relational Operators

- <
- <=
- >
- >=
- ==
- !=

Logical Operators

- &&
- ||
- !

Q Greatest of 3 numbers

Q Find a number is odd or even

Find person is young or not young

find grades for given marks.

Q Find radix of a number given in a string

Find a given year is a leap year

```
if (num.matches("[01]+"))
        Binary
elseif.
    (num.matches["[0-7]+"))
        Octal.
```

Q Display name of a day based on number.

find type of website and the protocol used.

```
Desc.   if (day ==1) { monday.}
        else if (day = = 2) { Tuesday.

String url = " http://www.google.com"
        url.substring(0, url-indexOf(":"));
    if (protocol.equals("http"))

String ext = url.substring (url.lastIndexOf(".")+1);
    if (ext.equals("com")) {

        }
```

Switch Case :-

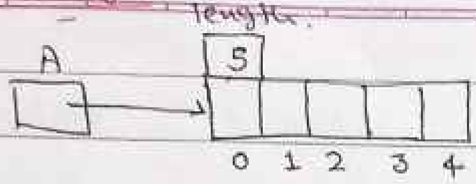
Menu driven program for arithmetic operation.

Loops

1. while loop
2. do..while loop.
3. for loop
4. for each loop.

```
while(condition){            do{
      ≡                        ≡
}                            }while(condition);
```

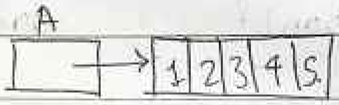
Arrays



```
int A[] = new int[5];
```

$\underbrace{\text{A[]}}_{\text{Refrence}}$ = new $\underbrace{\text{int[5];}}_{\text{object.}}$

```
Sout(A.length);
int A[] = {1,2,3,4,5};
```



```
for(int i=0; i < A.length; i++) {
  Sout(A[i]);
}
```

oreach.

```
for(int x : A) {
   Sout(x);
 }

for(int i=0; i< A.length; i++) {
     if(A[i] == Key) {
       S.O.P(i);
        System.exit(0);
     } }
  S.O.P("Not found");
```

```
max = A[0];
for(int i=0; i<A.length; i++){
   if(A[i] >max) {
      max = A[i];
    }
}
S.O.P(max);
```

2nd largestElemet.

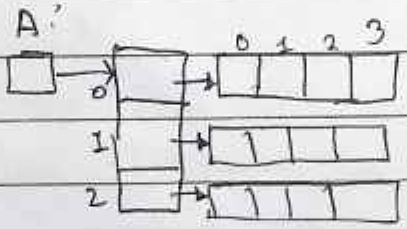
A [3|9|7|8|12|6|15];

```
max~~max~~max2=A[0];

if(A[i]>max1){
    ~~Max~~ max2= max1;
      ~~M~~max1= A[i];
    }
 else if ( A[i]>max2
    max2=A[i]
   }
```

2D Arrays

```
int A[][] = new int[3][4]
```

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



```
int A[][];
 A = new int[3][4];
```

Jagged array.

```
int A[][].
    A = new int[3][];
    A[0] = new int[2];
    A[1] = new int[4];
    A[2] = new int[3];
```

$C[0][0] = A_{00}B_{00} + A_{01}B_{10} + A_{02}B_{20}$

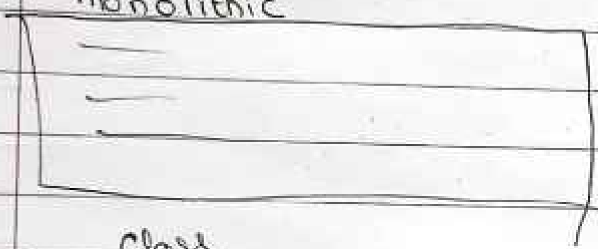
$C[0][1] = A_{\underset{i}{0}0}B_{0\underset{j}{1}} + A_{\underset{i}{0}1}B_{1\underset{j}{1}} + A_{\underset{i}{0}2}B_{2\underset{j}{1}}$

```
for (i - - - )
  for (j - - - )
    for (k=0; k<      ; k++) {
    c[i][j] = ΣA[i][k] * B[k][j];
  }
```

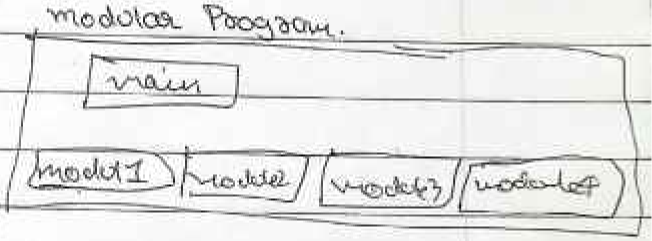
Java.util.Arrage.sort(arr);

Methods :-

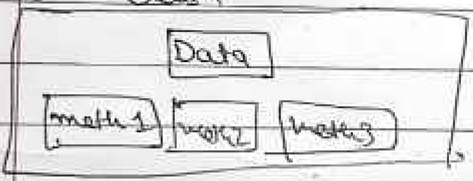
monolithic



modular Program.



Class



Signature → return Type methodName (parameter List)

```
{
   ≡
}
```

Parameter Passing :-

```
int add (int x, int y) {        } formal parameters.
    int z;
    z=x+y;
    return z;
}
psv main (---) {
    int a=10, b=5, c;
    c= add(a,b);               } actual parameters.
    S.O.P(c);
```

method call.

variable Arguments varargs

```
void show (int ...x){

}
```

```
void show(int x, int ...y){

}
```

Object Oriented programming

Principle :-

1. Abstraction:- Hiding internal details, showing only required things.

2. Encapsulation:- Everything in a box.

Specialization
3. Inheritance :-

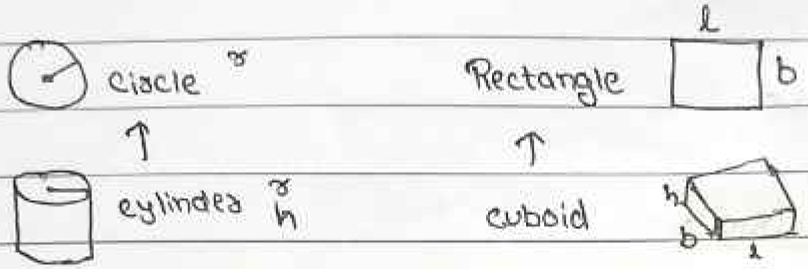
Generalization.
4. polymorphism:-

Class vs Object

Inheritance:- Process of acquiring features of existing class into a new class.

features:- Properties & methods.

Example:-



code:-

```
class Circle {
  private double radius
  public circle (){
       radius = 0.0;
   }
  public double area(){ }
  public double perimeter() { }.
 }
class cylinder extends circle {
  private double height;
   public cylinder(){
     height=0.0;
    }
   public double volume(){}
 }
class Test{
 public static void main().{
    Circle c1= new Circle();          c1.area();
    cylinder c2 = new Cylinder();     c2.area();
   }
 }
```

Constouctor in inheritance

1. 1st parent class constouctor is called then child constouctor.

Parameterized Constouctoo.

```
Class parent {
    parent() {
        Sout("Non-Para parent");
    }
    Parent(int x) {
        Sout ("Para parent");
    }
}
class child{
    child(){
        Sout(" Non-para child");
    }
    Child(x){ Sout("Para child");
    }
    Child(x,y) { super(x); // call parameterized constouctor of parent class
                                                                with one parameter
               Sout("2-para child");
    }
}
```

this vs super

```
class Rectangle {
    int length;
    int breadth;
    Rectangle (int l, int b){
                r1
        this.length = l;
          r1
        this.breadth = b;
    }
    void display(){
                                           r1
        system.out.println("Length:"+this.length);
                                           r1
        System.out.println(" Breadth:"+this.~~bredt~~h);
    }
}.
Rectangle r1 = new Rectangle(10,5);
r1.display();
```

this is a refrence to ~~of an~~ present or current object upon which this method is called.

- Super is a refrence to a Super class.

Super.x //to call x of super class

Method Overriding

→ redefining the method of super class in subclass

Dynamic method Dispatch

- method will be called depending upon the object not on the refrence

  Superclass refrence holding object of subclass

→ It is usefull for acheiving run time polymorphism using method Overriding

Ex:-

```
class Super{
    void meth1(){ SOP("meth1");}
    void meth2(){ SOP("super meth2"); }
}

Class Sub extends Super{
    void meth2(){ SOP("sub meth2");}
    void meth3(){ SOP("meth3");}
}

Class Test {
    P.S.V main(){
            Ref.         object
        Super S = new Sub();   // can call only those object available
                                  in super class.
            S.meth1();   - meth 1
            S.meth2();   - sub meth2
      X  ~~S.meth3();~~
```

Do's & Don'ts

→ signature must be Same

→ return type should be same

→ cannot override final & static methos.

→ we cannot have strict specifier in Subclass

Private

protected

public.

Abstract Classes

① abstract

② concrete

→ abstract keyword is used.

→ object of abstract class is not possible.

→ may or may not contain min. one abstract method.

abstract method :- only declared not defined.

Do-s & Don-t-s

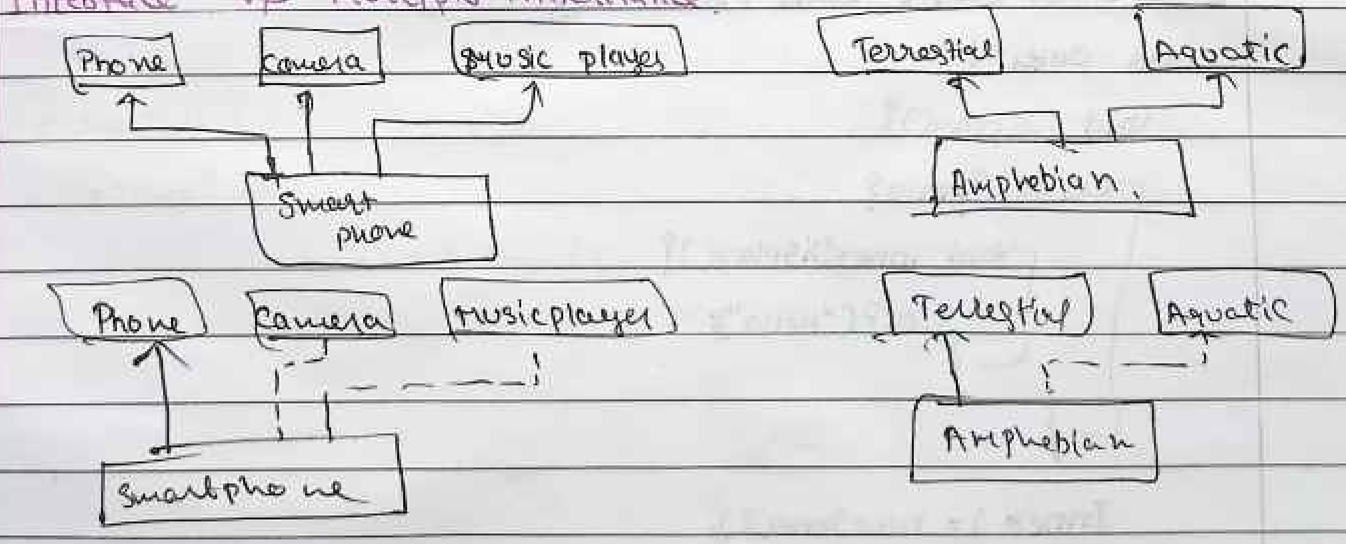
- cannot create object of abstract class.
- If class has abstract method then class must be declard abstract
- we cannot make final abstract class.

Interface:-

```
interface Test {
    void meth1();
    void meth2();
}
```

- completely used for acheiving polymorphism.
- contains only abstract method.
- implements keyword.
- class can implements many interfaces

Interface V/S Multiple inheritance



Inner class.

1. Nested
2. Local
3. Anonymous
4. static

Nested Inner class:- · we can use inner class by creating its object inside the outer class.

· outer class cannot access directly the variable of inner class.

```
class Outer{
  int x=10;
  class Inner{
      int y= 20;
      void innerDisplay(){
          S.O.P(x);
          S.O.P(y);
      }
  }
  void outerDisplay(){
    Inner i= new Inner();
     i.innerDisplay();
    S.O.P(i.y);
  }
}
```

Local Inner Class:- class inside a ~~function~~ method

```
Class Outer {
   void Display(){
     Class Inner{
         void innerDisplay(){
             S.O.P("Hello");
         }
     }
     Inner i= new Inner();
      i.innerDisplay();
   }
}
```

Anonymous Innerclass:- is defined at the time of creation of objects.

```
abstract class My{
      abstract void display();
   }

Class Outer{

      public void meth()
      { My m = new My(){   // Defining the message
          public void display(){
             S.O.P ("Hello");
          }
        };
        m.display();
      }
   }
```

Static Inner class:- are static members of outer class. object of SIC can be created outside the outer class & access from any where without creating object of ~~inner~~ outer class.

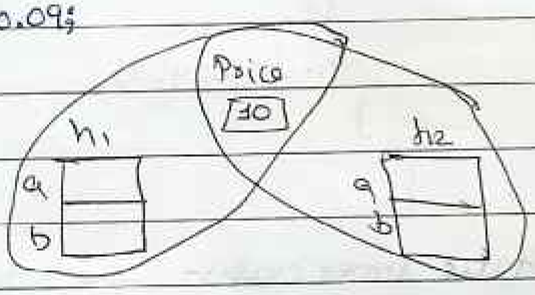
* static class can only access static member of class. cannot access not static member

```
Class Outer{
      Static int x=10;
          int y=20;
    static class Inner{
         void display(){
          ✓ S.O.P(x);
          ✗ S.O.P(y);
         }

 class Test{
   psvm(){
    Outer.Inner i= new outer.Inner();
      i.display();
    }
```

Static

Static keyword is used for representing metadata

```
class HondaCity{
    static long price = 10;
    int a,b;
    Static double onRoadPrice(String city){
        switch (city) {
            case "delhi":
                    return price + price*0.1;
            case "Mumbai":
                    return price+price*0.09;
            }
        }
}
```



```
Class Test. {
    P.S.V main(). {
    Honda city  h1 = new HondaCity();
    HondaCity  h2 = new HondaCity();
     S.O.P (h1.price); — 10
     S.O.P(h2.price); - 10.
     S.O.P( Honda City.price);
```

static block

- ~~This~~ block will execute as the class loads before creation of any object.

```
class My {
        static int s;
       Static {
             S.O.P("B-1");
             S = 10;
          }
           ⋮
        Static {
              S.O.P("Block2");
             }
              ⋮
```

final Keyword

final variable: value cannot be modified.

- final variables are written in Uppercase.

```
class My{
       final int MIN=1;
       final int NORMAL;
       final int MAX;
   Static{
       NORMAL=5;
    }
   MY(){
       MAX = 10;
   }
```

Final Method: It cannot be overrided.

```
class Super{
  final void meth1(){
    S.O.P("Hello"); } }
Class sub extends Super{
X  void meth1(){
   SOP("Hi"); }
  void meth2(){
    S.O.P("Bye");}
 } .
```

final class: This class cannot be extended

```
final class Super{ --- }
class sub extends Super{ - - }.
```

Singleton Class: more than one object of the class is not allowed.

```
Class CoffeeMachine{ private float coffeeQty; private float waterQty; Static private
                                                                      CoffeeMachine con
   private coffeeMachine(){
          coffeeQty=1; waterQty=1; }
   Static public coffeeMachine getInstance(){
      if(our==null)  our = new CoffeeMachine();
        return our;
        }
                                    coffee machine c = new CoffeeMachine()
                                    coffee Machine c = CoffeeMach.getInstance
```

Packages:-

- A package is a collection of classes, interfaces or other packages.
- are used for organizing the java project.
- Grouping of related classes interfaces & packages.

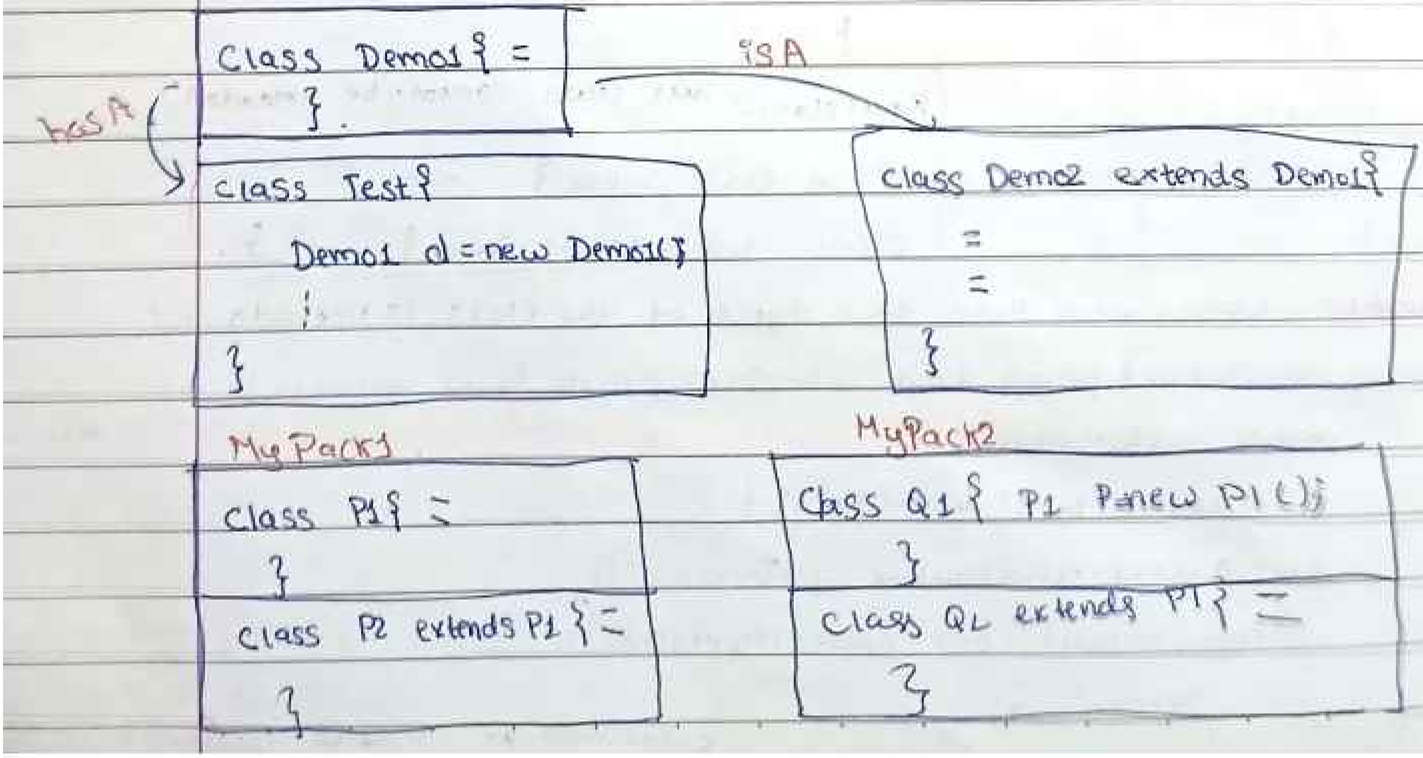
```
          location
javac -d . Demo.java
```

```
package mypack1;        package mypack1.inner;
```

Access Modifier:-

- default
- private
- protected
- public

```
class Demo{
      - int x;
any.  - void show(){ =
            }
      - class Inner{ =
           }
}
```



```
class Demo1{ =
   }.

class Test{
    Demo1 d=new Demo1()
   };
}

class Demo2 extends Demo1{
    =
    =
}
```

```
MyPack1
class P1{ =
  }
class P2 extends P1{ =
  }

MyPack2
class Q1{ P1 P=new P1();
  }
class Q2 extends P1{ =
  }
```

	default	private	Protected	Public
Same class	✓	✓	✓	✓
Same Pack Sub class	✓	X	✓	✓
Same Pack non-sub class	✓	X	✓	✓
Different Pack Sub class	X	X	✓	✓
Diff Pack non-sub class	X	X	X	✓

Exception Handling :-

→ Exception are runtime errors.

Types of Errors.

Programmer ←

Compiler
1. syntax error →

```
int x, y;
1. x = 10_
2. z = x + y;
```

Tracing / Debugger.
2. logical errors. →

① $r = \frac{-b}{2a}$ r = -b/(2 * a);

```
② for (int i = (1); i < A.length; i++) {
      S.O.P(A[i]);
  }
```

User —

Exception Handling.
3. Runtime errors (Exceptions)
① Bad input
② unavail Resources.

Exception Handling Construct :-

① multiple
② nested.

```
Class Test {
    P.S.V main() {
            int a, b, c;
        try { a = 10; b = 0; c = a/b;   S.O.P("Result is " + c); }
        catch (Arithmetic Exception e) { S.O.P("Division by zero" + e); }
    }
}
```

finally { } case with try but it will surely get executed
cleanup

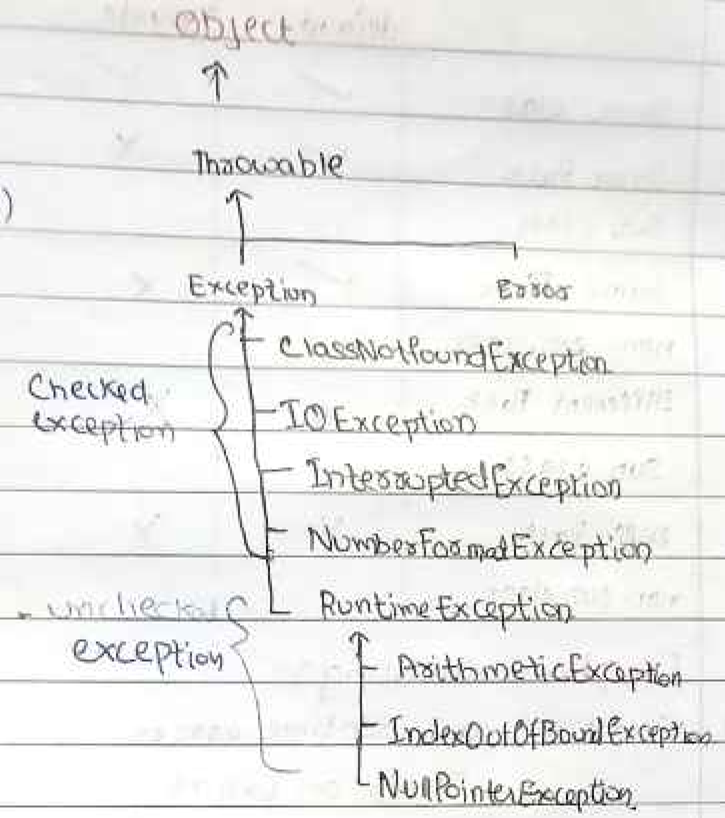
Exception classes

Class Exception

- String getMessage()
- string toString()
- void printStackTrace()

must handle them using try & catch

not compulsory



```
try { =
    }
        Sub class
catch ( Arith Exception e ) {
       =
    }
        super class
catch(Exception e) {

  }
```

Userdefined exception:-

```
Class MinBalanceException etends Exception {
        public string toString() {
                return "min balance should be500g try aggin with smalleramount";
        }
```

Throw vs Throws

Exception

```
PSV main(){

    meth1( );

}
```

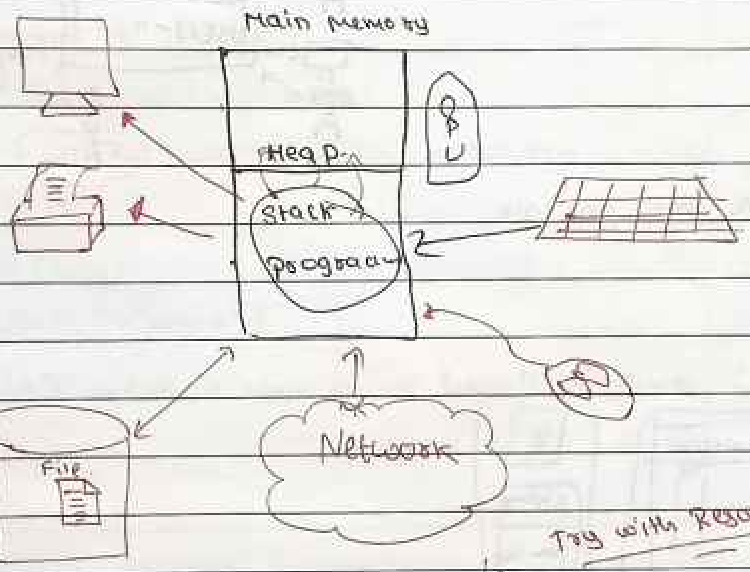
```
int meth1( ) throws Exception.
{try
 {
  int a= area(-10,5);
  SOP (a);}
  catch(Exception e) {S.o.p(e);}
}
```

if dont want to catch here.
then in signature throwsException

```
int area(int l, int b){throws
 {if (l<0 || b<0)
     Throw new Exception("
 int a = l*b;
 return a;
```

User defined Exception.
own Except

Try with resources



```
int meth1() throws Exception
{ FileReader f;

   try
   {
 → f= new FileReader("my.txt");
 → //use file
   xf.close(); //Clean up is not done
   xreturn result;
   }
  finally {
       f.close();
      }
  }
```

Try with Resources

```
int meth1(    ) throw Exception{
                     r1;r2;r3
   try (FileReader f =new FileReader("my
   {
      //use file
      return result;
   }
}
```

interface Autoclosable

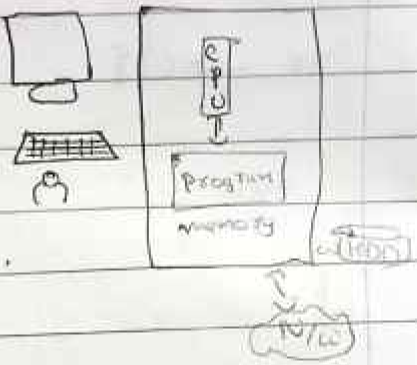
read only

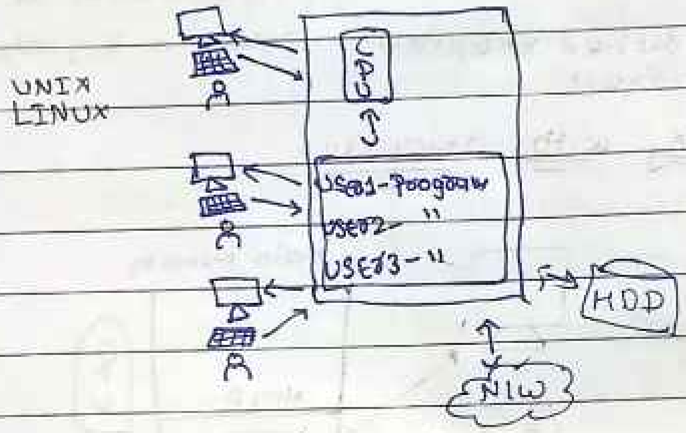
Multithreading :-

Multiprogramming

multi-user — MultiTasking

MultiTasking → Multithreading.

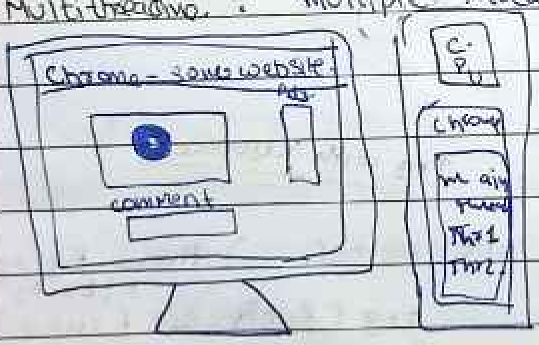




multitasking

running more than one task

Multithreading :- multiple thread in a single appliction



Control flow of program:-

→ One program basically have one flow.

Multithreading in java

① Thread class ② Runnable Interface

must override run method when extends Thread.

```
class MyThread extends Thread{
        public void run(){
                int i=1;
                while(true){
                    S.O.P( i+ "Hello");
                    i++;
                }
        }
}
```

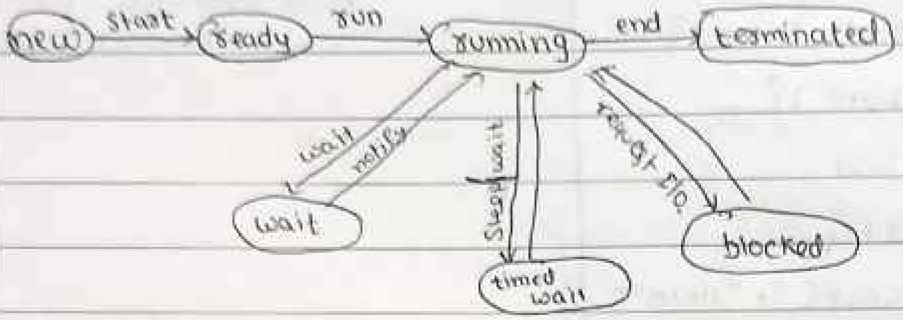
```
Class Test{
    P.S.V main(){
      MyThread t= new MyThread();
      t.Start(); // built in method to call run method.
      int i=1;
      while (true){
         S.O.P(i+"world");
         i++;
      }
    }
}
```

using Runnable interface

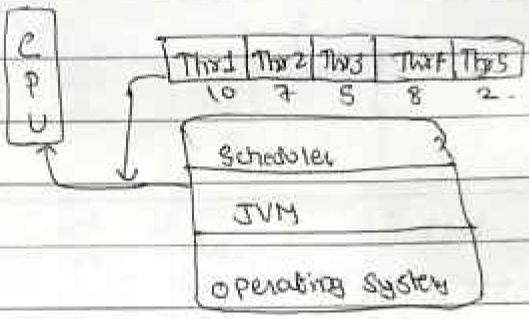
```
class My implements Runnable{
      public void run(){
          int i=1;
          while (true){ S.O.P(i+"Hello");i++;
          }
      }
}
```

```
Class Test{
      PSV main(){
  →   My m= new My();
  →   Thread t= new Thread(m);
      t.Start();
      int i=1;
      while(true){S.O.P(i+"world"); i++;}
      }
}
```

States of a Thread.



Thread Priorities.



Thread.MIN_PRIORITY = 1

Thread.NORM_PRIORITY = 5

Thread.MAX_PRIORITY = 10

Thread class

Constructors

Thread( )

Thread (Runnable r)

Thread (Runnable r, String name).

Thread (ThreadGroup g, String name).

Thread (String name)

Other methods

getxxx( ) / Setxxx( )	Enquiry
long getId( )	boolean isAlive( )
String getName( )	boolean isDaemon( )
int getPriority()	boolean isInterrupted( )
ThreadState getState( )	
ThreadGroup getThreadGroup ( )	

void setName (String name)

void setPriority (int P)

void. getDaemon (boolean d).

Instance Methods

void interrupt( )

void join( )

void join(long millis)

void run( )

void Start( )

Static Methods

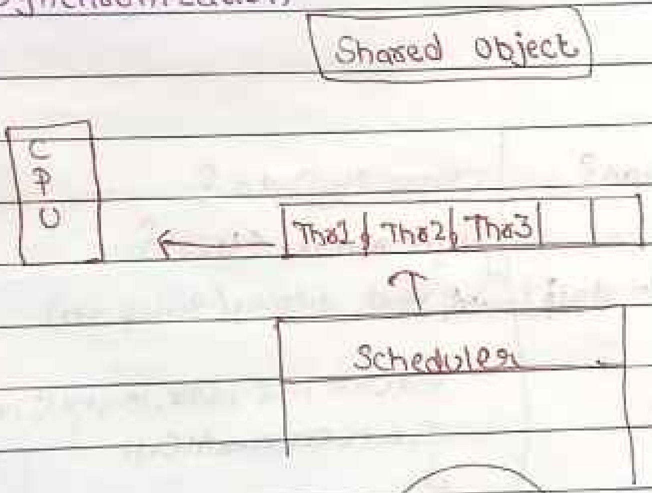
int activeCount( )

Thread currentThread( )

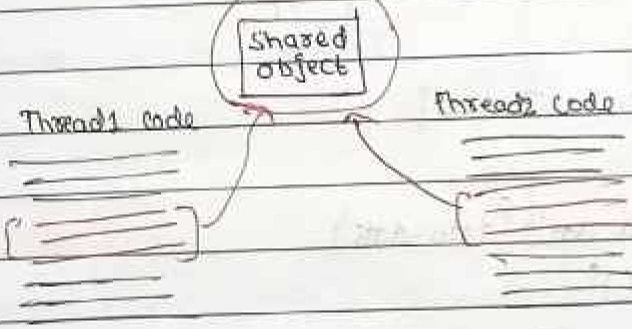
void yield( ) // hold for sometimed higher priority

void dumpstack( )

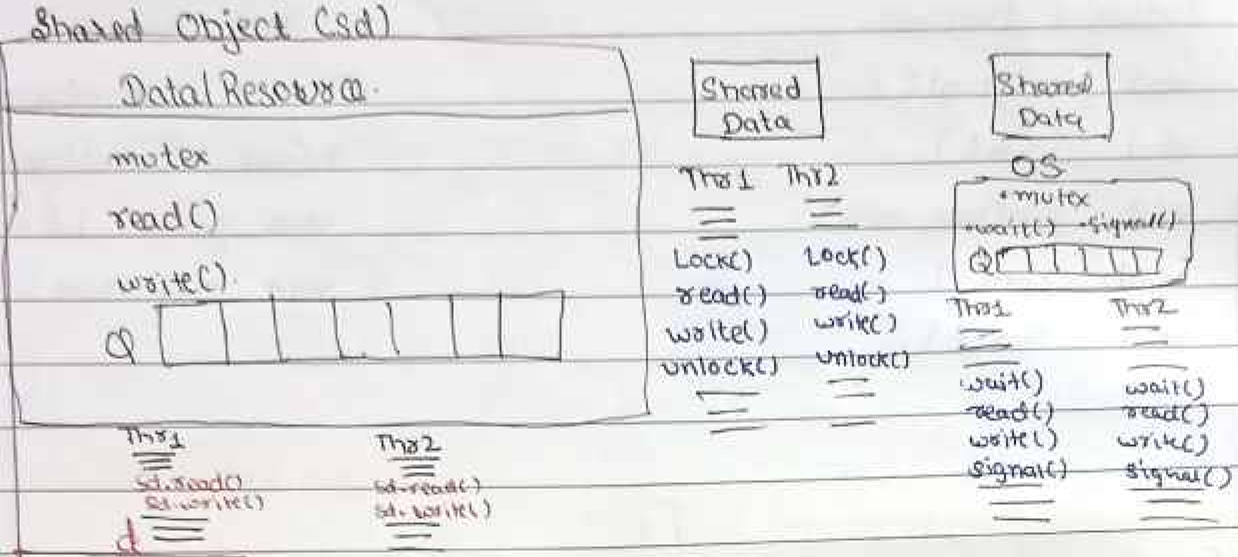
Synchronization

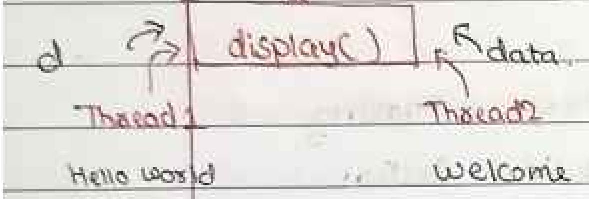


1. Resource Sharing
2. Critical Section
3. Mutual Exclusion
4. Locking / Mutex
5. Semaphore
6. Monitor
7. Race Condition
8. Inter-Thread Communication.



Monitor



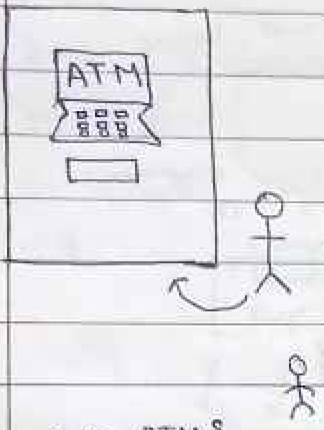


```
Class Mythread1 extends Thread{
    My Data d;
    MyThread1(My Data dat) {d= dat;}
    public void run(){
    d.display("Hello world");
    }
}
```

```
Class MyData {
 synchronized (this) {
   void display(String str)
   for(int i=0; i<str.length(); i++) {
   S.o.P(str.charAt(i));
   }
  }}
}
```

```
Class MyThread2 extends Thread{
    MyData data;
    MyThread2(MyData dat) {data=dat;}
    public void run(){
    dat.display("welcome");
     }
}
class Test {
 p.s.v main(...)
  MyData d= new MyData();
  MyThread1 t1 = new MyThread1(d);
  MyThread2 t2= new MyThread2(d);
  t1.start();
  t2.start();
```

Challenge Singelton class



```
class ATM {
syn   - Check Balance (name)
syn   - withdraw (name, amt);
        }
```

```
Class Customer {  ext Thread
   ATM atm;
   String name;
   int amt;
 - useATM () {
   checkBalance (--)
   withdraw (___, amount)
   }
```

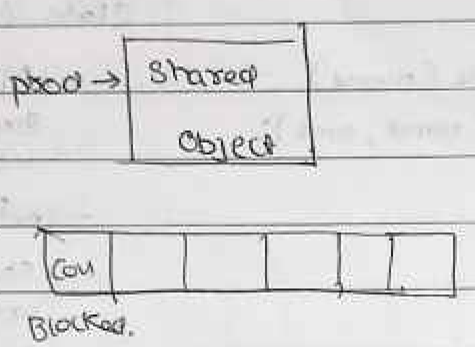
```
class ATM {
Synchronized public void checkBalance (String name) {
     try { Thread.sleep (1000); catch ( Exception e) { }
     Sopt ("Balance");
   }
Synchronized public void withdraw (String name, int amount)
       sout print ( name + " withdrawing");
       try { Thread. sleep (1000); } Catch (Exception e) { }
       System. out. println (amount);
     }
  }
```

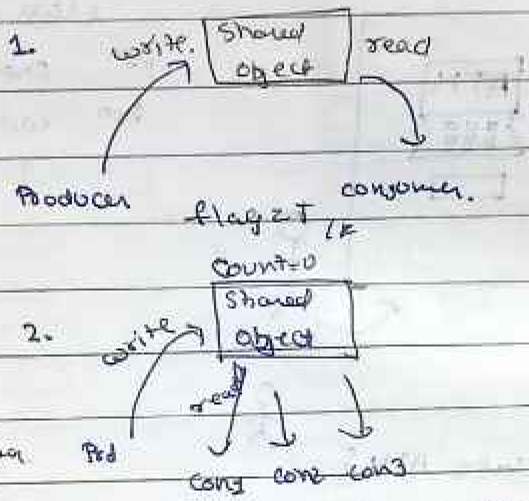
```
public class SCThread1 {
  public static void main (String[] ar
    ATM atm = new ATM();
  customer c1 = new customer("Smith", atm, 1
  Customer c2 = new Customer ("John", atm, 2
   c1.start ();
   c2.start ();
    }
  }
```

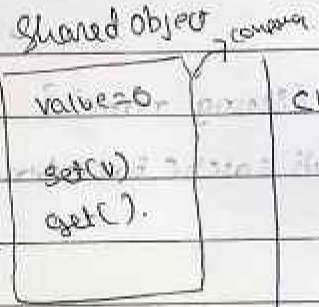
```
class Customer extends Thread {
     String name;
     int amount;
     ATM atm;
     Customer (String n, ATM a, int amt) {
        name = n;
        atm = a;
        amount = amt;
     }
     Public void useATM() {
       atm.checkBalance (name);
       atm.withdraw (name, amount); }
     Public void run() {
        useATM(); }
}
```

Page No.:
Date:
YOUVA

InterThread Communication.





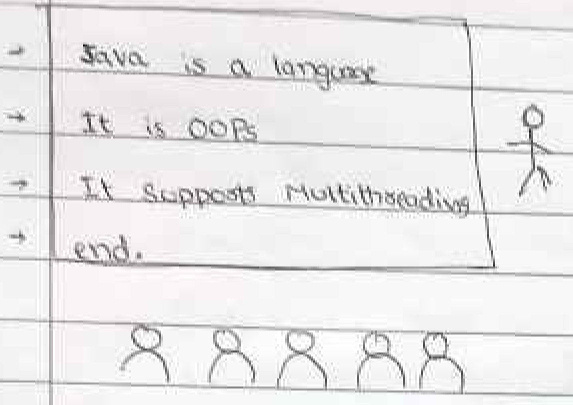


```
Class MyData{
  int value = 0;
  bool flag= true;// Prod turn

  Synchronized void set(int v){
    while (flag != true)
        wait();
    value = v;   || Producer done his job & set flag to false so consumer can do his.
    flag = false;
    notify();
  }
  synchronized int get(){
    int x=0;
    while (flag != false) wait();
    x= value;   || consumer has done his job & set flag
    flag= true;
    notify();
   ~~revalue;~~
    return x;
  }
}
```

```
class Consumer extends Thread{
    MyData d;
    consumer(MyData dat){ d=dat;}
    public void run() {
     while(true){
        S.O.P("con: " + d.get());
     }
    }
}
```

```
class Producers extends Thread{
   MyData d;
   producer(MyData dat) {d=dat;}{
    public void run()
      int i=1;
   while(true) {
     d.set(i);
   S.O.P("producer:"+i);
     i++;
   }
  }
}
```

Challenge
Student-Teacher & whiteboard



→ Java is a language

→ It is OOPs

→ It supports Multithreading

→ end.

```
Class whiteBoard{
  String text;
  int no.ofStudents=0;
  int counte=0; // how many hay cmpleted
  public void attendance(){
      noofStudents++;
  }
  Synchronized public void write(String t){
     sout("Teacher is writing "+t);
      while(count !=0) wait();
        text = t;
       Count=numberOfStudents;
       notifyAll();
  }
  Synchronized public String read(){
     while ((count==0)wait();
   String t=text;
   Count--;
    if(count==0)
         notify();
   return t;
  }
}
```

```
Class Studentt extends Thread{
   String name;
   whiteBoard wb;
   Public Student(String n, whiteBoard w){
      name=n;
       wb=w;
   }
   Public void run(){
      string text;
      wb.attendance();
    do{text = wb.read();
        sout(name+ "Reading "+text);
        s.out.flush();
       }while(!text.equals("end")); }}
```

```
Class whiteBoard{ String text; no.of student=0;
          Count;
         write(msg)

          String read(){
                 attendance(){
                      noofStudent++;
                 }
                 }
        }
Class Teacher extends Thread{
         run(){ }
   }
 Class Student extends Thread{
   String name;
     run{
        print
   }}
```

```
class Teacher extends Thread{
   whiteBoard wb;
   String notes[]={"Java is lang","It has OOPs",
                     "Platform Indepen","Thread"};
                                              "end"
  public Teacher(whiteBoard w){
       wb=w;
   }
   Public void run(){
      for(int i=0;i<notes.length;i++)
         wb.write(notes[i]); }
   }
```

```
public Class SCThread2{
   PSVM(String[] args){
     whiteBoard wb=new whiteBoard();
     Teacher t=new Teacher(wb);
   Student S1 = new St (            )
                 S2 =
                 S3 =
                 S4 =
               t.Start();
               S1.start();
               S2.start();
               S3.start
               S4.start();
```

java.lang Package

Object class:-

```
class MyObject {
   public String toString(){
        return "My Object";}
   public int hashCode(){
          return 100; }
    public boolean equals(Object o){
          return this.hashCode() = o.hashCode();}
        }
```

Wrapper class:-

```
public class Wrapper{
    public static void main(String[] args){
    Integer i= new Integer(10);
    Integer a=Integer.valueOf(10);
    Integer b=10;

     Byte c=15;
     Byte d=Byte.valueOf("15");
     byte bb=15;
     Byte e= Byte.valueOf(bb);
     Short f=Short.valueOf("123");
     Float g=12.3f
     Float h= Float.valueOf("123.5");
     Double j=Double.valueOf(123.456);
     Character k=Character.valueOf('A');
     Boolean l=Boolean.valueOf("true");
     Float x=h.floatValue();
     float y=h; //similar to above one also unboxing.
       int m=10;
      Integer n=m //autoboxing
       int p=n; //auto unboxing    similar n.intValue();
   }
}
```

e
bb=15

Integer class:-

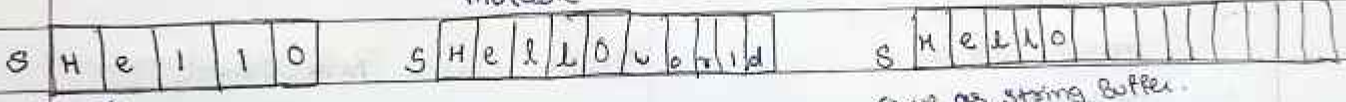
```
int m1 = 15;

Integer m2=m1;

Integer m3=15;

m2.equals(m1)    // true
m2.equals(m3)    // true

    Integer m2=Integer.valueOf("123");
    Integer m3= Integer.valueOf("A7",16);
    Integer m4 = Integer.decode ("0xA7");
     Sout( Integer.reverseBytes(128)== Integer.MIN_VALUE);
    }
}.
```

Float Class:-
Float Class

```
        float a=12.5f;
         Float b=12.5f;
     System.out.println(b.equals(a));
                    b.isInfinite();          b.isNaN();
```

String   String Buffer cap 16   String Builder.

mutable



immutable

HelloWorld

append()

insert()

s.append("world")

Thread-Safe // 2 threads cannot access SB simultaneously

Same as String Buffer.

not Thread-safe.

Math Class

Enum

```
enum Dept{ CS,IT,CIVIL,ECE} //Static, Public,final
main {
    Dept d=Dept.CS;
      d.ordinal()   //return index
      d.name()      //return name.
      Dept list[] = Dept.values()}
       for(Dept x: list)
          Soot (x);
```

- It can have constructor of Private or default tipe
- when enum is loaded for each constant constructor is called.

```
enum  Dept{ CS ("John", "Block A"), IT("Smith", "Block B"), CIVIL("Srinivas", "Block C")
               ECE("Dave", "Block D");
   String head;
   string location;
   private Dept(string head, string loc){
      this.head=head;
      this.location=loc;
   Public string getHead(){ return head;}
     "       "    getlocation() { return location;}
```

reflection package. (sub package of lang / java.lang.reflection).

```
import java.lang.reflect.*;
class My{
      private int a;
      protected int b;
      public int c;
             int d;
```

```
public class ReflectDemo{
    P.s.v.m(String[] args){
      Class c=My.class;
  or
     My m = new My();
     class c1 =m.getClass();
       c.getName();
   Field field[] = c.getDeclaredFields();
 Constructors con[] = c.getConstructors();
 Method ~~meth~~[] = c.getMethods();
```

Javadoc 15

Java Documentation:-

```
/** @author Abdul Bari
 * @vession 2.0
 * @Since 2018
 **/

package javadocdemo;
/**
* @author
* class for library Book
*/
public class Book{
   /**
    * @value 10 default value
    */
    static int val=10;
   /**
    * Parameterized Constructor
    * @param s Book Name
    */

public Book(String s){}
 /**
* Issue a Book to a Student
* @param roll roll number of a student
* @throws Exception if book is not available, throws Exception
*/
public void issue(int roll) throws Exception{}
/**
*check if boot is available
* @Param str Book name
* @return if boot is available returns true else false
*/
public boolean available(String str) {return true;}
 }
}
```

Annotations:- to provide metadata

1) Applied to code 2) Applied to Other Annotations

Inbuilt Annotations

1. @Override
2. @Deprecated
3. @FunctionalInterface
4. @SuppressWarnings
5. @SafeVarargs

```
import java.lang.annotation.Annotation;
@interface MyAnno{
        String name();
        String project();
        String version() default "13";
 }
        @MyAnno(name="Ajay", project="Bank")
```

in-built Annotations

1. Retention. (@Retention(RetentionPolicy. CLASS RUNTIME SOURCE)
2. Documented.
3. Target (@Target(value=ElementType. constructor field localVariable);
4. Inherited.
5. Repeatable.

Date:

Lamda Expression:-

```
Public class LamdaDemo {
  PSVM(String[] args) {
    MyLamba m = new MyLamdf() {
          P. v. display() {
          Sout ("Hello World");
          }
    };
    m.display();
```

OR

Anonymous Method. or Lambda Expression

```
                       () → {
    MyLambda m = ~~new [illegible]~~
          Sout("Helloworld");
          };
}
```

```
@FunctionalInterface
interface MyLambda {
   public void display();
}
class My implements MyLambda {
   P v display() {
   S.Out ("Helloworld"); }
}.
```

```
→ P V display (String str);
```

```
→ MyLambda m=(s)→{Sout(s);};

m. display ("Hello world");
```

```
MyLambda m = (a,b) → {return a+b;};
          or
MyLambda m = (a,b) → a+b;
```

Capture

can access local variables or capture local variables ~~which~~ only if they are final or they are ~~not~~ never modified inside the method.

method refrence — new for constructor

```
interface MyLambda { P.V display (String str); }
Public class Lambda demo { P S V m (String[] args) {
                        LamdaDemo::reverse
    MyLambda ml= System.out::println;
    ml . display ("Hello");
    }
```

LamdaDemo ld = new LamdaDemo();
MyLambda ml=ld :: reverse;

```
  P Ⓢ v reverse (String str) {
  StringBuffer sb = new StringBuffer(str);
  sb.reverse ();
  System.out.println (sb);
  }
}
```

IO Streams

java.io

Byte Stream

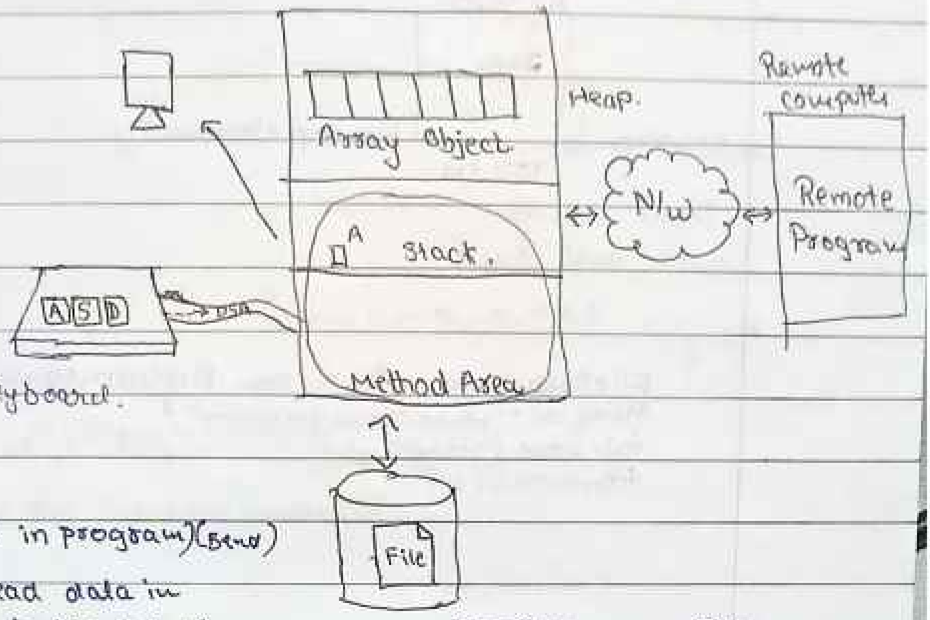
InputStream

OutputStream

Character Stream

Reader

Writer



Class InputStream :-

✓ int read() // read (data in program)(send)

✓ int read(byte[] b) // read data in b (send in b)

✓ int read(byte[] b, int off, int len) // off → which inderstand len → how much letter

✓ int available () // no of bytes available to read

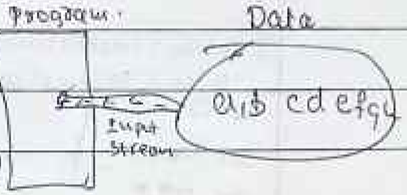
long skip(long n) // to skip & remove (just like discard)

void mark(int limit) // to mark (not gets removed from data). limit → how long its valid.

void reset () // to get to mark again

boolean markSupported() // ~~[illegible]~~ to check mark is supported.

✓ void close() . // to close resources.



Class OutputStream :-

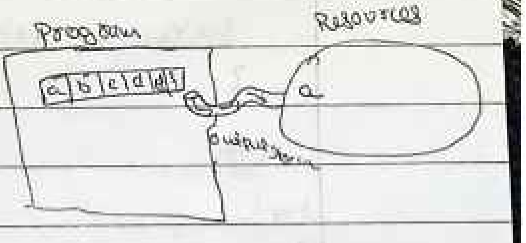
void write (int b)

void write (byte[] b)

void write (byte[] b, int off, int len)

void flush () // work on buffered output stream. ~~flush the data from buffer to Resources~~

void close ()



Java.lang.Object

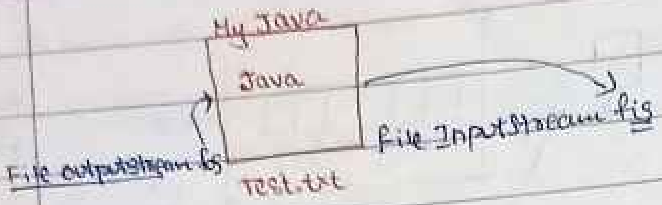
- java.io.InputStream
  - java.io.ByteArrayInputStream
  - java.io.FileInputStream
  - java.io.ObjectInputStream
  - java.io.Piped InputStream.
  - java.io. sequence InputStream.
  - java.io. StringBufferInputStream
  - java.io. ~~[illegible]~~ FilterInput Stream

java.io.OutputStream.

ByteArray OutputStream

Buffered

- java.io.[illegible]inputStream
- java.io. DataInputStream
- java.io. PushBack InputStream



```
import java.io.*
public class File {
        p.s.v.m(String[] args){
try {
   FileOutputStream fos = new FileOutputStream ("C:/MyJava/test.txt");
   String str = "Learn Java program";
   fos.write (str.getBytes())
   fos.close();



} catch (FileNotFoundException e){          }
   catch (IOException.
```

```
CMD
type test.txt
del T*.*
```

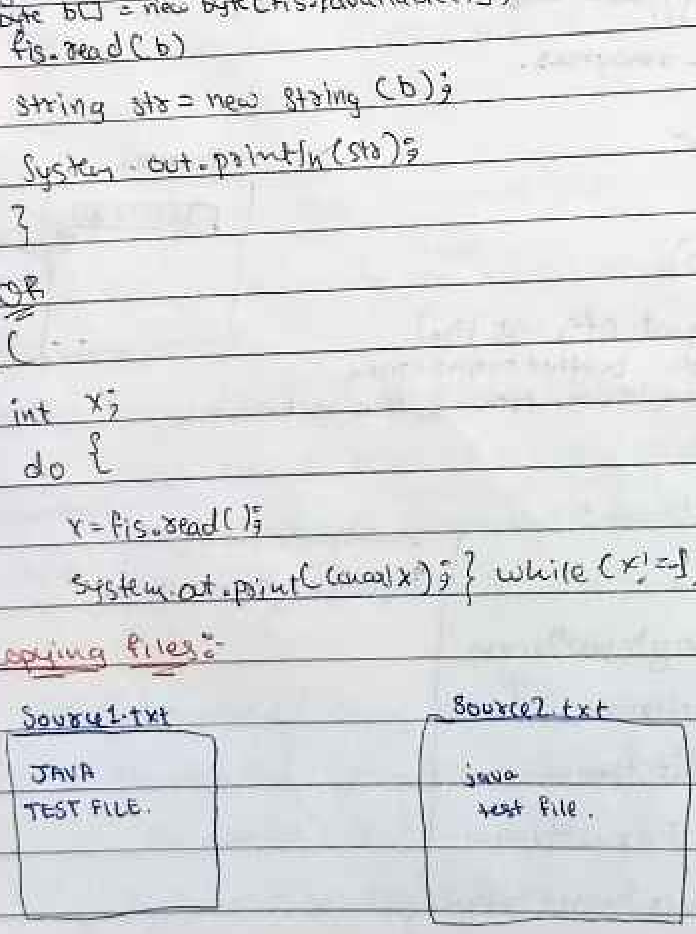
```
 try ( FileInput Stream fis = new FileInputStream ("C:/MyJava/Test.txt") ){
  byte b[] = new byte[fis.available()];
  fis.read (b)
   String str = new String (b);
   System.out.println (str);
  }
```

OR

```
try ( ..
{ int x;
   do {
       x = fis.read();
       System.out.print((char)x); } while (x != -1);
```

Copying files:-

65-90
97-122.



```
fis Input
fis Input
fos Output
SequenceInputStream sis = new SequenceInputStream(fis1, fis2);
 int b;
while ((b=sis.read())!=-1){
   fos.write(b);

 }
```

Byte Streams:-

```
Public class ByteDemo{
        psvm(String[] args).
   {  byte b[]={'a','b','c','d','e','f','g','h','i','j'};
ByteArrayInputStream bis=new ByteArrayInputStream(b);
 int x;
while  bis.readAllBytes();
```

Java Generics:

Introduction:

class Object

```
        Object obj = new String("Hello");
        string str = (string) obj;
public class GenericDemo<T> {
    T data[] = (T[]) new Object[3];
    P S V M(String[] args) {
        GenericDemo<String> gd = new GenericDemo();
          gd.data[0] = "hi";
          gd.data[1] = "bye";
  //      gd.data[2] = 10;
}
```

Reverse Echo Server using Socket

```
client                                              Server
                                     ServerSocket ss = new ServerSocket(2000)
Socket st= new socket (ip , 2000)    Socket st= ss.accept();
is                              ← os
os                              → is
keyb
```

```
import java.net.*;
import java.io.*;
 public class ReverseEcho {
     public static void main (String[] args) throws IOException {
       ServerSocket ss = new ServerSocket(2000);
       socket  Stk = SS.accept();
 BufferedReader br = new BufferedReader( new InputStreamReader(stk.getInputStr()));
 PrintStream ps =new PrintStream(Stk.getOutputStream());
   String msg;
   StringBuilder sb;
   do {
       msg =br.readLine();
       sb = new stringBuilder(msg);
       Sb.reverse()
       msg = Sb. toString();

 //       Ps.println(msg);
       } while (!msg.equals("dne"));
   }
 }
```

object for writting

object for displying

to send message to server

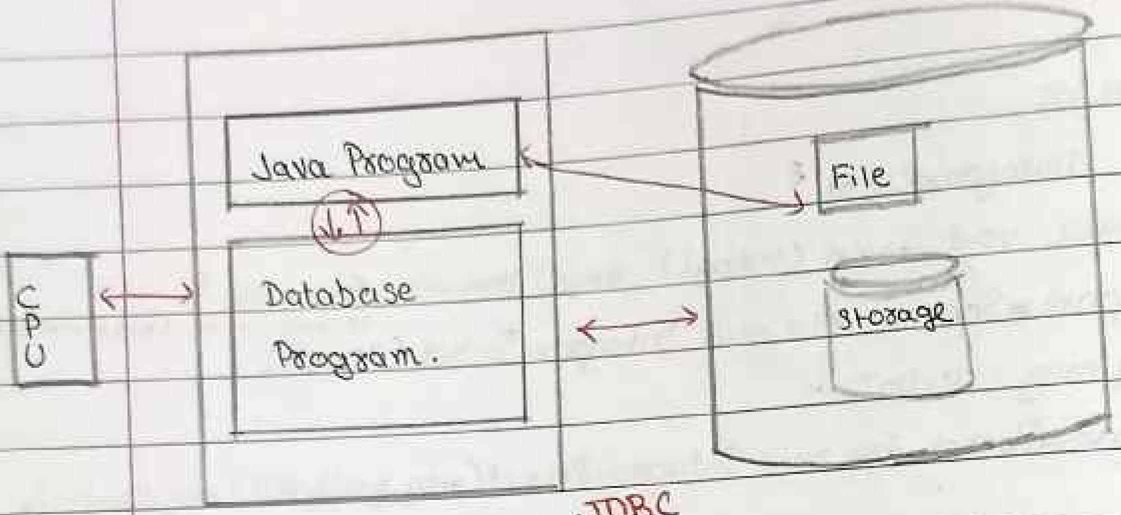
```
Class Client{
   public Static void main(String[] args) throws Except{
 socket Stk = new Socket ("localHost", 2000);
BufferedReader keyb= - -(new{ InputStrea (System.in)));
 Buffered Reader —
 PrintStream - - -
 String msg;
 do{ msg= keyb.readLine();
  ps.println(msg);
  msg= br.readLine();
  System.out.println("From server" + msg);
   } while(!msg.equals("dne"));
 Stk.close();
 }
}
```

Datagram Reverse Echo Server using Socket.

DatagramSocket
DatagramPacket

```
import java.net.io;
public class DatagramClient {
        public static void main (String[] args) throws Exception {
                                                  because we want to receive data.
           Datagram Socket ds = new DatagramSocket (2001);
           String msg = "Hello"; ✓
Sending  — DatagramPacket dp= new DatagramPacket(msg.getBytes(),msg.length(),
code
                                          InetAddress.getByName("localhost"),20
           ds. send (dp);
receiving  byte b[] = new byte[1024];
code       dp = new DatagramPacket(b,1024);
           ds.receive(dp);
getting    msg = new String(dp.getData()).trim();
           System.out.println ("From Server" + msg);
            ds.close();
        }
}
class Server {
         P.S.V main ( String[] args) throws Exception{
          Datagram Socket  ds= new  DatagramSocket (2000);
            receiving code
             getting
          StringBuilder  sb= new StringBuilder(msg);
          sb.reverse ();
           msg = sb.to String ();
             sending code
              ds.close ();
            }
       }
```

Databases



JDBC

1. Files vs Database –

↳ very difficult to organize largesize data.

- is a program which deals with storing & organizing data in permanent storage like HardDisk.

- It is collection of interrelated data may contain program for accessing. organised data in Tables.

3 famous Databases

- Oracle
- MySQL
- SQL server
- ✓ SQLite

Terminology

1. Relation :-
2. Schema :- Description of table (structure 1st row)
3. Field : :- Coloumn are Fields
4. Record (Tupple/row) :- a row
5. Primary Key :- Unique & NOT NULL Coloumn.
6. Relationship :-
6. Foreign key :-
7. constrainsts :

DDL	DML	Query	DCL
1. CREATE ✓	1. INSERT	Select	
2. DROP	2. DELETE	from	
3. ALTER	3. UPDATE	where.	
4. TRUNCATE			
5. RENAME			

Data Type of SQLite

NULL

INTEGER

REAL / FLOAT / NUMERIC. (5,2)

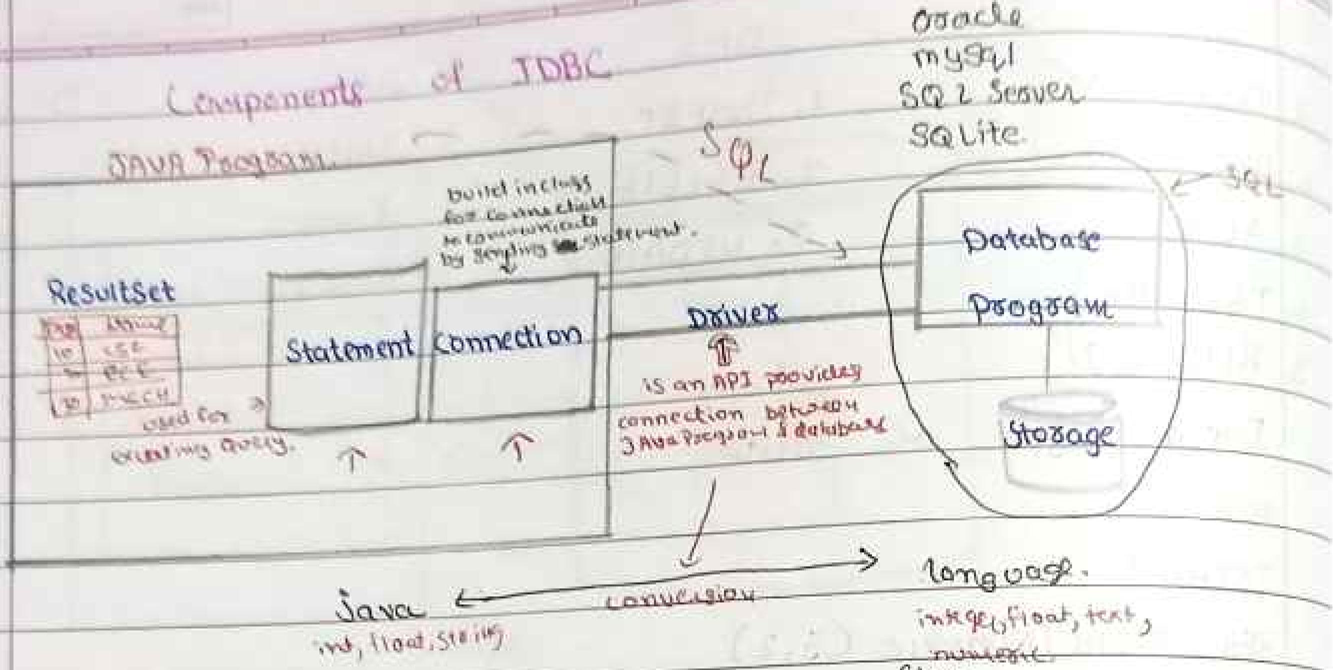
TEXT string

CHAR / VARCHAR

BLOB Binary Large Object. image video Audio

↳ formatless (Raw data)

Components of JDBC



Steps

1. Use Driver.

2. Establish Connection

3. Create Statement.

4. Execute Query.

5. Get Resultset.

1. Type1

JDBC-ODBC Bridge (partial)

2. Type-2

Native-API (partial)

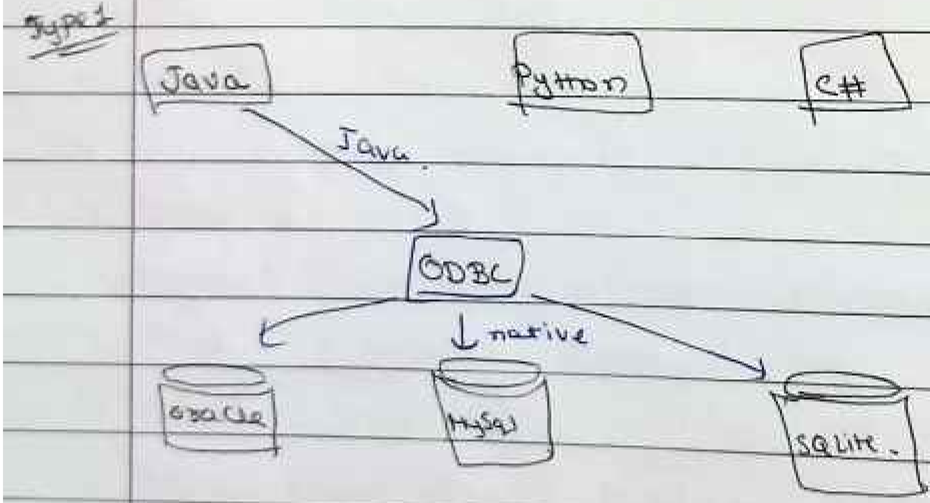
3. Type-3

Java-Net Protocol Driver. (pure)

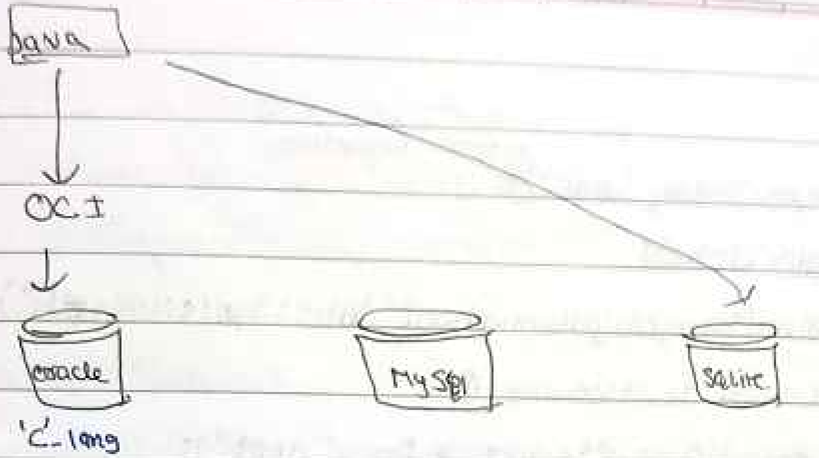
4. Type-4

Thin-Driver (pure)

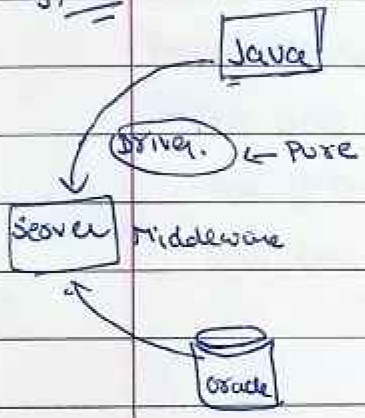
Type 1



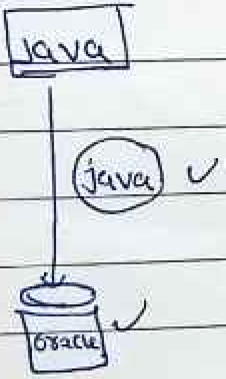
Type 2



Type 3



Type 4:-



```
import java.sql.*;
class Database {
    Public static void main(String arg[]) throws Exception.{
  // class.forName(org.sqlite.JDBC);
                                                            path
   Connection con = DriverManager.getConnection("jdbc:sqlite:univdb");
   Statement stm = con.createStatement();
   ResultSet rs = Stm.executeQuery("select * from dept");
int dno
String dname
/* while(rs.next()){
         dno = rs.getInt ("deptno");
         dname = rs.getString ("dname");.
         system.out.println(dno+" "+dname);
   } */
   stm.close();
   con.close();
}
```

okay

JAVA.sql interfaces

Statement

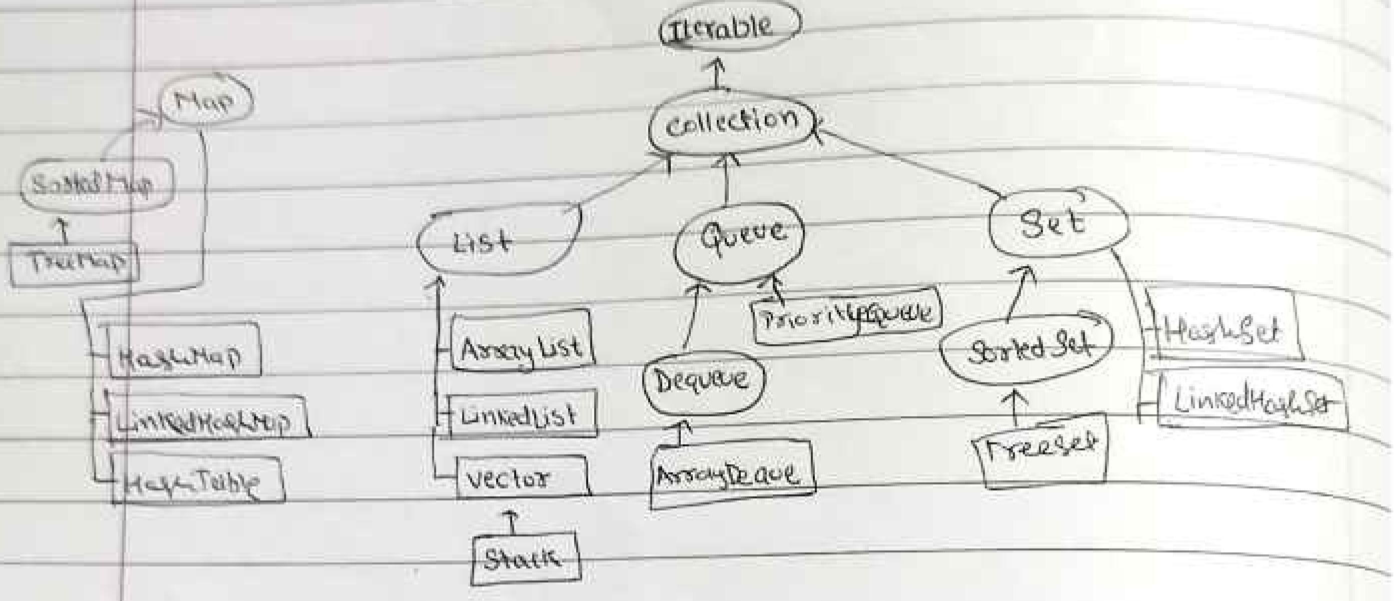
```
        Stm = con.createStatement();
    Stm.executeQuery ("select * from students");
    Stm.executeUpdate(" DML       ");
```

1. Statement
2. PreparedStatement
3. CallableStatement :- invoking stored procedure.

2. PreparedStatement

```
PreparedStatement  stm = con.prepareStatement("select * from Student where deptno=?
    int dno=20;
    stm.setInt(1,dno);
    stm.execute();/ Stm.executeQuery();
```

Collection Framework:



package java.util

- Collection
- List
- Set
- Queue

Collection Interface

add (E e)

addAll(collection <E> c)

remove (Object o)

removeAll (collection <E> c)

retainAll(collection <E> c)

clear( )

isEmpty( )

Contains (Object o)

ContainsAll (collection <E> e)

equals (Object o)

Size ( )

iterator ( )

toArray( )

List:-

interface List extends Collection.

add(int index, E e)

addAll(int index, Collection<E> e).

remove(int index).

get(int index)

set(int index, E e)

sublist(int from, int to).

indexOf(Object o).

lastIndexOf(Object o).

listIterator()

→ listIterator(int index).

Set:-

interface Set extends Collection.

Queue

interface Queue extends collection.

add(E e).

poll() null

remove() throws NoSuchElementException.

peek() null

element() throws NoSuchElementException