S. No. | Date | Title | Page No. | Teacher's Sign / Remarks

✱ Array Syntax →

```
int arr[] = new int[size];
    arr[0] = new int(); // object
```

✱ object array →

- Referenced →

```
Student S[] = new student[3]; // Ref. array
```

- object creation →

```
for (i=0 ; i<=2; i++)
{
    S[i] = new student(); // object associated to ref.
}
```

S. No. | Date | Title | Page No. | Teacher's Sign / Remarks

- local var memory allocated in stack.
- local var should initialized or have value otherwise we got error. (no garbage)

- Every constructor in Java have by default super method. So always constructor of parent will be called 1st.

```
super();
```

- super keyword represents parent class.

```
super.x;
```

JAVA

- by Tarun Sir

JAVA - CORE

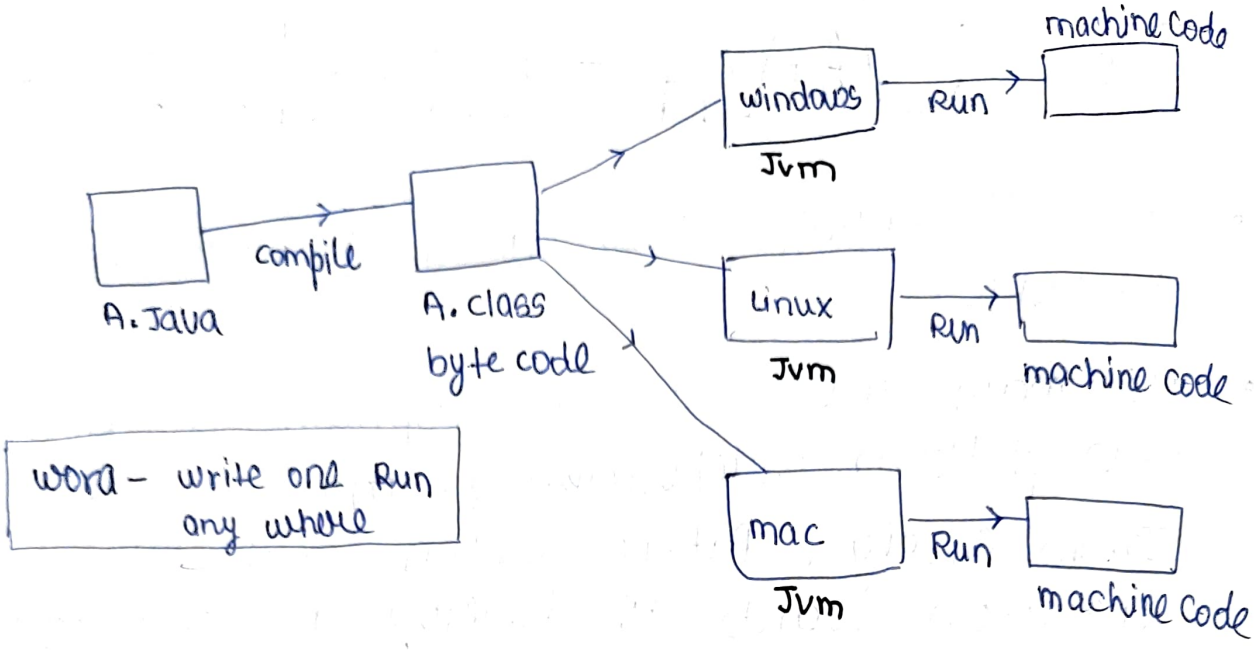
- Java is highly class based programming language.
- The main reason for development of Java was platform independency. that is the James Gosely, Patric Norton and nic sheraton wants to develop a language which can work on electronic appliances like set-up box.
- The first name (in starting) of Java was Green Talk and extension was .gt After that it becomes oak then finally its name changed to Java.
- The first official release of Java was in year 1995 (Jan)

features of Java / Buzz words →

1) Simple

Java is a simple language because it removes all the complex features of C and C++ like multiple inheritance, operator overloading & external pointers.

2] Platform Idependent -



Java is platform idependent because of Byte code i.e. when we compile our Java programe a '.class' file is created and this called Byte code and this Byte code can be read by 'JVM' installed on the machine.

3] secure -

Java is a secure language because every-time you execute programme a Byte code verifyer check the byte code. If Byte code is system generated it verify code and execute it otherwise Byte code verifyer throw a verifyer verifier error. Hence Java is secure language.

4] Robust (strong) —

Java is Robust language because it have strong memory management. In Java their is concept of garbage collector who execute automatically in background and free the memory occupied by unused object.

5] Multithreaded —

Java support multithreading that is we can execute a different parts of Java programme simultaneously and all this parts called threads.

6] Exceptional Handling —

Exceptions are unwanted events which disturbs the normal flow of programme. To Handle this exceptions their is a mechanism called exception Handling. Exception Handling handles the exception and maintain normal flow of programme.

7] Object-Oriented —

- Encapsulation (Access modifiers → Public, Private, default, protected)
- Polymorphism
  - ↳ compile time (Early binding)
  - ↳ Run time (Late binding)
- Inheritance
  - ↳ single level
  - ↳ multi level
  - ↳ Hirarichal
- Abstraction (Hiding Implementation)

8] Portable –

Java is portable ie. programe developed in one machine or OS can be port to the another machine or OS.

9] Architecture Nutral –

Java is a Architecture nutral demguage ie if we change in Architecture of the system it will not have any impact on the Java programme.

10] Dynamic –

Java is a dynamic language because classes are load dynamically.

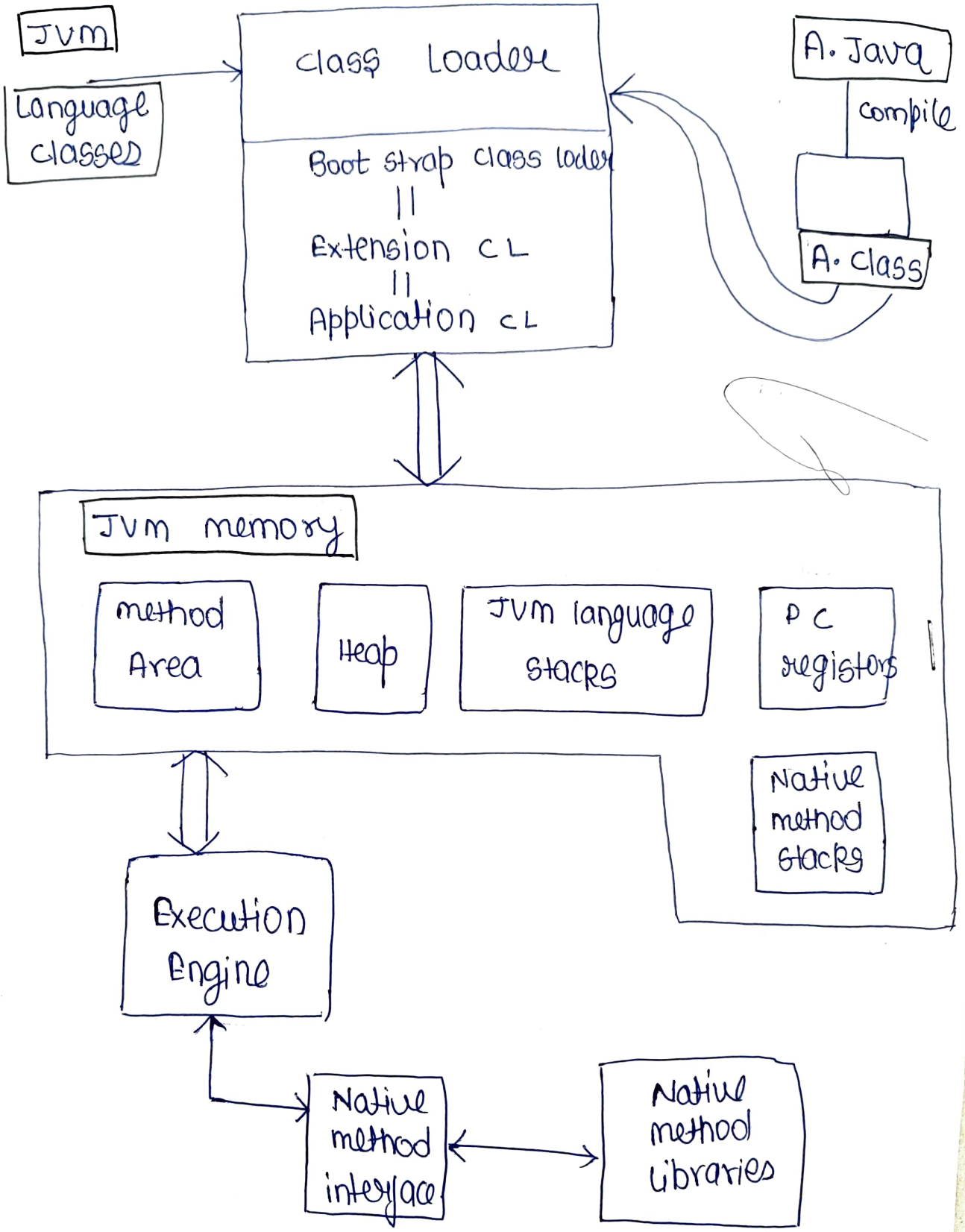
11] High Performance –

Java is high performance language because of 'Jit compiler' (JIT ⇒ Just in Time). whenever a function called multiple times a jit compiler store the converted code of that function into memory and whenever again its called it directly execute that function.

12] Distributive –

In Java we can develop a distributed applications (over the network)

JVM Architecture →



1) when we compile a .java file
file is generated

2) this .class file goes into various steps →

3) firstly, it goes to class loader sub system.

4) class loader mainly responsible for loading, linking, initialization.

- **loading** - The class reader the .class file and generate the binary data and save it in the method area. JVM stores following information in method area -
  - The fully qualified name of loaded class and its immediate parent class.
  - whether the .class file is related to class or interface or enum
  - name of variable their values and information about methods.

Linking - verification, preperation and resolution performed in the linking.

- verification - It insure that the corresponding .class file is properly generated by a valid compiler or not.

  If verification fail we get a run time exception - 'java.lang.verifyerror'

  This checking is done by Byte code verifier

1) When we compile a .Java file a .class file is generated

2) This .class file goes into various steps →

3) Firstly, it goes to class loader sub system.

4) class loader mainly responsible for loading, linking, initialization.

Loading - The class reader the .class file and generate the binary data and save it in the method area. JVM stores following information in method area -

- The fully qualified name of loaded class and its immediate parent class.
- whether the .class file is related to class or interface or enum
- name of variable their values and information about methods.

Linking - verification, prepération and resolution performed in the linking.

- verification - It insure that the corresponding .class file is properly generated by a valid compiler or not.

  If verification fail we get a run time exception - 'Java.lang.VerifyError'

  This checking is done by Byte code verifier

- preperation – JVM allocates fo memory for class static veriable and initializing the memory to dfault value.

- Resolution – It is the process of replacing symbolic reference from the type direct reference. It is done by searching into the method area to locate the referene entity.

Initialization – In this phase all the static variables are assigned with their values in code

JVM memory --→

5] Method Area – In this area all class level information like class name, immediate parent class name, method and variable information, infamation of class or enum or interface.
(Before Java 1.8 static var stored in method area now stored in heap)

6] Heap – Information of all objects and static var stored in heap memory area. String objects also stored in Heap.

7] Stack Area – For every thread JVM creates one run time stack which is stored here. Every block of this stack is called activation record or stack frame. All the local vars of method are stored in corresponding frame. After a thread terminates its run time stack will be destroyed by JVM.

8] PC Registors — It stores the address of curren-nt execution instruction of the thread. Each thread has seperate PC register.
(PC → Programme counter)

9] Native method Area — For every thread a seperate native stack is created. It store native information.

Execution Engine ---→

10] Execution Engine - It execute the .class file (ie byte code) It reads the byte code line by line. use the data and information present in various memory area and Execute the instruction. It has 3 Parts -

- Interpretor — It interprets the byte code line by line and than execute. It has a disadvantage that one method is called multiple times every time interpre-tation is required

- JIT (Just in Time) compiler — used to increase the efficiency of interpreter. The It compile entire byte code and change the it to the native code so whenever interpreater see reapeated method call JIT provide direct native code for that part. Rein

- Garbage collector — It destroy the unreferenced objects.

11] Native method Interface — It is an interface that interact with the native method libra-ries and provide native libraries required for execution.

12] Native method libraries — Its collection of native libraries (C, C++) are required by execution engine

MAIN - METHOD →

Syntax : Public static void main (String args[])

" class name always Block Start "

↳ public - because can be called from outside

↳ static - if a method declareed static because its not dependent on object - its dependent on the class

↳ void - Return type of the method

↳ main - main is name of method

↳ (String args[]) - String is class name and command line argument args[] is array as parameter can be x[] / a[] / etc. (name not matters)

" A programme can have n classes and programme should be saved with a class name which has command line argument "

- final - data type for declaring constant variable

'Hello world' Programme →

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

newline

- out is an object of class print stream
- System is a class

- Run and compile → (1st set Path)

** If class public then classname and filename should be same

compile → javac filename.java

Run → java filename

- File name → or classname in which main method present not for public class

File name is same as base class

or

```
class A {
    main ()
    }
```

Filename compile
Run by classname
not for public class

File name ⇒ A.java

USER - INPUT →

To take user input we have to import Scanner class of the util Package

syntax for import →

import java.util.Scanner ;

Scanner → class name capital

methods of Scanner class –

- ↳ nextInt()
- ↳ nextFloat()
- ↳ nextDouble()
- ↳ nextBoolean()
- ↳ nextLine()
- ↳ next().charAt(0)

} to take input diff data types

Float →
Float a = 3.14f ;

syntax for object creation –

classname (objectname) = new classname (—) ;

objectname → reference variable

Constructor

any command if req.

(next Page →)

WAP to Find Sum of 2 nos (Static)

```
class A
{
    public static void main (String args [])
    {
        System.out.println ("Hello");
        int a = 10, b = 20, c;
        c = a+b;
        System.out.println ("sum = "+(c));
    }
}
```

, not used
+ used in Java

[A.Java]

WAP to Find sum of 2 no's (Dynamic)

```
import java.util.Scanner;
class A
{
    public static void main (String args[])
    {
        int a;
        Scanner ip = new Scanner (System.in)
        system.out.println ("Enter a & b");
        a = ip.nextInt();
        b = ip.nextInt();
        c = a + b;
        system.out.println ("sum = "+c);
        ip.close();
    }
}
```

WAP to calculate Gross Salary →

```
import java.util.Scanner;
class Sal {
    double bs, ta, da, pf, gs;
    void set()
    {
        Scanner ip = new Scanner(System.in);
        System.out.println("Enter bs");
        bs = ip.nextDouble();
        ip.close();
    }
    void cal()
    {
        ta = (bs * 10) / 100;
        da = (bs * 20) / 100;
        pf = (bs * 20) / 100;
        gs = ta + da + bs - pf;
    }
    void show()
    {
        System.out.println("Gross sal = " + gs);
    }
    public static void main(String args[])
    {
        Sal ob = new Sal();
        ob.set();
        ob.cal();
        ob.show();
    }
}
```

WAP to convert $ ↔ ₹ (currency converter)

```
import java.util.Scanner ;

class C
{   int d, r ;

    void set ()
    {
        Scanner ip = new Scanner (System.in) ;
        System.out.println (" Enter amt in Rs") ;
        r = ip.nextInt () ;
        ip.close () ;
    }

    Void cal ()
    {
        d = r/80 ;
        System.out.println (" d = "+d ) ;
    }

    public static void main (String args[])
    {
        C obj = new C () ;
        obj.set () ;
        obj.cal () ;
        C.close () ;
    }
}
```

WAP to find perimeter of □

```
import java.util.Scanner;

class P{
  Double l, b, p;

  void set ()
  {
      Scanner ip = new Scanner (System.in);
      System.out.println ("Enter l & b");
      l = ip.nextDouble ();
      b = ip.nextDouble ();
      ip.close ();
  }

  void cal ()
  {
      p = 2*(l+b);
      system.out.println ("perimeter = " +p);
  }

  public static void main (String args[])
  {
      P ob = new P ();
      ob.set ();
      ob.cal ();
      ob.close ();
  }

}
```

WAP to convert °C ↔ °F (temp converter)

$C = (F - 32) 5/9$

```
import java.util.Scanner;
class GT {
    double C, F;
    void set()
    {
        Scanner ip = new Scanner(System.in);
        System.out.println("Enter temp in °F");
        F = ip.nextDouble();
        ip.close();
    }
    void cal()
    {
        C = (F - 32) * 5/9;
        System.out.println("temp in °C = " + C);
    }
    public static void main(String args[])
    {
        T ob = new T();
        ob.set();
        ob.cal();
        ob.close();
    }
}
```

Swapping logic using XOR operator

XOR → (^)

T	T	→
0	1	1
1	0	1
0	0	0
1	1	0

logic →

let $x = 5 \rightarrow 0101$

$y = 2 \rightarrow 0010$

$x \wedge y$ → 0111 (XOR)

→ (7)

swap →

$x = x \wedge y$

$y = x \wedge y$

$x = x \wedge y$

$x \rightarrow 0111$

$y \rightarrow 0010$

$y = 0101 \Rightarrow 5$

$x \rightarrow 0111$

$y \rightarrow 0101$

$x = 0010 \Rightarrow 2$

(Programme ⇒)

```
import java.util.scanner ;

class swap {
    int x , y;

    void set ()
    {
        Scamer ip = new scanner ( system.in);
        System. out.println (" Enter x & y ");
        x = ip. next Int ();
        y = ip. next Int ();

        ip. close ();
    }
    void swp ()
    {
        xc = x ^ y ;
        y = x ^ y ;
        x = x ^ y ;

        system. out. println (" swapped ");

    }
    Public static void main ( String args [])
    {
        Swap ob = new swap ();

        ob. set ();
        ob. swp ();

        ob. close ();

    }
}
```

WAP to calculate % of given marks → (using 2 files/classes)

```
import java.util.Scanner;
class Res{
    double p;
    void cal
    double cal (double h, double e, double m,
                double c, double s)
    {
        p = (h+e+m+c+s)/5;
        return p;
    }
}
```

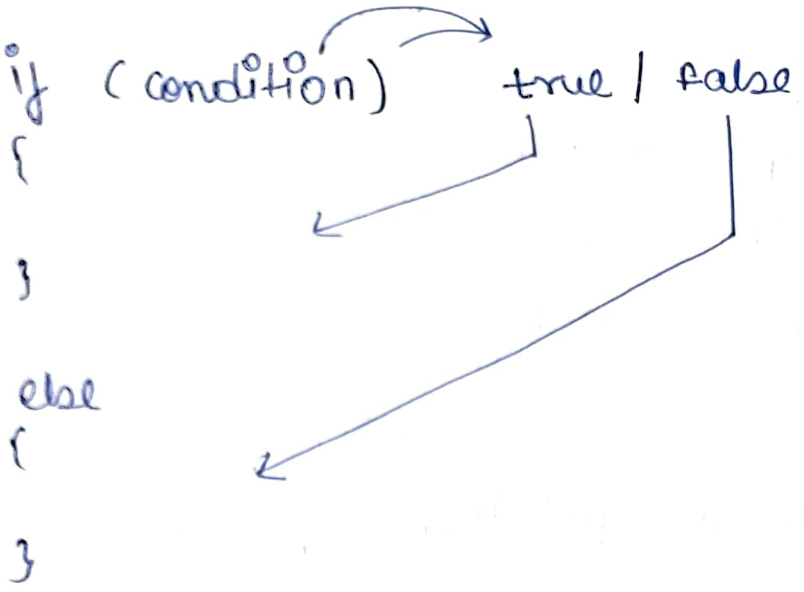
(Res.java)

Another file → from this file we will call above method cal which is saved in another file (~~#~~ (Res.java) (Student.java)

```
import java.util.Scanner;
class Student
{
    double h, e, m, s, c, per;
    void set()
    {
        Scanner ip = new Scanner(System.in);
        System.out.println("Enter marks of
                            h, e, m, s, c");
```

```
h = ip.nextDouble();
e = ip.nextDouble();
m = ip.nextDouble();
c = ip.nextDouble();
s = ip.nextDouble();
Res ob = new Res();          // call Res.Java
per = ob.cal(h, e, m, c, s);
System.out.println("percentage = "+ per);
}

public static void main (String args[]).
{
    Student ob = new Student();
    ob.set();
}

}
```

Conditional - Statements →



- Imp note on comparing strings →

eg → User = "admin"
U = "admin"

User == U will give **False**

∴ "==" operator also check reference (addd) in case of strings

So we use 'equals' method of string class to compare string.

Syntax → w.equals("username");

WAP to check username and password →

```
import java.util.Scanner;

class Login {
 String username = "admin";
 String password = "admin123";
 String u;
 String ps;

void set()
{
  Scanner ip = new Scanner(System.in);
  System.out.println("Enter uname & pass");
  u = ip.nextLine();
  ps = ip.nextLine();
  ip.close();
}

void checkLogin()
{
   set();
   if(u.equals(username) && ps.equals(password))
      {
         system.out.println("login success");
      }

   else
      {
         system.out.println("login fail");
      }
}
```

```
public static void main (String args[])
    {
        Login ob = new Login();
        ob.checkLogin();
    }
}
```

WAP to check vowel or consonant →
(2 classes in same file) Alphet.java

```
import java.util.Scanner;
class Vc
{
    char ch;
    void set()
    {
    Scanner ip = new Scanner(System.in);
    System.out.println("Enter char");
     ch = ip.next().charAt(0);
     ip.close();
    }
```

```
void check ()
{
  set ();
  if ((ch>='A' && ch<='Z') || (ch>='a'
          && ch<='z'))
  {
    if (ch=='A' || ch=='a' || ch=='E' ||
          ch=='e' || ch=='I' || ch=='i' ||
        ch=='O' || ch=='o' || ch=='U' ||
          ch=='u')
      {
       System.out.println ("vowel");
      }
    else {
       System.out.println ("consonant");
      }
  }
else if (ch>='0' && ch<='9')
   {
     System.out.println ("digit");
   }
else {
     System.out.println ("symbol");
    }
}
```

```
class Alpha
  {
    public static void main (String args[])
    {
      Vc obj= new Vc();
      Obj.check();
    }
  }
```

This - keyword →

- This keyword always points current class data.
- If local and class level variables have same name then compiler get confused to resolve this problem we use this keyword. This keyword always point to class level var.
- Syntax → 'this.varname'
- Same as this pointer in cpp

WAP to calculate Electricity Bill →

```
import java.util.Scanner;
class Bill
{ int u, r, billamt;
  void set (int u)
  {
    this.u = u;
  }
  void cal ()
  {
    if (u>0 && u<200)
    {
      r = 7;
    }
    else if (u >= 200 && u<500)
    {
      r = 12;
    }
    else
    {
      r = 20;
    }
  }

  public static voi

  void abc ()
  {
    billamt = u*r;
    System.out.println (billamt);
  }
```

```
public static void main (String args[])
{
    int unit;
    Scanner ip = new Scanner (System.in);
    unit = ip.nextInt();
    Bill ob = new Bill();
    ob.set(unit);
    ob.cal();
    ob.abc();
}
```

WAP to check leap year →

logic →

year %400 == 0 || year %100 != 0 .

&& year %4 == 0

Switch - Case →

Syntax is same as in C/C++

```
Switch (choice)
{
    case 1 :
    //code
    break;

    case 2 :
    // code
    break;

    case 3 :
    // code
    break;

    .
    .
    .
    .
    .

    default :
    // code
    break;
}
```

WAP for banking management -

```
import java.util.Scanner;

class Bank
{
    int amt, bal = 5000, ch;
    String user = "admin";
    String pass = "admin123";
    int flag = 0;
    Scanner ip = new Scanner (system.in);
    void deposit ()
    {
      System.out.println ("Deposit amt =" + amt);
      bal = bal + amt;
      System.out.println ("Total bal = " + bal);
    else {
      System.out.println ("Invalid amt");
    }
}

void withdrawal ()
{
    System.out.println ("Enter amt to withdraw");
    amt = ip.nextInt ();
    if (amt > 0 && amt <= bal)
    {
        System.out.println ("with amt = " + amt);
        bal = bal - amt;
    }
```

```
    else {
        System.out.println("Invalid amt");
    }
}
void showbalance()
{   System.out.println("Total bal = " + bal);
}
int login(String u, String p)
{   if(u.equals(user) && p.equals(pass))
    {   flag = 1;
    }
    else {
        flag = 0;
    }
    return flag;
}

Void menu()
{
    String u;
    String p;
    System.out.println("Enter username");
    u = ip.nextLine();
    System.out.println("Enter password");
    p = ip.nextLine();
    int f = login(u, p);
```

```
if (j==1)
{
    System.out.println("Press 1 for deposit");
    System.out.println("Press 2 for withdrawal");
    System.out.println("Press 3 for show balance");
    ch = ip.nextInt();
    switch (ch)
    {
      case 1 :
        deposit();
        break;

      case 2 :
        withdrawal();
        break;

      case 3 :
        ShowBalance();
        break;

      default :
        system.out.println("Invalid choice");
    }
}
else {
      system.out.println("Invalid choice");
    }
}
```

```
public static void main (String args[])
  {
    Bank ob = new Bank();
     ob.menu();
   }
}
```

WAP to cacluate max of 2 no's using switch case →

```
import java.util.Scanner;
class mx2
{   int a, b, ch;
    Void Set()
    {
        Scanner ip = new Scanner(System.in);
        System.out.println("Enter a & b");
        a = ip.nextInt();
        b = ip.nextInt();
        ip.close();
    }
    Void cal()
    {
        ch = a>b ? 1:0;
        Switch (ch)
        [
            case 1:
            System.out.println("a is max");
            break;
```

```
case 0:
{
    ch = a==b ? 1 : 0;
    Switch (ch)
    {
        case 1:
        system.out.println ("a is eqlual to b");
        break;

        case 2:
        system.out.println (" b is greater ");
        break;
    }
}     break;

public static void main (String args[])
{
    mx2 ob = new mx2();
    ob.set();
    ob.cal();
}

}
```

Constructor →

- Constructor is a special function who have same name as of class.
- constructor is automatically called when object is created.
- constructor is used to set default values.
- constructor don't have any return type.
- constructor should always declared as Public.
- Types of constructor →

① Default contructor →

```
X ( )
{
}
```

② Paramatrice constructor →

```
X (int a, int b)
{

}
```

③ Copy Constructor →

```
X (x ob2)
{

}
```

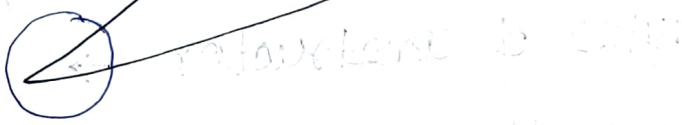
Loop →

Repeatetive statements there are 2 types of loops →

- Exit controlling – Do while
- Entry controlling – for , while

- For loop –

```
for ( intialisation ; condition ; increament/dec)
{

}
```



- while loop –

```
while ( condition)
{
    increament / dec ;
}
```

- Do - while loop –

```
Do {

}
while (condition);
```

WAP to calculate sum of series →

```
import java.util.Scanner;

class Sm
{ int no, i, sum;
   public Sm()
   { sum = 0;
     no = 5;
   }
  void add()
  {
     for (i = 1; i<=no; i++)
       {
         sum = sum + i;
       }
       System.out.println("sum = "+ sum);
  }
  public static void main (String args[])
   {
       new Sm().add();
   }
}
```

another way to create object

WAP for Factorial using Parameterized constructor

```
class Ps
{   int f, no, i; fact;
    public Ps (int no)
    {
        fct=1;
        this.no = no;
    }
    void fact ()
    {
        for (i=1; i<=no; i++)
        {
            fact = fact * i;
        }
    System.out.println ("Factorial = " + fact)
}

Public static void main (String args[])
{
    Ps ob = new Ps (no7);
}
}
```

WAP for copy constructor →

copy constructor copies value of object Example →

```
class cpy
{   int a;
    cpy(int a)      // Paramatrised constructor
    {  this.a = a;
    }

    cpy ( cpy ob)  // copy constructor
    {
      this.a = ob.a;
    }
    Void show ()
    {
      System.out.println ("value of a = " + a);
    }

    public static void main ( String args[])
    {
       cpy ob = new cpy (5);
       ob. show();
       cpy obj = new cpy (ob);
       obj. show();
    }
}
```

this programme also shows constructor overloading

WAP for Fibonacci series →

0, 1, 1, 2, 3, 5, 8 - - - - -

```
import java.util.Scanner;
class Feb
{
  int f1, f2, f, i, not ;
  Scanner ip ;  // reference var
  public Feb()
  {
    ip = new Scanner(System.in);
    f1 = -1;
    f2 = 1;
  }

  void cal()
  {
      not = ip.nextInt();
    System.out.println("Enter no of terms");
    for(i = 1; i <= not ; i++)
      {  f = f1 + f2;
         System.out.print(f + ", ");
         f1 = f2;
         f2 = f;
      }
}
```

```
public static void main ( String args[])
{
    Reb ob = new Reb();
    ob.cal();
}
}
```

Command line argument →

The java command line argument is an argument passed at the time of running of programme.

The argument passed as console can be recived in java programme and used as the input →

```
class cma
{
   public static void main (String args[])
      {
        System.out.println (args[0]);
      }
}
```

- while running → (How to pass it)

compile - javac cma.java

Run - java cma argument

eg → java cma Rahul

Example 2 →

```
class cma
{
  public static void main (String arg[])
  {
    for (int i=0 ; i<arg.length-1; i++)
      {
      System.out.println (arg[i]);
      }
  }
}
```

WAP to print sum of all command line arguments-

```
class cma
{ public static void main (String args[])
    int sum = 0;
  for (int i=0; i<arg.length-1; i++)
  {
    sum = sum + Integer.parseInt (arg[i]);
  }
 System.out.println (sum);
}
```

- Type casting / Parse method →

used convert valid string to int →

Syntax → Integer.parseInt (String)

Polymorphism →

Polymorphism means creating different forms or different behaviors. There are 2 types of polymorphism —

- Compile time (overloading) / static binding
- Runtime (overriding) / Dynamic binding

Method Overloading →

When we have 2 or more than 2 methods with same name but different parameters is called function or method overloading.

Example - WAP to overload main method →

```
class MO
{
    public static void main (int a)
    {
        System.out.println ("value of a = "+a);
    }
```

//PTO

```
public static void main (String args[])
{
    main (5);   // can't call by object
                :. declared as static

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

output → value of a = 5
Hello

WAP to overload area method →

```
class Ar
{
    double a ;
    Final double pi = 3.14;   // final means const
    Void area (double l, double b)
    {
        a = l * b;
        System.out.println ("area of □ = "+a);
    }
    Void area (double r)
    {   a = pi * r * r;
        System.out.println ("area of O = "+a);
    }
```

```
public static void main ( String args[])
{
    Ar ob = new Ar();
    ob.area(5);  ⟶ area of ○
    ob.area(2, 3.14); ⟶ area of □
}
```

WAP for hotel management 'menu'

bill → bill amount
q → quantity
r → rate

```
import java.util.Scanner;
class menu
{
    Int bill, q, r;
    Void order()
    {
        q = 1;
        r = 250;
        bill = r*q;
        System.out.println("You order a one simple thali and bill amount = "+ bill);
    }
```

```
void order (int q)
{
    r = 250;
    bill = r * q;
    System.out.println (" You order " + q +
        " simple thali and bill amt = " + bill);
}

void order (String dishname, int q)
{
    if (dishname.equals ("pizza"))
    {
        r = 550;
        bill = r * q;
        System.out.println (" you ordered "
        + q + " " + dishname + " and bill
        amount = " + bill);
    }
    else if (dishname.equals ("burger"))
    {
        r = 250;
        bill = r * q;
        System.out.println (" you order
        a " + q + " " + dishname + " and
        bill amount = " + bill);
    }
}
```

```
public static void main (String args[])
{  int ch, q;
   menu ob = new menu();
   Scanner ip = new Scanner(System.in);
   System.out.println("press 1 for one simple
                       thali");
   System.out.println("press 2 for two or more
                       simple thali");
   System.out.println("press 3 for other dish");
   ch = ip.nextInt();
   switch (ch)
{
   case 1:
      ob.order();
      break;
   case 2:
   System.out.println("Enter no. of thali
                       you want");
   q = ip.nextInt();
   ob.order(q);
   break;
```

```
case 3 :
System.out.println ("Press 1 for pizza");
System.out.println ("Press 2 for burger");
ch = ip.nextInt();
Switch (ch)
{
    case 1 :
    System.out.println ("Enter quantity");
    q = ip.nextInt();
    ob.order ("pizza", q);
    break;

    case 2 :
    System.out.println ("Enter quantity");
    q = ip.nextInt();
    ob.order ("burger", q);
    break;

    default :
    System.out.println ("invalid choice");

  }
  break;

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

Variable arguments → (VAR ARG)

The variable arguments allow the methods to accept zero or multiple arguments. Before var args either we use overloaded methods or take an array as a method parameter.

The following is syntax for the same →

Syntax → void methodname (int ... a)
{

}

for-each loop →

for (int x : (s)) ↳ array
{

}

ex → int ar[] = { 10, 20, 60, 80, 90};

for(i=0; i<len; i++) ⇐ for (int element : ar)
{
element = ar[i] ⇐ sop (element + " ");
}

o/p → 10 20 60 80 90

Example → for int →

```
class varg
{
    int sum = 0;
    void add (int... s)  // multiparameter
    {
        for (int x : s)
        {
            sum = sum + x;
        }
        System.out.println (sum);
        sum = 0;
    }
    public static void main (String args [])
    {
        varg ob = new varg ();
        ob.add (1);
        ob.add (2,3);
        ob.add (1, 2, 3, 4, 5);
    }
}
```

output →

1
5
15

Example → for strings →

```
class vareg
{  void show ( string ... s)
   {
      System. out. println ("* * * * * * *");
      for ( string x : s )
      {
         system . out. println (x);
      }
   }
   public static void main (string args[])
   {
      vareg ob = new vareg ();
      ob. show( "Rahul" );
      ob. show ("monu", "mehul", "Sumit");
   }
}
```

output →

```
* * * * * * *
Rahul
* * * * * * *
monu
mehul
Sumit
```

• Rules for variable Arguments →

↳ There should be only one vare arg in each method call.

```
Void ( int...a, String...b)
        X not valid
```

↳ vare arg is last argument of the method.

```
void (int a, int...b); ✓
Void (int...a, intb); X not valid
```

WAP for factorial using var arg →

```
class varg
{
  int fact = 1;
  Void facto (int...i)
  {
    for (int x : i)
    {
      fact = fact * x;
    }
    System.out.println (fact);
    fact = 1;
  }
```

```
public static void main (String args[])
{
    Varg ob = new Varg();
    ob.facto(2,3);
    ob.facto(1);
    ob.facto(1,2,3,4,5);
}
}
```

INHERITANCE → Inheritance shows IS-A relationship

↳ One class can access the property of the another class is called Inheritance.

↳ In Java we use 'extends' keyword to perform inheritance.

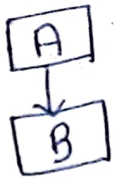
↳ Java supports only single level, multilevel and Hirarichal inheritance.

↳ Java does not support multiple and Hybrid inheritance.

↳ we can achive multiple inheritance by implementing multiple interfaces

↳ There are 2 types of class parent / base / super classes and child / sub / derive classes

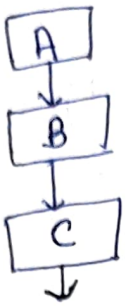
↳ single level →



singly level

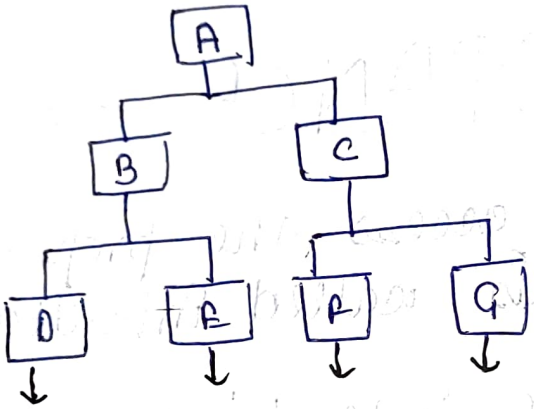
" object will be of child class "

↳ multi level →



more than one level

↳ Hiranchal →



tree

ARRAY →

- Array is collection of similar data which works on index system.
- Array index starts from 0 and start from end at n-1
- Size is fixed but java provides ArrayList to create dynamic array

↳ The biggest advantage of array is that in single var we can store multiple variables because of that readability increase.

Syntax –

```
int a [] = new int [5];
                  ↓
              size of array
```

Example →

```
import java.util.Scanner;

class Ar
{
  int a[] = new int[5];
  int i;
  Scanner ip;

  Ar()
  {
    ip = new Scanner( System.in );
  }

  void set ()
  {
      for ( i=0 ; i< a.length ; i++ )
        {
          a[i] = ip.nextInt();
        }
    }
  }
```

```
class Br extends Ar
{
    set();  void show() {
    for ( int i=0;  i<a.length ; i++)
        {
          system.out.println(a[i]);
        }
}   }

public static void main(String args[])
{
    Br ob = new Br();
    ob.show();
}
}
```

RANDOM - METHOD → java.util.random / java.lang.math

1) math.random –

Random method return a double value with positive sign greater than or equal to 0.0 and less than 1.0 (by default)

WAP to print random no. acc to the system –

```
import java.lang.Math;    // import this package

class Rn
{
   public static void main (String args[])
     { System.out.println (Math.random());
     }
};
```

- Formulla to generate Random no of our choice →

minimum + (int) Math.random() * ((max-min)+1))

WAP which prints random no's using our choice

```
import java.lang.Math;

class Rn
  {
```

```
public static void main (String args[])
{
    int min = 1000;
    int max = 99999;

    int otp = (min + (int)(math.random()*
                          ((max - min) + 1)));

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

WAP which takes user input in a guessing no and matching with system no's →

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

class An 1
{
    int min = 1 , max = 10;
    int SetNo()
    {
        int otp = (min + (int)(math.random()*
                              ((max - min) + 1)));

        return otp;
    }
}
```

```
class Rn extends Rn1
{  int no, otp1, count=3;
   void check()
   {
      otp1 = cho setno();
      Scanner ip = new Scanner(System.in);
      for (int i=1; i<=3; i++)
         {
          system.out.println("Enter no");
          no = ip.nextInt();
          if (no > otp1)
           {
             System.out.println("you enter large no");
           }
          else if (no == otp1)
          {
            system.out.println("now you are correct");
            break;
          }
          else {
            system.out.println(--count + "chances left");

   }
}
```

```
    public static void main (String args[])
    {
        Rn ob = new Rn();
        ob.check();
    }
}
```

WAP which takes user input of guessing a no with help of making object of Random class

```
import java.util.Scanner;
import java.lang.Random;
                util
class Rn1
{
    int otp;
    int setno()
    {
        Random rn = new Random();
        otp = rn.nextInt(100);
        return otp;
    }
}

class Rn extends Rn1
{
    int no, otp1, count;

    Void check()
    {
```

```
otp1 = get();
Scanner ip = new Scanner (System.in);
for (int i=1; i<=3; i++)
{
    System.out.println ("Enter the no");
    no = ip.nextInt();
    if (no > otp1)
    {
        System.out.println ("you enter a large no");
    }
    else if (no == otp1)
    {
        System.out.println ("wow! u r correct");
        break;
    }
    else {
    System.out.println ("you enter small no");
    }
System.out.println (--count + "chances left");
}
}
```

```
public static void main (String args[])
  {
     Rn ob = new Rn();
     ob.check();
  }
}
```

2) java.util.concurrent.ThreadLocalRandom;

↳ This is introduced in java 1.7 to generate random numbers of type integers, doubles.

↳ Its a static class

↳ It has nextInt(), nextDouble(), nextBoolean methods in it.

8 # WAP which takes user input of guessing no. using the above class -

```
import java.util.Scanner;
import java.util.Random;
Import java.util.concurrent.ThreadLocalRandom;

class Rn1
{
   int otp;

   int setNo()
   {
      otp = ThreadLocalRandom.current().nextInt(10);
      return otp;
   }
```

// Rest code is same as last

SUPER keyword and Method Overriding →

super keyword is used to call super class method, constructor and variables.

WAP of method overriding using super keyword -

```
class mor
 {   void show ()
     {  system.out.println ("Super");
     }
 }

class sp extends mor
 {      @override    // Annotation
        void show()
     {  super.show ()   // Here super is used to call super class method
        System.out.println ("Sub");
     }
     public static void main (String args[])
        {  sp ob = new sp ();
           ob.show ();
        }
 }
```

- @Override →

The @Override annotation is one of a default Java annotation and it can be introduced in Java 1.5 version. The @override annotion indicates that the child class method is overriding its base class method.

The @Override annotation can be useful for 2 reasons -

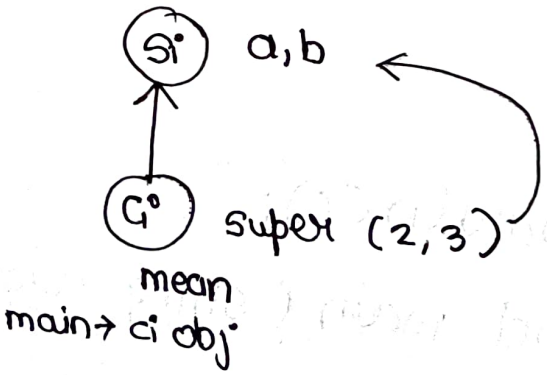
↳ It extracts a warning from the compiler if the annoted method dosn't actually override anything.

↳ It can improve the readablity of source code.

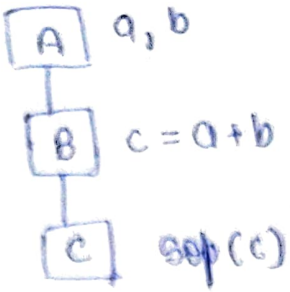
WAP for single level inheritance → ⭐⭐

```
import java.util.Scanner;
class si{
    int a,b;
    si (int a, int b)
    { this.a= a;
      this.b = b;
    }
}
```

```
class C1 extends S1
{ int c ;
  C1 ()
  {
    super (2,3) ;  // will call constructor of super class
  }
  void mean()
  {
    c = (a+b)/2 ;
    System.out.println (c) ;
  }
  public static void main (String args[])
  {
      C1 ob = new C1();
      ob.mean() ;
  }
}
```



WAP for multi level inheritance -




```
import java.util.Scanner;
class multi {
    int a, b;
    multi(int a, int b)
    {
        this.a = a;
        this.b = b;
    }
}
class Inh extends multi {
    void call() {
        int c;
        c = a + b;
    } System.out.println(c);
    public static void main(String args[])
    {
        Inh ob = new Inh(2,5);
        ob.call();
    }
}
```

final Keyword →

- If a class declareed final with final keyword it can't be Inheriited →

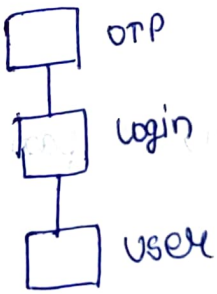
  eg → final class A { }

- If a method declareed final with final keyword it can't be overrided.

- If a variable declareed final with final keyword it becomes constant.

  eg → final int pi = 3;

WAP for OTP generation and Login →

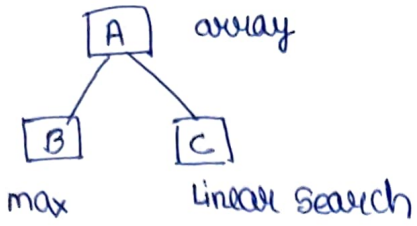



```
import java.util.Random;
import java.util.Scanner;

class otpgen
 { int otp;
    Random rn = new Random();
    int otpgenerate()
     {
       otp = rn.nextInt(9999);
       return otp;
     }
 }
```

```
class Login extends otpgen
{
    Scanner ip = new Scanner (System.in);
    String user = "admin";
    String pass = "admin123";
    int flag = 0;
    int o;
    int log (String u, String p)
    {
        System.out.println ("your otp = "+otp);
        System.out.println ("Enter otp");
        o = ip.nextInt();
        if (o == otp)
        {
            if (user.equals (u) && pass.equals (p))
            {
                flag = 1;
            }
            else {
                flag = 0;
            }
            return flag;
        }
        else {
            System.out.println ("invalid otp");
            return flag;
        }
    }
}
```

```
class user extends login
{
    String u1;
    String p1;
    int k2;
    void set ()
    { System.out.println ("Enter username");
      u1 = ip.nextLine ();
      System.out.println ("Enter password");
       p1= ip.nextLine ();
    }
    void checklogin ()
       { set();
         l3 = log (u1, p1);
         if (l3==1)
           { System.out:println ("login");
           }
         else { system.out.println ("login fail");
           }
    }
    public static void main (String args ())
       { user ob = new user ();
         ob.checklogin ();
    }
}
```

WAP for Hirarichal inheritance →

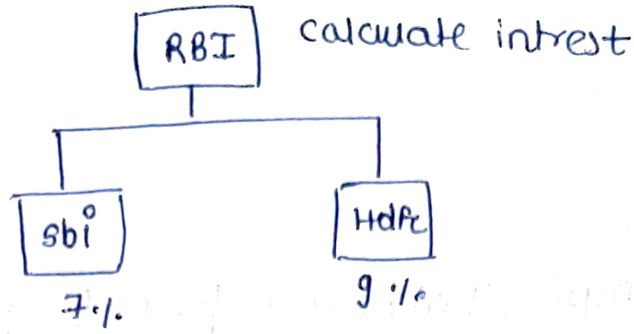



```
import java.util.Scanner;
class Ar
{
    int a[] = new int[5];
    int i ;
    Scanner ip;
    public Ar()
     {
        ip = new Scanner(System.in);
     }

    Void set()
     {
       System.out.println("Enter array elements");
       for(i=0; i<a.length; i++)
            {
                a[i] = ip.nextInt();
            }
     }
}
```

```
class mx extends Ar
{   int max;
    void maximum()
    {
        set();
        max = a[0];
        for(i=1; i<a.length; i++)
        {   if(a[i]>max)
            {
                max = a[i];
            }
        }
        system.out.println("max element = "+max);
    }
class search extends Ar
{   int no;
    void srch()
    {
        set();
        system.out.println("Enter the no");
        no = ip.nextInt();
        for(i=0; i<a.length; i++)
        {
```

```
        if(a[i] == no)
        {
            System.out.println("no. found");
            break;
        }
    }
    if (no > a.length-1)
    {
        System.out.println("no. not found");
    }
}
}

public class Hie {

    public static void main(String args[])
    {
        Search ob = new Search();
        ob.Srch();

        Mx obj = new Mx();
        obj.maximum();
    }
}
```

WAP for Hirarichal inheritance →




```
import java.util.Scanner;
class Rbi
{
   double Si;
   Scanner ip = new Scanner (System.in);
   double simple_Intrest (double p, double r, double t)
   {
      Si = (p*r*t)/100;
      return Si;
   }
}
class Sbi extends Rbi
  {
      double r, p, t, s;
      void set()
      {
         r = 7;
         System.out.println("Enter p & t");
         p = ip.nextDouble();
```

```
        t = ip.nextDouble();
    }
    void cal()
    {
        set();
        s = simple_Intrest(p, r, t);
        System.out.println(s);
    }

class Hdfc extends Sbi
{
    double r, p, t, s;
    void set()
    {
        r = 9;
        System.out.println("Enter p & t");
        p = ip.nextDouble();
        t = ip.nextDouble();
    }
    void cal()
    {
        set();
        s = simple_intrest(p, r, t);
        System.out.println(s);
    }
}
```

```
public class Bank {
    public static void main (String[]args())
    {
      Sbi ob = new Sbi ();
       ob.cal();
      Hdfc obj = new Hdfc();
       obj.cal();
    }
}
```

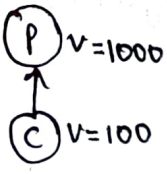
Runtime- Polymorphism →

The parent class can hold reference to both the parent and child object. If a parent class variable holds reference of child class and the value is present in both the classes in general the reference belongs to the parent class variable this is due to the run time polymorphism characteristic in Java

WAP to demonstrate run time polymorphism -

```
class Parent {
        int value = 1000;
    }

class Child extends Parent{
        {
          int value = 10;
    }
```

```
class GFG {

    public static void main (String args[])
    {
        child obj = new child();
        System.out.println ("Ref of child = " + obj.value);
                                                  type
        parent par = obj;
        System.out.println ("Reference of parent
                                  type : " + par.value);
    }
}
```



GFG
C obj
P ob = obj

O/P
10
1000

ABSTRACTION →

Hiding the details is called abstraction or hiding the (levels of) implementation.
In java abstraction can be achieved by →

1) By creating abstract class
2) By creating interface

- ## ABSTRACT CLASS →

A class which is declareed as abstract keyword is known as abstract class

It can have abstract and non abstract method or both. Abstract class is necessary to extend (inherit)

↳ Abstract class must be declared with abstract keyword

↳ It can have both abstract or non abstract methods

↳ Abstract class can have constructor and static method also

↳ The non abstract method of abstract class can be declare final

↳ we can also declare variable in abstract class

↳ Abstract method can't be declared final

**

↳ If a class inheriting the abstract class then it is cupulsory for child class to override the abstract method of abstract class and there child class have to implement the abstract method

WAP of abstract class with single abstract method

```
import java.util.Scanner;
abstract class Abs
{
  abstract void prime (int no);
}
class Prm extends Abs
  {  int i;
     @override
```

```
void prime (int no)
{
    for (i=2; i<= no ; i++)
    {
        if (no%.i == 0)
        {
            break;
        }
    }
    if (no == i)
    {
        System.out.println ("prime");
    }
    else {
        System.out.println ("not prime");
    }
}
public static void main (String args[])
{
    Prm ob = new Prm ();
    int no ;
    Scanner ip = new Scanner (System.in);
    System.out.println ("Enter a no");
    no = ip.nextInt ();
    ob.prime (no);
}
}
```

Abstract class reference var can hold the object of child class

eg → If abs is an abstract class and prm is a child class than →

```
abs ob = new Prm();
```

abs → reference var

Prm → object

WAP which has abstract class and have a concreate and a abstract methods →

```
import java.util.Scanner;

abstract class Abs1
{
    int no1, no2, h;
    void set()  ⟵ concrete method of abstract class
    {
        Scanner ip = new Scanner(System.in);
        System.out.println("Enter the no → ");
        no1 = ip.nextInt();
        no2 = ip.nextInt();
    }
    ct

    abstract void ncf(); ⟶ abstract method of abstract class

}
```

```
class Hc extends Abs 1
{
    int min i ;
    @override
    void hcf ()
    {
       min = no1 < no2 & no1 : no2 ;
      for (no1 % i == 0 && no2 % i == 0)
          {
             h = i ;
          }
       }
        system. out. println ("Hcf = "+h);
    }
   public static void main (String args[])
        {  Cbi ob = new Cbi
Ref. var  Abs 1 ob = new Hc () ;
                                    object
            ob. set();
            ob. hcf();
        }
    }
```

#WAP to create constructor of abstract class →

```
abstract class cons
 [
      public cons ()
     [
       system. out. println ("welcome");
      }
}
```

```
class chi extends cons
{
    public static void main (String args[])
    {
        chi ob = new chi();
    }
}
```

WAP which has static class method in abstract class

```
abstract class cons
{
    static void show()
    {
        System.out.println("welcome");
    }
}
class chi extends cons
{
    public static void main (String args[])
    {
        cons.show()          or  // chi.show()
    }
}
```

//static method called without object using class name

we can call by this class also

WAP for creating abstract class variable →

```
abstract class cons
{
    int i=5;
}
class chi extends cons
{
```

```
void show()
{
  System.out.println(i);
}
public static void main (String args[])
  {  chi ob = new chi();
     ob.show();
  }
}
```

ANONYMOUS CLASS IMPLEMENTING ABSTRACT CLASS →

A class without a name is called anonymous class. The class is associate with the object of super class

example –

```
abstract class cons
{  int i = 5;
   abstract void show();
}
class chi e
{  public static void main (String args[])
   {  cons ob = new cons() ;
      {
         @override
         void show()
         {
          System.out.println ("value of i = "+i);
         }
      };
      ob.show();
   }
}
```

anonymous class

WAP to count no. of digits using annonymous class concept.

```
import java.util.Scanner;

abstract class Abse
{
    Scanner ip = new Scanner (System.in);
    int no;
    void set()
    {
        system.out.println ("Enter no");
        no = ip.nextInt();
    }
    abstract void digitcount ();
}
class Digit
{
    public static void main ( String args[])
    {
        int count = 0;
        Abse ob = new Abse()
        {
            void digitcount ()
            {
            while (no>0)
            {
                count ++;
                no = no/10;
            }
            system.out.println ("no of digit = " + count);

            }
        };
        ob.set();
        ob.digitcount();
    }
}
```

INTERFACE →

↳ An Interface in Java is a blueprint of class.

↳ The Interface in Java is a mechanism to achive abstraction.

↳ Java Interface represent 'is-a' relationship.

↳ Interface is defined by 'interface' keyword

↳ To implement (~~interi~~) a interface we use 'implement' keyword.

↳ one class can implement multiple interface.

↳ one interface can Inherit other interface

- Reasons for creating a Interface →

① To achive multiple inheritance

② To achive abstraction

- Some imp points regarding interface →

↳ Before Java 1.8 only abstract method is there in interface from 1.8 verson Java added the static and dfault method in interface.

↳ After Java 1.9 private method concept also added in interface

↳ The variable declare in Interface is always public static final type.

↳ An abstract method of interface is always public (public abstract void show())

- Types of Interface →

1] Normal Interface - An Interface with multiple abstract and concrete methods is called normal Interface.

2] Functional Interface - An Interface with only single abstract method is called functional interface (SAM). This Interface is denoted by @functional interface annotation.

3] marker Interface - It's an empty interface. eg → serializable clonable.

IMP-NOTE : Private method of interface is always called from default method of interface.

- Syntax of Interface →

```
interface Abs
{
    void show();   // by default public abstract void show();
    int a = 5;     // by default public static final int a = 5;

    default void show() {
    }
    static void show() {
    }
    private void show() {
    }
}
```

- implementation and inheritance syntax →

```
interface i.
 {
 }
 interface j
{
}
class A implements i, j
 {

 }
```

- Example for interface →

```
@ functional Interface
interface i
{
  public abstract void linearsearch ();
  // or we can write void linearsearch ();
}
// following code can be in other file or in same file
import java.util. Scanner ;
class L9 implements i
{
    int n0, j ;
    int ar[] = {1, 2, 4, 5, 6} ;

    public void linearsearch ()   // Always use public to implement method of interface
    {
        Scanner ip = new Scanner (ip.next) ;
```

```
System.out.println("Enter no to be search");
no = ip.nextInt();
for (j=0; j<a.length; j++)
    {   if (a[j] == no)
            System.out.println("no found");
            break;
        }
    }
    if (j > a.length-1)
        {   System.out.println("no not found");
        }
    }
    public static void main (String args[])
    {   Ls ob = new Ls();
        ob.linearsearch();
    }
}
```

WAP which can implement the interface using the annonymous class →

```
@FunctionalInterface
interface Inf
{
    void show();
}
```

```
class Bin {
    public static void main (String args[])
    {
      Inf ob = new Inf () {
          public void show()
          {
            System.out.println ("welcome");
          }
      };
    }     ob.show();
}
```

Lambda functions →

- Lambda functions are used to define the abstract method of functional interface.
- It only works on functional interface.
- Its the feature of Java 8 version.
- Its used to concise the length of code.

Syntax / Example →

```
Interface Inf
  {
     void show();
  }
```

```
class Lambda {
    public static void main (String args[])
    {
        Inf ob = () -> {
                System.out.println ("welcome");
        };
        ob.show;
    }
}
```

Reference of Interface

// This is called Functional Programming Approach

WAP to calculate max using lamba functions by passing parameters in it →

```
@FunctionalInterface
interface Inf {
    void max (int a, int b);
};

class Bin {
    public static void main (String args[])
    {
        Inf ob = (a, b) -> {
            if (a > b)
                System.out.println ("a is max");
            else
                System.out.println ("b is max or equal");
        };
```

```
Bin ob
ob. max(2,4);
    }
}
```

WAP for Armstrong no using lambda function

```
@functional Interface
interface Arm
{
  public abstract void armstrong (int no);
}
class Arml
{
    public static void main (String args[])
    {   int sum, no1, r;
        Arm ob = (no) -> { no1 = no; sum = 0;
```

we can initialize here

```
        while (no>0)
        {
            int r = no % 10;
            r = no % 10;
            sum = sum + r*r*r;
            no = no / 10;
        }
        if (no1 == no sum)
        {
            System.out.println
                ("Armstrong");
        }
```

```
        else {
            System.out.println (" not armstrong ");
            }
    };
    ob.armstrong (153);
  }
}
```

- Implementing Multiple Inheritance → using multiple interface

```
interface I1 {
    void set ();
}
```

```
Interface I2 {
    void Digit_count ();
}
```

```
import java.util.Scanner;
class Dg implements I1, I2 ;
{
    @override
public void set () {
            }

    @override
public  void Digit_count () {
                    }

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

Full code →

```
interface I1
{
    void Set();
}

interface I2
{
  void Show();
}

import java.util.Scanner;

class Dg implements I1, I2
 {       int a;
        @override
       public void Set()
         {
           Scanner ip = new Scanner (System.in);
               int a = ip.nextInt();
         }

       @override
        public void Show()
         {
            System.out.println ("value of a =" + a);
         }
```

```
public static void main (String args[])
{
    Dg ob = new Dg();
    ob.set();
    ob.show();
    ip.close();
}
}
```

Extending / Inheriting Interfaces →

class will always implement child interface

example –

```
interface i4 {
    public abstract void set();
}
interface i5 extends i4
    public abstract void binsearch();
}
import java.util.Scanner;
class Ie implements i5
{
    int f, l, m, no;
    int a[] = {1, 3, 5, 6, 7};

    @Override
    public void set()
    {
        f = 0;
        l = a.length - 1;
    }
```

```
no = ip.nextInt();
m = (f+l)/2;
```

```
@override
public binsearch()
{
    while (f<=l)
    {
        if (a[m]<no)
        { f= m+1;
        }
        else if (a[m]==no)
        { system.out.println("no. found");
        } break;
        else
        { f= l= m-1;
        }
    m= (f+l)/2;
    }
  if (f>l)
    { system.out.println("no. not found");
    }
}

public static void main (String args[])
  {
      Ie ob = new Ie();
      ob.set();
      ob.binsearch();
  }
}
```

static method of Interface → called by the name of Interface example →

```
import java.util.Scanner;
interface I6
{
    public static final int no = 5;

    static void show()
    {
        System.out.println("value of a = "+no);
    }
}

class Id implements I6
{
    public static void main(String args[])
    {
        I6.show();
    }
}
```

private and default method of interface →

private is always called from a default method of the interface. for example →

```
interface I7
{
    private void show()
    {
        System.out.println("welcome");
    }

    default void show1()
    {
        show();
        System.out.println("default");
    }
}

class Dm implements I7
{
    public static void main(String args[])
    {
        Dm ob = new Dm();
        ob.show1();
    }
}
```

PACKAGE →

- A Java Package is group of similar types of classes interfaces and subpackages.
- Package can be of two types :

Built-In Package

eg ⇒ LAN, AWT, Javax, String, IO, util, sql etc.

User-Defined Package

which is created by user itself

```
import package.classname ;
```

Example to form a Package →

In diff folders

A.Java

```
package p ;
public class Abc
{
public int sqrt (int no)
  {
   return no * no ;
  }
}
```

"name of package is always small and method of package always public"

B.Java

```
import p.Abc ;
class B
{
  public static void main
                (String args []).
  {
    int a ;
    Abc ob = new Abc ();
    a = ob.sqrt (no);
    System.out.println (a);
  }
}
```

- MULTIPLE PACKAGE →

WAP of multiple package which prints the series of prime number →

```
package p ;
import java. util. Scanner;
public class A
{
    int no ;
    public int set ()
    {
        Scanner ip = new Scanner (System. in);
        System. out. println ("Enter no ");
        no = ip. nextInt ();
        return no ;
    }
}
```

```
package q ;
import p. A;
public class B
{   int i, j, no ;
    public void prime ()
    {
        A ob = new A();
        n = ob. set();
        for ( i=1 ; i<=no ; i++)
            {
                for ( j=2 ; j<=i ; j++)
                {
```

```
            if (i%j == 0)
              { break;
              }
          }
        if (i==j)
          { System.out.println (i+" ");
          }
      }
  }
```

```
import p.A;
import q.B;

class mn
{ public static void main (String args[])
    {
        B ob = new B();
        ob.prime();
    }
}
```

- NOTE →

Importing or accessing classes with same name name in different package we use fully qualified name

WAP for multiple package in which the class name is same →

(PTO)

```
package p;
public class D
{
   public void show()
     {
       System.out.println("welcome");
     }
}

package q;
public class D
{
   public void show()
     {
       System.out.println("Bye");
     }
}
class mn1
{   public static void main(String args[])
    {
        p.D ob = new p.D();
        ob.show();

        q.D obj = new q.D();
        obj.show();
    }
}
```

- Sub - Package →

```
package p.t;

public class Result
{
    double per, t;

    public double cal (double h, double e,
                       double m, double s,
                       double c)
    {
        t = (h+e+m+s+c);
        per = t/5;
        return per;
    }
}
```

```
import pt.Result;
import java.util.Scanner;

class Res
{
    double h, e, m, s, c;
    Scanner ip = new Scanner (System.in);

        // (PTO)
```

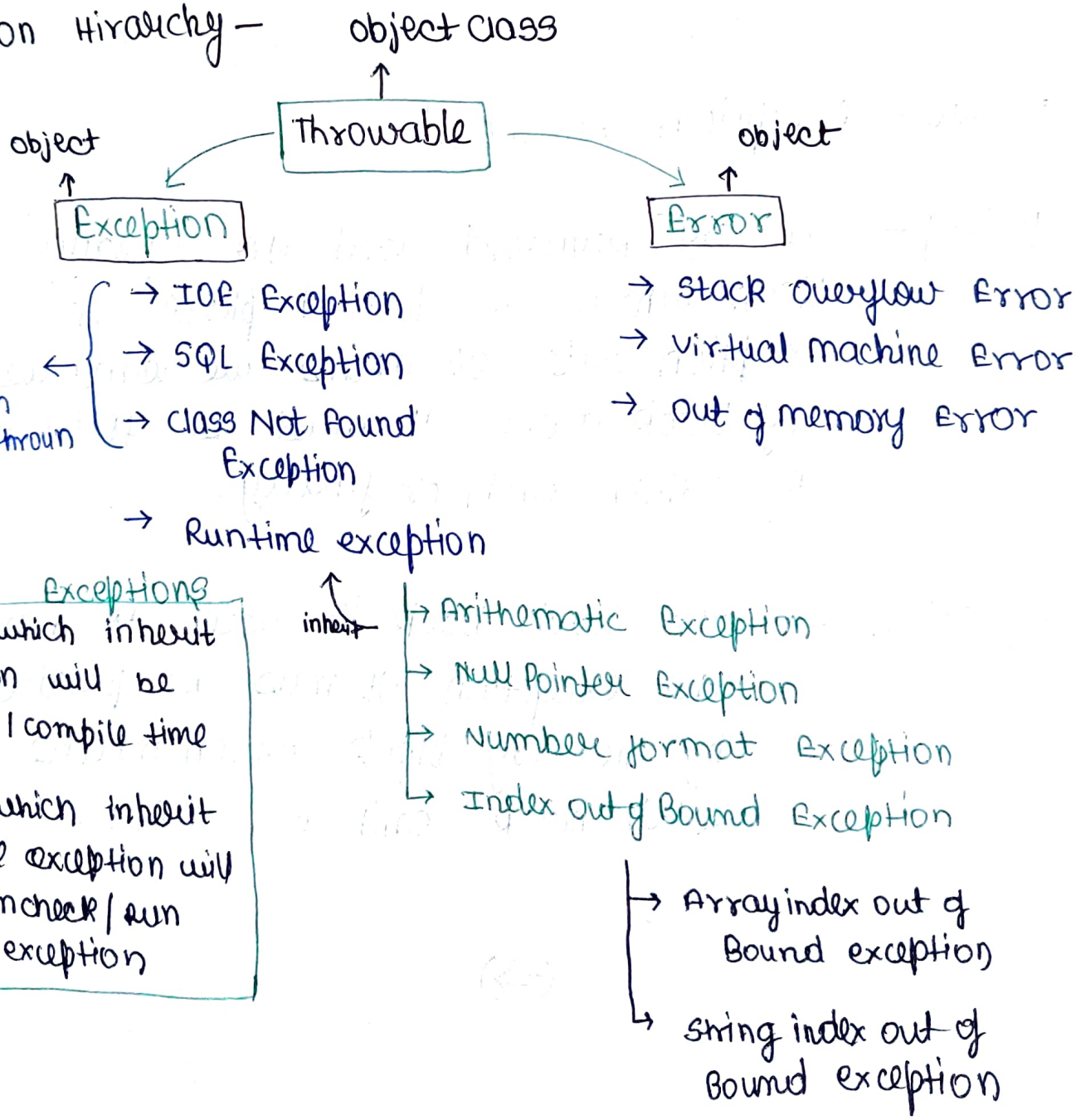
```
Void set ()
{
System.out.println ("Enter marks of
                     h, e, m, s, c");

   h = ip.nextInt ();
   e = ip.next Double();
   m = ip.next Double ();
   s = ip.nextDouble ();
   e = ip.nextDouble ();
}
Void show ()
{
   Result ob = new Result ();
   System.out.println ( ob.cal (h, e, m, s, c) );

   public static void main (String args[])
   {
      Res obj = new Res ();
      obj.set ();
      obj.show ();
   }
}
```

EXEPTIONAL HANDLING ⟹

The Exceptional handling in Java is one of the most powerfull mechanism to handle runtime errors so that normal flow of programme can be maintained.

- Exception-Exceptions are abnormal conditions they disturb the normal flow of programme.

- Exception Hirarchy –



CUSTOM Exceptions

Class which inherit Exception will be checked / compile time

class which inherit runtime exception will be uncheck / run time exception

- Types of Exceptions →

1) Checked Exception - The class that directly inherit the throwable class is called checked exception.
   eg → IO exception, SQL exception etc
   They are called compile time exception.
   (need to be handled compulsory)
2) Unchecked Exception - The class that inherit a runtime exception called unchecked
   eg → Arithematic, Null Pointer exception etc
   They are called Runtime exception.
3) Error - Error is irrecoverable.
   eg → stack overflow, out of memory error etc.

- Java Exception Keyword →

1) Try - The try keyword used to specify a block where we should place an exception code. It means we can't use try block alone it must be followed by either catch or finally.

2) catch - The catch block is used to handle the exception. It must be provided by the try block which means catch block can't be alone.

(⇒)

• Types of Exceptions →

1) Checked Exception - The class that directly inherit the throwable class is called checked exception.
eg → IO exception, SQL exception etc
They are called compile time exception.

2) Unchecked Exception - The class that inherit (need to be handled compulsory)
a runtime exception called unchecked
eg → Arithematic, Null Pointer exception etc.
They are called Runtime exception.

3) Error - Error is irrecoverable.
eg → stack overflow, out of memory error etc.

• Java Exception Keyword →

1) Try - The try keyword used to specify a block where we should place an exception code. It means we can't use try block alone it must be followed by either catch or finally.

2) catch - The catch block is used to handle the exception. It must be preceded by the try block which means catch block can't be alone.

(⇒)

3) Finally – The finally block is used to execute the necessary block of programme. It is executed wheather exception handled or not.

4) Throw – The throw keyword used to throw an exception.

5) Close – The close keyword used to decleare exceptions. It specify that there may occur a exception in the method. It dosn't throw an exception. It always use with method signature.

InputMismatch Exception →

This exception occurs when input mix match eg → we give string as input but in code int was to be given as a input. Not predefined so import it in java.

example - WAP for reverse no with Exception handle →

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

class Inme
{
    int no, r, sum;
    public Inme()
    {
        sum = 0;
    }
```

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

Void set ()
{
    try
    {
        System.out.println ("Enter no");
        no = ip.nextInt ();
    }

  catch ( InputMismatchException e )
                                    //object
    {
      e.printStackTrace ();
} }                  // this method print the exception
                        it tells type, why exception
                        occur and

void sum ()
{
    while (no>0)
    {
        r = no %10;
        sum = sum + r;
        no = no /10;
    }

    System.out.println ("sum);

}
```

```
public static void main (String args[])
  {
     Inme ob = new Inme();
     ob.set();
     ob.rev();
  }
}
```

Arithematic Exception →

This handles the exception with arithematic logics. This is predefined in Java so no need to import this exception. example 'divide by zero'

ex → WAP to divide 2 no's which handles input mismatch and arithematic exception —

```
import java.util.Scanner;
import java.util.InputMismatchException;
class Ari
{  int a, b, c;
   Scanner ip = new Scanner(System.in);

   void set()
   {  try {
         System.out.println("Enter value of a & b");
         a = ip.nextInt();
         b = ip.nextInt();
      }
```

```
catch ( InputMismatchException e)
    {
        e.printStackTrace();
    }
}
Void div()
{
    try
    {
     c = a/b;
    }
    catch ( ArithematicException e)
     {  e.printStackTrace();
     }
}
Void show()
  {  System.out.println (c);
  }
public static void main (String args[])
    {    Ari ob = new Ari();
         ob.set();
         ob.div();
         ob.show();
    }
}
```

NumberFormatException →

when we try to convert an invalid string into the integer than numberformatException will occure. It means when its not possible to convert a string on any numeric type this exception will be thrown

NuberFormatException is subclass of IllegalArgument Exception which is predefined in Java

eg → WAP to handle NumberFormatException that is thrown while converting a string to numeric →

{ String s = "123" valid → can be convertible
String s = "ab3" non valid → chare can't convertible }

```
class Nfe
{
    String s = "123g"  // invalid string ;
    int no ;
    Void convert ()
    {
      try
      {
        no = Integer.parseInt(s);
        System.out.println ("no is = "+ no );
      }

      Catch ( Exception e)
        {
          e.printStackTrace ();
        }
}
```

method of wrapper class used to convert string to numeric

we can also use NumberFormat Exception e but this can be more generic it can print any kind of exception

```
publicstatic void main (string args[])
    { Nfe ob = new nfe();
      ob. convert();
    }
}
```

Finally Block and Null Pointer Exception →

If a data structure (linked list) is pointed with NULL and we are trying to execute its length, size etc then it will throw null pointer exception

Ex → WAP to handle finally block above exception

```
class Npe
{
  String S = NULL;
  void show()
  {
    try
    {
      system.out.println(S.length());
    }
    catch (NullPointerException e)
    {
      e.printStackTrace();
    }
    finally
    {
      System.out.println("this code run
                          every time");
    }
      // finally always run either exception
         comes or not come
}
```

```
public static void main ( string args[])
{
    Npe ob = new Npe();
    ob.show();
}
}
```

Throw keyword →

used to throw an exception manully →

example → WAP for banking system using throw keyword

" Exception thorwn by throw keyword is always handled by caller ~~by~~ method."

```
import java.util.Scanner;
class Thr
{
    int bal, amt;
    Scanner ip;
    public Thr()
    {
        bal = 5000;
        ip = new Scanner(System.in);
    }
    Void set()
    {
        try {
            system.out.println("Enter amount");
            amt = ip.nextInt();
        }
```

```
    catch ( Exception e)
      {
        e.printStackTrace ();
      }
}

void withdrawal()
{   if ( amt <0 || amt > bal )
    {
      throw new ArithematicException (" invalid
                                        Amount ");
    }
                  // object of exception
                    thrown to caller method
    else
    {
      System.out.println (" amt withdrawn = "+amt);
      bal = bal - amt ;
      System. out.println (" bal amt = "+ bal);
    }
}
public static void main (String args[])
  { Thr ob = new Thr ();
    ob. set ();
    try {
          ob. withdrawal ();
        }
```

```
catch (Throwable e)
{
    e.printStackTrace();
}
}
```

// we can also use exception keyword but throwable class is superclass of exception class

// Here main is caller method of withdraw method so main method will handle the exception

~~throw~~ Throws keyword →

It identify the exception. It never throw a exception.
It tells that particular method can throw a exception.

- WAP for uncheck exception using ~~close~~ throws keyword -

```
import java.util.Scanner;
class Age
{
    Scanner ip = new Scanner ("System.in");
    int age;
    void set()
    {
        try {
            system.out.println("Enter age");
            age = ip.nextInt();
        }
        catch (Exception e)
        {
            e.printStackTrace();
        }
}
```

```
Void check () throws ArithmeticException
{
    if (age < 18)
    { throw new ArithmeticException ("under age");
    }
    else {
        System.out.println ("you can vote");
    }
}
public static void main (String args[])
{   Age ob = new Age ();
    ob.set ();
    ob.check ();
}
}
```

// Handled here only by throw

- Checked Exception → (with throws keyword)
  warning on compile time if throws not used

```
import java.io.*; // * used to import all class of io package

class Fl
{
    File fp;
    public Fl()
    {
        fp = new File ("A.txt");
    }

    Void set () throws IOException
    {
        if (fp.createNewFile())
        {
            System.out.println ("Filename =" + fp.getname());
        }
    }
```

// not handled here so thrown to caller ie main

```
public static void main
    ( String args[])
{
   A1 ob = new A1();
   try {
        ob.set();
       }
   catch (Exception e)
       {
        e.printStackTrace();
       }
}
}
```

```
public static void main
( String args) throws
 IOException
{
    A1 ob = new A1();
    ob.set();
}
```

not handled here so go to JVM

throws → यहाँ exception आ सकता है। या तो यहा Handle करो या फीर caller method को send किया जाएगा।

ArrayIndexOutOfBoundsException →

checks size of array - Invalid Index

eg →

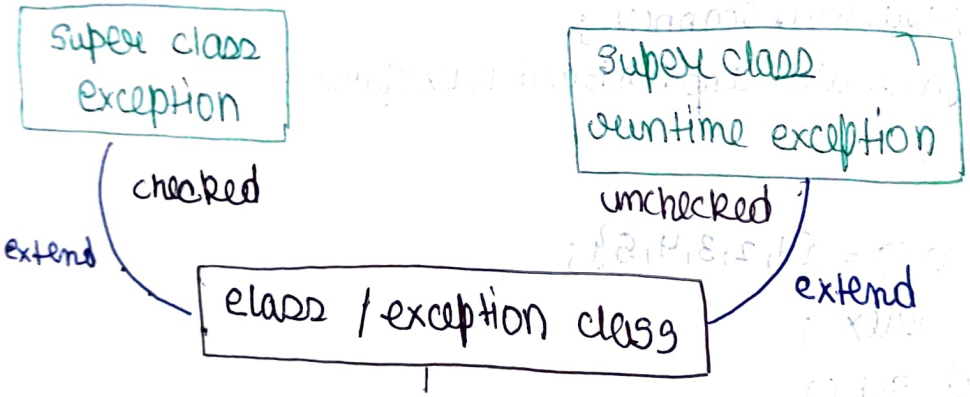
```
import java.util.Scanner;
import java.util.InputMismatchException;

class A1
{
    int a[] = {1,2,3,4,5};
    int index;
    void set()
    {
        try
        {
            Scanner ip = new Scanner(System.in);
            System.out.println("Enter the index");
            index = ip.nextInt();
            System.out.println(a[index]);
        }
```

```
catch (ArrayIndexOutOfBoundsException e)
{
    System.out.println(e);
}                                   // will print exception
```

// we can use multiple catch block

```
catch (InputMismatchException e)
{
    e.printStackTrace();
}
}

public static void main (String args[])
{
    Ai ob = new Ai();
    ob.set();
}
```

Custom Exception → (User Defined Exceptions)



If exception class inherits **exception** class as super class then compile time or checked exception will be formed

If our exception class inherits **runtime exception** class as super class then runtime or unchecked exception will be formed.

" we have to always throw custom exceptions they never come automatically "

WAP to form a checked Exception which is user defined →

```
import java.util.Scanner;
class InvalidAgeException extends Exception
{
    InvalidAgeException()
      {  super();  }                    //will call constructor of super class i.e exception class
    InvalidAgeException(String msg)
      {  super(msg);  }                 //will call parameterised constructor of super class ie exception class
}
class Vt
{ int age;
  void set()
  {  try { Scanner ip = new Scanner(System.in);
           System.out.println("Enter age");
         } age = ip.nextInt();
    catch (Exception e)
        { e.printStackTrace();
        }
  }

  void check() throws InvalidAgeException
  {
      if (age<18)
          {throw new InvalidAgeException("Underage");
          }
      else {
          System.out.println("you can vote");
          }
  }
```

```
public static void main (String args[])
{
    vt ob = new vt();
    ob.set();
    try {
        ob.check();
    }
    catch (Exception e)
    {
        e.printStackTrace();
    }
}
}
```

:- MULTI - THREADING :-

- multithreading in Java is a process of executing multiple threads simultaneously.
- Java multithreading is mostly used in games and Animations.

Advantage of multithreading →

1] Because Threads are independent you can perform multiple operations at same time.

2] As threads are independent it doesn't affect other threads if an exception occur in single thread.

Thread →

A Thread is a light weight subprocess. the smallest unit of processing.

Thread Lifecycle →

- **Stage - 1 → New / Bound State** whenever a new thread is created it always in a new state.
- **State-2 → Active State -** when a thread invokes the start method it moves from new to active state. The active state contain two states → Runnable and Running states. which are as follows —

  ↳ **Runnable -** A thread that is ready to run is moved in runnable state.

  ↳ **Running -** when the thread gets the CPU it moves from runnable to running state

- State-3 → Blocked/waiting state - whenever the thread is inactive for a span of time then the thread is in Block/waiting state.
- State-3 4 → Time waiting state - sometimes waiting for a lead to starvation for eg → A thread 'A' has entered in critical section of code and is not willing to live leave that critical section. In such ~~there~~ scenerio another thread 'B' has to wait forever which leads to starvation. To avoid such scenerio a time waiting state is given to thread 'B' thus thread lies in a waiting state for a specific time not forever.
- Step-4 → Terminated state - A thread reach to terminated state due to following reason.
  - ↳ when a thread finish its job then its exits or terminates normally.
  - ↳ Abnormal termination - whenever some un-usual event such as unhandled exception or segmentation fault is there.

← THREAD INTRUPTION →

Creating thread by Extending thread class →

- current Thread method - This method is static method of thread class (means called by class name ie thread). This method is used to get ~~print~~ the thread name on console.
- getname method - This method prints name of the thread.

- State-3 → Block/waiting state – whenever the thread is inactive for a span of time then the thread is in Block/waiting state.
- State-4 → Time waiting state – sometimes waiting for a read to starvation for eg → A thread 'A' has entered in critical section of code and is not willing to the leave that critical section. In such ~~Sneio~~ scenerio another thread 'B' has to wait forever which leads to starvation. To avoid such scenerio a time waiting state is given to thread 'B' thus thread lies in a waiting state for a specific time not forever.
- Step-4 → Terminated state – A thread reach to terminated state due to following reason-
  - ↳ when a thread finish its job then its exits or terminates normally.
  - ↳ Abnormal termination – whenever some un-usual event such as unhandled exception or segmentation fault is there.

← THREAD INTRUPTION →

Creating thread by Extending thread class:

- current Thread method – This method is static method of thread class (means called by class name ie thread). This method is used to get ~~print~~ the thread name on console.
- getname method – This method prints name of the thread.

```
class myth extends Thread
{
    public static void main(String args[])
    {
        System.out.println(Thread.currentThread().getName());
    }
}
```

output → main

- Run method – jb written thread method we make overrided form of it. its called by using start method.
- start method – jb responsible for calling run method. jis put thread to Active thread state

Example →

```
class myth extends thread
{
    @Override
    public void run()
    {
        System.out.println(Thread.currentThread().getname());
    }
    public static void main(String args[])
    {
        myth t = new myth(); // t in new / block state
        t.start();  // t is in now active state
        System.out.println(Thread.currentThread().getname());
    }
}
```

will call st run method

output – main
thread = 0
(default names)

child thread

- setname method → This method used to set the name of child threads in programme

```
class myth extends Thread
{   @override
    public void run()
    {  system.out.println(Thread.CurrentThread().getname());
    }
    public static void main(String args[])
    {
        myth t = new myth();
        t.setName("rahul");
        t.start();                          } thread-1 rahul

        myth t1 = new myth();
        t1.setName("suraj");
        t1.start();                         } thread-2 suraj

        System.out.println(Thread.CurrentThread().
                                getname());
    }
}
```

output →

rahul	main	rahul	main	suraj	suraj
main	rahul	suraj	suraj	rahul	main
suraj	suraj	main	rahul	main	rahul

These are possible outputs which depends on jvm sequencer to whom 1st system is allocated.

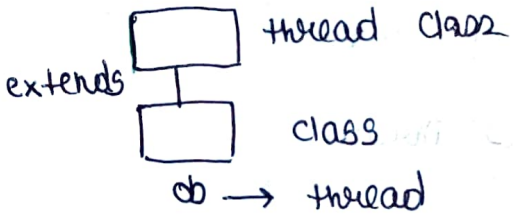
- programme to change name of current / main thread →

```
class myth extends Thread
{ public static void main (String args[])
  { Thread.currentThread().SetName("my main Thread");
    System.out.println(Thread.currentThread().
                       getName());
  }
}
```

output → my main thread

IMP NOTES →

- In every programme (even in Hello world) atleast one thread is created by jvm called main thread
- Not every ~~not~~ object is thread only object of class which is child class of Thread class is a thread.



- How multithreading works?

There are 3 threads A, B, C A takes 5 min, B take 2 min C take 1 min to complete tasks.

A switch B — C — B — A — C — A — B - - -

time allocate by jvm system	2 ns	1 ns	1 ns	2 ns	1 ns	2 ns	1 ns	3 ns	- - -

System given to each task (thread) for nano sec. then switch the thread till particular task is given. There is very fast switching so we can feel that all are simultaneous.

setting a thread name by thread constructor →

```
class my extends Thread
{
    my (String msg)
    {
        super(msg);
    }

    public void run()
    {
        System.out.println(Thread.currentThread().getName());
    }
    public static void main (String args[])
    {
        my t = new my("Child Thread");
        t.start();
    }
}
```

- isAlive() method → this is used check that thread is alive or not it returns true if thread is alive and false when thread is in terminated state

```
class my extends Thread
{       @override
    public void run()
    {
        System.out.println(Thread.currentThread().getname());
    }

    public static void main (String arg[])
    {
        my t = new my();
        t.start();
```

```
    system.out.println(t.isAlive());
}
```

It will give true ∵ thread is alive

ALTER check False (thread termination)

```
t.start();
For(int i=0; i<=10; i++)
{
}
System.out.println(t.isAlive());
```

∵ जब तक loop चलेगा t child thread terminate हो जाएगा।

will give false

Creating thread by using Runnable Interface →

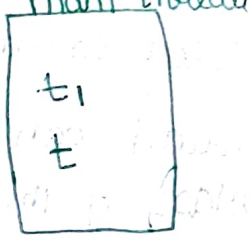
Thread can be created by two ways → object of child class of Thread class and other by implementing Runable Interface. Example – " object will always of Thread class "

```
class Thr implements Runnable
{
    public void run()
    { system.out.println(Thread.currentThread().getName());
    }
    public static void main(String args[])
    {   Thr t = new Thr();
        Thread ob = new Thread(t);   // ob will always of thread class to create thread
        ob.start();
    }
}
```

thread का object बनाकर t (अपनी class) object को pass करना thread के object से start call करेंगे

WAP for multithreading →

```
class multi extends Thread
{   int i;
    public void Run ()
    {   for ( i=1 ; i<=10 ; i++)
        {   System.out.println ( Thread.currentName().
                                  getName() );
        }
    }
    public static void main (String args[])
    {   multi t = new multi ();
        multi t1 = new multi();
        t.start();
        t1.start();
    }
}
```



Thread Intruption

- sleep method → 'sleep()' → its static method (called by name of class ie Thread). It stops the execution of thread for some seconds. It throws the checked exception ie intrupted exception which you need to always handle. It takes nanosec as parameter

```
import java.util.Scanner;
class slp extends Thread
{   int i, t, no;
    Scanner ip;
    public slp (int no)
    {   ip = new Scanner (System.in);
        this.no = no;
    }
```

```
public void run()
{  for (i=1; i<=10; i++)
   {  t = no*i;
      System.out.println(no+"*"+i+"="+t); // table printing
      try { Thread.sleep(2000);
          }
      catch (Exception e)
          { e.printStackTrace();
          }
   }
}

public static void main (String args[])
{   slp t = new slp (5);
    t.start();
}
```

Another syntax of sleep → Thread.sleep(2000, 1000);
older was native this one is deloped in java. (2000 → millisec, 1000 → nanosec)

- Join method → when two threads are depending on each other ex → one thread cal result and other thread show it so this can't run simultaneously. Join method tells that if one thread stops execution then other thread should join the programme. It also throws interrupted (checked) exception (need to handle)

```
class Res extends Thread
{
    static int sum = 0, i;

    public void run()
    {
        for (int i = 10; i <= 100; i += 10)
        {
            sum = sum + i;
        }
    }
```

```
public static void main (String args[])
{
    Res t = new Res ();
    t.start ();
    try {
        t.join ();
    }
    catch (Exception e)
    {
        e.printStackTrace ();
    }
    System.out.println ("Sum = "+sum);
};
```

NOTE other form of join is → t.join (2000); in which 2000 is nanosec as parameter. It means after 2 sec. Join another thread means loop will run till 2 sec (not depend on i) after that whatever the sum is will be printed ∴ after 2sec another thread joined ie main thread in above programme which prints sum.

~~Thread Priority~~ →

- yield method → This method send a request to JVM that I can stop my execution and wait for other threads. It totally depend on JVM that it accepts its request or not. In 99% cases JVM deny the request

```
class Ye extends Thread
{
    public void Run()
    {
        for (int i=0 ; i<10 ; i++)
        {
            System.out.println (Thread.currentThread
                                ().getName());
        }
```

```
public static void main (String args[])
{   Ye t = new Ye();
    Ye t1 = new Ye();
    t.start();
    t.yield();
    t1.start();
    System.out.println("main Thread");
  }
}
```

Demon Threads →

- Demon Thread in Java is a low priority thread that works / Run in a background to perform task eg- garbage collector. They are slave thread which works for other thread. eg- autocorrector, spell checker in ms word etc.

- Demon Thread is a service provider thread that provide a service to user thread. Its life also depends on ~~user~~ mercy of user thread. When all the threads die Jvm terminates demon thread automatically.

- Always Demon thread created before start if we create it after start it will throw IllegalThreadStateException. so main thread can't be set as a ~~Dem~~ Daemon Thread.

- setdaemon() method → used to set a thread as daemon thread in a programme.

- isdaemon() method → used to check whether a thread is daemon or non daemon. It returns true or false.

```
public static void main (String args[])
{   Ye t = new Ye();
    Ye t1 = new Ye();
    t.start();
    t.yield();
    t1.start();
    System.out.println("main Thread");
  }
}
```

Demon Threads →

- Demon Thread in Java is a low priority thread that works / Run in a background to perform task eg- garbage collector. They are slave thread which works for other thread. eg- autocorrector, spell checker in ms word etc.

- Demon Thread is a service provider thread that provide a service to user thread. Its life also depends on ~~user~~ mercy of user thread. when all the threads die JVM terminates demon thread automatically.

- Always Demon thread created before start if we create it after start it will throw IllegalThreadStateException. so main thread can't be set as a ~~Dem~~ Daemon Thread.

- setdaemon() method → used to set a thread as daemon thread in a programme.

- isdaemon() method → used to check whether a thread is daemon or non daemon. It returns true or false.

" If there is nothing in foreground the demon thread will not run "

example –

```
class Dmn extends Thread
{  public void Run ()
   {  System.out.println( Thread.currentThread ().
                                    isDaemon());
   }
   public static void main ( String args[])
   {
      Dmn t = new Dmn ();
      t.setDaemon (true);
      t. start ;
      System. out. println ("welcome");
   }
}
```

✓ Daemon works always in background

If we not write this so there will nothing in foreground of t (ie demon) so it will not run. So to run Daemon thread there must be a thread in foreground

Intruption methods →

1) Intrupt method – the Intrupt method used to Intrupt the thread. If any thread is in sleeping or waiting state otherwise it will ~~throw~~ intrupte not work. It throws Intrupted Exception when we intrupt the pro thread

example -

```
class Inter extends Thread
{ public void Run()
    try { for (int i=0; i<=10; i++)
          { system.out.println(i);
            Thread.sleep(2000);
          } }
    catch (Exception e)
        { e.printStackTrace(); }
  }
  public static void main (String args[])
  { Inter t = new Inter();
    t.start();
    t.interrupt();
  }
```

manditory to put thread in waiting state to use interrupt

2) isInterrupted method - This method of thread class is an instance method that test whether the thread has been intrupted. It return the value of internal flag either th true of false. If thread is intrupted it will return true otherwise it will return false. example →

```
class Inter extends Thread
{
  public void run()
  {
    System.out.println(Thread.currentThread().
                       isInterrupted());
    for (int i=1; i<=10; i++)
    {
      System.out.println(i);
      Thread.sleep(2000);
    }
```

example -

```
class Inter extends Thread
{ public void Run()
    try { for (int i=0; i<=10; i++)
            { system.out.println(i);
              Thread.sleep(2000);
            } }
    catch (Exception e)
        { e.printStackTrace(); }
  }
  public static void main (String args[])
  { Inter t = new Inter();
    t.start();
    t.interrupt();
  }
```

manditory to put thread in waiting state to use interrupt

2] isInterrupted method - This method of thread class is an instance method that test whether the thread has been intrupted. It return the value of internal flag either th true of false. If thread is intrupted it will return true otherwise it will return false. example →

```
class Inter extends Thread
{
  public void run()
  {
    System.out.println(Thread.currentThread().
                       isInterrupted());
    for (int i=1; i<=10; i++)
    {
      System.out.println(i);
      Thread.sleep(2000);
    }
```

```
System.out.println(Thread.currentThread().
        isInterrupted());
    }
    catch (Exception e)
    {
      e.printStackTrace();
    }
public static void main (String args[])
  {  Inter t = new Inter();
     t.start();
  }  t.interrupt();
```

3] Interrupted method - It also prints that thread is intrupted or not by print returning true or false. It change the state of thread means if thread is in intrupted state then it change it to non-intrupted state example -

```
class Inter extends Thread
{ public void run() {
    try { System.out.println(Thread.
          currentThread().interrupted());
    for (int i = 1; i<=10; i++)
      {
        System.out.println(i);
        Thread.sleep(2000);
      }
    System.out.println(Thread.currentThread().
          interrupted());
    }
    catch (Exception e)
      {
        e.printStackTrace();
      }
}
```

Output →
True
1
2
3
4
5
6
7
8
9
10
False

```
public static void main (String args[])
{   Inter t = new Inter();
    t.start();
    t.interrupt();
}
```

Sychronisation in multithreading →

- Synchronization in Java is capablity to control the access of multiple thread to any share resource.
- Sychronization is mainly used to prevent thread interference and to prevent consistancy problem.

- Types of sychronization –

1) Process Sychronization
2) Thread Sychronization
   - → mutual Exclusive Sychronisation
     - → syn. method
     - → syn. Block
     - → static syn.
   - → cooperation / Inter Thread communication

- An Anology To understand Synchronisation by movie Ticket Example →

Dg and Dt are two friends they are booking tickets for a movie in which there are 100 seats available now Dg books 70 and Dt books 80 tikets at same time so software will show tickets booked to both but its can't possible ∵ only 100 tickets are there. So Sychronization locks the object so if Dg is accessing so Dt will be in waiting state and vise versa.

```
public static void main (String args[])
{   Inter t = new Inter();
    t.start();
    t.interrupt();
}
```

Sychronisation in multithreading →

- Synchronization in Java is capablity to control the access of multiple thread to any share resource.
- Sychronization is mainly used to prevent thread interference and to prevent consistancy problem.

- Types of sychronisation–

1) Process Sychronization
2) Thread Sychronization → mutual Exclusive Sychronisation
   - → syn. method
   - → syn. Block
   - → static syn.
   - → cooperation / Inter Thread communication

- An Anology To understand Synchronisation by movie Ticket Example →

Dg and Dt are two friends they are booking tickets for a movie in which there are 100 seats available now Dg books 70 and Dt books 80 tikets at same time so software will show ticket booked to both but its can't possible ∵ only 100 tickets are there. So Sychronization locks the object so if Dg is accessing so Dt will be in waiting state and vise versa.

- Synchronization method → (movie book Application)

```
class Tba        (method level sychronisation)
{   int ts = 10; //(Total Seats)
    synchronized void booking ( int seats)
    {   if ( seats <= ts)
        {
            System.out.println ( Thread.currentThread().
                getName() + " seats book successfully " +
                    + seats );
            ts = ts - seats;
            System.out.println (" seats left = "+ ts) ;
        }
        else
        {
            System.out.println (Thread.currentThread ().
                getName() + " Sorry can't book since
                seats left = " + ts );
        }
    }
}

class mba extends Thread
{
    static tba b;   // Ref var static as formed in
                    another class
    Int seats ;

    mba ( string msg )
    {
        super (msg);
    }
```

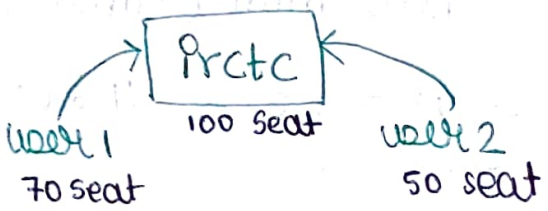
```
public void run()
{   b.booking(seats);
}
public static void main (String args[])
{       b = new Tba();
        mba amit = new mba("amit");
        amit.seats = 7;          // seats is class var
        amit.start();

        mba Rahul = new mba("Rahul");
            Rahul.seats = 8;
            Rahul.start();
    }
}
```

output → amit seat books sucessfully = 7
seat left = 3
Rahul sorry can't bok since seats left = 3

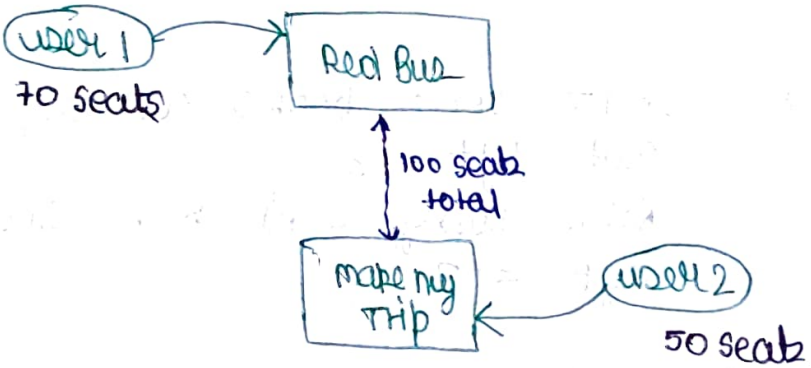
- Synchronised Block →
Rather to synchronise full method we can synchronize a part of method by using the Synchronized Block

```
Void booking (int seats)
{
    System.out.println("hello and welcome");
    Synchronized (this)        // this used to refer current object
    {
        if()
        else{
            }
    }
}
```

- Static synchronization / class level Synchronization →
- Consider two Apps → RedBus and BookMyTrip to book tickets to handle the same situation of movie theater here we need Static Synchronization.
- method level synchronization →



class level / static synchronization →



Example → movie booking system :

```
class Tba                                    // software
{
    static int ts = 10; // static :: used in a static method
    static synchronized void booking (int seats)
    {
        if (seats <= ts)
        {
            system.out.println (Thread.currentThread().
            getName() + " seats book successfully " + seats);
            ts = ts - seats ;
```

```
        system.out.println ( "seats left "+ ts) ;
    }
    else {
        system. out.println (Thread . currentThread () . getName
            () + " seats book success "+seats); " sorry
            can't book seats because seats left = "+ ts);
    }
}
}
class mythh1 extends Thread          // Red bus
{   Tba b ;
    int seats ;
    mythh1 ( Tba b , int seats )
    {  this.b = b ;
       this. seats = seats ;
    }
    public void run ()
     {
       b. booking (seats);
     }
}

class mythh2 extends Thread          // make my Trip
{   Tba b ;
    int seats ;
    mythh2 ( Tba b , int seats )
    { this.b = b ;
      this. seats = Seats ;
    }
     public void run ()
         {   b. booking (seats);
         }
    }
```

```
class mba extends Thread    // main method
{
   public static void main (String args[])
   {  Tba b1 = new Tba ();
```

2 thread hom Redbus

```
      mythh1 amit = new mythh1 (b1, 7);
      amit.start ();

      mythh2 rahul = new mythh2 (b1, 8);
      rahul.start ();
```

```
      Tba b2 = new Tba ();
```

2 thread hom makemy trip

```
      mythh1 ajay = new mythh1 (b2, 7);
      ajay.start ();
      mythh2 Sunil = new mythh2 (b2, 8);
      Sunil.start ();
   }
}
```

INNER - CLASSES →

- Java inner-classes or nested classes is a class that is declared inside the class or interface.
- we use inner classes to logically group classes and interface in a one place to be more readable and maintainable.
- inner classes can access all the members of other class including private data member & methods
- Need of Java Inner classes —

sometime user need to programme a class in such a way so that no other class can access it therefore it would be better if you include within other classes.

- Types of Inner / Nested classes —

1) Non static Nested / Inner class
   - → member Inner class
   - → Annonymous Inner class
   - → Local Inner class
2) static Nested / Inner class

- How do we know its inner class?

when class file is created it is created with $ symbol followed by out keyword named inner (with extention .class)

class file name → classname$Inner.class

member Inner class →

```
class out
{   private int a = 10;
    class Inner
    {
        void show ()
        {
            System.out.println (a);
        }
    }
    public static void main (String args[])
    {
        out ob = new out ();
        out.Inner inn = ob.new Inner();
        inn.show ();
}   }
```

object of inner class is created with help of outer class

Annonymous Inner class → (class without name)

```
abstract class student
{   abstract void show();
}
class mn
{   public static void main (String args[])
    {
        student ob = new student();
```

```
{   void show ()
    {  System.out.println (" welcome ");
    }
};
do.show ();
}
}
```

no need method of abstract class use and the anonymous class

Local Inner class →

```
class U
{  int a = 10;
   void show ()
   {  class LL
      {   void disp ()
          {  System.out.println ("value of a = "+a);
          }
      }
      LL do = new LL ();
      do.disp();
   }
   public static void main (String args[])
      {  U obj = new U();
         obj.show();
      }
}
```

Local Inner class — If a class is defined in a method then that class is called local inner class (eg above)

```
{  void show ()
    {  system.out.println (" Hellow world");
    }
};
ob. show ();
}
}
```

To access method of abstract class we use the anonymous class.

Local Inner class →

```
class Li
{  int a = 10;
   void show ()
   {  class Lii
      {   void disp ()
          {  system.out.println ("value of a = "+a);
          }
      }
      Lii ob = new Lii ();
      ob. disp();
   }
   public static void main (String args[])
      {  Li obj = new Li();
         obj. show();
      }
}
```

Local Inner class — If a class is defined in a method then that class is called local Inner class (eg above)

static Nested class →

If a class is defined static inside outer class than it called as static nested class.

Example –

```
class ot
{
    static int a = 10; ⟶ //var used in static class should be static
    static class In1
    {
        void msg()
        {
            System.out.println("value of a = "+a);
        }
    }

    public static void main(String args[])
    {
        ot.In1 obj = new ot.In1();
        obj.msg();
    }
}
```

// static so no need of outer class object. class name is sufficient

STRING - HANDLING →

- Java support string handling system.
- collection of character is called string.
- In Java string is bassically a object that represents sequence of character value.
- Array can be converted into string using string constructor
- Java provides tostring, string buffer and string builder.

creating string by converting array →

```
class St
{ public static void main (String Args[])
  [ char ch[] = {'a', 'b', 'c', 'd', 'e'};
    string s = new string (ch);
              // string constructor
    System.out.println (s);
} ]
```

output → abcde

creating string using new keyword →

```
class S1
{ public static void main (String args[])
  {
     String s = new String ("indore");
     System.out.println (s);
  }
}
```

using obj of String class

creating string using string literals →

```
class S1
{
   public static void main (String args[])
   {
      String s1 = "bhopal";
      System.out.println (s1);
   }
}
```

why to use literals rather than object →

- let s_1 created by object of string and s_2, s_3 are created by string literals.

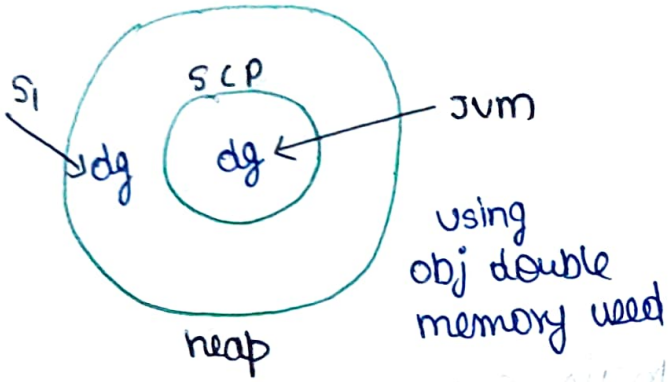
(next page →)

- whenever we create object then at 2 places memory are allocated → In heap and in scp
- we create ~~obj~~ String by literals than only at one in SCP memory is allocated

- If multiple reference have same string literals both points to the same literal
- Java string objects are immutable. but literals are also immutable. we can't change string objects / literals once we created.

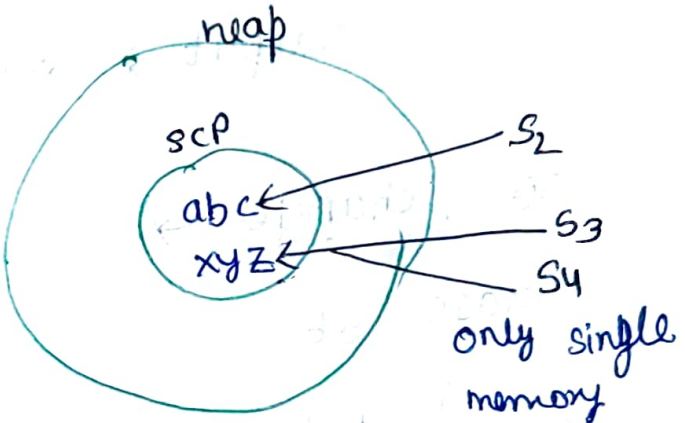
Diagramed →

we know objects created in heap and literals created in scp which is also in heap



using obj double memory used

$s_1 = dg$

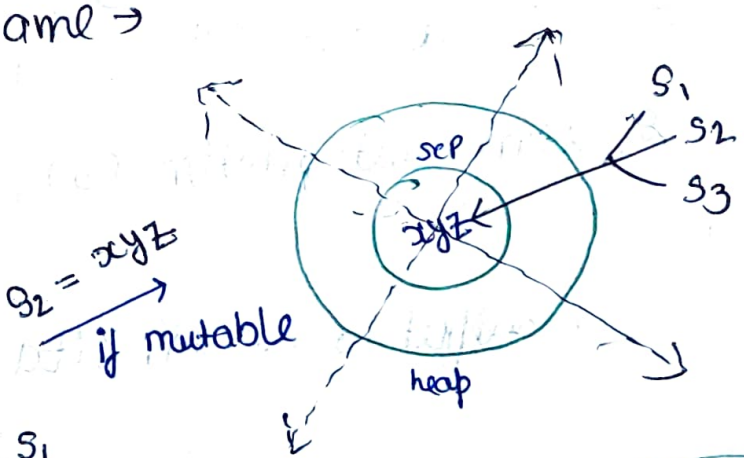


only single memory used

$s_2 = abc$
$s_3 = xyz$
$s_4 = xyz$

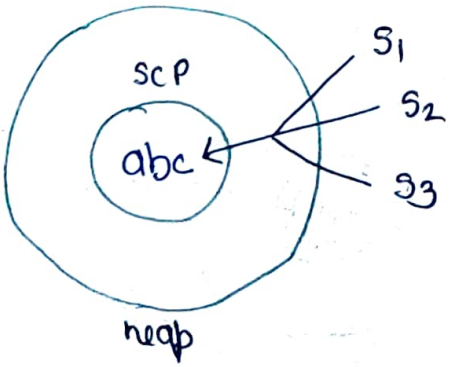
mutablity diagrame →

$s_1 = abc$
$s_2 = abc$
$s_3 = abc$

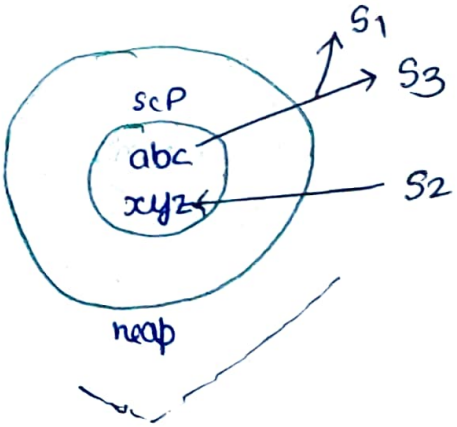


$s_2 = xyz$ → if mutable

s_1 & s_3 also got changed



$s_2 = xyz$ → but ~~not~~ immutable

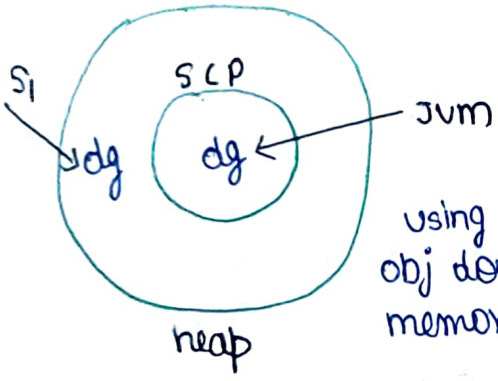


(∵ literals are immutable)

- If multiple reference have same string literals both points to the same literal
- Java string objects are immutable. but literals are also immutable. we can't change string objects / literals once we created.

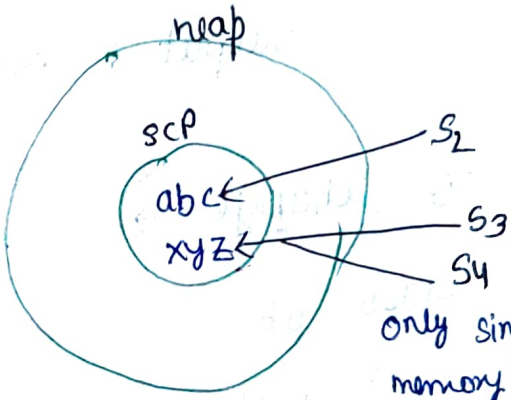
Diagrames →

we know objects created in heap and literals created in scp which is also in heap



using obj double memory used

$s_1 = dg$

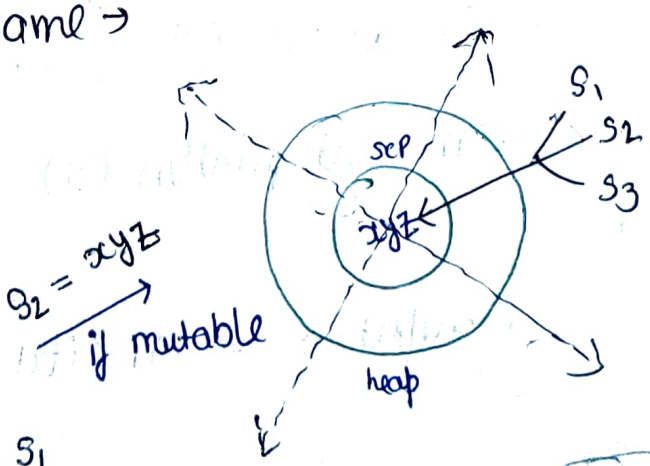


only single memory used

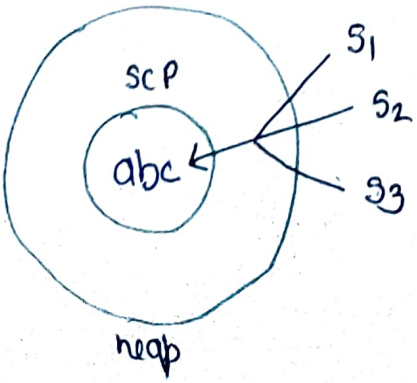
$s_2 = abc$
$s_3 = xyz$
$s_4 = xyz$

mutablity diagrame →

$s_1 = abc$
$s_2 = abc$
$s_3 = abc$

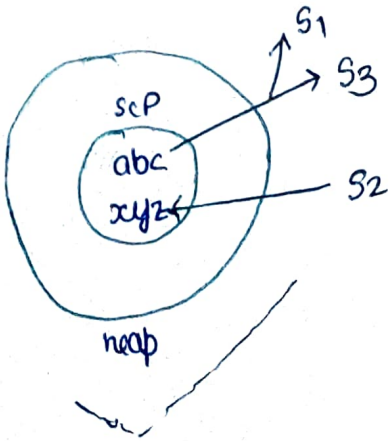


s_1 & s_3 also got changed



$s_2 = xyz$

but ~~not bad~~ immutable



(∵ literals are immutable)

WAP to check Immutability of strings →

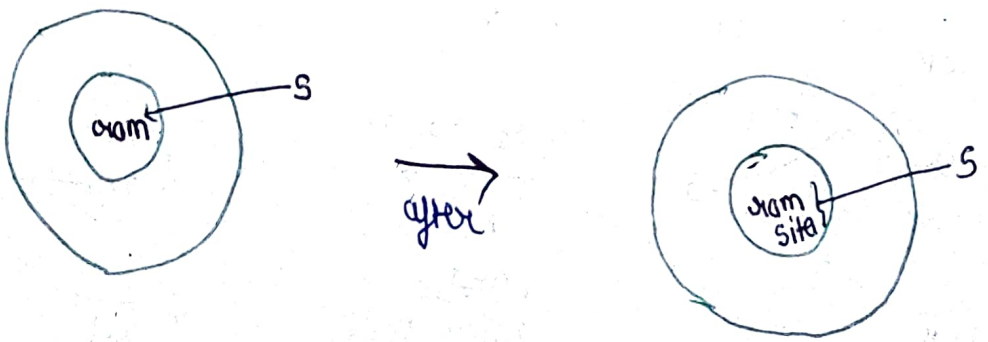
```
class St
{
    public static void main (String args[])
    {
        String s = "ram";
        s.concat ("sita");
        System.out.println (s);
    }
}
```

output → ram

To change →

```
class St
{
    public static void main (String args[])
    {
        String s = "ram";
        s = s.concat ("sita");
        System.out.println (s);
    }
}
```

output → ram sita



'==' vs 'equals method' [comparison] →

== compare reference also but equals method compares only content.

VIMP NOTE →

- Strings created by object are always on different reference so == returns always false
- String created by literals are created with same reference so == returns always true

Example -

```
class St {
    public static void main (String args[])
    {
        String s = new String ("rahul");
        String s1 = new String ("rahul");   } diff ref

        System.out.println (s==s1); //false
        System.out.println (s.equals(s1)); // true

        String s2 = "Devansh";
        String s2 = "Devansh";   } same reference

        System.out.println (s2==s3); //true
        System.out.println (s2.equals(s3)); // true
    }
}
```

output →

false
true
true
true

WAP to Explain all functions of strings →

```
class Str
{
    public static void main (String args[])
    {
        String S = "JavaCore";   // 0 1 2 3 4 5 6 7
        String S1 = "JAVA";
        String S2 = "    Java ";
        String S3 = " ";
        Sop (S.length());          // print length
        Sop (S.toUpperCase()); // Convert uppercase
                                   similarly lowercase
        Sop (S2.trim());       // remove extra space
        Sop (S.substring(2));  // 0-2 string remove
        Sop (S.substring(2,4)); // 2-3 string remove
                                   (2 = incl, 4 = not incl)
        Sop (S.replace('a','r'));  // replace
        Sop (S.startsWith("J"));  // checks starting letter
        Sop (S.endsWith("a"));  // checks end letter
        Sop (S.charAt(3));     // at 3 what letter ?
        Sop (S.indexOf("a"));  // returns index
        Sop (S.lastIndexOf("a")); // last time a
        Sop (S.equals("Java"));  // compare
        Sop (S.equalsIgnoreCase("JAVA"));
                                   // compare not case sensitive
```

// Sop ⇒ System.out.println

```
        sop (s3.isEmpty());   // check empty string
        sop (s1.concat("core")); // joins strings
    }
}
```

output →

- 8
- JAVACORE
- {java (if lowercase)}
- JaVa
- VaCore
- va
- JrvrCore
- true
- false
- a
- 1
- 3
- false
- false
- true
- JAVA core

Two-String-Method →

- If you want to represent any object as a string Two string method comes to existence
- The Two string method returns string representation of the object. (by default comes in # code)

Example →

```
class student
{   String Name;
    int age;
    int roll_no;

    public student (string name, int age, int roll_no)
    {
        this.name = name;
        this.age = age;
        this.roll_no = roll_no;
    }
    @override   // object class की tostring को override
    public string tostring()
    {
        return name + " " + age + " " + roll_no;
    }

    public static void main (string args[])
    {
        student ob = new student ("abc", 5, 10);
        Student obj = new student ("xyz", 11, 15);
```

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

STRING BUFFER →

- String Buffers objects are mutable & synchronised (slow)

methods of string Buffer →

```
public class stbuff {
    public static void main (string args[])
    {
        StringBuffer buffer = new StringBuffer ("hello");   → (obj of buffer)
        buffer.append ("world"); // appends string
        sop (buffer);
```

```
StringBuffer sb = new StringBuffer ("Hello");
sb.insert (1, "Java");   // insert / append at index
sop (sb);
```

```
String Buffer sb1 = new stringBuffer ("Hello");
                                      0 1 2 3 4
sb1.replace (1, 3, "Java");   // replace bet 1 to 3
                    ↳ not incl
sop (sb1);
```

```
StringBuffer sb2 = new stringBuffer ("Hello");
sb2.delete (1,3);   // del bet 1 to 3
Sop (sb2);
```

```
String Buffer sb3 = new stringBuffer ("Hello");
sb3. reverse ();   // reverse string
Sop (sb3);
```

```
StringBuffer sb4 = new stringBuffer ();
sop ( sb4. capacity ());   // capicity by default
        // (default)          16
sb4. append ("Hello");      when 16 char
                            entered
sop ( sb4. capacity());     then 16*2+2
        // now 16

sb4. append ("Java is my fav lang");
sop ( sb4. capacity());
        // now 16*2+2
```

```
}
}
```

capicity formula → 16 * 2 + 2

old cap * 2 + 2

ie 34

than 34 * 2 + 2

STRING - BUILDER →

	STRING-BUFFER		STRING-BUILDER
①	String Buffer is sychronised ie thread safe it means two thread can't call method of buffer simultaneously	①	String Builder is non sychronised ie non thread safe iti means two thread can call the method of builder simultaneously.
②	String Buffer less efficient than string builder since method is sychronised	②	more effiencient than string
③	Introduced in Jala 1.0	③	Introduced in Jala 1.5

- methods of string Builder →

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

    StringBuilder sb = new stringBuilder ("Hello");
    Sb. append ("Java");   // appends string
    Sop (sb);
```

```
    StringBuilder Sb1 = new StringBuilder ("Hello");
    Sb.Insert (1, "Jaa");   // insert at index
    Sop (sb);
```

```
StringBuilder sb2 = new StringBuilder("Hello");
sb2.replace(1,3,"Java"); // replace bet 1 & 3 index
sop(sb2);
```

```
StringBuilder sb3 = new StringBuilder("Hello");
sb3.delete(1,3); // delete bet index
sop(sb3);
```

```
StringBuilder sb4 = new StringBuilder("Hello");
sb4.reverse(); // reverse String
sop(sb4);
```

- FILE - HANDLING → I/O

- File Handling mechanism used to perform read write operations using files ie read, write, delete, data in the files.
- File Handling written in io package with File class
- File always throw checked exception, that is ie exception

creating a file and Some methods of file →

- createNewFile() → used to create a new file. called with obj of File class.
- getName() → throws name of file if already created
- getAbsolutePath() → throws path of the file
- length() → shows size of file
- canRead() → shows file is readable or not
- canWrite() → shows file is writable or not
- setWritable(true/false) → change permission if true than file will writable else false than file would not writable

Example (PTO)

Example -

```
import java.io.*;
class A1
{
    public static void main (String args[])
    {
        try {
            File fp = new File ("A.txt"); // open File (File pointer)
            if ( fp.createNewFile())
            {
                System.out.println ("File created");
            }
            else {
                Sop ("File name "+ fp.getName());
                Sop ("File Path "+ fp.getAbsolutePath());
                Sop ("File Size "+ fp.length());
                Sop ("is redeable"+ fp.canRead());
                Sop (" ");
                fp.setWritable (false);
                Sop (" is writable "+ fp.canWrite());
                Fp.setWritable (True);
            } } Sop ("is writable" + fp.canWrite());
        catch (Exception e)
        {
            e.printStackTrace();
        }
    }
```

Writing Data to File using File writer class →

- A file writer class have write method to write data in file and close method to close file.
- object will be of FileWriter class.
- it also throw io exception ie checked exception.

- write () - write the data in file
- close () - close the file

Example -

```
import java.util.Scanner;
import java.io.*;

class Fw
{
 public static void main (String args[])
  {  String name;
     try {
     Scanner ip = new Scanner (System.in);
     FileWriter fw = new FileWriter ("F.txt", true );
```

VIMP NOTE

FileWriter ("Name") → write mode overwrite data

FileWriter ("Name", true) → append mode not overrite / append data

```
            System.out.println ("Enter name");
            name = ip.nextLine ();
            fw.write ("\n" + name);
            fw.close ();
        }
    catch (Exception e)
        {
            e.printStackTrace ();
        }
    }
}
```

Reading Data →

- we can read data using scanner class.
- it also throws i/o exception.
- used io package for using file class.
- hasNextLine () → check wheather file have next line or not
- nextLine () → Reads the line from file

Example →

```
import java.io.*;
import java.util.Scanner;
```

```
class Fr
{ public static void main (String args[])
  [ try {
        File fp = new File ("A.txt");
        Scanner ip = new Scanner (fp);
        while ( ip.hasNextLine ())
        { String data = ip.nextLine();
          System.out.println (data);
        }
      ip.close ();
  }
  catch (Exception e)
     [ e.printStackTrace();
     ]
}
```

open file

we send send obj of file class to scanner class

deleting data from file →

- delete() → this method used to delete the file

```
import java.io.*;
class Del [
    { public static void main (String args[])
      {
        File fp = new File ("A.txt");
        if (fp.delete ())
          [ System.out.println (" file deleted "+
```

```
            + fp.getName());
        }
        else
        {
            System.out.println
                ("unable to delete file");
        }
    }
}
```

Writer class →

- Writer class used to insert / write data in file.
- Writer class is parent class of FileWriter class.
- Same methods and throws same exception.
- This is abstract class so object will of FileWriter and ref. var will of writer class

Example →

```
#include java.io.*;
class wr
{
    public static void main (String args[])
    {
        try {
```

```
        Writer wr = new FileWriter ("A.txt");
        wr.write ("welcome");
        wr.close();
    }
    catch (Exception e)
        { e.printStackTrace();
        }
  ]
}
```

if not created will create & if created than open it

Java Reader Class →

- class used to read the characcater.
- methods of subclass read & close..
- Read data in byte / binary form so we make logic accorindly. means we typecast data to char. & throw i/o exception

Example →

- read() → read single character

```
import java.io.*;
class Rd {
    public static void main (String args[])
    {
      Reader rd = new FileReader ("A.txt");
      int data = rd.read();   // read single char
                              ie 1st char
      // (ITO)
```

```
while ( data != -1)          // typecast
{
    system.out.println((char) data);
    data= brd.read();
}                    // (read char till data != -1)
brd.close();
}
catch (Exception e)
{
    e.printStackTrace();
}
}
}
```

Buffer-Writer-Class →

- Its used to provide buffering for writer instance. (Buffer Reader class)
- It makes performance fast
- It Inherits writer class so throws i/o excep.
- object and ref. var will of File Writer class and pass it to obj of Buffer writer class.

example →

```
import java.io.*;

class Bwc {
```

```
public static void main (String args[])
{
    try {
        FileWriter fr = new FileWriter ("A.txt");
        BufferedWriter bw = new BufferedWriter
                                (fr);
        bw.write ("welcome to File handing");
        bw.close();
        fr.close();
    }
    catch (Exception e)
        { e.printStackTrace();
        }
}
}
```

Buffer-Reader - class →

- It used to provide buffering for Reader instance. (Buffer Reader class)
- It makes performance fast.
- It Inherits Reader class so throw i/o exception
- obj & ref var will of file reader class and pass it to object of buffer reader.

example -

```
import java.io.*;
public class Byr {
```

```
public static void main (String args[])
    throws Exception {
        FileReader fr = new FileReader ("A.txt");
        BufferedReader br = new BufferedReader (fr);
        int i;
        while ((i = br.read()) != -1) {
            System.out.print((char(i));
        }
        br.close();
        fr.close();
    }
}
```

Reading data from console by Inputstream Reader and Buffer Reader →

- Inputstream Reader is class which is used for Inputs. It throws exception
- object of inputstream passed in obj of buffer Reader

example -

```
import java.io.*;
class Bufd
{
    public static void main (String args[])
    throws Exception
    {
```

```
String name;
InputStreamReader i = new InputStreamReader
                      (System.in);
BufferReader br = new BufferReader(i);
System.out.println("Enter name");
name = br.readline();
System.out.println(name);
br.close();
i.close();
}
}
```

// console if file name than in file

we can take input without scanner class by this way

Print-Writer-class →

- Print writer class is implementation of writer class.
- Its used to print formated representation of objects to text output stream.
- we can print on console ass well as on file.
- Flush() → This method used to clear stream memory.

example-

```
import java.io.*;

class Pwc{

    public static void main (String args[])

        {
```

```
PrintWriter w = new PrintWriter (System.out);
w. write ("welcome");
w. flush ();
w. close();

PrintWriter wl = new PrintWriter ("A.txt");
wl. write ("hello & welcome");
w. flush();
wl. close();
} }
```

on console

In file

Programme to read write object data int the file →

- File output stream → used to open file and used to write data in the file.

- object output stream → used to write data of object in file.

- write object () → This method used to write data. method of object output stream

- X.dat file → Binary file with .dat extention

- File input stream → used to open file and used to read data from file

- object input stream → used to read data of object from file.

- readobject() → This method used to read data. method of objectinput stream.

Example →

```
import java.io.*;
class std43 implements serializable
{
   string name;
   int age;
   public std43 ( string name, int age)
      {  this.name = name;
         this.age = age;
      }
   public string tostring ()
      { return name + " "+ age;
      }
   public static void main (string args[])
   throws exception
      {  try {
```

```
FileOutputStream fos = new FileOutputStream ("A.dat");
ObjectOutputStream oos = new objectoutputstream ("fos");
    Std43  ob = new Std43 ("akahul",23);
    oos.writeobject (ob);
    oos close ();
  }
  catch (IOException ex)
     { ex.printStackTrace ();
     }
Try {
  FileInputStream is = new FileInputStream ("A.dat");
  objectInputStream ois = new ObjectInputStream (is);
    Std43 st = (Std43) ois.readobject ();
      ois. close ();
      is. close ();
      System.out.println ( st.toString ());
    }
  catch (IOExcpHon e)
      {
        e. printStackTrace;
      }
}
```

JAVA - SWING ⟶ (NOT - USED - NOW)

- we can use Java swing & javax package to create GUI. (Forms etc)
- JFrame → This class sets the frame of GUI This class have many methods.
- we can create Button, Drop Down etc. in our GUI. Nowadays GUI's are created by other technologys like HTML, CSS, JS, Php

- Example using JFrames →

```
import Javax.Swing.*;
public class Simple {
 JFrame f;
 Simple() {
  f = new JFrame();
  JButton b=new JButton("click");
  b.setBounds(130,100,100,40);
  f.add(b);
```

```
        f.setSize(400, 500);
        f.setLayout(null);
        f.setVisible(true);
    }

    public static void main
            (String args[])
    {
        new simple();
    }
}
```

RELATIONSHIPS BET. CLASSES

There are Two Types of Relationship →

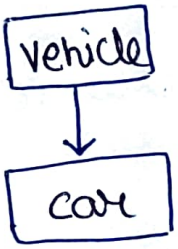
1] IS-A Relationship

2] HAS-A Relationship

IS-A Relationship →

This is also known as Inheritance

we use extends keyword.



eg → car IS A vehicle

HAS-A Relationship →

This is also known as Association

eg → car HAS a engine

In this one class have a object of other class means car class will have obj of engine class in above example

more example → student has a ROLL NO

student has a Name

example →

```
import java.util.Scanner;
class Eng
{
    int f=0;
    int start(int R)
    {
        if(R==1)
        {f=1;
        }
        else {
            f=0;
        }
        return f;
    }
}

class Car
{
    int key,s;
    Eng ob = new Eng();
    Scanner ip = new Scanner(System.in);
    void car_start()
    {
        System.out.println("press 1 to start car");
        key = ip.nextInt();
        s = ob.start(key);
```

```
        if ( s == 1 )
        {  System.out.println (" car start");
        }
        else {
          System.out.println (" error in car start");
        }
    }
    public static void main ( String args[] )
        {
            car ob = new car ();
            ob.car_start ();
        }
}
```

Types of Association / Has A relationship →

There are Two types of Association →

- composition
- Agregation

- composition – Tightly bound relation. (Objects are tightly coupled) eg → body & heart, car & engine. one can't live / exist without other

- Aggregation – Loosly bound relation. one does not depend fully on other (life cycle not depends) eg → body & one kidney, car & AC, etc.

POJO Class :-

- POJO class should be containg all private data members.
- It should contain atleast one non paramatrized or paramatrized constructor.
- It should contain getter and setter methods.

Get and set method are used to control and change private data members from outside the class.

Example →

```
class student {

    private int roll_no;

    public student ( int roll_no)
    {
        this. roll_ no. = roll_no. ;
    }
```

```
public int getroll_no ()
{
    return roll_no;
}
public void set_rollno (int rollno)
{
    this.roll_no = roll_no;
}
}
class main {
    pubjic static void main (string args[])
    {
        student s1 = new student (19);
        s1.setRollno (32);  // change roll no.
        System.out.println ("roll no. is,"+
                            s1.getrollno());
    }
}
```

Printing object →

println (s1) // example

o/p →

com. packagename classname @ 134 Pe

↙

hash code of obj by JVM

IMP Note →

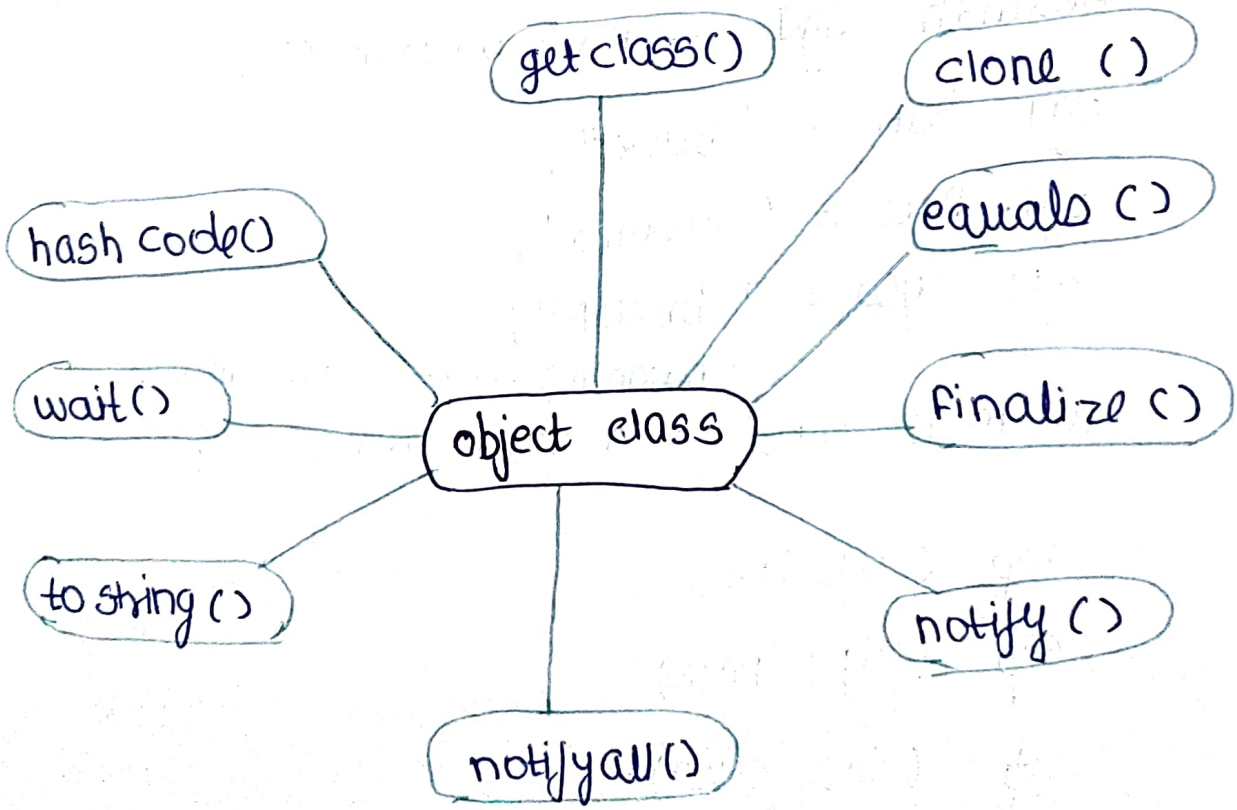
All class here inherited by object class by Default →

Java. lang. object

OBJECT - CLASS

This is one special class, object defined by Java.
All other classes are subclasses of object.
object class is super class of all other class.
A reference variable of type object can refer to an object of any other class.
Since arrays are implemented as classes, a var of type object can also refer to an array.

methods of objet class →



① objectclone() →

creates a new object that is same as the object being cloned. 2 diff objects have same content / prop. but address is diff.

Example –

```
class Devansh
{
    string hair;
    string face;
    string eyes;

    public static void main (string[] args)
    {
        devansh dg1 = new Devansh();
        dg1.hair = "black";
        dg1.face = "round";
        dg1.eyes = "brown";
                    (Devansh) ——→ //Type cast
        Devansh dg2 = (Devansh) dg1.clone();

        sop (dg2.hair);
        sop (dg2.face);
        sop (dg2.eyes);
    }
}
```

o/p

black
round
brown

handle clone not support exception to use clone method.

② boolean equals (object object) →

compares the calling object with another object and returns true if equal otherwise returns false. Compare location & memory so always override it

example →

```
public class demo {

    p. s. v m. ( s args [] )

    [
        student a = new student ("mary", 19,
                                 1122);

        student b = new student ("mary", 19,
                                 1122);

        if (a. equals (b))
            {     sop ("equal");
            }
        else {
            sop ("not equal");
            }
    }
}
```

o/p e: not equal

o/o comparing references of objects.

we will override it to make a & b equal objects if content is same

```
class student {

    private int studentID;
    private int Age;

    public student ( int a1 , int b1)
    {
        a1 = studentID;
        b1 = Age;
    }
    @override
    public boolean equals ( object obj )
    {
        student s = (student) obj;
                    a1
        if ( studentID != s.studentID )
            return false;

        if (b1 != s.Age)
            return false;

        return true;
    }
}
```

upcasting

Downcasting

```
public class Demo {
    public static void main (string args[])
    {
student a = new student (19, 1122);
student b = new student (19, 1122);
if (a.equals (b))
        sop ("equal");
else    sop ("not equal");
    }
}
```

o/p equal

" cont. in Adv. Java "

```
public class demo {
    public static void main (string args[])
    {
        student a = new student (19, 1122);
        student b = new student (19, 1122);
        if (a.equals (b))
            sop ("equal");
        else    sop ("not equal");
    }
}
```

o/p equal

"cont. in Adv. Java"

Runtime Polymorphism Imp's

```
class Big {
    void show ()
    {
        sop ("Big");
    }
}

class small extends Big
{
    @override
    void show()
    {
        sop ("small");
    }
}

class main {
    public static void main (String args[])
    {
        Big bm = new small();
        bm.show();
    }
}
```

O/P
small

bm will look in Big if find show than look in childs for override forms

If there is no show in parent than there error

object is of small we can call only those method given by parent to child

Runtime Polymorphism Imp's

```
class Big {
    void show ()
    {
        sop ("Big");
    }
}

class Small extends Big
{
    @override
    void show ()
    {
        sop ("small");
    }
}

class main {
    public static void main (String args[])
    {
        Big bm = new Small ();
        bm.show ();
    }
}
```

bm will look in Big bf find show than look in childs for overriding forms

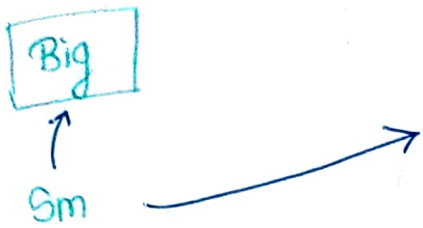
If there is no show in parent than there error

o/p
small

object is of small we can call only those method given by parent to child

sm Big का object है और
small के reference को hold करेगा

Big sm = new Small();



ref. Parent class का object child class के obj reference को hold कर सकता

(Parent का obj child को access नहीं करेगा)

Here Parent's object holding reference of chid class

1st sm will check & show in Big class

But In Java there is by dyault virtual keyword # so than it will check in small so o/p = small

X Small ob = new Big(); Invalid

VIMP → child class का ref. object parent class के obj reference को hold नहीं कर सकता

```
class Big {
    void show()
    {
        sop("Big");
    }
}

class Small extends Big {
}
    void show()
    {
        sop("small");
    }

class main
{
    public static void main(String args[])
    {
        Big ob = new Small();
        ob.show();
    }
}
```

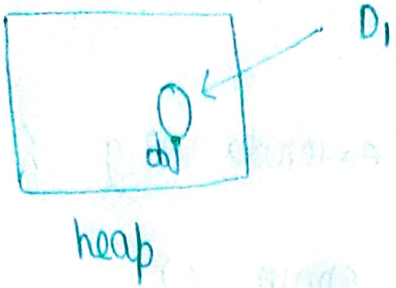
O/P → Big

∵ it will check in parent class ∵ no method in ↳ child so show of Big will Run

```
class Big {
        }


class small extends Big {

        void show ()
            {   sop ("small");
            }

}

class main {
{
    public static void main ( string args[])
    {
        Big ob = new small ();

        ob. show ();

    }                    :. no show in parent

}
```

o/p → error

Demo D1 = new Demo ();



heap

D1 is reference holding object of Demo

VIMP Notes →

child class object can't का reference parent class के object को hold नहीं कर सकता।

But

Parent class का reference child class के object को hold कर सकता है

```
child obj = new Parent();
```

```
Parent obj = new child();
```

Type Casting →

seq → byte, short, int, long, float, double

"left वाले को right में assign कर सकते but right वाले को left में assign नहीं कर sakte"

eg →
```
byte b = 100;
int x = b;
```
} ✓ allowed

but we need typecasting when →

```
int x = 100;
byte b = (byte) x;
```
(byte) x ; — type cast

Parse int used to

getter & setter methods →

POJO class
plain old java object

```
class student {
    private int rollno;
    private string name;

    # paramatrized constructor

    public int getRollno() {
        return rollno;
    }

    public void setRollno( int rollno)
    {
        this.rollno = rollno;
    }
}

class main {
    public static void main(
        student s1 = new student (31, "Rahul");
        s1.setRollno.(32);
        sop ("rollno. is ", + s1.getRollno());
    }
```

object print →

com.package name@ 1234 ← hex code

Class name

hash code of object assigned by JVM

All class by default inherited by object class.

java.lang.Object

functional programming

new → scope project / project

eclipse is a

interface is always void show();

package name →

com.pojo.object

name

multiple abstract methods is called

② An interface with only single abstract method is called functional interface (SAM)

③ this interface is denoted by @functional interface annotation.

④ marker interface → is an empty interface eg → serializable clonable

note → Private method of interface is always called from default method of interface.

492 Canvid

$$P = (A_{11} + A_{12})$$

Interface →

- An Interface in java is a blueprint of class
- The interface in java is a mechanism to achive abstraction
- java interface represent 'is-a' relationship.
- Interface is defined by interface keyword.
- To implement a interface we use implements keyword

- one class can implement multiple interface
- one interface can inherit other interface

Reasons for creating a interface →

↳ To achive multiple inheritance

↳ To achive abstraction

- Before Java 1.8 only abstract method is there in interface from 1.8 version java added the static and default method in interface
- ~~After~~ In java 1.9 private method concept also added in interface
- The variable declare in interface is always public static final type
- An abstract method of interface is always public. (public abstract void show())

- Types of interface →

① And Interface with multiple abstract and concreate methods is called normal interface.

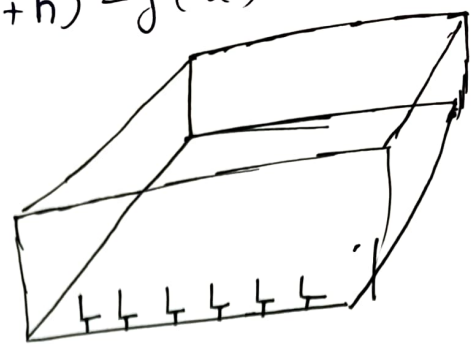
if a class inheriting the abstract class then its compulsory for child class to override the abstract method of abstract class otherwise you have to make a child c/s as abstract class for then child have to implement the abstract method.

```
abstract class Abl
{
  int a = 7 ;
}

class Ba extends Abs 1 {

 void show ()
 { a++;
   Sop (a);
}
 public. static void main ( String arg [])
   {
      Ba ob = new Ba();
     ob. show();
   }
}
```

$\Delta f(x) = f(x+h) - f(x)$

$\Delta f e^{ax}$

Δ^2



Sp

~~Hassle~~ Sombre

up the down stair case

Hassle → Problem परेशानी

winsome → attractive मनोहर ~~वर~~

ill will → bitterness

somber → dark or dull in tone निराशाजनक

non chalant → appearing casually बेपरवाह

magnanimous → forgiving easily उदार

suave

conceited

gregarious

Engrossing

~~Glum~~ Grumpet

Topy turvy

Tumultous

How to be a non chalant in a ~~Hassle~~