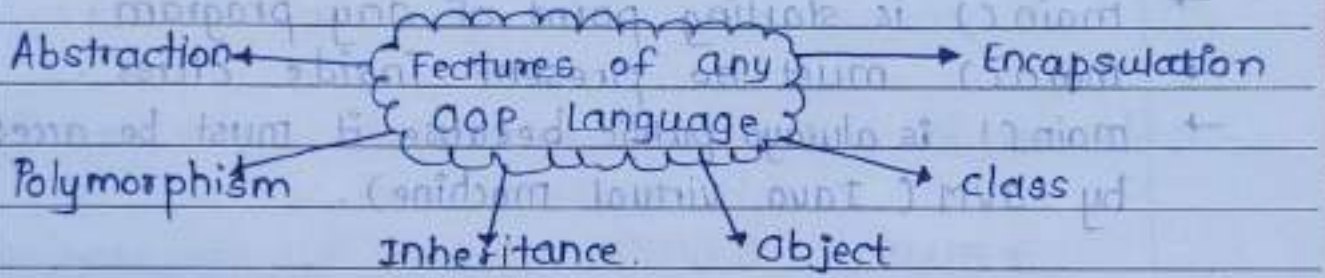
17/Aug-2021 Day: 01

Topics to learn:

- → Intro and Features
- → Classes and Objects
- → Packages and Interfaces
- → Multithreaded Programming and exception handling.
- → Graphics programming and internet
- → File Ilo and collection framework

CH: 1 Introduction and Features of Java



- → any language which supports these features is called as OOP lang.
- → So, Java is also called as Pure object oriented Programming language

C++

```
class A
{
    mem_fun();
    data_members;
};
int main()
{
}
```

Java

```
class A
{
    mem_fun();
    data_members.
    public static main(String[] arg
    {
    }
}
```

Note: In Java main function and all the code must be present inside the class.

main function syntax:

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

  }
```

→ main() is starting point of any program.
→ main() must be present inside class.
→ main() is always public because it must be accessed by JVM (Java virtual machine).
  → Public: Accessible through out the program.
→ main() must be static because main() function is executed before object creation.
  → static: They can be accessed before creation of object
→ Void means returns nothing.

OOPS features:

① Class: Collection of data members and member function. Blueprint for object.

Syntax:

```
class A
{ data mem;
  member fun();
  P.S.V main (String [] args)
}
```

Javad

② Object : It is runtime entity.
→ In Java objects are always created at run time / Dynamically.

Common Syntax :

```
Class_name obj = new class_name ();
```

③ Abstraction : Hiding background details.

④ Encapsulation : Wrapping up of data into single unit.
In java class is wrap up for member functions & data members

⑤ Inheritance : Acquiring properties of one class into another class.

⑥ Polymorphism : Ability to represent more than one form.

⑦ Dynamic Binding :
Binding : Resolving method call.
→ ① Static binding / Early binding : Method call resolve at ~~run time~~ (Compile time)
→ ② Late binding / Dynamic binding : Method call resolve at run time

⑧ Message Passing : two objects communicates with each

⑨ Modularity : Dividing complex task into small tasks

⑩ Reusability : Reusing classes using inheritance.

⑪ Resilient to change : changes

18 Aug 2021

Day : 02

FIRST Java program

① Type your program in any text editor.

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

Here,
- System is class
- out static object
- println is method

② Save your program with .java extension.
standard practice to save your program with classname.java [eg. Your_class.java]

Location of Classname.java File

D:\
↳ temp
↳ Your_class.java

③ Compile program using java compiler.

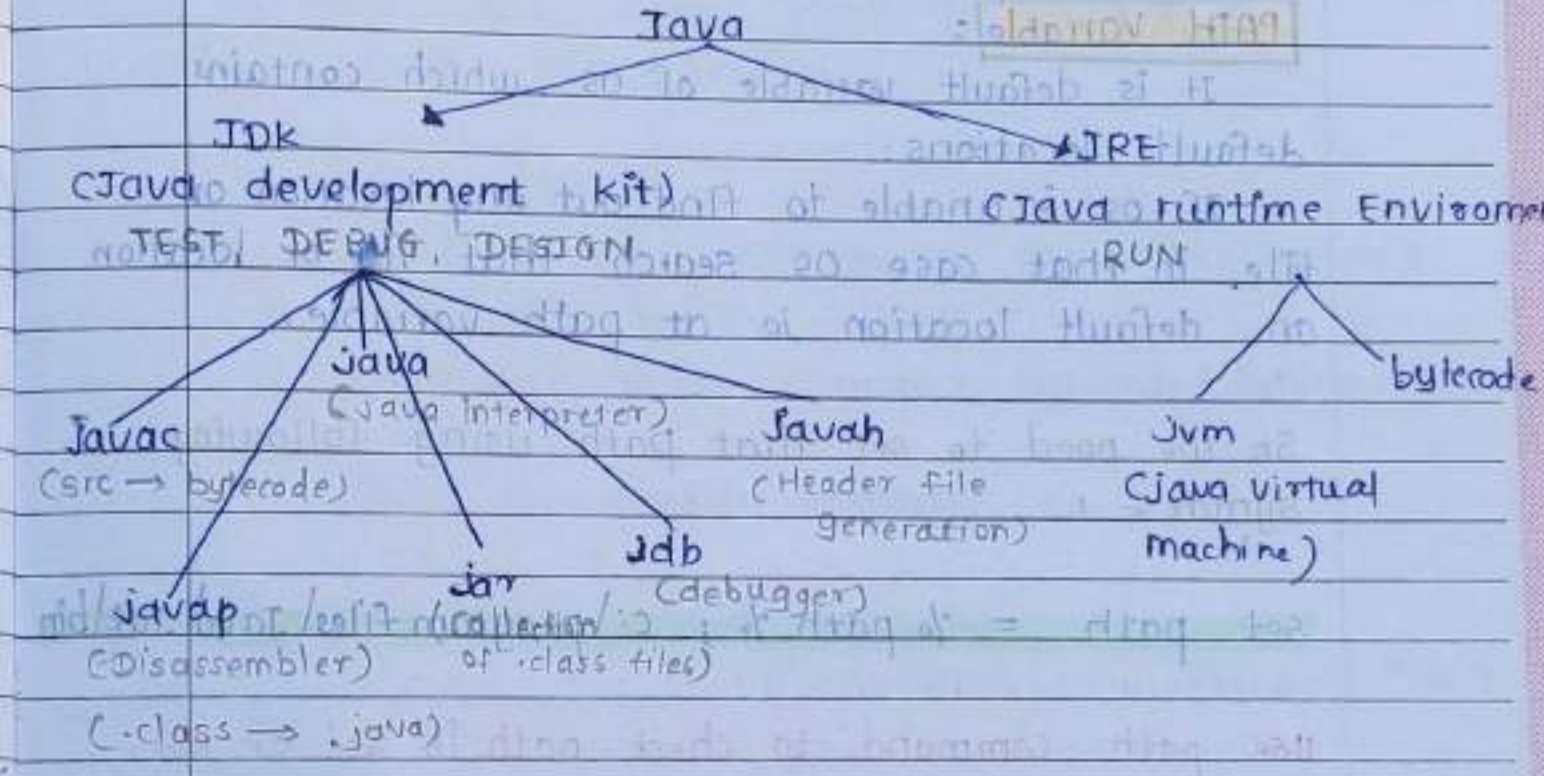
Java technologies

1] Java SE → Standard Edition → Core programming

2] EE → Enterprise Edition → Collection of (API)
JSP, JDBC, Networking, servlet
Application Programming Interface
(Collection of classes)

3] ME → Mobile Edition → Collection of API

Components of Java



Download java setup (Java SE 8) then you will get that setup on following location.

C:\
↳ Program files
↳ Java
↳ jdk
↳ bin [javac, java]

Program

For running and compiling program we needs java, javac & command prompt. But we have javac and java on another path/location and program is on another location, so after executing - javac programname.java

- java programname

it will throw error as java/javac are on another location and program is on another location.

19/08/2021

Day : 3

So solution for that problem?

↳ We can use path variable.

PATH Varrable:

It is default variable of os, which contains default locations.

If os is unable to find out any location or file, in that case os search that file or location on default location ie at path variable.

So we need to set that path using following syntax:- (on command prompt)

```
set path = %path% ; c:\program files\Java\Jdk\bin
```

Use path command to check path is set or not.

Here setting path through command prompt is set path is temporary so we can save that path permanently using and method.

Go to Advance system setting
↳ Click on Enviroment variable
↳ and click on path then click edit
↳ paste the path of bin folder
↳ into blank column and click ok
↳ And by clicking ok-ok close all the dialouge boxes.

Finally, open command prompt and open select your program file path using
↳ D:
then enter

↳ cd temp
enter
↳ Now your folder path is selected. So compile your program
↳ D:/temp> javac your_class.java
(command for compilation)
↳ After successful compilation .class file is generated which contains bytecode (intermediate code)
Now run your program
↳ D:/temp> java Your_class
(command for executing file)

20/8/2021 Day 4

@ Benefits of oops

- Reusability
- Data hiding
- Reduced complexity of a program
- Easy to maintain and upgrade
- Message passing
- Modifiability
- Security

Pop and OOP features / Difference :

Procedure Oriented	Object Oriented
i) Program is divided into functions.	i) Program is collection of object.
ii) Importance for functions	ii) Importance for data
iii) follows top-down approach	iii) Bottom-up
iv) Don't have access specifiers	iv) Support access specifiers
v) Don't support oops features.	v) Support oops features.
eg. C, pascal	eg C++, java

* Difference between C++ and Java:

C++	Java
1) Extension of c with object Oriented behaviour.	1) Pure object Oriented lang.
2) Provides template classes	2) Do Not support templae classes.
3) Global Variables can be used	3) Do not support to global variables.
4) Concept of pointer	4) No support to pointer
5) One step compiliation process.	6) Two step process- compiliation & Interpretation.
6) less secure	6) More secure.

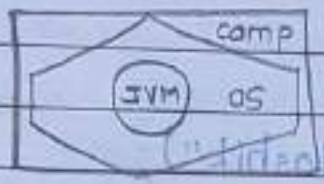
Features of Java:

→ Simple and Object Oriented
→ Platform Independent & Portable

Dir Commands shows all folders/files within particular folder.

Platform Independent: Java code execute on any type of platform

Portable: You can transfer java code from one comp. to another.



Javac program_name.java (compile)

Generation of .class file

(src code → bytecode → Machine code)

This Java command submit ← java program_name that .class file to JVM and interpreter present inside JVM converts that .class file into Binary code (Machine code) → this file executed by OS.

without JVM java is platform dependent.

28/08 Day 5

* Sample Addition program:

```
class Addition
{
    void add()
    {
      int a, b, c;
      a = b = 20;
      System.out.println("Addition " + (a+b));
    }
    public static void main(String args[])
    {
      Addition obj = New Addition(); // object creation
      obj.add() // method call
    }
}
```

.class file always generated with name of class

Day : 6

24/03/

Program

```
public class Student
{
    Void add ()
    String str = new String ("Nashik");
    int if, cm, total;
    if = cm = 50;
    total = if + cm;
    System.out.println (" Total = " + total);
    System.out.println (" City = " + str);

    public static void main ( String [] args)
    {
        Student obj = new Student ();
        obj.add ();
        Teacher objs = new Teacher ();
        objs.show ();
    }

class Teacher
{
    void show ()
    {
        System.out.println (" GPN Teacher's ");
    }
}
```

→ Save file using its classname but which contain main method

→ Compile file, after compilation 2 .class files will generated within program where existing classes

→ Run .class file.

Compiler - whole block / faster / more memory

Interpreter - Line by Line / slower / less memory

25/08

Day : 07

Features of Java :

→ Simple and object oriented
→ Compiled and Interpreted
→ platform Independent and portable
→ Robust and Secure
  ↳ supports so many features
→ High performance
→ Distributed
→ Multithreaded

25/08

Day : 08

Java program structure :

// → single line comment
/* */ → Multiline comment

① Documentation : suggested [dates, authors, title etc]
② Package Statement : optional [way to group classes into single unit].
③ Important statement : optional [way to include other classes into our class.
④ Interfaces Statement : optional

```
interface myinterface
{
    void add();
}
```

⑤ class definition : optional
⑥ Main class definition : Essential
⑦ Define main method : Essential to run program

Program Structure

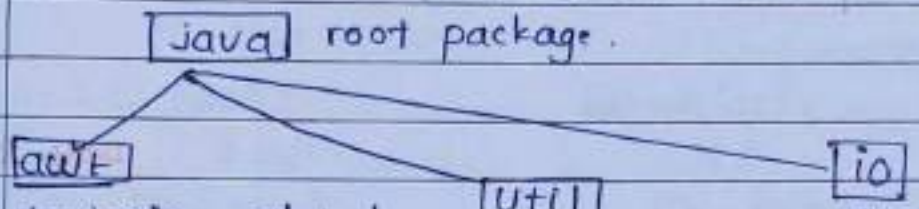
```
// sample program structure

// 1. Documentation

package com.company.student

import java.util.scanner

/* interface My_interface
   {
     void add();
   }
*/

class A
{
}

class B
{
}

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

27 | 08 | 21

Page No.:
Date:
YOUVA

Readymade classes provided by java.

1) java package - Consist core programming classes
2) javax package - Extended. Consist Advance programming classes.



(GUI designing related classes)
Frame
Button
Label
List

calendar
date
Scanner etc.

filereader
filewriter

Command line argument :

Data pass to the program at the time of running is called command line argument. The input data is get stored in the form of string into string array specified in main method.

java programname 10 20 30 Nashik

data pass while running

String array | 10 | 20 | 30 | Nashik |

Conversion of data:

data is in the form of string so to convert it into integer or float we use following function.

① Integer.parseInt(value) — conversion to integer
② Integer.parseFloat(value) / Float.parseFloat(value) — conversion to float

28 Aug

Day-10

In java to find out length of array java provide length property.

```
int a = arr_name.length;
```

a var contains length of array.

If array don't not contain any data and if we try to access array location beyond it will throw ArrayIndexOutOfBoundException

Data types :

Type of data stored in variable.

① Int : → int - 4B
→ Short - 2B
→ long - 8B
→ byte - 1B

② Real → float − 4B   use f at end eg k=2.0f
→ Double − 8B

③ character → char − 2B
④ Boolean → boolean − 1B ⟨ True / False

To store character
① ASCII   ② Unicode
8 byte
[0 - 255]   [0 - 65535]

Day 11

31 Aug **Variable** : Name given to storage area. Named memory location for atomic data (single value)

*) **Types** : 1) Local − Defined inside block
2) Instance − When objects are created.
3) Static − Using static keyword.

* **Rules and convention**

Coding convention : Standard practices.

① package com.companyname.modulename.submodulename
extencion of company domain name

② Classname : Classname must be discriptive. (Holds [illegible] eg. NewClass

③ method name : discriptive. two or more words. first word small case & remaning Camel case. rollNo

Instance — Variables — Local

Class / static

21 sept

Day: 13

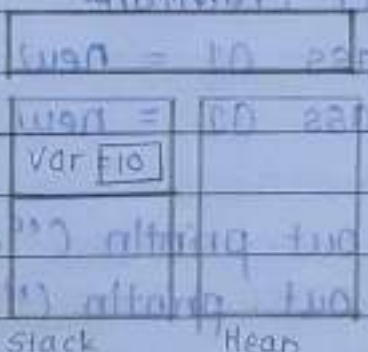
i) Local Variable

→ The variable declared inside body of method.
→ Scope of that variable is only with that particular block.
→ Local variable cannot be static
→ Local variables are implemented internally in method area i.e in stack memory.

Example:

```
class ConceptOfVar
{
    void MyMethod()
    {
        int var = 10;   // Var is local variable
        System.out.println("value = " + var);
    }

    public static void main(String [] args)
    {
        ConceptOfVar o1 = new ConceptOfVar();
        o1.MyMethod();
    }
}
```

Scope of var is local



a) Instance Variable:

→ Instance variables are the variables declared inside class without static keyword

→ They are declared inside class but they are created when object is created and destroyed when object is destroyed.

→ Instance variables can be accessed only by creating objects. (objname.varname)

→ Instance variables are created inside Heap memory but reference to them are present in stack memory area.

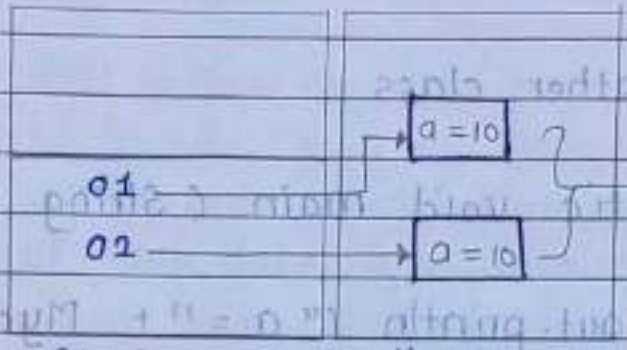
→ They have default values as 0

→ Separate copy of instance variable is created for each object

Sample Program:

```
class MyClass
{
int a = 10;  // a is instance variable.

public static void main (String [] args)
  {
    // Object Creation
    MyClass o1 = new MyClass();
    MyClass o2 = new MyClass();

    System.out.println ("object 1 = " + o1.a);
    System.out.println ("object 2 = " + o2.a);
  }
}
```

common



3) **Static Variables**:

- → Also known as class variable.
- → Declared inside class usind 'static' keyword.
- → They have class level scope.
- → When we have to commonly data in that we can use static variables.
- → It creates same copy for each class.

Sample program 1

```
class MyNewClass
{
    static int a = 20;    // static /class variable
    public static void main (String [] args)
    {
        System.out.println ("a = " + a);
    }
}
```

Sample Program 2 (different)

```
class MyClass
{
 static int a = 10;      // static variable
}
public class Another class
{
 public static void main (String [] args)
 {
  System.out.println ("a = " + Myclass.a);
 }
```

syntax := classname.varnam

program 3

```
class A
{
 void fun ()
   {
   System.out.println ("open your eyes !!");
   }
}

class B
{
static A obj = new A ();   // static object
}

public class ImpClass
 { public static void main (String [] args) {
     B.obj.fun ();
 }
}
```

class static object function of class of static obj

* **Type casting or Type conversion:**

long → int ✓
int → long ×
int → float ✓
float → int ×

→ Conversion of one data type into another is called casting.

→ Assigning one type of value to variable of another type is called Type casting.

04|Sept

Day : 14

Programming lang according to variable
→ ① Loosely coupled
→ ② Tightly Coupled

→ Loosely coupled – General declaration of variable and the type is considered according to value stored in it. - javascript, VBscript.

Var a = 10 — int type
Var a = 'k' — char type

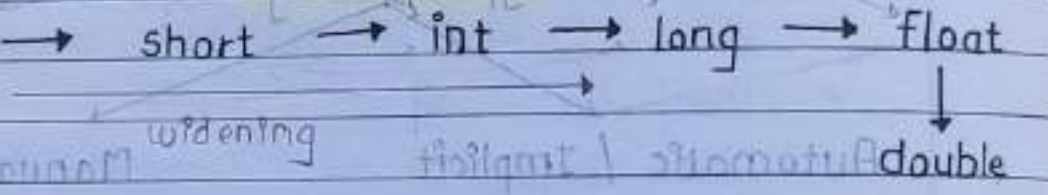
→ Tightly coupled – Variable type is defined at the time of declaration. eg: Java, C++, C (Partial tightly coupled)

int a = 10
float a = 5.5

Type casting :

① Automatic / Implicit / Widening casting :

→ Assigning data type of lower size to data type of higher size.
→ Perform by JVM (Java Virtual Machine).
→ No data loss
It takes place when ?
→ Two types are compatible
→ Target type is larger than source type.



Example :

```
// Automatic casting by JVM
class WideningClass
{
 public static void main (String [] args)
    {
      int a = 10 ;
      long b = a ;   // Int to long
      float c = b ;  // long to float

      System.out.println ("Integer = " + a); // 10
      System.out.println ("long = " + b); // 10
      System.out.println ("Float = " + c); // 10.0
    }
}
```

char to int = ASCII value

② Manual / Explicit / Narrowing

→ Assigning data type of higher size to lower size of data type by explicit casting.
→ It is not performed by JVM, it requires explicit casting performed by programmers.
→ Higher sized narrowed to lower.
→ If we do not perform explicit casting while assigning higher size data to lower then compile throws error

Byte ← Short ← int ← long ← float ← double

Narrowing

Example :

```
// Manual Casting
class NarrowingClass
{
    public static void main ( String [] args)
    {
        double a = 122.66 ;
        long   b =(long)a ; // double to long
        int    c = (int) b ; // long to int

        System.out.println (" Double = " + a ); // 122.66
        System.out.println (" Long   = " + b ); // 122
        System.out.println (" Integer = " + c ); // 122
    }
}
```

Day : 15
6|Sept|2021

Driver - class containing main method
Concept - Functionality classes

* **Literals :**

Literals : i) Boolean
ii) string
iii) Integer
iv) character

anstant values that appear
program It is directly assigned

* Types of operators in Java

operator = Symbol use to perform operation on operands

operand = Value on which operator performs operation

1) Arithmetic operator :
2) Relational -11-
3) Logical -11-
4) Bitwise -11-
5) Assignment -11-
6) Conditional -11-
7) Special purpose -11-

7|Sept|2021

→ **Bitwise operator :**

1) & (Bitwise AND)

Truthtable :

0	0	= 0
0	1	= 0
1	0	= 0
1	1	= 1

example :

10 = "1010"
&12 = "1100"
8 = "1000"

Multiplication is performed

2) | (Or)

Truthtable :

0	0	= 0
0	1	= 1
1	0	= 1
1	1	= 1

eg

10	1010	
|12	1100	
14	1110	

Addition is performed.

3) ^ (XOR)

Truthtable
0 0 → 0
0 1 → 1
1 0 → 1
1 1 → 0

Example :

10	1010
^12	1100
6	0110

4) << (Left shift) : shift all bits to left and insert 0 at end

4 << 1 ⇒ 0100
= 8 → 1000

5) >> (right shift) : shift all bits to right and insert 0 at beginning and ignore last bit

4 >> 1 ⇒ 0100 (ignore)
= 2 → 0010

6) ~ (Not)

Truthtable
0 → 1
1 → 0

Example :

5 = 0101
~5 = 1010 ⇒ (10)

→ Special purpose operator

instanceOf operator :

→ Used for object reference variable.
→ It checks whether the object is of particular class type / interface type.
→ If object is instance of class then return true else gives error

syntax : object_name instanceof cls_name.

→ **Decision making and Branching Statement :**

Flow control Statements

① **Decision Making :**
IF , IF - Else , Else if ladder, Nested if, Nested if-else , switch case

② **Loop control :**
while, do-while, for, for each

③ **Jump control**
break, continue. (goto is not supported by java)

→ **Enhanced for loop i.e for each loop :**

Syntax :

```
for (Datatype Var : List_name)
{
//code
}
```

Program :

```
public class ForLoop
{
  public static void main( String [] args)
  {
    int [] a ={ 10,20,30,40,50};
    for (int i:a)
    {System.out.println (i) ;
     System.out.println ("","");
    }
  }
```

output 10,20,30,40,50,

08/Sept/2021

→ Math class in java :

Math class ⇒ provided under java.lang package

↓ It provides all static method thus we don't need object to call them. we can call them using syntax.

Math.methodname();

① abs() :- Return absolute value of double, long, float, double

Math.abs(-101) ⇒ 101 (absolute value)

② exp() :- returns Euler's number e raised to the power of double value

double x = 5

Math.exp(x) = 148.4131591025766 (result)

③ max() : Returns max values of double, int, float, long data type

Math.max(106, 206) ⇒ 206

④ min() : Returns minimum value of double, int, float, long data type

Math.min(206, 106) ⇒ 106

⑤ round() : Return round up or closet value

⑥ sqrt() : Returns sq. root.

⑦ pow() : Returns power of value

Etc there are many math class functions.

09|sept|2021

CH : 02 CLASSES AND OBJECTS.

* CLASS :

→ Everthing in java is encapsulated in class.
→ Core of java is class
→ It template / blueprint that describes behaviour/state of a particular entity.
→ It is user defined data type.
→ This data type is use to create object of that type.
→ It is declared using 'class' keyword.

state => data members
behaviour => member functions

```
class MyClass
{
    // data members
    // members function
}
```

* Object :

→ An object is entity which has state and behaviour.
→ Object is an instance of class.

Creating object :

Syntax :

```
class_name object_name = new class_name();
```

cls.name obj ;
This statement only create refference variable

reffernce var.
allocates memory area called as object

```
Myclass ob = new Myclass();
```

reffernce variable
allocation of memory is called object

→ Call using object

```
new class_name().data_member;
new class_name().member_function();
```

→ call using refference variable of class type.

```
ref_variable.data_member;
ref_variable.member_function();
```

1) class_name ref_var; ← only creation of ref variable of class type.
2) new class_name(); ← object. (constructor called)
3) class_name ref_var = new class_name(); ← creation of object and refference of that object is get stored in ref_var

19 Sept 2001

→ Array : Collection of similar element.

Syntax :

```
datatype Arr_name = new datatype[size]
```

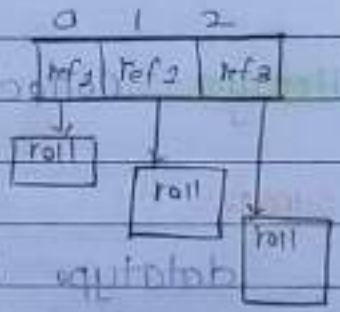
Program :

```
import java.util.*;
class ArrayClass
{
    public static void main(String a[] args)
    {
        int a[] = new int[5];
```

```
Scanner sc = new Scanner (System.in);
System.out.println ("Enter :");

for (int i=0; i<5 ; i++)
 {
  a[i] = sc.nextInt();
 }
```
↳ converts console input into int.

```
System.out.println (" Elements :");
 for ( int i=0 ; i<5 ; i++)
  {
   System.out.println (a[i]);
  }
}
```

→ **Array of objects**: collection objects having same data type.




```
class A
{
 int roll;
}
public class B
{
   public static void main (String args [])
    {
 A ob[]= new A[3];

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

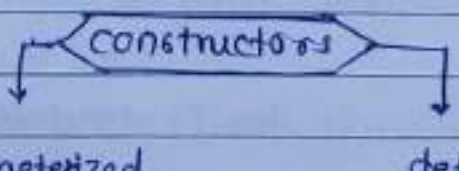
```
        ob[i].roll = 100;
    }
    for (int i=0 ; i<3 ; i++)
    {
        System.out.println(ob[i].roll);
    }
}
```

14/08/2024

Constructor :

- Constructor is special method use to initialize an object members.
- Every class has a constructor.
- If we don't specify constructor explicitily then complier provides default constructor.
- key points :
  i) It does not have return type
  ii) same name as the class name.
  iii) abstrad, static, Final or Synchronized. These modifiers are not allowed for constructors.
  iv) they are invoked automatically at the time of object creation.

* Types of Constructor :



1) **Default Constructor :**

→ If we do not provide any constructor then by default java complier provides default constructor which will set int value as 0, float as 0.0 and String as null.

→ or If you provide constructor without parameters then it is called default constructor.

Program :

```
class MyClass
{
  int a;
  float c;
  public MyClass()  // constructor
    {
      a = 10;
      c = 10.23f;
    }
  public void show()
    {
      System.out.println(" a = " + a);
      System.out.println(" f = " + f);
    }
  public static void main(String[] args)
    {
      MyClass ob = new MyClass();
      ob.show();
    }
```

output

a = 10
c = 10.23

} du to constructor

2) Parameterized Constructor :

* The constructor which has specific number of parameters is called Parameterized Constructor.

program :

```
class NewClass
{
  public int rows ;
  float length ;
  public NewClass ( )
   {
     rows = 2 ;
     length = 5.2f ;
   }
  public NewClass( int r, float l)
   {
     rows = r ;
     length = l ;
   }
  public void show()
   {
     System.out.println (" rows = " + r);
     System.out.println (" length = " + l);
   }
  public static void main (String [] args)
   {
     NewClass o1 = new NewClass (100, 2.3);
     o1.show() ;
     Newclass o2 = new NewClass() ;
     o2.show() ;
   }
```

→ **Constructor Overloading in java:**

→ Technique of having more than one constructor with diff. parameter.

→ Each constructor performs diff. task.

→ Compiler identify them with the help of no. of parameter and thier types.

program :

```
class Cricketers
{ string name;
  String role;
  int totalMatches;
  public Cricketers ()        // First
  {
     name = "Hardik";
     role = "All_rounder";
     totalMatches = 1280;
  }
  public Cricketers ( int M)  // Second
   { name = "Krunal";
     totalMatches = M;
    role = "Bowler"; }
  public Cricketers (String s, String k, int L)
   {
     name = "Krunal"          // third
     role = k;
     totalMatches = 190;
  }

  public void show ()
  {
```

```
System.out.println (name + " " + role1 + " " + totalMatches);
}
public static void main (String [] args)
{
    Cricketers c1 = new Cricketers ();  // first constructor
    Cricketers c2 = new Cricketers (510);  // second
    Cricketers c3 = new Cricketers ("Dhawan", "Batsmen", 1560);  // third
    c1.show();
    c2.show();
    c3.show();
}
}
```

16/09/2021

→ this keyword
- → Used to reffer current object
- → It is always a reference to the object on which method is invoked.
- → Used to invoked current class constructor
- → Can be pass as an argument to another method.

Uses:
- → Used to refer current class object.
- → this() can be used to invoked current class constructor.
- → Used to invoked current class method (implicitly)
- → pass as an argument to method / constructor call
- → Used to return current class object.

Example :

① without this

```
class Sample
{
  int roll;
  Sample (int roll) {
  roll = roll;
  }
  void show()
  {
  System.out.println("roll = " + roll);
  }
  public static void main()
    {
      Sample s = new sample(20);
      s.show()
    }
}
```

Here, expected output is 20 but from above program output will be 0 so solution for above problem is we have to do this

② Use of this

```
class Sample
{
  int roll;
  Sample (int roll)
   {
  this.roll = roll ; }
  void show() {
   system.out.println("roll = " + roll);
}
```

```
public static void main (String [] args)
{
    Sample s = new sample (20);
    s.show();
}
```

output :- roll = 20

⑨ Calling parameterized constructor from default one:

```
class Animal {
    String name;
    public Animal()
    {
        this ("Cow");
    }
    public Animal (String A)
    {
        Name = A;
    }
}
```

④ Call to method:

```
public void show()
{
    System.out.println ("show");
}
public void display()
{
    this.show();
}
```

Remote Method invocation : calling methods of server from client. (But we don't call constructor present on server)

⑤ Use to return current object

```
public cls_name show()
{
    return this;
}
```

⑥ passing current obj to call method of another cls

```
class ThisClass
{
    void show(ThisClass ob)
    {
        System.out.println("ob.a+" is value of a");
    }
}

public MainCls
{
    int a = 100;
    void demo()
    {
        ThisClass o1 = new ThisClass();
        o1.show(this);
    }
    public static void method(String[] args)
    {
        MainCls obj = new MainCls();
        obj.demo()
    }
}
```

output = 100

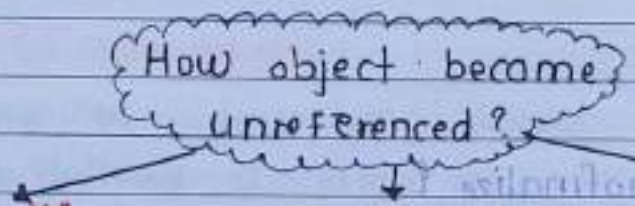
20/sept/2021

method call resolved at compile time

objects are created in Heap/Dynamic memory and referenced to them

Garbage Collection :

- Garbage in java is unreferenced object
- Garbage collection is process of reclaiming the runtime unused memory automatically.
- way of destroying the unused objects.
- Makes java memory efficient
- It is automatically done by garbage collector present in java



① By nulling reference

```
A obj = new A();
obj = null;
```

② By assigning reference to another object

```
A o = new A();
o = new A();
```

③ By anonymous object.

```
new A();
```

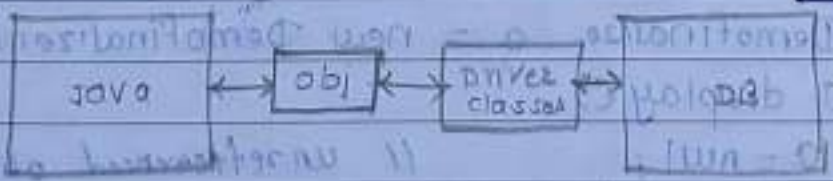
21/sept/2021

- **gc()**
  - Used to call garbage collector explicitly.
  - It request the JVM for garbage collection
  - This method is present in System & Runtime class.

```
System.gc();
```

- **finalize()**:

Object is superclass of all java classes



Provides finalize() method

when object becomes unreferenced, it garbage collector destroyed that object, but that connection with DB remains open and it is not secure thus we have finalize method

→ finalize method is used to perform work just before destroying an object.

→ It performs cleaning work can be closing file handler, database connection, closing network sockets, stopping web services, etc.

→ It get called automatically by gc just before destroying object

→ It's the last chance for any object to perform cleanup utility.

Example:

```
class Demofinalize {
  int roll;
  Demofinalize ()
  {
    roll=1357;
  }
  void display ()
  {
    System.out.println("roll=" +roll);
  }
protected void finalize () //finalize().
  {
    system.out.println ("finalize()");
  }
public static void main(String [] args)
{
    Demofinalize o = new Demofinalize()
    o.display();
     o = null;          // unreferenced obj.
    System.gc();        // garbage collector
```

```
System.in.read();  // waiting mode
int c = 20;
System.out.println("c:" + c);
}
}
```

output : roll = 1351
finalize()
c : 20

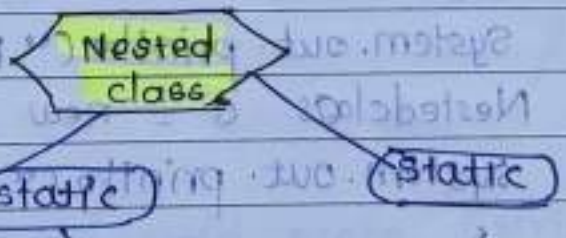
→ **Points to remember :**

- → Present in java.lang.Object class.
- → It is ~~defined~~ declared as protected inside Object class.
- → It is get called only once by GC threads.

22/09/2021

Nested Classes :

→ class defined inside another class is called as Nested class



Member class
- inside any class.

Local class
- inside member fun.
- inside block.

Anonymous class
- class without any name.

1) Static Nested class :

→ It cannot refer non-static member of class without creating object.

static members of inner static class can be accessed outside using classname.datamem;

Program :

```
class NestedDemo
{
    int a=15;
    void display()
    {
        System.out.println("a="+a)
        InnerStatic ob = new InnerStatic
        NestedDemo ob = new NestedDemo();
        System.out.Println("
```

Program :

```
class NestedClass
{
    int a=10;
    static class Inner{
        int b=20;
        public void show()
        {
        System.out.println("b="+b);
        Nestedclass o = new NestedClass();
        system.out.println("a="+ o.a);
        }
    }
public static void main (String [] args)
    {
        Inner ob = new Inner();
        ob.show();
    }
}
```

output :

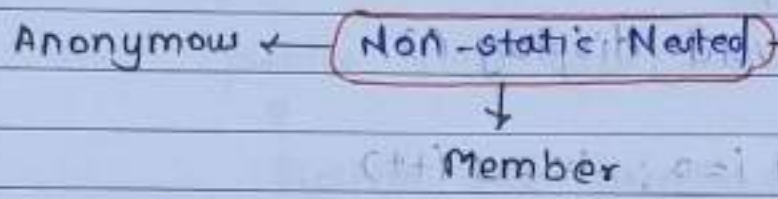
20

10

23/09/2021

2) **Non- Static Nested Class :**

→ It has access to all methods and variable and may refer them directly. but reverse is not true. (outer class cannot directly access members of inner class)



① **Member class :** we can directly access all member of outer class inside member class directly.

```
class Outer {
    void show()
    {
        Inner oi = new Inner();
        oi.display();
    }
// Member class
class Inner
{
    public void display()
    {
        System.out.println("Inner class");
    }
}

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

output :
Inner class

② Local class : class defined inside any block.

→ class defined inside the inner class is also Local class.

```
class Outer
{
  int cnt;
  public void display()
  {
    for(int i=0; i<2; i++)
    {
      class Inner {
        void show()
        {
          System.out.println("Inner " + (cnt++));
        }
      }
      Inner in = new Inner();
      in.show();
    }
  public static void main(String[] args)
    {
      Outer ot = new Outer();
      ot.display();
    }
}
```

output :
Inner 0
Inner 1

③ Anonymous class : class without any name is called Anonymous class.

```
interface Animal
{
  void type();
}
```

```
public class ATest
{
  public Static void main(String [] args)
  {
    Animal an = new Animal()
    {
      public void type()
      {
        System.out.println("Anoonymous class");
      }
    };
    an.type();
  }
}
```

Anonymous class

④ Creation of object of inner class outside outer class :

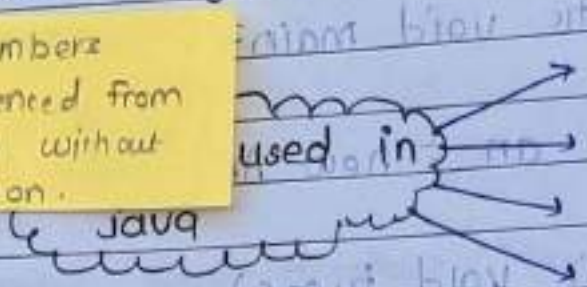
```
class outer {
  public void show() {
    System.out.println("outer method");
  }
  class Inner {
    public void display() {
      system.out.println("Inner method");
    }
  }
}
class Test {
  public static void main(String args[])
  { outer ot = new outer();
    outer.Inner in = ot.new Inner();
    in.display();
    ot.show();
  }
}
```

24/09/2021

Static Members:

↳ static keyword used mainly for memory

non-static members can't be referenced from static context without object creation.



To access members without object creation.

① Static Variables:

- Static variables can be used to refer the common property of all objects.
- It saves memory.
- Accessing static variable inside class
  variable_name;
- outside class
  class_name.var_name;

```
class StaticVar
{
    static int c = 10;
    public static void main (String [] args)
    {
        System.out.println (" c = " + c);
    }
}
```

c = 10

② Static Method:

- It get invocked without object creation
- It access static data member and can change their value.

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

static method

③ Static blocks

- Used to initialize static data members.
- It is executed before main method at the time of class loading
- Initialization before object creation

Syntax :

```
static
{
}
```

Program

```
class StaticDemo
{
  int roll;
  static
  {
    roll = 101;
    System.out.println("Block");
  }
  public static void main(String [] args)
```

```
{
    StaticDemo o = new StaticDemo();
    System.out.println("roll = " + o.roll);
}
}
```

→ Block

→ roll = 101;

27/09/2021

→ Method Overloading :

→ **Method** - Collection of statements group together to perform an operation.

→ Syntax =

```
return_type method_name (Parameter-list)
{
    ---
}
```

→

```
void add (int a, int b)
{
    int c;
    c = a+b
    System.out.println("c = "+c);
}
```

→ Parameter Vs. Arguments :

Parameter : Variable use to define a particular value during a function definition.

Arguments : Values passed during function call.

```
public void add (int a, int b)
{
    --
}
add (20, 10)
```

Parameters

Arguments

→ Call by value and Call by reference.

Call by value - ↳ Copy of an argument value is passed
↳ no effect on the original arguments

Call by reference - ↳ reference of an argument is passed.
↳ changes made inside method affects original argument.

In java, when you pass a primitive type to a method it is pass by value.

→ Call by reference example

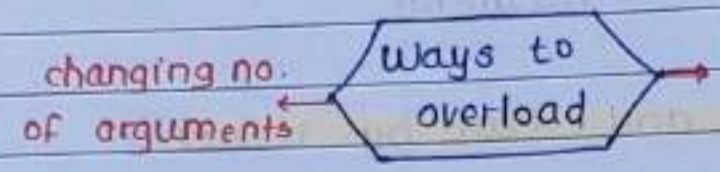
```
public class Test
{
    int x = 10;
    public void call (Test t)
    {
        t.x = 10+5;
    }
    public static void main (String [] args)
    {
        Test obj = new Test ();
        obj.call (obj);
        System.out.println ("x = " + x);
    }
}
```

output = 15

Method Overloading

→ Class having methods with same name but different parameter

→ improves readability.



① changing no of arguments

```
class OverloadingDemoClass
{
void Add (int a)
  {
    int res = a+10;
    System.out.println ("Addition:" + res);
  }
 void Add (int a, int b)
  {
    int res = a+b;
    System.out.println ("Addition:" + res);
  }
 void Add ()
  {
    int res = 15+20;
    System.out.println ("Addition:" + res);
  }
 public static void main (String [] args)
{
  OverloadingDemoClass o = new OverloadingDemoClass ();
```

```
o. Add();
a. Add(10);
o. Add(20, 50);
}
}
```

Addition : 85
Addition : 20
Addition : 70

② Changing data types

```
class OverloadingDemoClass
{
 void add(int a, int b)
  {
   int res = a+b;
   System.out.println("Addition =" + res);
 }
 void add(float a, float b)
 {
   System.out.println("Addition =" + (a+b));
 }

public static void main(String[] args)
 {
   OverloadingDemoClass o = new OverloadingDemoClass();
   o.add(10, 20);          // 30
   o.add(10.2f, 20.3f);    // 30.5
 }
}
```

```
Addition = 30
Addition = 30.5
```

28/09/2021

→ Chain Constructor : Calling multiple constructor by creation of single object.

```
class ChainConstructor
{
   int a;
   String name;
   public ChainConstructor ()
    {
      this (30);
    }
   public ChainConstructor (int p2)
    {
      this (p1, "ABC");
    }
   public ChainConstructor (int p2, String P3)
    {
      a = p2;
      name = P3;
    }
   void show ()
    {
     System.out.println ("a = " + a);
     System.out.println ("name = " + name);
    }
   public static void main (String [] args)
    {
     ChainConstructor o = new ChainConstructor ()
      o.show()
    }
}
```

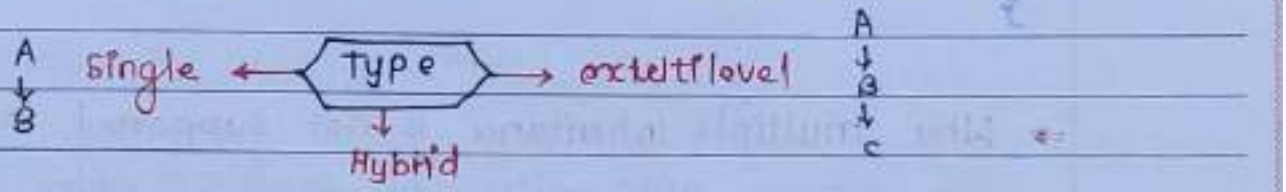
a = 30

name = ABC

→ extends keyword

Inheritance Basics : represents IS-A rlnship (parent-child)

- process in which one class acquires properties of another class.
- class which inherits properties is called sub class
- class from which properties get inherited is called super class.
- In java inheritance is mechanism in which one object acquires all properties of parent object.
- extends and implements are the keywords used to describe inheritance in java.



→ extends : Used to inherit the properties of one class to anoth
inherit the properties of a class.

```
class Super
{
}
class sub extends super
{
}
```

Example :

```
class parent
{
    void demo()
    {
     System.out.println("Parent class method");
    }
}
```

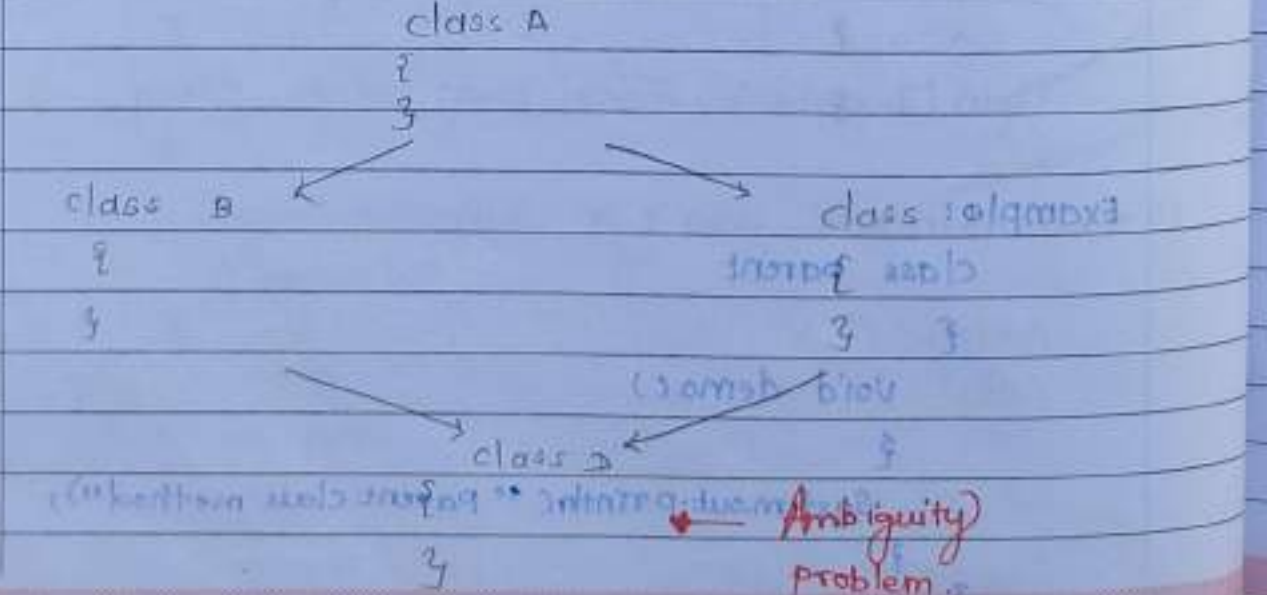
```
Public class child extends parent
{
    public void show()
    {
        system.out.println ("child class method");
    }
    public static void main (String [] args)
    {
        child ob = new child();
        ob.demo()          // parent class method
        ob.show()          // child class method
    }
}
```

→ Why multiple inheritance is not supported in java?

To remove ambiguity, To provide more maintainable and clear design

```
class C extends A, B
```

this is wrong
not allowed



30/09/2021

directly all members as it is access the super except private members of super class in sub class

Is-a relationship : If you are able to access properties of one class into another without object creation then that relationship is called Is-a rlnship.

Has-a relationship : If you access properties of one class to another by object creation then it is Has-a relationship.

① Single level inheritance :

one parent and one child.

```
class A          // super class
{
   int a = 10;
}
class B extends A    // single level
{                    // sub class
    void show()
    { System.out.println("a from super class:"
                +a);
    }
}
class Driver  // driver class
{
  public static void main (String [] args)
    {
      B ob = new B();
      ob.show();
    }
}
```

② Multilevel inheritance :
sub class act as super class for another sub class.

A → B → C

```
class A // multiple inheritance
{
    String fname = "khushi";
}
class B extends A.
{   void show() {
    String Lname = " Nikam";
    System.out.println (fname+" "+Lname);
} }
class C extends B
{
    void demo()
    {
        show();
    }
    public static void main (String [] args)
    {
        C obj = new C();
        obj.demo();
    }
}
```

- Extends – class to class inheritance
- implements — interface to class inheritance

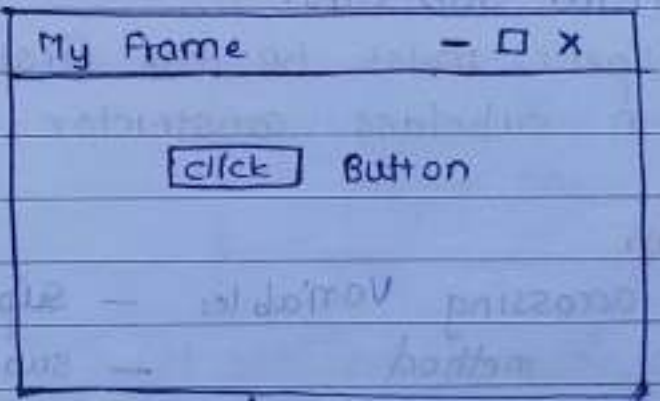
Page No.
Date:
YOUVA

Inheritance example :

* Window creation

```
import java.awt.*;

class MyFrame extends Frame  // Frame is class provided
{                            // under awt package
    public MyFrame()
    {
        setSize(500,500);
        setVisible(true);
        setLayout(new FlowLayout());
        Button b1 = new Button("Click");
        add(b1);
        setTitle("My Frame")
        Label l1 = New Label("Button");
        add(l1);
    }
  public static void main(String[] args)
    {
      Myframe O = new MyFrame();
    }
}
```

output



Applet : Top level window | web browser display.

Frame : Top level window | desktop display

1|10|2021

Page No.:
Date:
YOUVA

→ Superclass default constructor get called through default constructor of sub class after creation of sub class object.

```
class A
{
}
class B extends A
  {
  }
class C extends B
  {                          // D is childmost class
  }                          // A is topmost class
class D extends C            // C immediate superclass
  {                          // A & B indirect superclass
  }
```

SUPER Keyword :

→ Use to access immediate super class members
→ Use to differentiate of super class from its sub class if they have same member name
→ Use to invoke constructor of super class from sub class
→ Super() must be the first statement in subclass constructor.

Syntax :

accessing Variables — super.Varname
method — super.method()
constructor — super() super(param)

Program :

① → Accessing ~~method~~ variable from super class.

```
class super
{
   String name = "Khushi";
}
class sub extends super
{
   String name = "Nikam";
    void show()
    {
      System.out.println("Name from super class=" + name);
      System.out.println("Name from sub class =" + name);
    }
   public static void main (string [] args)
   {
      sub o = new sub()
       o.show()
    }
}
```

Name From super class = Nikam
Name From sub class = Nikam

→ Here we expected khushi thus to solve this problem we Super keyword.

super.name

It will pointing to member of immediate super class.

It will provide output as Khushi

② Accessing super class method :-

```
class Super
{
   void show()
   {
   System.out.println ("from Super class");
   }
}
class sub extends Super
{
   void show ()
   {
   System.out.println ("from sub class");
   }
 public static void main (String [] args)
   {
      sub o = new sub();
      super.show ()          // Super class method
      show()                 // Sub class method
   }
}
```

→ From super class
→ from sub class

4/10/2021

③ Accessing Constructor of Super class :

① Default constructor of super class is get called after creation of object of sub class :

Example :

```
class Super {
    String FName ;
      public Super ( )
       {
         FName = "ABC";
       }
  }
class Sub extends Super {
    String LName ;
      public Sub ()
       {
         LName = "XYZ";
       }
    public static void main (String [] args)
       {
         Sub Obj = new Sub ();
        System.out.println ("creation of object ");
        System.out.println ("Super class constructor is called");
         obj.show()
       }
      Void show ()
       {
        System.out.println (FName + LName);
       }
}
```

ABC XYZ

② To access parameterized constructor of super class, you need to use (super) keyword.

Program:

```
class Super {
    int roll;
    public Super (int roll)
    {
        this.roll = roll;
    }
}

class Sub extends Super
{
    int RollNo;
    public Sub (int RollNo)
    {
Must be → super (195163); // immediate superclass Constructor
first statement
        this.RollNo = RollNo;
    }
    void show()
    {
        system.out.println ("roll =" + roll);
        System.out.println ("RollNo =" + RollNo);
    }
    public static void main (String [] args)
    {
        Sub ob = new Sub (195101); // sub class Constructor
        ob.show();
    }
```

roll = 195163
RollNo = 195101

5/10/2021

Page No.: YOUVA
Date:

Polymorphism

Compile time polymorphism — Runtime polymorphism

① Method overloading ② Method overriding

Method overriding :

If sub class has same method as declared in parent class, it is known as method overriding in java.

→ Usage :

→ Provide specific implementation of a method that is already provided by super class.

→ Rules :

- → Same name
- → parameter must be same
- → Must be in inheritance.

Example.

```
class Animal {
    public void eat ()
    {
        System.out.println ("Animal");
    }
}
class pet extends Animal
{
    public void eat ()
```

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

→ static method cannot be overridden because, a static method is bounded with class and instance method is bounded with object.

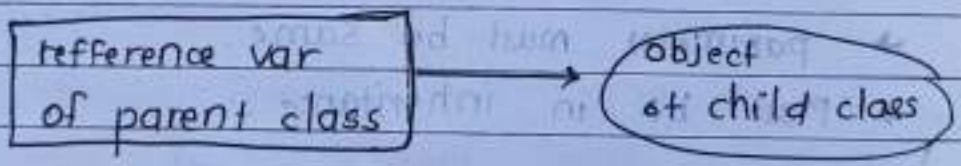
6/10/21

Dynamic Method Dispatch :

→ Mechanism by which call to overridden method is resolve at runtime

→ This is how java implements runtime polymorphism

→ Java determines which version of method to execute based on the type of object refferanced by referrence variable




```
Parent obj = new Child()
```

upcasting → When parent class refference variable refferd to object of child class.

7/10/21

Final keyword :

→ Use to restrict the uses

Can we define constructor as final : ? No we can't define.

→ It can be used in many context → variable, Method, class.

* → value of final variable is constant
* → final method can not be overridden in sub class
* → final class cannot be inherited by any class.

HOLIDAYS 8/10 18/10

ABSTRACT CLASS & METHODS :

WAYS to achieve abstraction in java

i) Abstract class
ii) Interface

i) Abstract class :
+ We can't create object of Abstract class

ii) Abstract Method :
+ It doesn't have implementation.
+ declared with abstract class.
+ subclass must implement abstract method otherwise subclass will be abstract too.

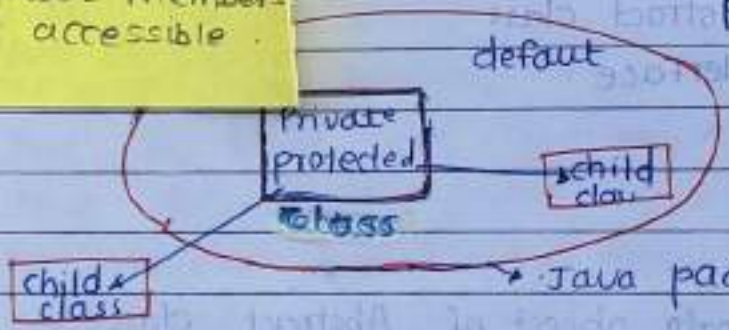
Note : Implement abstract method in subclass without abstract keyword.

→ class containing setter and getter method called bin dao (using set) → (using get)
pass parameter | returns value

→ Visibility Controls:
Java provides access modifiers to set access levels. We can assign limits

↳ Default : Access within same package.
↳ Public : Accessible everywhere
↳ Private : Only within same class
↳ Protected : Within same package as well as sub classes

Different package and default members are not accessible.



↳ package P1.P2.P3;
class Access1
{ int a = 10;
}

P1
↳ P2
↳ P3

Same package and default members

↳ package P1.P2.P3
class Access2
{
public static void main(String [] args)
{ Access1 ob = new Access1();
System.out.println("a = " + ob.a);
}
}

compile and running class From package.
① compile - Javac -d.

↳ If we want to access class present in different package then we have to import that package

class never be protected or private it can be public or default

Java Bin class :- Setter() and getter() methods.

↳ It is class which is public class and it contains all private members

Bin class example :

```
public class Bin_Class
{
    private int Roll;
    private string name;
    public void setRoll(int rn)
    { this.Roll = rn;
    }
    public void setname(String s)
    { this.name = s
    }
    public int Roll getRoll()
    {
        return this.Roll;
    }
    public string name()
    {
        return this.s;
    }
}
```

Setter():
↳ setVarname(para)
↳ keyword

Getter()
↳ getVarname()
↳ get keyword

↳ protected Members - Accessible in same package and all Sub classes.

↳ Private members - Only within same class

19/10/2021 By using super keyword we can access protected & public members

Quick Guide.

	same package	same class	different package	sub class
private	✓	✓	×	×
protected	✓	✓	×	✓
public	✓	✓	✓	✓
default	✓	✓	×	×

STRING CLASS

→ In java String is not primitive type.

String is not primitive.
→ object of string class is immutable.

a.

object contains array of characters.

```
String s1 = "Khushi";
```

↳ compiler will implicitly convert into

```
String s1 = new String("Khushi");
```

Reference Variable

object of String class

string Creation

① using literal =

```
String s1 = "ABC";
```

② using obj Statement

```
String s1 = new String("ABC");
```

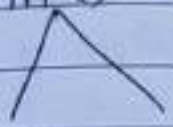
String class methods returns us new object

index of string = start from 0.

charAt() ல index value invalid dili ன exception yeil

string methods

① charAt (index) ⇒ returns char value at index.
② length () ⇒ returns string length. (Count first from 1)
③ string substring (Begin, End) ⇒ returns substring from Begin index to End index.
④ string1.Concat (string2) ⇒ It concats two strings.
⑤ trim () ⇒ It removes leading space and trailing space from string.



Ltrim (left triming) Rtrim (right triming)

```
Str1.ltrim()
Str1.rtrim()
```

⑥ indexOf () ⇒ returns indexOf specified character (First Occurrence)
⑦ toLowerCase () ⇒ converts to lower case
⑧ toUpperCase () ⇒ converts to upper case

STRING BUFFER CLASS

if we have to perform many operations on single string then unnecessary many string objects will created which will leads to memory wastage

↓ so, we can use STRING BUFFER CLASS

stringBuffer class is used to create a mutable string object.

→ It represents growable & writable character sequence.

↓ adding, modifying

↓ adding, modifying

Mutable means - it can be change or modify exsisting string object without creati-

To create string using string buffer we can't use syntax of string literal.

- Syntax of string creation in stringBuffer class :

  → StringBuffer str1 = new StringBuffer ("kn");
  (constructor calling)

- Constructors :

  → StringBuffer() - Empty string Buffer with initial capacity 16
  → StringBuffer (String str) - string buffer with str.
  → StringBuffer (int capacity) - Empty string Buffer with specified capacity as length

→ String Buffer class methods -

① public synchronized StringBuffer append (String s)
appends string in exsisting string object.

→ str1 = "ABC"
str1.append ("XYZ"); → ABCXYZ

② public synchronized StringBuffer insert (int offset, String s)
To insert string
index location जिथे string insert करायची
string जी insert करायची

③ public synchronized StringBuffer
To delete string
delete (int start, int end);
start to delete string
till end

→ Synchronized keyword is use to restrict parallel method calling.

④ public [synchronized] StringBuffer reverse():
to reverse string.

⑤ public int capacity()
It returns the current capacity.

String	StringBuffer
→ immutable Object → slow an consumes more memory as every time it creates new object. → overrieds equals() method	→ mutable object → Fast and consumes less memory → do not overrides equals() method.

Vector Class

→ array contains primitive values of similar type.
→ Collection framework ⇒ contains list of different types of object.

Vector = We can maintain list of different objects.
Arraylist = —— " ——

→ Provides various methods to manage objects list.
→ Provides dynamic list.

Vector (100% increment)	Arraylist (50% increment)
→ Legacy (version 1)	→ class of collection frame work. (Version 1.2) → fast
→ slow	
→ Not provided common mechanism For list	→ Provides common mechanism
→ synchronized	→ Not synchronized
→ Iterator / Enumeration	→ Iterator.

→ Vector class is provided in util package

Example

```
Vector v = new Vector()
v.add (10)
v.add (10.5)
v.add ("XYZ")
A o = new A();
v.add (o);
```

Different types of element

add ×

addElement () ✓

addElement of vector class to add element in list

initial capacity of vector class = 10

→ all element stored in vector as object.

→ Vector is class it implements dynamic array

→ Vector is synchronized

→ Useful when we don't no array size in advanced or changable size of array.

Constructors in vector class:

Increment 100 size when we insert 101 element

→ Vector() - default vector with 10 capacity.

→ Vector (int size) - Creats vector with specified size.

→ Vector (int size, int inc) - Creats vector with incrementation done with

size() - no of elements currently present

Capacity() - How many elements it can hold

C) - Create Vector that contains the elements of collection.

Methods:

① Capacity():

② size():

③ v.addElement (new Integer(1)) : adds element

④ v. firstElement() : Returns first element ↘ Type cast required.

⑤ v. lastElement() : Returns last element ↗

⑥ v.contains() : Whether obj present or not

return → true / false

⑦ Enumeration vEnum = v.elements()

↳ returns Iterator object.

① hasMoreElement = method of iterator. Checks presence

java.lang contains 8 classes

of element.

2) nextElement() : prints element

(8) elementAt() : returns element at index

(9) removeAllElement() : remove all vector elements.

Wrapper Class

→ Mechanism use to convert primitive into object and vice versa

→ Present in java.lang package

int a;

Integer ob = new Integer(a);

autoboxing → automatic conversion of primitive into object.

Integer ob = new Integer(100)
int a = ob;

unboxing → Conversion of objects into primitive types.

Primitive		Wrapper
boolean	→	Boolean
char	→	Character
byte	→	Byte
short	→	Short
int	→	Integer
long	→	Long
float	→	Float
double	→	Double.

primitive type ला obj मध्ये convert केल तर आपण wrapper class च्या methods call करू शकतो

Example:

int a = 10

Integer ob = new Integer(a);

or

Integer ob = a;

conversion of int into Integer class obj.

22/10/2021

```
import java.util.*;
public class Sample
{
   public static void main(String[] args)
   {
      int a = 10;
      Integer ob=new Integer(a);
      System.out.println("Float = " + ob.toFloat
   }
}
```

23/10/21

remove()– method of vector class. To delete element at specified place

elementAt()–to print element of specified location

25/8/21

Interfaces & Packages

PACKAGE : 1] Package definition

→ Group of similar types of ~~object~~ classes, interfaces, enumeration and Sub-packages

→ Use to avoid name conflicts & to control access of class, interface and enumeration

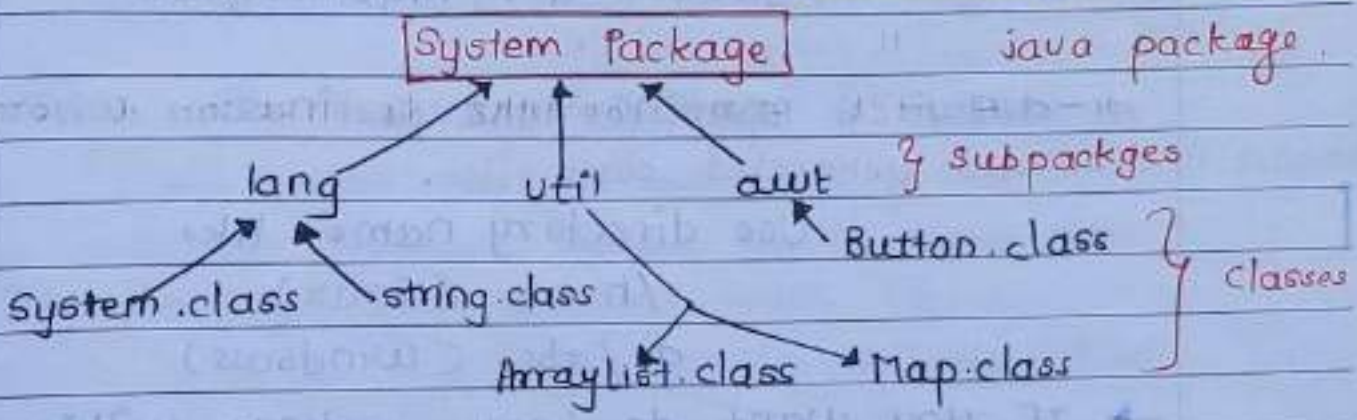
→ It is helpful to locate the related classes

Built-in ← Two forms → User-define

java, lang, sql, net

awt, javax, swing, io, util

Package Use - ① organising files ② Useful for project having multiple packages ③ Resolve name conflict



2] Creating Package :

→ include package command followed by Package name as first statement in program

syntax : pakage PKg_Name ;

or

pakage pack1.pack2.pack3 ;

root/Main package

Sub packages

Example: pakage pack_project ;
public class MyClass {
—II—
}

Java uses File System directory to store package.

Compiling java package -

Two ways of compiling

① By creating Directory -
→ Create director under development enviroment
→ compile java file (javac MyClass.java)
→ put .class file in created directory & Run program (java MyPack.MyClass).

② By using -d attribute -
i) ↳ javac -d diredory javafile.java

Example :

(compiling) >javac -d . MyClass.java

-d switch specifies the destination where to put generated class file.

Use directory name like
/home (linux)
d:/abc (windows)

→ If you want to keep package within same directory, you can use (.)

(Running) ii) ↳ java mypack.Myclass

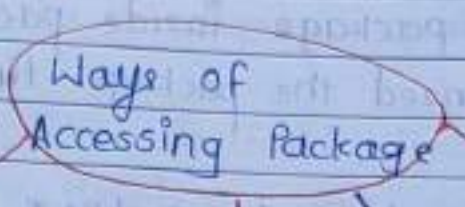
3] Adding class to a package :

पहली package statement
~~दूसरा~~ import statement.

* To add classes into package write package followed by package_name at the top of program

26/10/21

4] Accessing Package :



① import package.* ;
All interfaces and packages will be accessible. (except subpackages)

② import package.cls_name;
only that specified class will be accessible

② fully qualified name (only for a particular block)
Pack.AC obj = new pack.AC()

import pk1.pk2.*;
all packages (pk2) members are accessible

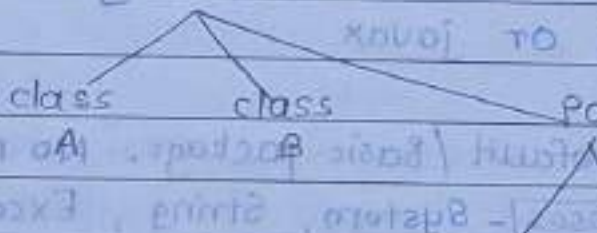
27/10/21

① ⇒ import java.util.*; (all classes)

② ⇒ import java.io.FileReader ; (only File reader)

③ ⇒ java.util.Scanner sc = new java.util.Scanner();

Example



i) import pack1.*
↳ Class A and B

ii) import pack1.pack2.*
↳ class

iii) import pack1.pack2.pack3.*
↳ class D & E

Example :

import pack1.A ;
Import Pack1.C ;

all class use करायचे असतील तर

→ **Subpackage** :

→ package inside package.

→ to categorized the package further.

```
package pack1.pack2.pack3;*;
public class Simple
{
public static void main (String [] args)
{
System.out.println ("Hello");
}
}
```

compile - javac -d . Simple.java

Run - java pack1.pack2.pack3.Simple

output - Hello

SYSTEM PACKAGE

All classes & interfaces that come with installation of JDK are put together are known as API (Application Programming Interface).

All Java API packages are prefixed with java or javax.

java.lang → default / Basic package. No need to import

classes - System, String, Exception, Thread, Wrapper classes, etc.

java.util → Random, Date, Stack, Vector, LinkedList, stack HashMap, GregorianCalendar, etc.

→ Utily classes

java.io → FileInputStream, FileOutputStream, FileReader, FileWriter, RandomAccessFile, BufferReader, BufferWriter, etc

→ I/o operation performed using this.

java.applet → AppletContext, Applet, AudioStub, AudioClip
→ Required for developing applets

java.awt → Button, Choice, Textfield, Frame, List, checkbox, etc
→ Essential for developing GUI applications

java.awt.event → ActionEvent, MouseListener, ActionListener, WindowAdapter
→ Use to handle events generated by GUI component

java.sql → DriverManager, Statement, Connection, ResultSet
→ Database access in JDBC applications

java.Net → Socket, ServesSocket, URL, DatagramPacket, AudioStub, AudioClip
→ For Socket programming

Naming Convention
company name. project name. peosonalname.

STATIC IMPORT
→ static import is feature that expands the capabilities of import keyword
→ Help to import static members of class

① import single static membes
import static java.lang.Math.Sqrt;
② import all static members
import static java.lang.Math.*;
③ Example without static import
Math.Sqrt(9);

28/10

Interface :

limitation of multiple inheritance → ambiguity.

1) Interface definition :-
   - → It is like class containing methods and variable but difference is that an interface can define only abstract methods & final field.
   - → doesn't contain constructors & can't be instantiated.
   - → No method definition (It is responsibility of class to define this method inside them otherwise that class will become abstract class and cannot be instantiated.
   - → Blue print of a class.
   - → Contains static constant & abstract methods only.
   - → Helps to achieve abstraction / Multiple inheritance
   - → Is-A relationship

syntax - interface interface_name { }

Example-

```
interface Interface1
{
    void show(); // only declaration
}
```

Why interface →
- achieve abstraction
- Multiple inheritance feature
- loose coupling.

```
interface print
{
  int min = 5;
  void show();
}
```

public static final keywords before variables

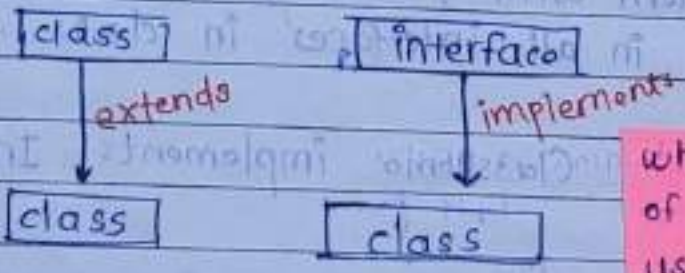
compiler

Java compiles abstract adds public, keyword before interface method

```
interface print
{
public static final int min = 5;
public abstract void show();
}
```

Relationship betn classes interfaces

If we don't inherit interface then there is no meaning of declaring interface



while implementing method of interface in class then use public access specifier

→
```
interface Addition
{
  void add();
}
```

Child class — Parent class

→
```
class Demo implements Addition
{
```

```
public void add()
{
  int a,b;
  a=b=100;
  System.out.println("Addition :"+(a+b));
}

public static void main(String args[])
{
  Demo o = new Demo();
  o.add()
}
}
```

output
200

↳ We can define multiple interfaces in single file. save that file with any one interface name.

multiple inheritance

implementation of more than one interface in single class separate them with , (comma) and define methods present in all interfaces in class.

Syntax ↳ public class ClassDemo implements In1, In2, In3

↳ We can also create reference of interface to store object of subclass/class where we implement interface.

* **Interface inheritance**

extends keyword use to extends one interface in another.

Example

```
interface A
{ void methodname();
}

interface B implements A extends A
{
  void methodname();
}
```

package p1 p2
import —"—
package and import must present before interface in program code

```
interface interface_name
{
}
```

ABSTRACT	INTERFACE
✓ multiple inheritance not supported	✓ supported
✓ extends keyword	✓ implements keyword
✓ may include concrete methods also	✓ all must be abstract methods
✓ All access specifier supporting except private	✓ public only
✓ abstract, and access modifiers should be written	✓ If ommitted public & abstract are taken
✓ main can included	✓ not included
✓ constructor supported	✓ Not supported

31/10/21

4 Multithreading & Exception Handling

Multithreading

→ Executing multiple threads simultaneously

It is light-weight subprocess, smallest unit, Basic component of multithreading, Separate path of execution.

used to achieve: multithreading, multiprocessing → multitasking

Multithreading Advantages:
- Multiple operations at same time
- Many operations together performed so saves time
- Independent (thus doesn't affect other threads if exception occurs in single thread.

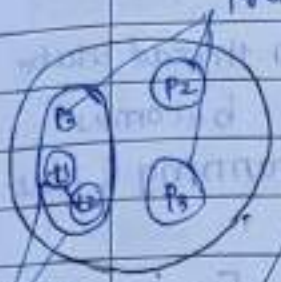
Multitasking

Executing multiple tasks simultaneously

(1) Process-based (Multiprocessing)	(2) Thread-based (Multithreading)
• Each process having separate memory area.	→ Thread share same memory area
• Heavy weight & cost of comm. high	→ Lightweight & cost of comm is low, switching from one
• switching from 1 process to other takes some time!	

What is Thread

- light weight sub process.
- Smallest unit of processing.
- Separate path of execution.
- Threads are independ.
- Shares common memory area.
- Thread is executed inside process
- context switch between threads
- one process can have multiple threads.



At a time only one thread executed.

- Asynchronus programming
  ⇒ parallel execution
- Sequencial programming

Thread class ⇒ provided in java.lang
⇒ Thread class object is called Thread.
⇒ We can convert object to thread.
⇒ Object of Thread class is control internally by JVM.

Life cycle of thread :

→ There are 5 states of thread.
According to sun there are only 4 states (new, runnable, non-runnable, terminates)
→ But for understanding consider 5 states (ie running & all 4 States).
→ Life cycle controlled by JVM.

1. New (born)
2. Runnable
3. Running
4. Non-Running (wait / Block)
5. terminates.

31/10/21

Thread life cycle use round robin scheduling algorithm.

① New/Born - Beginning of thread life cycle. is state until start method

= new Thread(); ← Born thread state

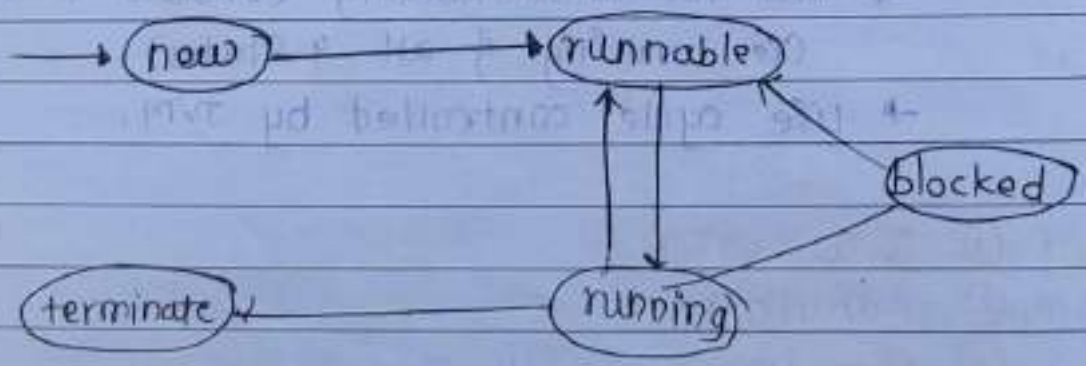
② Runnable - T1.start() → after this thread becomes Runnable. Threads are ready to running, but not running.

③ Running - If Thread scheduler selects thread for executing from the ready queue.

④ Waiting / Blocked - (Non runnable)

- If thread is waiting for another thread to perform task
- Thread remains non runnable until special transition occurs.
- → Thread do not go directly from non-runable to runnable thred for running.
  - ① sleeping
  - ② Blocked for I/o
  - ③ Blocked for join completion
  - ④ waiting for notification
  - ⑤ Block for acquire lock of an object
- → JVM executes the thread based on priority.

⑤ Terminate /dead: after complete execution thread reaches to dead state.



main() Thread

→ Even if you don't create any thread in your program, a thread called main thread is always created.

→ ही जरी automatically create होत असेल तरी, You can control it by obtaining a reference to it by calling CurrentThread() method.

Key points
- It is thread from which other threads will be produce
- It is always last thread to finish execution.

Creating Threads
- extending Thread class
- implementing Runnable interface
- Creating thread object

① Thread Class :

Provides constructors & methods to perform operations on a thread. Thread class extends Object class & implements Runnable Interface.

Constructors - Thread() →
Thread (String s) → logical name to thread
Thread (Runnable r) → converting object to Thread
Thread (Runnable r, String s) →

common Thread class methods :

① public void run - perform action for a thread.

② public void start - start the execution of thread. JVM calls run() method on thredd.

③ public void sleep (long milliseconds)
- currently executing thread will sleep for specified miliseconds.

④ public void join() - It waits for thread to die

⑤ public void join (long miliseconds) - waits for thread to die for specified miliseconds

⑥ public int getPriority()
return thread priority.

⑦ public int setPriority (int priority) -
changes the thread priority

⑧ public String getName() -
returns the name of thread

⑨ public void setName (String name)
changes thread name

⑩ public Thread currentThread() -
returns reference of currently executing thread

⑪ public int getId() - returns thread Id

⑫ public Thread State getState() - returns the state of thread

⑬ public boolean isAlive() - test if thread is alive

⑭ public void yield() - pause current thread & allow other thread to run

⑮ public void suspend() - suspends thread

⑯ public void resume() - resumes suspended thread

⑰ public void stop() - Use to stop the thread

⑱ public boolean isDaemon() - test if thread is Daemon

⑲ public void setDaemon (boolean b) - marks thread as Daemon / User thread.

⑳ public void interrupt() - interrupt the thread

㉑ public boolean isInterrupted() - checks thread has been interrupted.

㉒ public static boolean interrupted() - check if the current thread has been interrupted.

Example Using Thread class - class extends Thread class

Thread class

```
class MyClass extends Thread
{
  public void run()
  {
    System.out.print("Hello!");
  }
  public static void main (String args[])
  {
    MyClass t1 = new MyClass();
    t1.start();
  }
}
```

We have to compulsorily implement run method whenever we implement runnable interface or

जेव्हा आपण Thread class च्या properties inherite करतो आपल्या class च्या object ला threading च behaviour apply होत

Public void run() :
- Can call other methods, can use other classes
- We can declare other

Key points -

① Starting Thread = Use of start() method.

Tasks of start()
- New thread starts with new call stack.
- Thread moves from new state → Runnable
- When thread gets chance it targets to run() method.

Who makes your class as thread object.

When you create object of your class, your class constructor is invoked by compiler from where thread class constructor is invoked (by super() as first statement).

Thus your cls object is thread object.

01/11/21

Runnable interface –

→ જેના આપણામાં પહેલાથી class ના extends પણ કરાયેલ હોય and ત્યાં class મારે multithreading પણ implement કરાવવો હોય

so, we can use Runnable interface instead of using Extending Thread class.

→ Runnable interface consist only run() method

public void run() – use to perform action for a thread

Example – implementing the Runnable interface.

```
class Super {
    void printing()
    {
        System.out.println("Hello! Super");
    }
}

class Sub extends Super implements Runnable
{
    public static void main (String args [])
    {
        Sub o1 = new Sub();
        Thread T1 = new Thread (o1);
        T1.start();
        o1.printing();
    }

    public void run()
    {
        System.out.println("Hello Thread!");
    }
}
```

If you are not extending Thread class then your obj wont treat as Thread object. So you need to explicitly create Thread class object.

Here, We are passing the object of our class that implements Runnable so that our class run() method may execute.

Note: You must have to implement run() method in your class where you have implemented interface

THREAD SCHEDULER =

- It is the part of JVM which decides which thread should run.
- There is no gurantee which thread will be run.
- at a time only one thread from process will run.
- Thread Scheduler mainly uses preemptive or time slicing scheduling to schedule the threads

Scheduling

Preemptive Scheduling	Time slicing Scheduling
→ Highest priority tasks executes until it enters the waiting or dead states or higher priority task comes into pictures	→ Under time slicing, a task executes for defined slice of time & then reenters into ready tasks.

main() < t1 (thread), t2 (thread) — Spawaning

→ METHODS IN JAVA MULTITHREDING :

① Sleep method – It is use to sleep a thread for specified time.

– It is static method.

Syntax to call – Thread.sleep (time in miliseconds)

Example – Thread.sleep (5000);

Here thread will sleep for 5 seconds

↳ at a time only one thread is executed. If you sleep a thread for specified time, the thread schedular picks up another thread & so on

② Naming Thread & Current Thread

1) Naming Thread :

→ Thread class provides methods to change and get the name of thread.

→ By default each thread as thread-0, thread-1 and so on

Instance methods

changing name → threadobj.setName("khushi")

Getting name → threadobj.getName()

threadobj → ThreadObject

2) Current Thread -

It returns reference of currently executing thread.

static method

syntax = Thread.currentThread()

Example = Thread.currentThread();

↳ It returns output in following manner.

Assuming thread name as khushi

[Thread_name, priority, parent_Thread]

↳ [khushi, (5), main]

By default priority is 5

③ stop method -

instance method.

↳ It will stop that thread permanently.

thread_object.stop();

④ interrupt method -

→ interrupt are applied only on executing / Running thread only. (instance method)

t1.interrupt() / t1.isInterrupted / Thread.interrupted (static method)

→ If any thread is in sleeping or waiting state, calling Interrupt method on the thread, breaks out sleeping or waiting state throwing InterruptException.

पण जर thread sleeping or waiting state मध्ये नसेल तर calling interrupt() will behave normally and doesn't interrupt the thread but sets interrupt flag = true.

⑤ join() method.

→ It gives other thread chance for execution and after complete execution of that thread it will starts its own execution, (instance method)

i.e one thread waits for other thread to die.

→ We can also provide time, till what time particular thread will execute.

① t1.join() — complete execution of t1

② t1.join(5000) — execution for 5 seconds of t1

Example

```
class myclass implements Runnable
{
    public void run() {
        System.out.println("Hello");
    }
    public static void main(String args[])
    {
        Thread t1 = new Thread(new myclass(), "t1");
        Thread t2 = new Thread(new myclass(), "t2");
        try {
            t1.join(2000);
        }
```

join method ज्या thread वर call झाली असेल that thread executes for spe

```
        catch (InterruptException e)
        {
            e.printStackTrace();
        }
        t1.start();
        try {
            t1.join();
        }
        catch (InterruptException e)
        {
            e.printStackTrace();
        }
    }
}
```

Join() method give chance to another thread to complete its execution

⑥ isAlive() method -

instance method

Checks is thread is alive or not?
returns false - if termination of thread is done
true - if thread is alive

⑦ Suspend() and resume():

instance methods

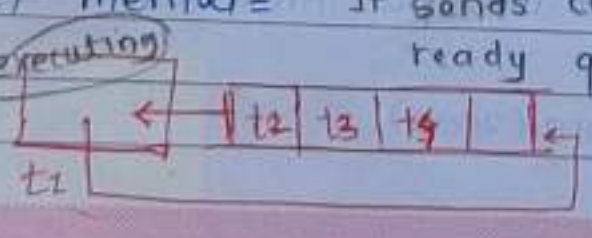
① suspend() : It suspends the thread i.e
→ we can only suspend that thread which is in execution
→ After suspending thread goes in ready queue

② resume() : to resume suspended thread.
We can resume only those threads which are suspended.

yeild and join — join() gives chance to the thread on which it is called but yeild gives chance to the

⑧ yeild() method = It sends current thread in tail of ready queue and executes another thread having priority higher

static



Page No.: YOUVA
Date:

Thread priority

- Priorities are represented by numbers between 1 and 10.
- Default priority of thread is 5 (NORM_PRIORITY)
  Min priority i.e MIN_PRIORITY is 1
  Max priority i.e MAX_PRIORITY is 10

→ These are the three static constants defined in thread class

instance methods

i) getPriority() -
   returns thread priority in integer form

ii) setPriority() -
   It changes the thread priority.

Synchronization

→ Controling the access of multiple threads to any shared resource.

→ Sychronization allow only one thread to access the share resource.

→ In critical resource only one thread can access it

Why Synchronization
- preventing thread interference
- prevent consistency problem

Types
- Process Synchronization
- Thread Synchronization
  - Mutual Exclusive
  - Inter thread communication

sleep and suspend =
suspend thredd res
when we resume it

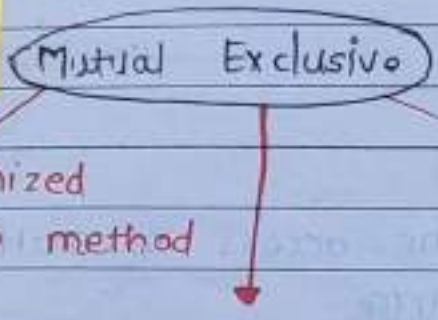
① Mutual Exclusive

→ Using synchronized keyword for particular method, block or using static synchronization that
→ so that block or method can access only one thread at a time

② Interthread communication (Co-operation)

→ Two thread communicates with each other whenever they want to access critical resource.
→ Using some methods like
wait(), notify(), notifyAll()

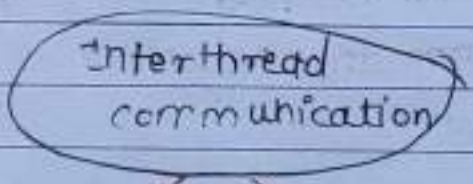
Thread Exceptions =
Security exception - If thread unable to create thread
IllegalArgumentException - negative value



① using synchronized keyword before method

synchronized keyword before static method

② using synchronized keyword before block



wait()
→ other threads will wait for completion for execution of current thread.

Notify()
→ It will notify that execution of thread is completed.

Daemon Thread - Thread executing at background.
- It is service provider thread that provides services to the user thread.

→ When all user threads dies, JVM will terminates this thread automatically.

→ Example: Spell checker, finalizer, gc, etc

→ You can see all details by typing jconsole in command prompt.

jconsole tool provid... about loaded class... memory usage and... running threads, etc...

Why JVM terminates Daemon thread

→ purpose of daemon thread is to provide service for users thred for background supported tasks.

→ so जब user thread लास्ट तक का सपोर्ट Daemon thread run मध्ये देवाचचा so, JVM terminates it.

Instance Daemon thread method

① public void setDaemon (boolean status)
makes current thread as daemon thread if we pass true value as status

public boolean isDaemon()
check current thread is daemon or not and returns true/false.

→ If you want to make user thread as Daemon thread it must not be started otherwise it throws IllegalThreadStateException.

THINK

7/11/21

→ Exception Handling

Exception -

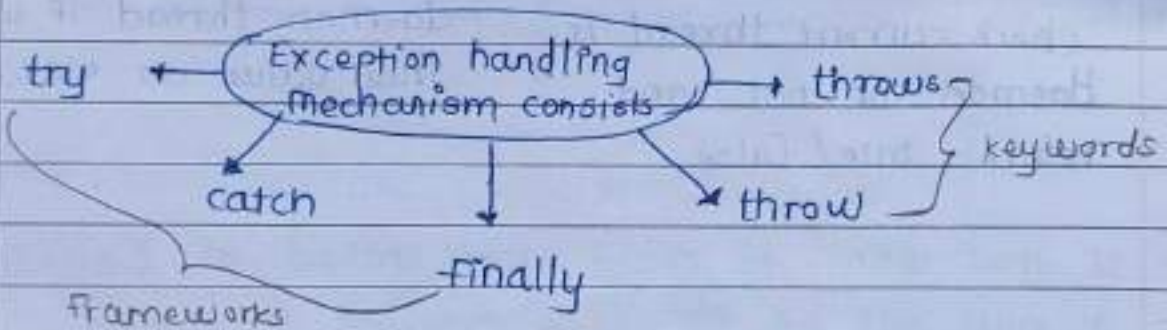
→ Exception is an abnormal condition occurs in a program at runtime & causes abnormal termination of program

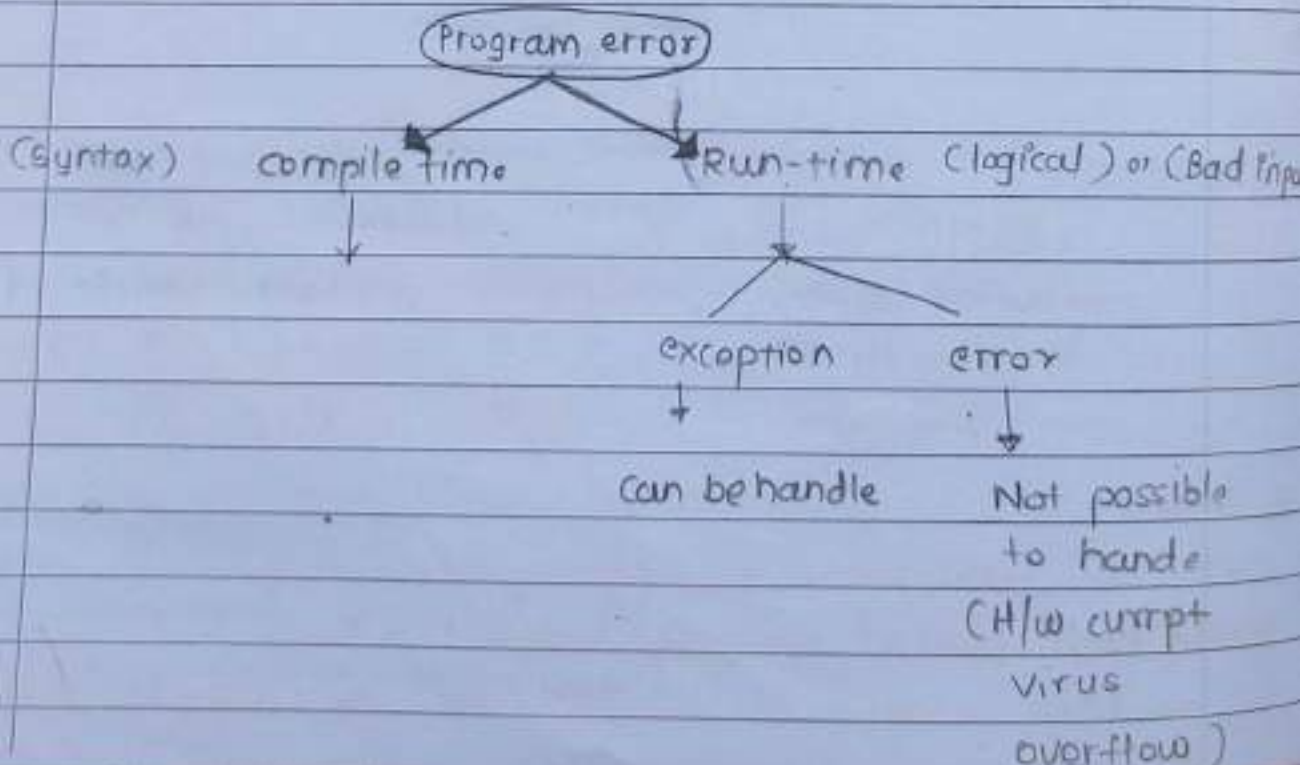
→ It is event that disturbs normal flow of program.

Exception handling :-

→ Mechanism used in java to effectively handle these abnormal conditions & maintains smooth flow of program even if abnormal condition occurs.

* Core advantage of exception handling is to maintain normal flow of application





Java try-catch block :

i) Java try block -

→ It is use to enclosed the code that might throw an exception.

→ It must be followed with catch or finally

```
try {
        ——— 1
        ——— 2
runtime-error ——— (3) ——→ या Line वर JVM identify करेल कोणत्या type ची abnormal
}                          condition उद्भव आणि त्या wise
catch (Exception e) ←      त्या class चा object create
{                          करेल, आणि तो obj throw देईल
}
```

(ex.)

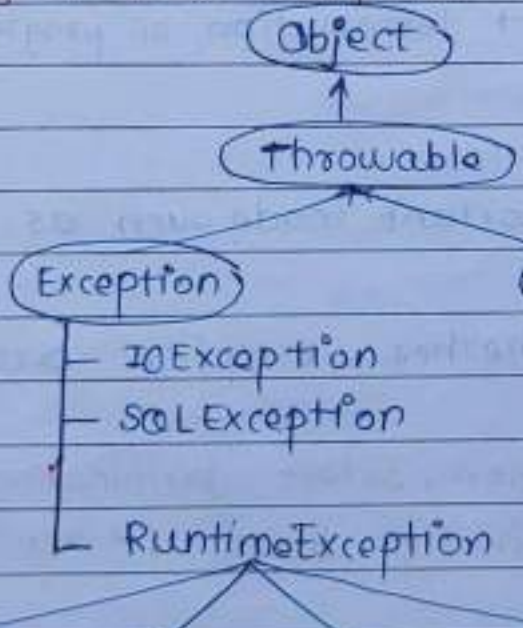
```
throw new ArithmeticException("/ by zero");
```

(String msg)

ही obj catch block catch करेल

```
Exception e = new ArithmeticException();
```

→ Hierarchy of java Exception -



- RuntimeException & all sub classes are unchecked exception
- All sub classes of Excep

ii) Java catch block -

→ Use to handle the exception. Must used after try block only

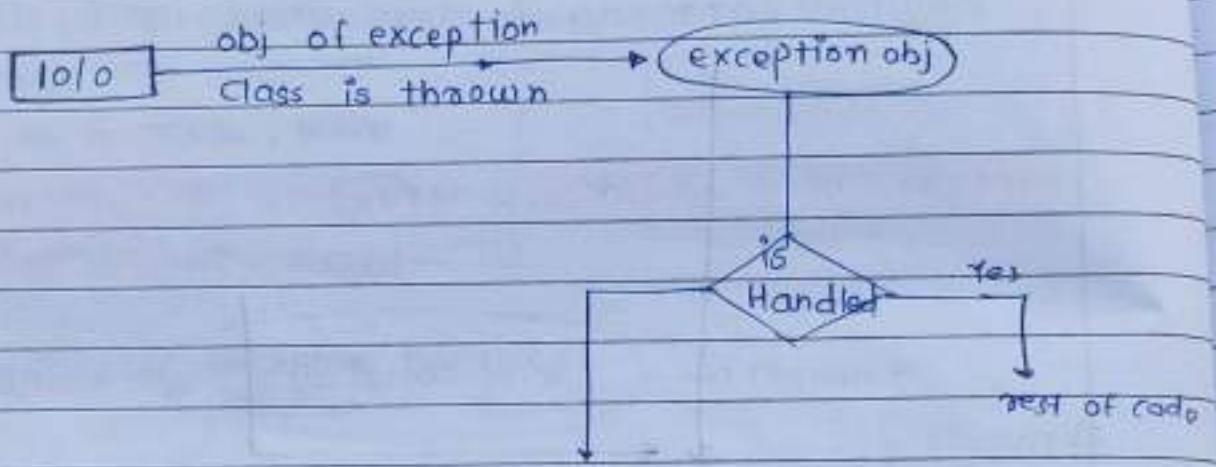
→ multiple catch block with single try

try-catch-finally - Handling exception

throw - manually throwing exception

throws - propogate exception.

WORKING OF TRY-CATCH :



→ exception description

→ prints stack trace

→ termination of program

iii) finally block -

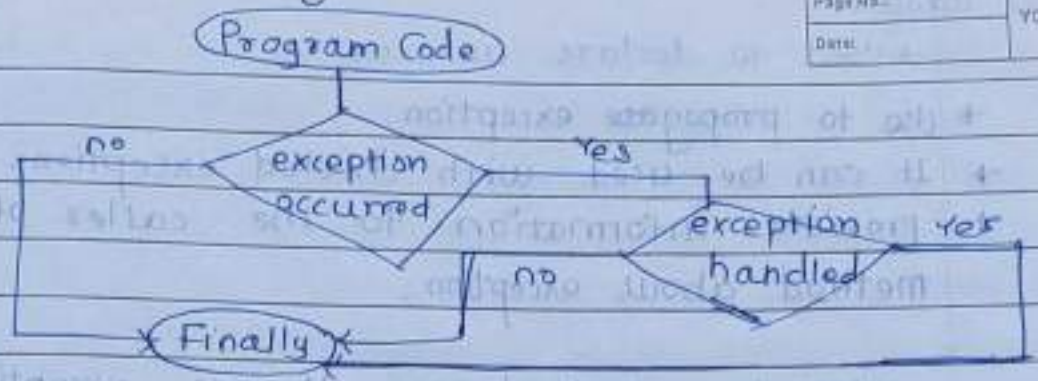
→ used to execute important code such as closing connection, Stream, etc.

→ It always execute whether exception occurred or not.

If you don't handle exception, before terminating the program, JVM executes finally block. (If any).

Used to put "cleanup" code such as closing a File, closing connection, etc.

usage of Java finally :



→ Multiple catch-block :

```
try {
    int a[] = new int [5];
        a[5] = 30/0;
}
catch (ArithmeticException e)
{
}
catch (ArrayOutOfBoundException e)
{
}
catch (Exception e)
{
}
```

→ Here only one exception occurred and only one catch block will execute

→ All catch block must be ordered from most specific to gen

g(ii)

iv) throw =

→ It is use to throw exception (checked / unchecked)

→ Use to throw custom exception

syntax : throw ExceptionClassObject ;

Example : throw new IOException ("Invalid")

v) throws :

→ used to declare an exception.

→ Use to propogate exception.

→ It can be used with checked exception.

→ provides information to the caller of the method about exception.

```
return_type method_name throws exception_class.
{
}
```

throw Vs throws :

	throw	throws
→	Use to explicitly throw an exception	→ It is use to declare an exception
→	Followed by instance	→ Followed by class
→	Used within method	→ Used with method Signature.
→	Cannot throw multiple exception	→ You can declare multiple exception.

10/11/21

Final, finally, finalize difference :

final : Use to apply restrictions

finalize : Used to perform clean up task when object is suppose to destroy.

finally : Used in exception handling, used to perform important task whether exception occurred or not

Java Custom Exception / User defined Exception :

→ Exception created by programmer.

→ You can create custom Exception by inheriting properties of exception class.

PROGRAM

```
class MarksException extends Exception
{
    MarksException (String s)
    {
        super (s);
    }
}
class MyClass {
    static void show ( int marks ) throws MarksException
    {
        if (marks < 35)
            throw new MarksException ("Fail");
        else
            System.out.println (" Welcome to College");
    }
    public static void main (String args [] )
    {
        try {
            show (45 );
        }
        catch ( Exception e )
        {
            System.out.println (e);
        }
    }
}
```

11/11/21

(v) GRAPHICS & INTERNET PROGRAMMING

Internet Programming (Applet):

① JAVA Applet :
→ It is program of java embedded in the webpage to generate the dynamic content.
→ Runs inside the browser and works at client side.

Applet : class extending java.applet.Applet class
→ It doesn't have any main() method.
→ It is not self executable program
→ It can be executed inside the web browser or appletviewer

appletviewer - It is a browser like tool, use to test appet, provided by Java.

Applet class :
→ java.applet package of java contains Applet class which provides life cycle methods for java applet

The methods which are control, executed, called by another application (ie browser appletviewer)

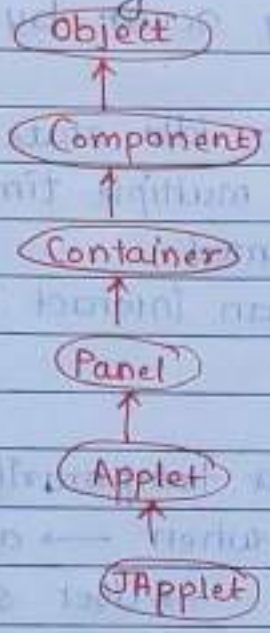
Life cycle methods :
- public void init()
- public void start()
- public void stop()
- public void destroy()

Applet must be public - Because applet program Browser मध्ये Run होतो at

Java.awt.Component class method
- public void paint(Graphics g)

All methods are called as callback.

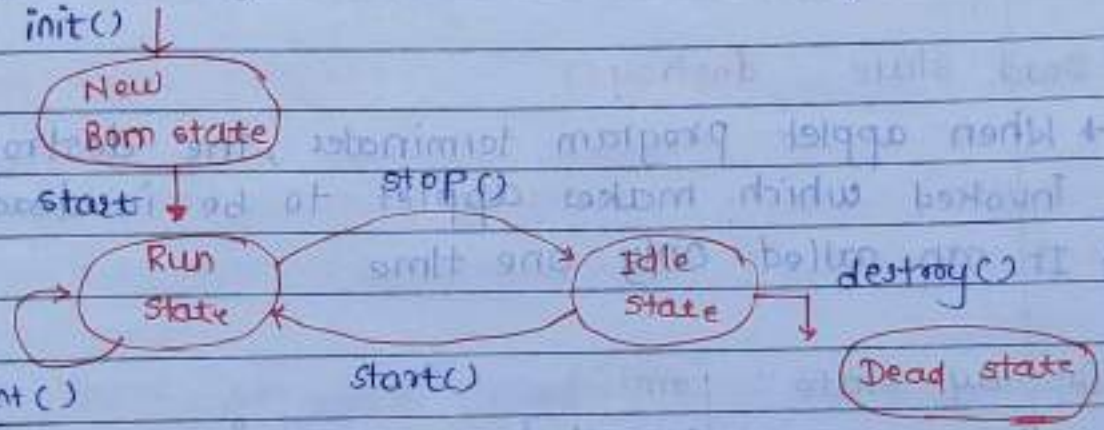
Applet class hierarchy:



→ Object class is super class of all classes
→ Component class provides GUI components
→ Container class provides window which holds other components.
→ Panel is sub container holds GUI components
→ JApplet is a swing version of Applet class.

APPLET LIFE CYCLE :

Applet life cycle methods are called by JVM.



① New Born state : init()
→ life cycle begins when applet is first loaded into browser & called init() method.
→ Called only one time.
→ Called to read the PARAM tag in HTML file.
→ It retrieve the passed parameter through the PARAM tag of HTML file using getParameter() method.
→ Used to initialize
→ User interact with the Applet after initialization.

③ Running state : start()

→ This state will automatically occur by invoking start() method
→ This state also occurs from idle state.
→ This method may be called multiple times when the applet need to be started or restarted.
→ In the start method user can interact within applet.

④ Idle state stop()

→ It halts execution of applet temporarily.
→ Applet moves to this state when → applet minimized
→ User switches to another page.
→ stop method is invoked at this time. (It can be called multiple times)
→ Applet can move to running state from this state.

④ Dead state : destroy()

→ When applet program terminates, the destroy method is invoked which makes applet to be in dead state.
→ It can called only one time.

⑤ Display state : paint()

→ When paint() is called applet is in display state.
→ Used to display o/p on screen.
→ Can be called any no. of times.
→ This method takes Graphic object as argument.
→ Use to draw on applet window.

12/11/21

→ Applet program structure :

```
import java.applet.Applet;
import java.awt.Graphics;
```

```
<applet code = "Applife1.class" width="900" height = "250">
</applet>

public class Applife1 extends Applet
{
    public void paint ( Graphic g)
    {
        g.setColor ( Color.PINK);
        g.fillRect (100,100,300,200);
        g.setColor ( Color.BLUE);
        g.fillOval (150,300, 100,100);
        g.drawOval (250,300, 100,100);
    }
}
```

→ How to run Applet program :

Compile : same as java program
i.e [javac program_name.java]

Running : There are (two) ways.

① Using java compatible browser
→ Create HTML file in same directory.
→ Inside body tag include following code

```
<applet code = "MYApplet.class" width = 400 height = 400>
</applet>
```

Execution :

↳ Create applet compile it
↳ Create HTML file & place applet code in HTML file
↳ And click on HTML file.

② Using appletviewer tool :

↳ Create an applet that contains applet tag in comment and compile it

```
c:\> javac filename.java
c:\> appletviewer filename.java
```

→ <applet> and <param> tag

i) <applet> tag

ii) <param> tag

<param name = "nameofparameter" value = "value of para">

Param tag is used to provide input at the time of loading to the applet.

Application :

Applet	Application
i) Small program	i) Large program
ii) Run a program on client browser.	ii) Can be run on standalone comp system.
iii) portable & can be executed by any Java supported browser	iii) Need JRE, JVM, JDK installed on client machine.
iv) It has 5 methods which will be automatically invoked after occurrence of particular event.	iv) Single start point ie main() method.
v) Created by extending Applet class.	v) created by writing public static void main(String[] args) method.
vi) Not self executable	vi) Self executable.

→ Local applet vs Remote applet

Local applet :

→ Stored and applet developed in local system.

→ Web page will search the local sys. directories,

find local applet & execute it. (doesn't require internet connection)

<applet codebase = "path" code = "NewApplet.class" width = 120 height = 200 > </applet>

Remote applet

→ developed and stored in remote system.

→ Requires internet connection to locate and load applet from remote computer.

<applet codebase = "https://gpn.com" code = "NewApplet.class" width = 120 height = 200 > </applet>

13/11/21

Graphics Programming:

Graphics class methods:

i) public abstract void drawString (String s, int x, int y)
- draws string

ii) public abstract void drawRect (int x, int y, int w, int h)
- draw rectangle / square

public abstract void fillRect (int x, int y, int w, int h)
- fill rectangle with color.

iii) public abstract void drawOval (int x, int y, int w, int h)
- draws oval
- same w & h then draws circle

public abstract void fillOval (int x, int y, int w, int h)

iv) public abstract void drawLine (int x1, int y1, int x2, int y2)
- draws line.

v) public abstract void drawPolygon (int x[], int y[], No_of_points)
array.

v) public abstract boolean drawImage (Image img, int x, int y, ImageObserver observer)
- used to draw specified image.

vi) public abstract void drawArc (int x, int y, int w, int h, int startAngle, int arcAngle)
-11- fillArc -11-
Arc drawn in clockwise direction

vii) public abstract void setColor (Color c)
- used to set graphics current color to specified color.

viii) public abstract void setFont (Font f)
- used to set graphics current font to specified font.

Introduction to AWT Classes :-

1) Color -
→ Constants of Color class :
+ black, red, green, yellow, blue, pink, orange

→ Important Constructors :
Color (int, int, int) — value betn 0-255
Color (int rgbValue)
Color (float, float, float) — 0.0 - 1.0

→ Important Methods
→ Color brighter ()
→ Color darker ()
→ Int getBlue ()
→ Int getGreen ()
→ Int getRed ()
→ Int getRGB ()

→ Setting the current Graphics class :
Void setColor (Color newcolor)
Color getColor ()

→ Foreground & Background of frame or component :
→ Color getBackground ()
→ Color getForeground ()
→ void setBackground (Color c)
→ void setForeground (color c)

Font :-

Font have family name (General name) - courier
logical Font name (category) - Monospaced
Face name (specific font) - Italic

→ Important Constructor
Font (String name, int style, int size)
Courier — Font.Style (Italic, Bold) — Size of font

→ Important methods :
→ String getfamily() — returns font family
→ String getFontName() — returns face name
→ String getName() — returns logical name.
→ int getSize() - size of font
→ int getStyle() - style of font
→ Boolean isBold() - } Returns true or false
→ — " — isItalic() -
→ — " — isplain() -

→ Setting or obtaining font :
Font getfont()
void setfont (Font f)

14/11/21

* Available fonts :

getAvailableFontFamilyNames () =
→ Returns availabe supported Fonts of java.
→ Returns array.

```
import java.awt.*;
class AppletEx
{
public static void main (String args [])
{
GraphicsEnvironment ge = GraphicsEnvironment.getGraphics
                         LocalEnvironment ();
String font [] = ge.getAvailableFontFamilyNames();
for ( String f : fonts)
{
 System.out.println (f);
}
}
}
```

→ SWING

16/11/21

Swing :

Creation of swing application :

→ Inherit properties of JFrame class
→ Set basic properties of new frame such as size, title, visibility.
→ Create object of components which you want to use.
→ set basic properties of components
→ Register event with component if required
→ Add components on Frame using add() method.

javax.swing.*

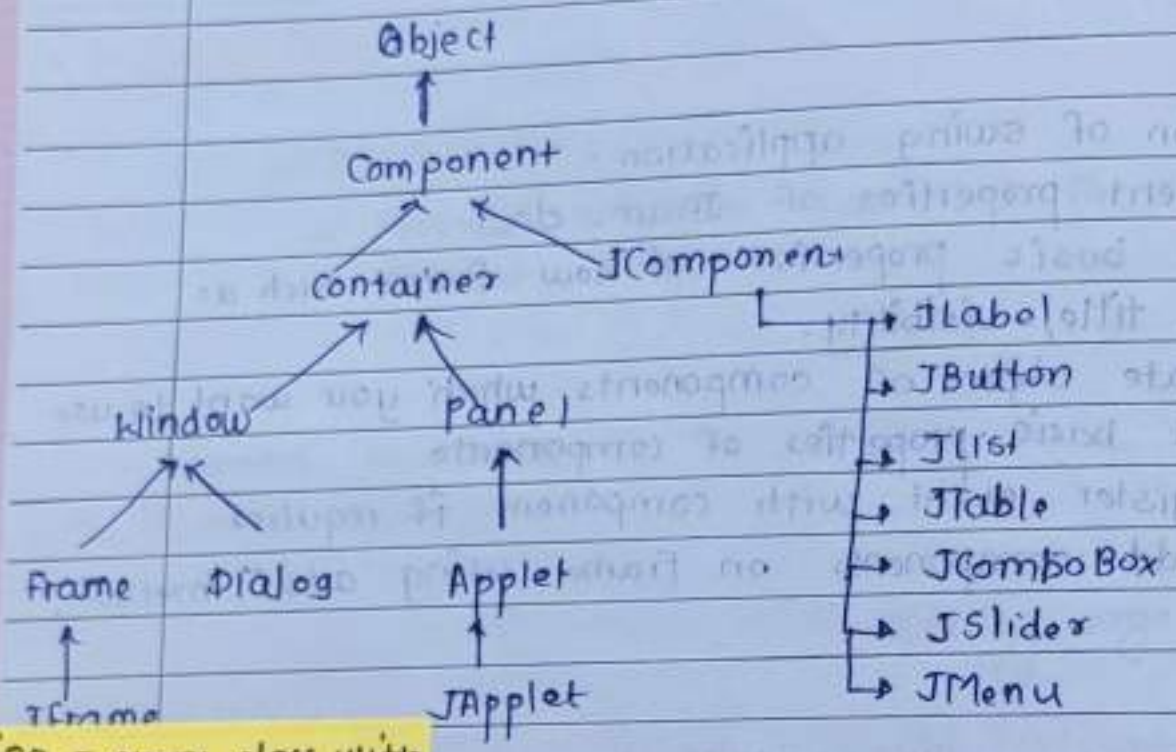
container → container hierarchy (container / subcontainer)

component → stores components (Button, label, etc)

→ API (Collection of classes & interfaces)
→ Use to design GUI based application
→ provides platform-independent & lightweight graphical components such as button, label, textfield
→ Javax swing package provides classes for java swing API such as JButton, JTextfield, JTextArea, JRadioButton, JCheckBox, JMenu, JColorchooser

Features:
- Platform independent
- Customizable
- Extensibe
- Configurable
- Lightweight.

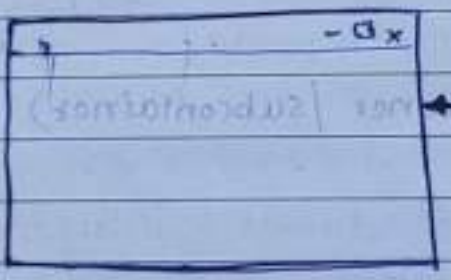
swing classes



swing = every class with J

awt = no J

Swing API



(Rectangular surface) contains components

manually (x, y, width, height)

Automatic

Use setLayout() method

→ pass null parameter to setLayout, to add component manually.

Automatic layout

FlowLayout

BorderLayout

GridLayout

CardLayout

Program for manual layout

```
import java.awt.*;
import javax.swing.*;

class Sample2 extends JFrame
{
    public Sample2
    {
        setTitle("Program");
        setSize(400,400);
        setVisible(true);
```

} Basic properties

```
        setLayout(null);
```

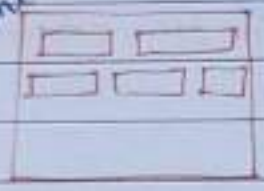
} Here parameter is null thus we can add components manually

```
        JLabel l1 = new JLabel("Name:");
        l1.setBounds(20,20,70,10);
        add(l1);

        JTextField t1 = new JTextField();
        t1.setBounds(100,20,50,10);
        add(t1);

        JButton b1 = new JButton("Done");
        b1.setBounds(20,30,50,20);
        add(b1);
    }
public static void main(String args[])
    {
        Sample2 s = new Sample2();
    }
}
```

Problem with manual layout :
जेव्हा आपण window resize करतो, ते components automatic adjust होत नाही, minimize केली window तर ते hide होतात.
so soln is to use automatic layout.

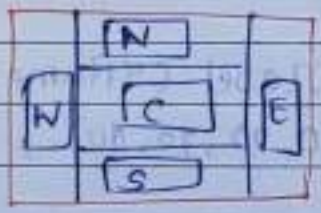
Automatic Layout :-
Following are layout manager class

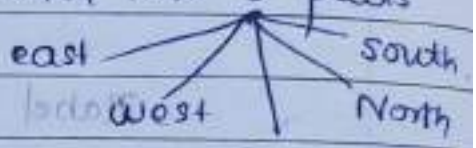
1) Flowlayout - arrange components like text, ie right, left or center arrangement.



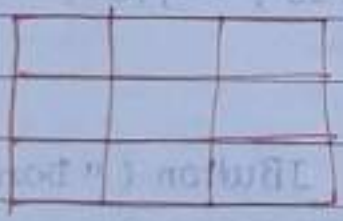
we can pass these as parameter while creating object of Flowlayout

2) Borderlayout - container gets divided into 5 parts





3) Gridlayout - container gets divided into row / column



4) Cardlayout -

18/11/21

File I/O & Collection Framework

Java I/O and streams :

Java I/O -

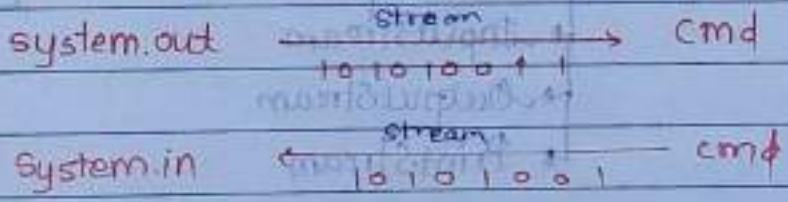
- Used to process the input and produce output based on the input.
- To make I/O operation fast, Java uses the concept of stream.
- java.io package contains all the classes required for input / output operations.
- We can perform file handling in java by java io API

Stream :

- A stream is a sequence of data
- stream is composed of bytes.

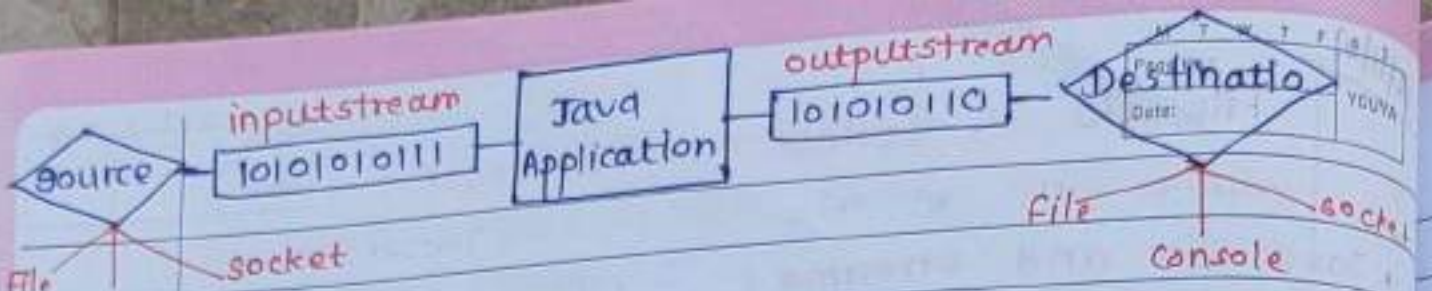
In java 3 streams are created for us automatically.

- All streams are attached with console.
  - System.out : standard output stream
  - System.in : standard output stream
  - System.err : standard error stream



① OutputStream :
Use to write data on destination (data → file, array, peripheral, device or socket)

② InputStream :
Use input stream to read data from a source.



Java encapsulates stream under java.io package. Two types of streams in java:

Byte stream

- Helps to handle input and output of byte
- one symbol = 8 bits

Character Stream

- Helps to handle input and output of characters
- One symbol = 16 bits.
- Unicode sy

ByteStream

- InputStream
- OutputStream

Character Stream

- Reader
- Writer

1] Byte Stream Classes

Byte stream classes →
- BufferedInputStream
- BufferedOutputStream
- DataInputStream
- DataOutputStream
- FileInputStream
- FileOutputStream
- InputStream
- OutputStream
- PrintStream

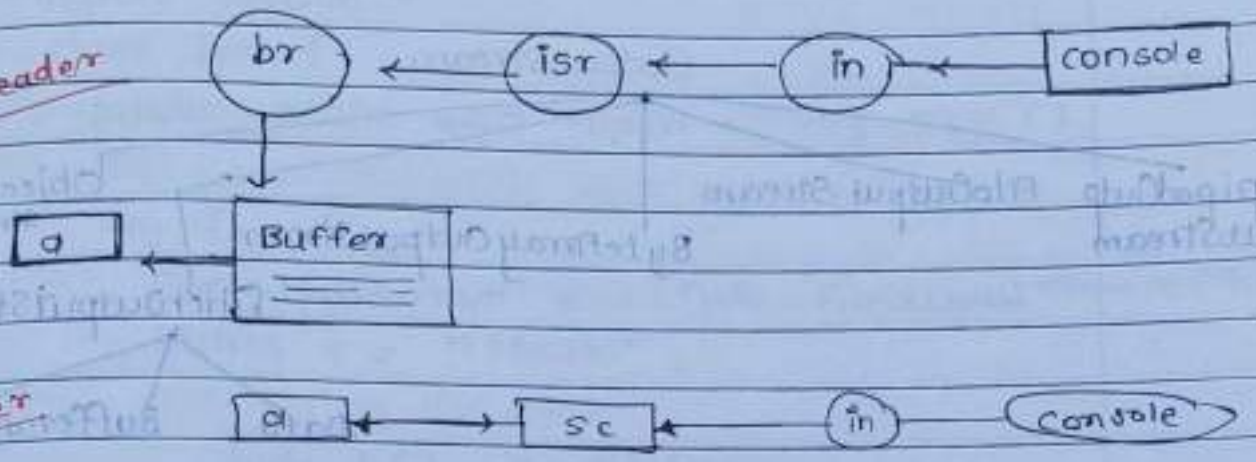
Two key methods:
read()
write()

2] Character Stream Classes:

- BufferReader
- BufferedWriter
- FileWriter
- FileReader
- InputStreamReader
- OutputStreamReader
- PrintWriter
- Reader
- Writer

3] Reading Console input :

BufferedReader br = new BufferedReader (new InputStreamReader (System.in));
↳ converts bytes to char



i) Reading Characters : (Reading method : read())
→ Used with BufferReader object to read characters
→ need of typecast to convert it into char as read() returns integer type value.

```
char c = (char) br.read()
```

ii) Reading Strings
readline() used with BufferReader object.

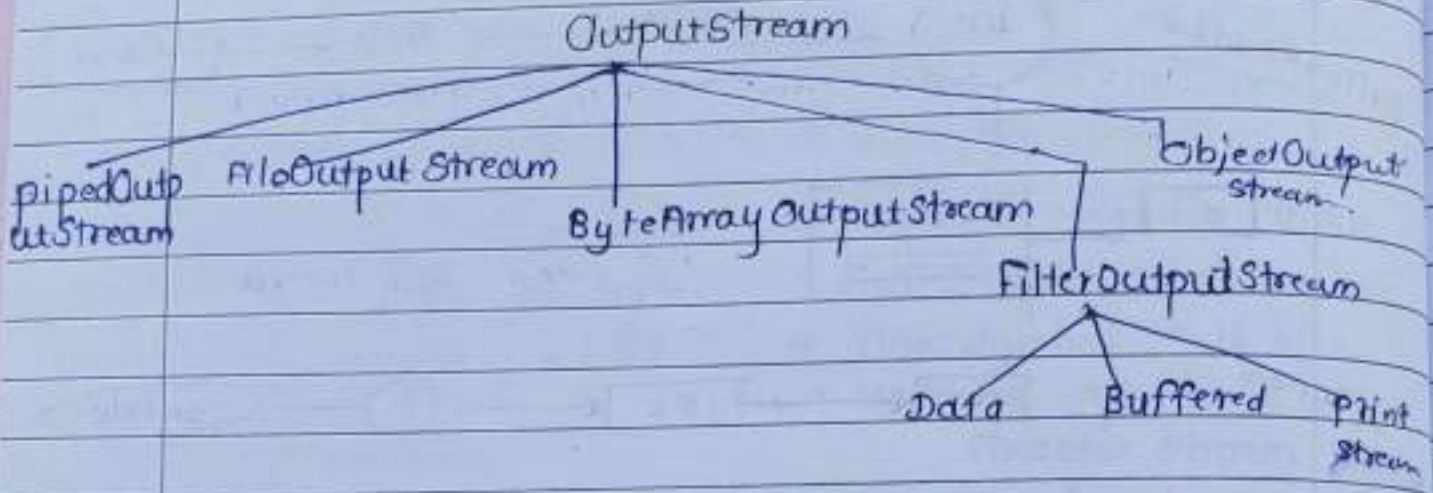
```
String s = br.readLine();
```

* Byte Stream Classes : ~~(Main~~
① OutputStream & InputStream : (Main classes)

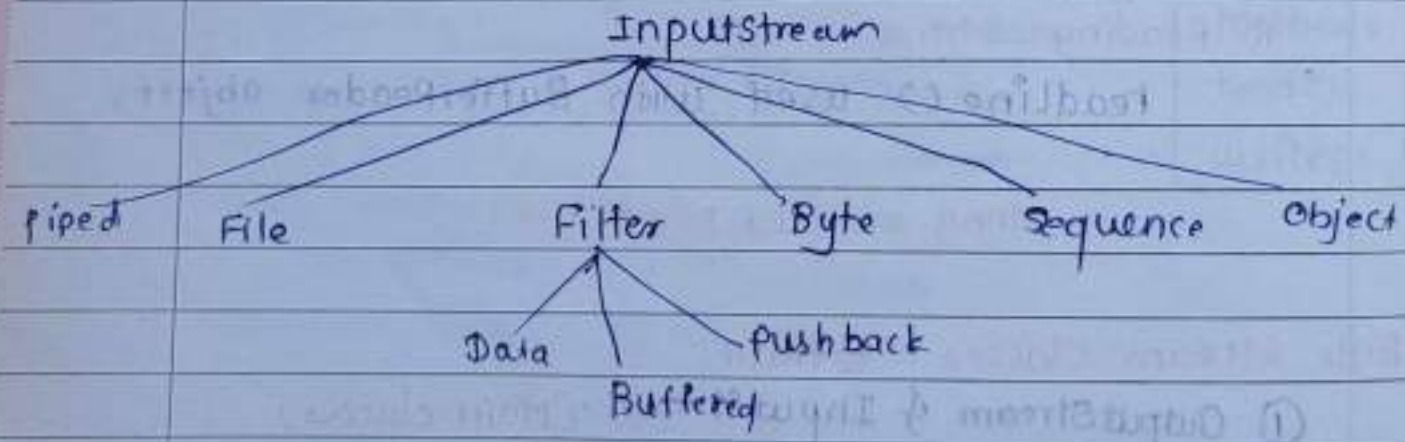
OutputStream :
abstract class & superclass of all classes representing an output stream of bytes.

Methods :

write (int) = use to write byte to current output stream
write (byte[]) = — " — array — " —
flush () = flushes the current output stream.
close () = close current output stream.



InputStream :
abstract class, SuperClass of all input stream of bytes
read()
available()
close()



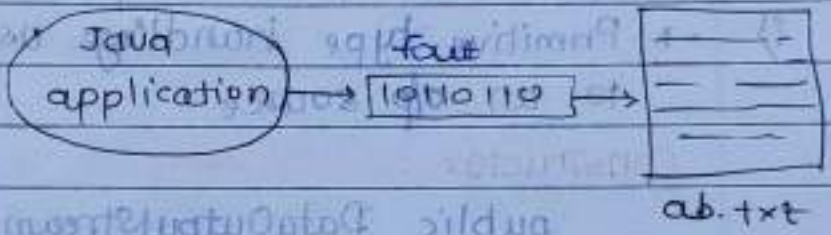
② FileInputStream and FileOutputStream (Handling)
→ Use to read and write data to or from file
i.e file handling.

FileOutputStream :

→ Used to write data to file.
→ To write primitive values

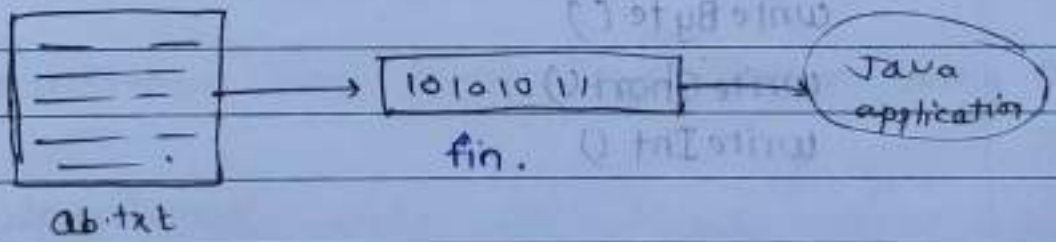
Example :

```
import java.io.*
class Test {
public static void main (String args[])
{
try {
FileOutputStream f = new FileOutputStream("ab.txt");
String s = "Khushi";
// String to byte
byte b[] = s.getBytes();
f.write(b);
f.close();
System.out.println("Done...!");
} catch (Exception e)
{
}
}
}
```



FileInputStream :

→ Used to read data from file.
→ obtains input bytes from a file.



fin.

using read()

→ Reading data from current file & writing it to another file

```
import java.io.*;
class Temp{
public static void main (String args[]) throws Exception
 {
FileInputStream fin = new FileInputStream ("ab.txt");
FileoutputStream fout = new FileOutputStream ("cd.txt");
int i=0
while (i=fin.read() != -1)
    {
      fout.write ((byte)i);
    }
   fin.close();
 }
}
```

③ DataInputStream & DataOutputStream (Primitive type handling)

i) → Primitive type handling use dto write primitive to an o/p source

Constructor :

```
public DataOutputStream (Outputstream out){ }
```

* Some methods

```
writeBoolean()
writeByte()
writeShort()
writeInt()
```

ii) Used to read primitives.

Constructors :

```
public DataInputStream ( InputStream in)
```

* some methods:
→ readByte()
→ readBoolean()
→ readShort()
→ readLong()

Program:

```
import java.io.*;

public class Temp{
public static void main(String args[]) throws IOException
{
    DataOutputStream do;
    do = new DataOutputStream(new FileOutputStream
                                ("abc.txt"));
    do.writeUTF("Hello");

    DataInputStream di;
    di = new DataInputStream(new FileInputStream("
                                abc.txt"));
    while (di.available() > 0)
        {
            String k = di.readUTF();
            System.out.println(k);
        }
}
}
```

Writing into file

Reading from file

④ BufferedOutputStream and BufferedInputStream (Using Buffer)

i) BufferedOutputStream:
→ Uses Internal Buffer to store data.

ii) BufferedInputStream:
→ —11— to read data.

Program :

```
import java.io.*;
class Temp {
  public static void main (String args[]) throws Exception
  {
    FileOutputStream fout = new FileOutputStream ("f1.txt");
BufferedOutputStream bout = new BufferedOutputStream (fout);
    String s = "Hello World";
    byte b[] = s.getBytes();
    bout.write(b);
    bout.flush();
    bout.close();
    fout.close();
    System.out.println("done!");

    FileInputStream fin = new FileInputStream ("f1.txt");
BufferedInputStream bin = new BufferedInputStream (fin);

    int i;
    while ((i = bin.read()) != -1)
      {
        System.out.println((char)i);
      }
    bin.close();
    fin.close();
  }
}
```

writing to file using buffer

reading from file using buffer

⑤ PrintStream :

→ It provides methods to write data to another stream

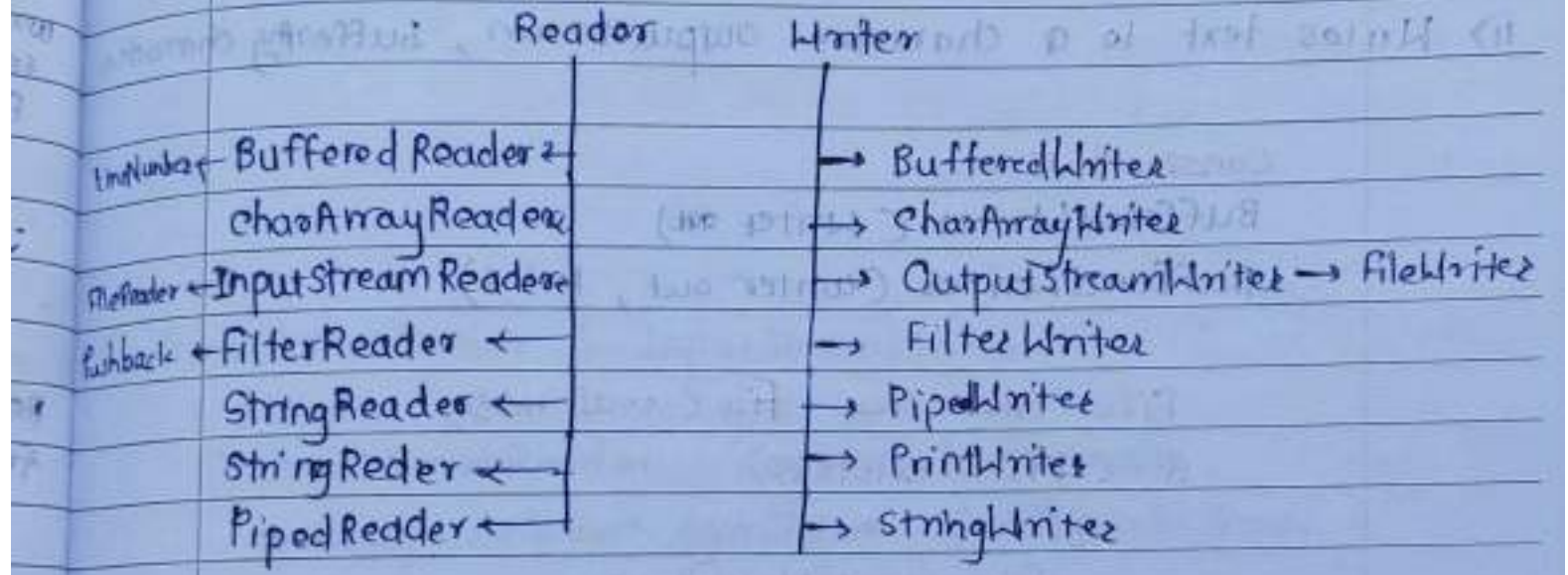
→ It automatically flushes the data.

→ doesn't throw IOException.

Methods :

print / println (boolean b)
print / println (char c)
print / println (char[] c)
print / println (int i)
print / println (long l)
print / println (float f)
print / println (double d)
print / println (String s)
print / println (object o)

→ Character Stream Class :



① Reader & Writer :

① Reader :
abstract class for reading character stream

② Writer
abstract class for writing character stream

② BufferedReader & BufferedWriter.

i) Reads character - input stream, buffering characters.
- → Buffered size may be specified, or may be used default.
- → BufferedReader extends Reader.

Constructors:
BufferedReader (Reader in) - default size buffer
BufferedReader (Reader in, int i) - buffer with i size

```
InputStreamReader r = new InputStreamReader (System.in);
BufferedReader br = new BufferedReader (r);
```

ii) Writes text to a character - outputstream, buffering characters

Constructors →
BufferedWriter (Writer out)
- BufferedWriter (Writer out, int i)

```
File file = new File ("ab.txt");
if (!file.exists())
  {
    file.createNewFile();
  }
FileWriter fw = new FileWriter ("abc.txt");
BufferedWriter bw = new BufferedWriter (fw);
```

③ FileWriter & FileReader
- → File handling
- → If you have to read or write textual information from file then use FileWriter & FileReader classes instead of FileInputStream & FileOutputStream classes.

① FileWriter :
use to write character oriented data to file.

Program :

```
FileWriter fw = new FileWriter ("abc.txt");
fw.write ("Hello");
fw.close();
```

② FileReader :
use to read data from file.

```
FileReader fr = new FileReader("abc.txt");
int i;
while ((i = fr.read()) != -1)
{
    System.out.println ((char) i); }
fr.close;
```

④ InputStreamReader & InputStreamWriter

① InputStreamReader : Converts byte→character.
bridge between byteStream & CharacterStream.

② OutputStreamWriter : Converts character → byte

Scanner:
used to read input

nextInt()
nextLine()
nextByte()
nextShort()
nextInt()
nextLong()
nextFloat()
nextDouble()

→ Serialization & De-serialization:
→ process of converting an object into a sequence of bytes = Serialization
→ reverse process of creating object from sequence of byte = desenialization
→ class must implement Serializable interface.

① Marker Interface
→ Interface without any field & method
→ It informs compiler that class implementing it has special behaviour.

some example
→ Serializable
→ Cloneable
→ Remote
→ Random Access

② Signature of writeObject() and readObject()

writeObject() :- It is method of ObjectOutputStream class Serialize object

readObject() :- It is method of ObjectInputStream class desenialize object

declare as static if u dont wany any field to be part of object state.

Program :

Serializing

```
import java.io.*;
class Student implements Serializable {
   String name;
   int roll;
   static int per;
   public Student ( String n, int Roll, int p)
    {
      this.name = n;
      this.roll = Roll;
       per = p;
    }
}
class Test {
   public static void main (String args [])
    {
     try {
        Student ob = new student ("khushi", 104, 90);
     FileOutputStream fos = new FileOutputStream ("s.ser");
     ObjectOutputStream oos = new ObjectOutputStream (fos);
      oos.writeObject (ob);
      oos.close();
      fos.close();
     }
     Catch (Exception e)
       {
       }
    }
}
```

Deserializing

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

```
{
student si = null;
try {
FileInputStream fis = new FileInputstream ("s.ser");
ObjectInputstream ois = new ObjectInputStream (fis);
   si = (Student) ois.readObject();
}
catch (Exception e)
   {
   }
System.out.println (si.name);
System.out.println (si.roll);
System.out.println (si.peo);
}
```