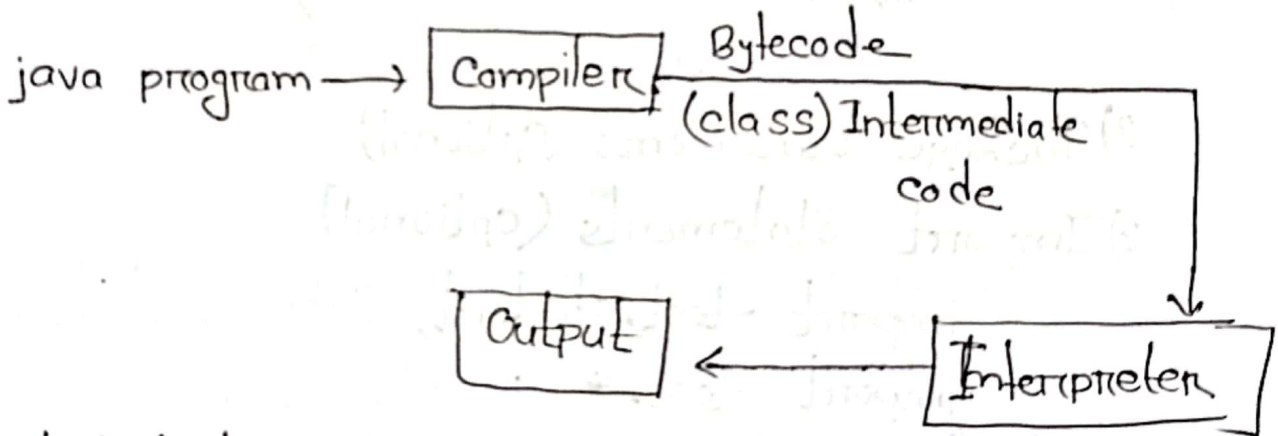
Introduction :-

- Developed by James Gosling in 1991 through Sun Microsystem.
- Publicly published first version in 1995; Java 1.0
- Starting name of java - 'Oak'.
- Java aquired by oracol in 2010.

Feature of java :-

- Simple & Secure. (Bytecode make it secure)
- Portable (run on multiple OS) - (using bytecode)
- Compiled & interpreted. (Support both)



- Distributed
- Multithreated (Java.lang এর under thread class থাকে)
- Robust & Secure
- Dynamic

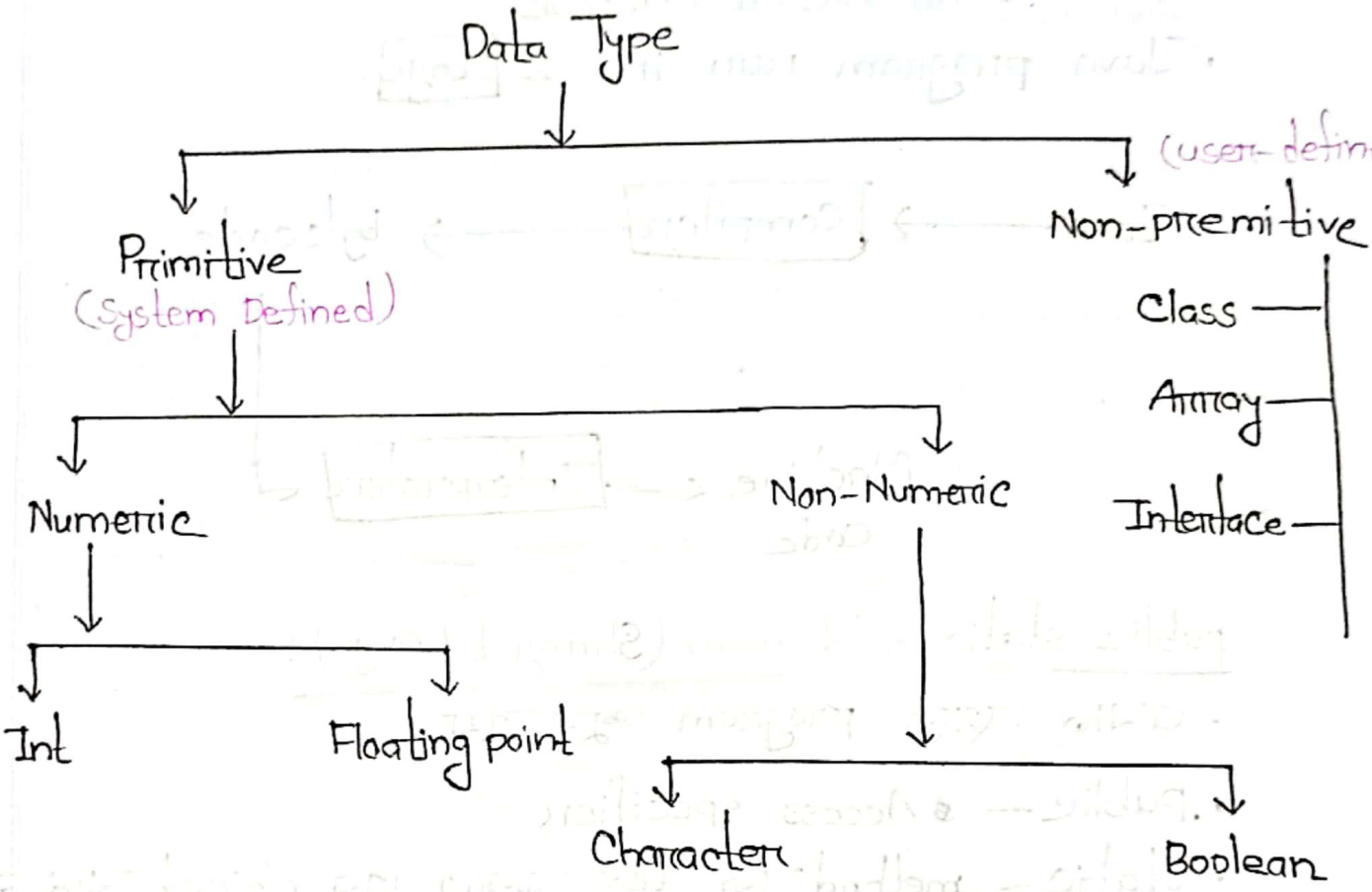
Object Oriented Programming:-

i) Encapsulation
ii) Object
iii) Class
iv) Inheritance
v) Polymorphism
vi) Dynamic- binding
vii) Message communaction
viii) Abstraction

Basic Structure:-

1) Documentation Section (optional)
   i) // single line comment.
   ii) /* Multi-line; define about program */
   iii) /** ... ... -- **/ (Automatic documentation create করে)
2) Package Statements (optional)
3) Import Statements (optional)
   import test.student; // Access student class
   import test.* ; // Access all
4) Interface Statements (optional) // To do multiple inheritan
5) Class Defination
6) Main class Defination + define main(). (Essential)

Data Types :-



Int :-

short – 2 byte
int – 4 byte
long – 8 byte
byte – 1 byte

Floating point :-

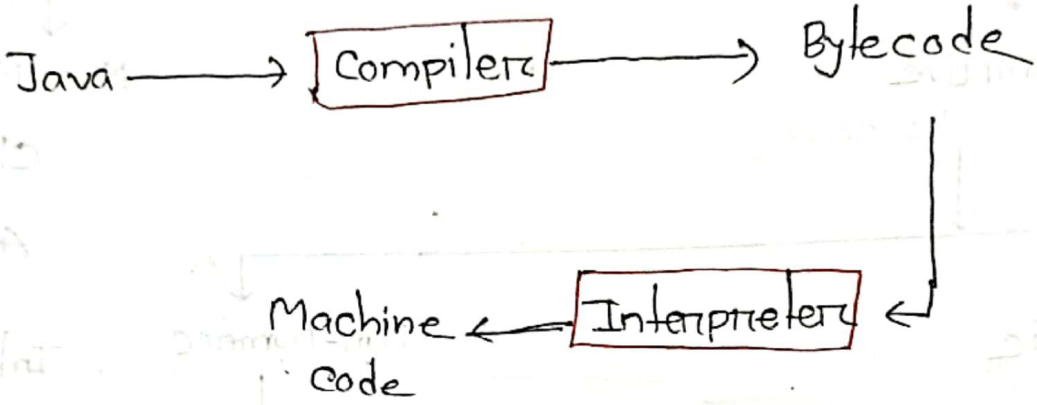
Float – 4 byte
double – 8 byte

Non-Numeric :-

char – 2 byte
boolean – true/false (1 bit)

JVM:-

- JVM – Java Virtual Machine
- Java program run in 2 stage.



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

- এখান থেকে program শুরু হয়।
- public – ⊛ Access specifier
- static – method কে use করতে এর object বানানোর দরকার নেই।
- void – no return value
- String – এটি pre-define class
- args – variable name (এটি change করা যাবে)

System.out.println :-

System.out.println("Text"); // Cursor next line এ যাবে

System.out.print("Text"); // Cursor same line এ থাকবে

System – pre-define class

out – object

println() — method.

System → Java.lang package এর ভিতর defined

out → public static final PrintStream out;

println() → এই class a define করা

PrintStream → Type (pre-defined class)

Compile & Run a program:-

1. Program
2. Save → class_name.java [Main class & java file এর নাম একই হতে হবে।]
3. Javac Test.java [Javac – compiler [bin folder এ]]
4. Java <class_name> Test.class
   [Java Test]
5. Output

Simple Java program:-

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

User input :-

```
import java.util.Scanner;
```
pre-define class

```
Scanner obj = new Scanner(System.in);
```
create object

```
int a;
a = obj.nextInt();
```

Decision making & Branching :-

i) IF statement
ii) Switch statement
iii) Conditional statement

→ For Branching

if state-ment :-

- Programe ঘ 4 ভাবে use হয়।
  - Simple if
  - if ... else ...
  - if ... else ... if
  - Nested if else

Jumps in Loop :-

Jumping out of loop — break;
Skipping a part of loop - continue

Define a class :-

```
[Access Modifier] class classname
    {
    Field declaration;
    Methods declaration;
    }
```

members

Constructor :-

- Class name & method name same হবে।
- কোন return type থাকবে না।
- Constructor 2 প্রকার। যথা:-
    i. Default constructor
    ii) Parameterized constructor

Method Overloading :-

- একই নামে Method হবে কিন্তু Parameter আলাদা হবে।
- একে compiletime polymorphism বলে। (static banding)
- Check whild overloading :-
    i) Number of Arguments.
    ii) Type of Argument.
    iii) Sequence of Argument.

```
class Test
  {
    int a, b;
    void calc (int x)
      {
        a = x;
        System.out.println("Square : " + (a*a));
      }
    void calc (int x, int y)
      {
        a=x;
        b=y;
        System.out.println("Addition :" + (a+b));
      }
  }
class Main
  {
  public static void main( String[] args)
    {
      Test obj = new Test();
      obj. calc (10,20);
      obj. calc(10);
    }
```

Output :-
Addition: 30
Multiplication: 100

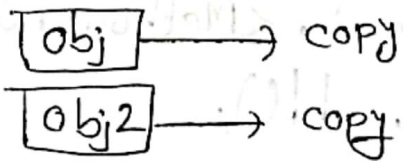
Static Member in java :-

```
class Student
  {
    int a, b;   <-- instance varriable
    void getData(int x, int y)  <-- instance method
      {
        a = x;
        b = y;
      }
  }
```

- instance method এর যখন object declare করা হয়, তখন এর একটা copy তৈরি হয়।

```
Student obj = new Student();
Student obj2 = new Student();
```

- এখানে obj ও obj2 দুটি object. সবার স্বতন্ত্র copy তৈরি হয়



- যদি এমন method বানানোর দরকার হয় যে, শুধু object এর সাথে নয় বরং পুরো class এর সাথে সুসম্পর্কিত থাকবে সেক্ষেত্রে static use করতে হবে।

eg:-

```
Static int a ;  ← class variable
Static void geldata(); ← class method
```

- class variable & Method access করতে কোন object দরকার নেই

```
class Student
 {
  static int a; ← class এর level এ থাকতে।
  :
 }
```

```
obj. Variablename = value
obj. Methodname();
```

← instance variable & Meth
access করতে object লাগে

```
classname. variablemame =value
classname.. Methodname().
```

← class var & Method
access করতে object লাগে না।

Static Method :-

<class_name> . <Method_name>

↳ Student. add();

Restriction :-

i) Static Variable & static Method direct call করা যা
ii) instance variable & instance method direct call করা যায় না।

iii) object call করার প্রয়োজন নেই।

iv) this, super keywords use করা যাবে না।

Program :-

```
class Test
  {
   static int cube(int x)
      {
        return x*x*x;
      }
  }

class Main
  {
   public static void main(String [] args)
      {
        int res= Test.cube(5);
        System.out.println ("Cube :" + res);
      }
  }
```

Output :-

Cube : 125

Nesting method:-

— একটি Program এ এক method আরেক method কে call দেয়, একে nesting method বলে।

Example:-

```
class Student
  {
    void disp()
      {

      }

    void getinp()
      {
        disp();
      }
  }
```

Program:-

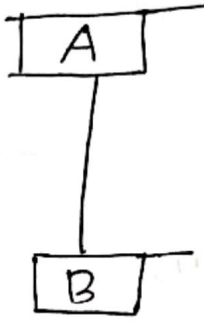
```
class Nesting
  {
    int m,n;
    Nesting(int x, int y)
      {
        m=x;
        n=y;
      }
```

```
    int largest()
      {
        if (m>n)
          {
            return m;
          }
        else
          return n;
      }
    void disp()
      {
        int ans = largest();
        System.out.println("value : "+ ans);
      }
}

class Main
  {
   public static void main(String [] args)
      {
       Nesting ob = new Nesting (10,15);
       ob.disp();
      }
  }
```

Output :-
value : 15

Inheritance :-

~~Type of i~~



Base class/Super class/parrent class

Sub class/derrive class/child class

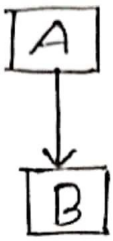
Sub_className extends Superc_class_Name

```
{

}
```

Type of inheritance:-

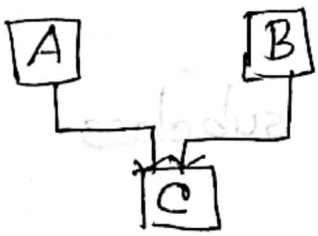
i) Single
ii) Mutti-level
iii) Muttiple
iv) Hierrarchial
v) Hybrid

Single :- একটা super & one sub class

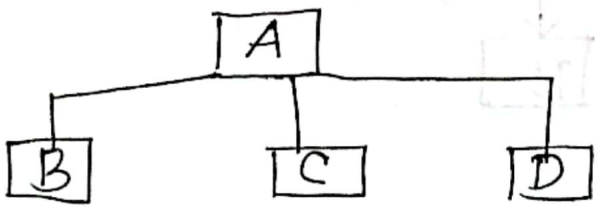


Multi-level :-

Multiple :- এক/একাধিক base class. Java তে direct use করা হয় না। এর জন্য interface use করা হয়।



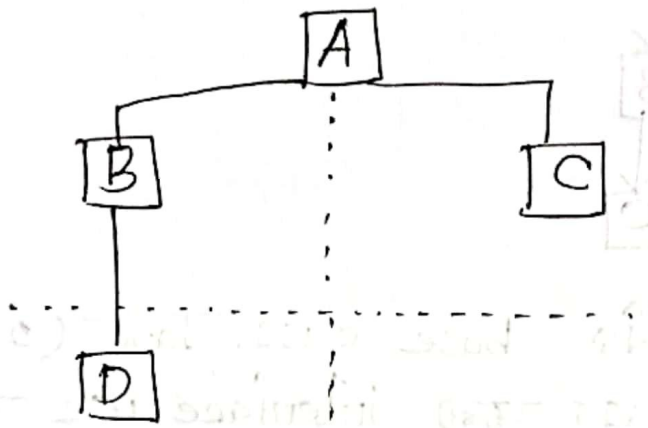
Hercarchical :-



Hybrid:- Combination of above inheritance.

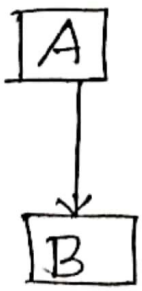
Java তে —

- Single
- Multilevel
- Hierarrical interface use করা হয়।



Single Inheritance:-

→ One Super & one subclass



Program:-

```
class A
  {
    int x, y;
```

```
    void getData(int a, int b)
      {
       x=a;
       y=b;
      }
     int add()
       {
        System.out.println("Super class A");
        return x+y;
       }
  }
//Derive class
class B extends A
  {
   int mult()
     {
      System.out.println("Sub class B");
      return x*y;
     }
  }
```

```
class Main
{
 public static void main (String[] args)
   {
     B. obj = new B();
     int add, mul;
     obj.getData(5, 3);
   add=obj.add();
       System.out.println("Addition:" + add);
       mul = obj.mult();
       System.out.println("Multiplication:" + mul);
```

Multilevel Inheritance:-

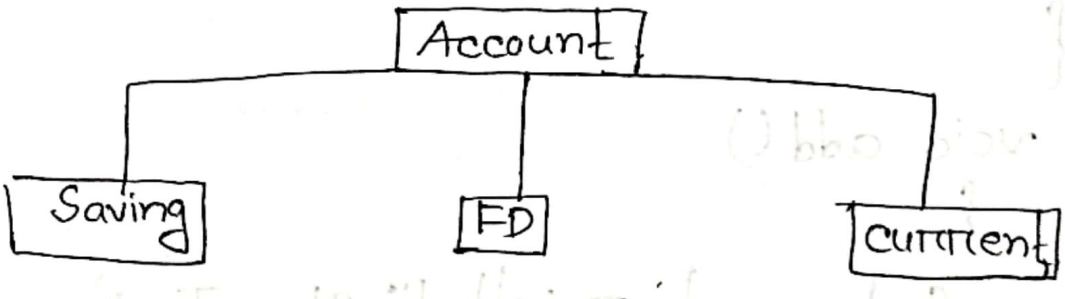
Program:-

```
class A
  {
   int roll
   void getRoll(int x)
     {
      roll = x
     }
   void putRoll()
     {
      System.out.println("Roll no :" + roll);
     }
}

class B extends A
  {
    int m1, m2;
    void getMark(int x, int y)
```

```
        {
         m1 = x;
         m2 = y
        }
      void putMark()
        {
         System.out.println("Test 1 : " + m1);
         System.out.println("Test 2: " + m2);
        }
}

class C extends B
  {
    int total;
    void disp()
      {
       putRoll();
       putMark();
       total = m1 + m2;
       System.out.println("Total: " + total);
    }
```

```
class Main
{
 public static void main(String [] args)
    {
      C ob = new C();
      ob.getRoll(101);
      ob.getMark(80,85);
      ob.disp();
    }
}
```

Hierarrical Inherritance :-

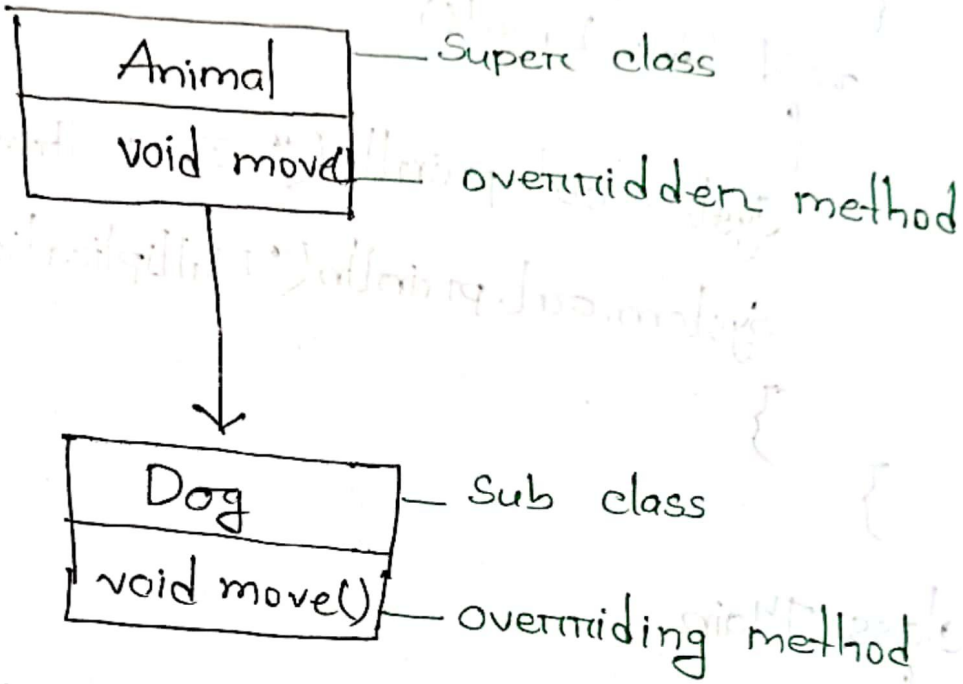


```
class One
 {
  int x = 10, y = 20;
  void disp()
    {
     System.out.println(" Class One");
     System.out.println("Value of x:" +x);
     System.out.println("Value of y:" +y);
    }
 }

class Two extends One
 {
  void add()
    {
     System.out.println(" Class Two");
     System.out.println("Addition:" +(x+y));
    }
}
```

```
class Three extends One
  {
    void ~~disp()~~ mul()
      {
        System.out.println(" class Three");
        System.out.println(" Multiplication : "+ (x*y));
      }
  }
class Main
  {
    public static void main(String [] args)
      {
        Two   ob1 = new Two();
        Three ob2 = new Three();

        ob1.disp();
        ob1.add();

        ob2.mul();

      }
  }
```

Method Overridding:-

- Name, parameter, return type same



Program:-

```
class Super
 {
  int x=10;
  void disp()
    {
     Sytem.out.println(" Super class"+x);
    }
 }

class  Sub extends Super
 {
    int y =20;
```

```
    void disp()
      {
        System.out.println(" Super: " +x);
        System.out.println(" Sub: "+y);
      }
  }

class Test
  {
   public static void main(String [] args)
     {
       Sub ob = new Sub();
       ob.disp();
     }
  }
```

Output :-
Super : 10
Sub : 20

Override হবে না:-

i) Method private হলে
ii) Constructor override হবে না।
iii) Static Method override করা যাবে না।
iv) Final Method override করা যাবে না।

Overloading	Overriding
Method Name একই হলেও parameter list, return type আলাদা হয়।	Method Name, return type, parameter list একই হবে।
Inheritance এর প্রয়োজন নেই।	Inheritance হতে হবে।

Abstract Method :-

- Abstract class এর object creat করা যায় না।

```
abstract class X
  {
   abstract void disp();
   void display()
    {
     System.out.println("Method from x class");
    }
}

class Y extends X
 {
  void disp()
    {
     System.out.println("Method in Y");
    }
```

```
}
class Main
 {
  public static void main(String[] args)
    {
      Y ob = new Y();
      ob.disp();
      ob.display();
    }
}
```

Final Variable Methods & class :-

- Final is a keyword. That used with variables, classes and Methods.
- Variable :- (Value constant হয়ে যায়)

```
final int COUNT = 100;
```

- Variable name capital letter a হবে।

Method :-

- Final variable ২ ধরণের।
  - i) instance
  - ii) local

Rules of final variable :-

Class variable :- (Static যুক্ত)

- Static keyword যুক্ত হবে।
- পুরো class এর সাথে সম্পর্কিত।

```
static int COUNT
or
static final int COUNT = 100;
or
static final int COUNT;
```

```
static
{
  COUNT = 100;
}
```
— static block

Instance variable :- (class এর ভেতর)

```
class X
{
  int a;
  final int COUNT = 10;
  final int CNT;
  X()
  {
    CNT = 10;
  }
}
```

- `int a;` — instance variable
- `final int COUNT = 10;` — Use final keyword ~~variable~~
- `final int CNT;` — constructor এর ভেতরে initialize করা যায়।

local variables :- (function এর ভিতর)

```
class X
 {
  void disp()
    {
     int a ;—— local variable
     final int COUNT = 10; —final keyword সহ।

     final in CNT;
     CNT = 10;
    }
 }
```

} অথবা এভাবে initialize করা যায়।

Parameter / Argument :-

```
void disp(final int a)
   {

   }
```

Final Method :-

```
final void disp()
   {
   :
   }
```

• একে override করা যাবে না।

Final class :-

```
final class x
    {

    }
```

- একে inherite করা যাবে না। কোন next class একে 'extands' করে inherite করতে পারে না।

Interface :-

- এর সব method abstract হয়।

Syntax :-

```
interface interface_name
    {
     variable;
     method();
    }
```

- method abstract হয়, তাই body থাকে না।

- public static final type var_name = value
  - public static final int A = 100; — এভাবে কাজ করে
  - int A = 100; — keyword গুলা লিখার দরকার হয় না

- public return_type method_name(parameter)
  - public int add (int x, int y);
  - int add (int x, int y); — by default public হয়।

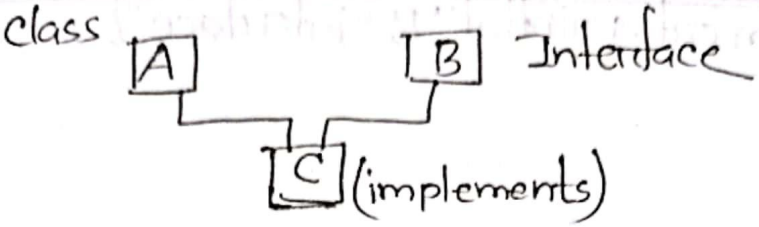
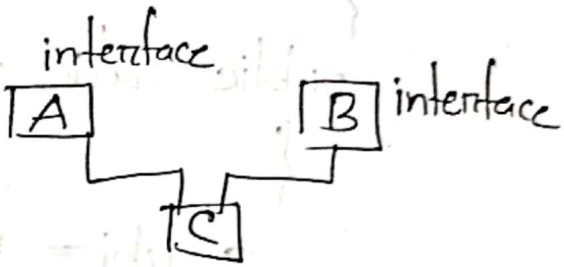


Fig- Multiple inheritance using interface.

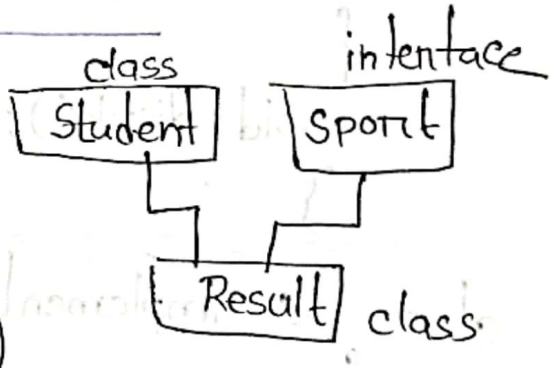
Program:-




```
interface A
   {
    int roll=101;
    void dispA();
    }
interface B
    {
     void dispB();
    }
class C implements A,B
    {
public void dispA()
        {
         System.out.println("Roll no:" + roll);
        }
      public void dispB()
        {
```

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

class Main
  {
  public static void main(String [] args)
    {
      C obj = new C();
      obj.dispA();
      obj.dispB();
    }
```

Multiple Inheritance using interface:-




```
class Student
  {
    int m1, m2;
    void getMark(int x, int y)
      {
        m1 = x;
        m2 = y;
      }
    void putMark()
      {
        System.out.println("Mark1: " + m1);
```

```
        System.out.println("Mark 2: " +m2);
    }
}
interface Sports
    {
     int sp=6;
     void spMarks();
    }
class Result extends Students implements Sports
    {
     public void spMark()
       {
       System.out.println("Sport Marks:" +sp);
       }
     void disp()
       {
         putMarks();
         spMarks();
         int total = m1+m2+ sp;
         System.out.println("Total Marks :" + total);
       }
    }
```

```
class Main
 {
  public static void main(String[] args)
    {
      Result res = new Result();
      res.getMark(80, 85);
      res.disp();
    }
 }
```

Array in Java :-

- Collection of similar type of data.

- Declaring the Array :-

Syntax :-

```
type arrname [];   → Form-1
type [] arrname;   → Form-2
```

eg :-

```
int roll [];
int [] roll;
```

- Creating Memory location :-

Syntax :-

```
arrname = new type [size];
```

```
roll = new int [30];
```

Combined Syntax:-

```
type. arrname = new type [size]
```

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

Initialization of Array:-

Syntax:-

```
arrname [index_no] = value;
```

```
roll [3] = 100;
```

Array length:-

```
arrname.length;
```

```
int s = roll.length;
```

program:-

```
import java.util.Scanner;

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

    Scanner sc = new Scanner (System.in);
    System.out.println("Enter Size:");
    int n = sc.nextInt();

    int marks [] = new int [n];
```

```
System.out.println("Array Length:"+ marks.length);
System.out.println(" Enter" + n + " Elements");
for(int i=0; i<marks.length; i++)
    {
     marks[i] = sc.nextInt();
    }
int total = 0;
for(i=0; i< marks.length; i++)
    {
     System.out.println("Marks [i]);
     total = total+ marks[i];
    }
  System.out.println("Total Value = " + total);
 }
}
```

Super keyword :-

- Inheritance এর সময় immidate super class কে refer করতে super keyword use করা হয়।
- super 2 ভাবে use করা যায়।
  - i) Accessing super class member.
  - ii) Calling super class constructor.

i) Accessing Super class Member :-

- Instance variable/method উভয়কেই use করা যায়।
- super ও derive class এ একই নামের variable থাকলে super class (Parent class) এর member access করা যায় :-

```
super.Variablename;
super.methodname();
```

eg :-

```
class A
  {
   int no;
   :
  }
class B extends A
  {
   int no;
   void disp()
     {
      no = 100;
      no = 100
      super class এর variable
      value hide করে দেয়।
      super.no = 200;
      ↳ এভাবে super class এর
      variable access করা যায়।
     }
  }
```

Program:-

```
class A
  {
   int no;
   void message()
     {
      System.out.println("Super no:" + no);
     }
  }
class B extends A
  {
   int no;
     B(int x, int y)
       {
        super.no = x;
         no = y;
       }
    void message()
       {
        System.out.println("Super 2:" + no);
       }
   void disp()
       {
```

```
        super.message();
        message();
      }
  }
class Test
  {
   public static void main(String [] args)
     {
       B obj= new B(100, 200);
```

super.no = a;
no = b;
→এভাবে কাজ করতে

Output:-
super no :- 100
super 2 :- 200

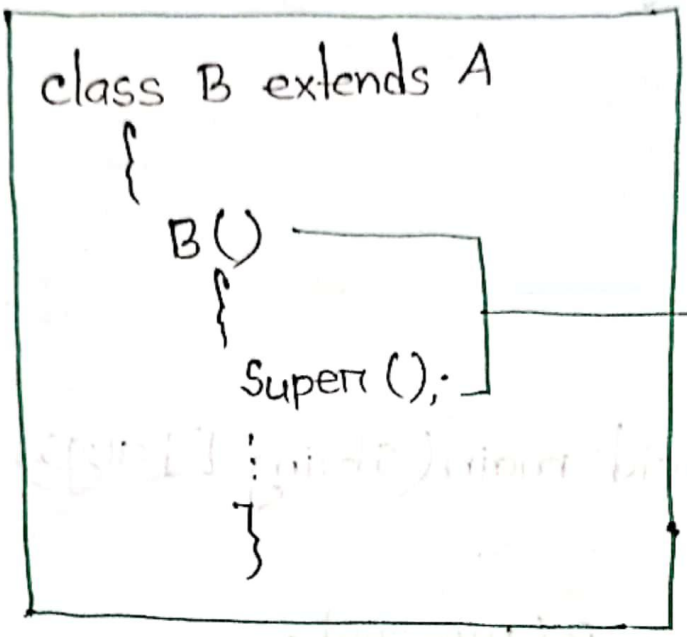
```
       obj.disp();
     }
}
```

Calling super class constructor:-

```
class A
   {
     A()
       {
       }
   }
class B extends A
  {
    B()
      {
      :
      }
  }
```

super keyword 2 ভাবে call করা যায়।
i) super();- no parameter
ii) super(Parameter_list);

• Sub class constructor এর প্রথম লাইনে লিখতে হবে।



Program :-

```
class A
  {
   int a;
   A()
    {
     System.out.println ("Constructor A");
    }
  }

class B extends A
  {
   B()
    {
     System.out.println("Constructor B");
```

Output :-

Constructor A
constructor B

implicite way te print hobe

```
    }
}
```

Program – 02 :-

```
class A
  {
    int a;
    A (int x)
      {
        System.out.println("A :"+a);
      }
  }
```

```
class B extends A
  {
    int b;
    B (int x, int y)
      {
        super(x);
        b = y;
        System.out.print("B :"+b
      }
  }
```

```
class TMain
  {
    public static void main(String [] args)
      {
        B obj = new B(10, 20);
      }
```

Output :-

A : 10
B : 20

Access Modifiers:-

i) At the top level (class level Access modifiers)
ii) Members level (Members level Access Modifiers)

1) Class level access modifiers:-

- Public, default হিসাবে থাকে।
- default কে Package privat/friendly বলে।

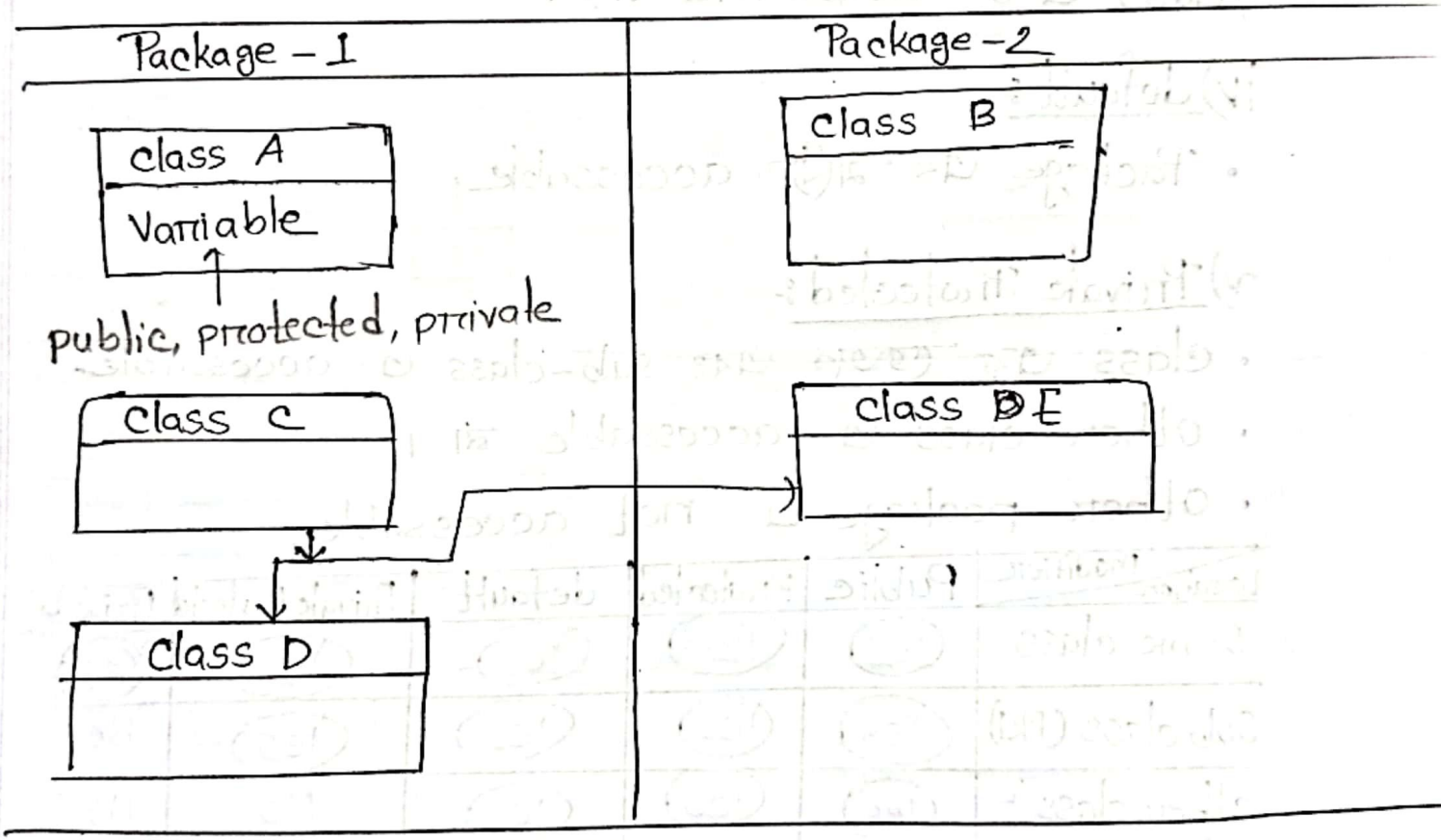
class Test:— এখানে default use করা হয়েছে.
{ — এটি কেবল একটি package এর ভিতর কাজ করে।
}

Package-1	Package-2
class A { }	A ob = new A(); — not accessible (Package import করলেও)
public class A { }	A ob = new A(); — accessable (সব জায়গায়)

i) একটা source code এ একটিমাত্র public class হবে।
ii) public class এর নামে filename save করতে হবে।

ii) Member level Access Modifier :-

- Variable, methode — member level
- Modifier :-
  - i) Public
  - ii) Private
  - iii) Protected
  - iv) default
- 2টি keyword একসাথে use করা যায়।
  - v) private protected



i) use public :-

- ঐ package এ যেকোন class access করতে পারে।
- ঐ package এর বাইরেও use করা যায় (inheritance/direct)

ii) Protected :-

- নিজ package এর সকল call এ use করা যায়।
- অন্য package এ use করা যায় না। (direct)

• Package এর বাইরে শুধু inheritance দিয়ে use করা যায় অন্য package এ।

iii) Private:-

• Height level protection provide করে।
• নিজ class এর মধ্যে accessable.
• Inheritance এর মাধ্যমে same packge এ অন্য class এও accessable নয়।

iv) default:-

• Package এর মধ্যে accessable.

v) Private Protected:-

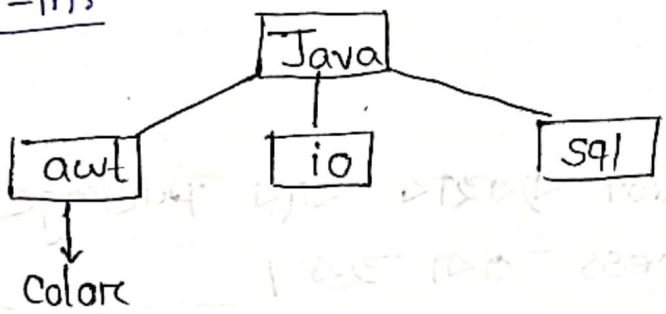
• class এর ভেতর এবং sub-class এ accessable.
• other class এ accessable না।
• other package এ not accessable.

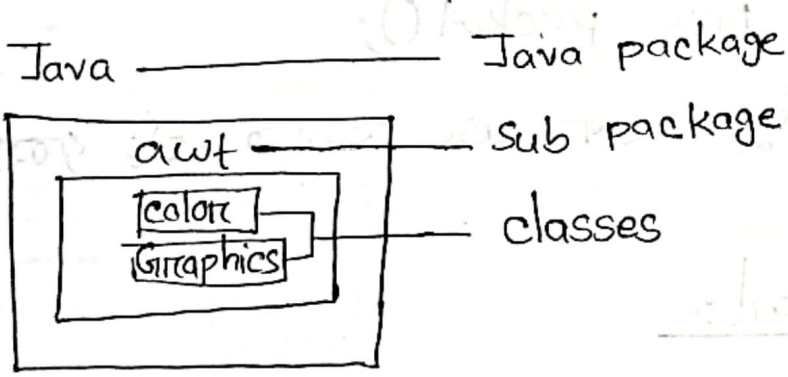
Location \ Modifier	Public	Protected	default	Private Protected	Private
Same class	Yes	Yes	Yes	Yes	Yes
Sub class (Pk1)	Yes	Yes	Yes	Yes	No
other class	Yes	Yes	Yes	No	No
Sub class (Pk2)	Yes	Yes	No	Yes	No
other class (Pk2)	Yes	No			
Non-sub (Pk2)	Yes	No	No	No	No

Packages :-

- Reuseablity OOP এর একটি বৈশিষ্ট্য।
- Similar type এর class, interface যেখানে group করে রাখা হয়, তাকে package বলে। এক container হিসাবে use করা যায়।
- Type of packages :-
  - i) Built-in packages (Java.io, java.lang)
  - ii) User defined package

i) Built-in :-





ii) User defined :-

- Name unique হতে হবে।
- lowerCase use করতে হবে। কিন্তু এটা Restricted না।

Advantage of package :-

i) Re-usablity (একই code অনেক জায়গায় use করা যায়)

ii) Access Control

iii) package এ (আলদা) class name same হতে পারে।

Use package:-

i) Fully qualified class name

ii) Import statement

i) Fully qualified class name:-

- অন্য Package এ dot ব্যবহার করে package এর element access করা হয়।

```
pack.A obj = new pack.A();
```

- Package এর কিছু অংশ use করতে এটি ব্যবহার করা হয়।

ii) Import Statement:-

- একে 2 ভাবে use করা যায়।

i) import package.*; → Access all element

ii) import package.class_name; → Access only one class of other class

Creating & Using Package :-

Create :-

1. Packge keyword এর সাথে package_Name

```
package MyfirstPackage;
```

2. এর নিচে class define করতে হবে।

```
public class classname
  {
  :
  }
```

3. public class এর নামে File Name Save করতে হবে।
4. Package নামের একটা directory create করতে হবে। directory name ও package folder name একই হবে
5. Compile the program
6. classname.java folder এ class file save হবে।

Program :-

```
package pack1;
public class A
  {
   public void dispA()
     {
      System.out.println(" Shakil");
     }
  }
```

```
import pack1.*;
class Test
 {
  public static void main(String[] args)
    {
      pack.A obj = new pack.A();
        obj.dispA();
    }
 }
```

Create package inside Another package:-

- একে Package Hierarchi concept ও বলা হয়।

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

Java util]—(Packages)

`import java.*;` — Access All packge of under Java only in Java pack

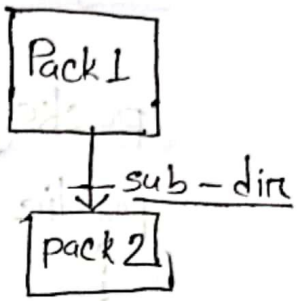
Java package এর ভেতরে অন্য কোন sub-package access করতে পারবে না। But java package এর অন্য সব accessable

```
package pack1.pack2;
public class Test
  {
  }
```

Program:-

Test.java

```
package pack1.pack2;

class
public class Test
  {
   public void show()
     {
      System.out.println(" Hi, there is me");
     }
  }
```



Main.java

```
import pack1.pack2.*;

class Main
  {
   public static void main(String [] args)
     {
      Test obj = new Test();
       obj.show();
     }
  }
```

Program :-

Test.java

```
package pack1;
public class Test
 {
   protected int val=100;
   public void disp()
     {
      System.out.println("class Test");
      System.out.println("value:" + val);
     }
 }
```

Main.java

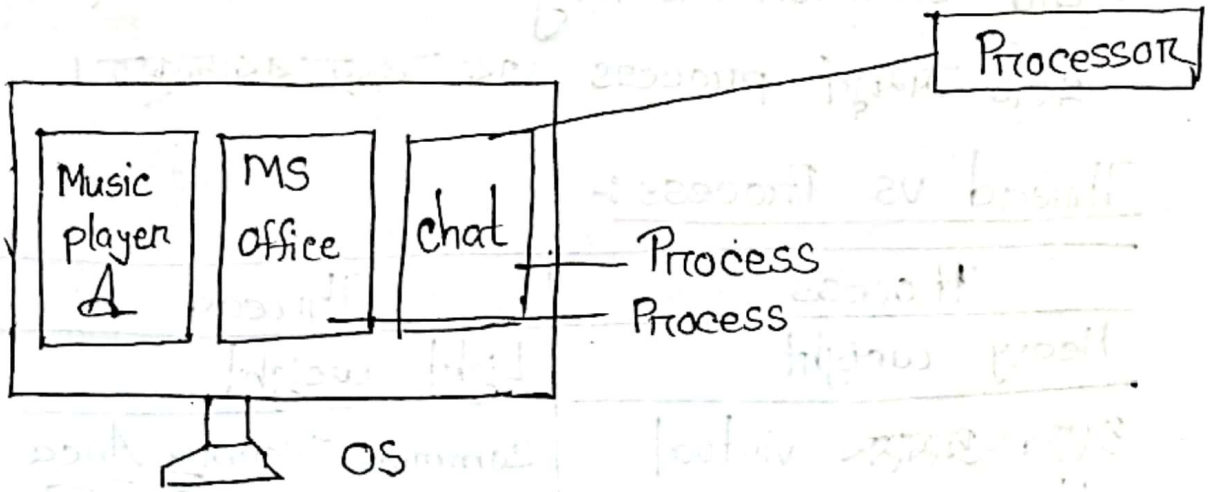
```
import pack1.Test;
class sub extends Test,
  {
    int n=200;
    void dispSub()
      {
       System.out.println(" Class Sub");
       System.out.println("value' +val);
       System.out.println(" N:"+n);
```

```
    }
}
public class Main
{
    public static void main(String [] args)
    {
        Sub obj = new obj();
        obj.disp();
        obj.dispSub();
    }
}
```

output:-
class Test
val : 100
class sub
val : 100
n : 200

Threads :-

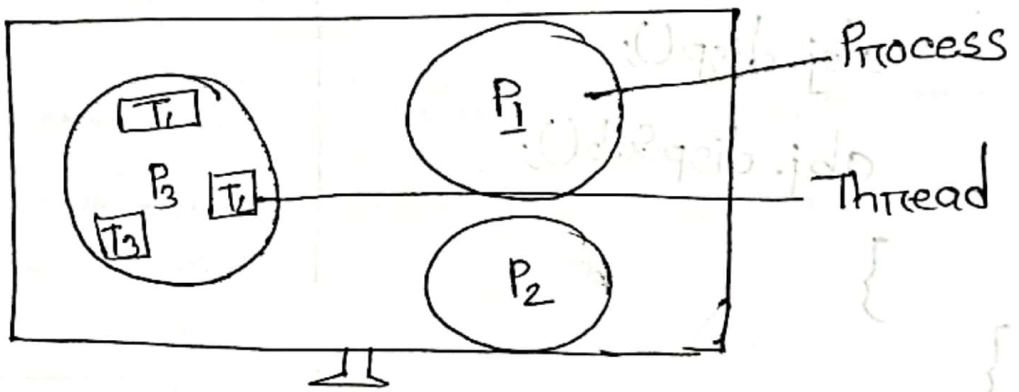
- Process
- Thread
- Process vs Thread



Processor at a time যেকোন একটি process continue করে। But switch এর মাধ্যমে সবগুলো handle করে।

- Process একটি সম্পূর্ণ Application run করে।
- এদের সবার আলাদা Address থাকে।

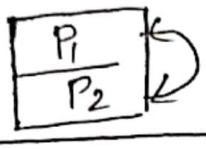
- আর একটা Application এ একসাথে অনেকগুলো কাজ হয়। যেমন- MS word চালু করলে Typing input নেয়, word count করে, spell checker তার কাজ করে। এদের Thread বলে।



- Thread হলো light weight process.
- এটি একটি sub process.
- এটি common memory area allocate করে যা একটি সম্পূর্ণ process এর জন্য বরাদ্দকৃত।

Thread vs Process :-

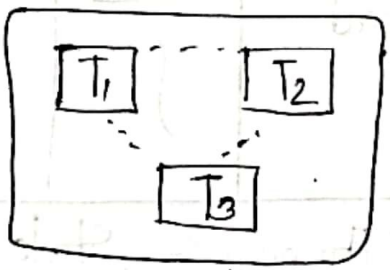
Process	Thread
Heavy weight	Light weight.
প্রত্যেক প্রসেসের virtual address আলাদা P1 P2	Common Memory Area share করে।
এক Process অন্য Process এর সাথে সম্পর্যুক্ত করতে	Common Memory Area share করায় Direct

Inter-communaction system use করে, যা OS নির্ভর। 	Communicate করতে পারে।
নিজস্ব Data Sagment থাকে।	Data & Code share করে।

- Thread code এর part যা independently execute হতে পারে।

Multi-Threading :-

- Thread is a part of process.



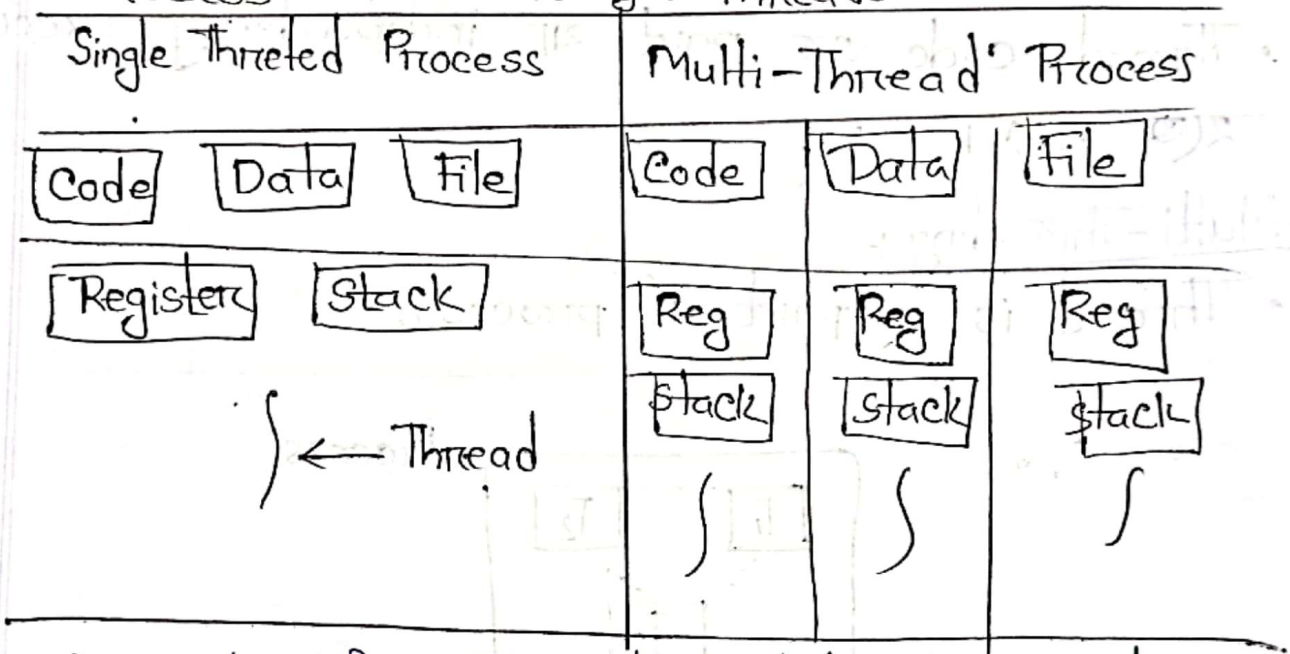
word

- প্রত্যেক thread independent. এরা আলাদাভাবে execute হতে পারে।
- Single thread এ একাধিক task থাকলেও, তা একটি sequence এ execute হয়। একটি task শেষ না হলে পরেরটি শুরু হয় না।
- Muti-Thread cpu কে proper utilization করে ফলে এর capability বাড়ে।

- Muli-Tasking 2 ধরণের। যথা:-
  - i) Process based (At least one thread)
  - ii) Thread based

java.lang.Thread

Process Flow through Threads:-



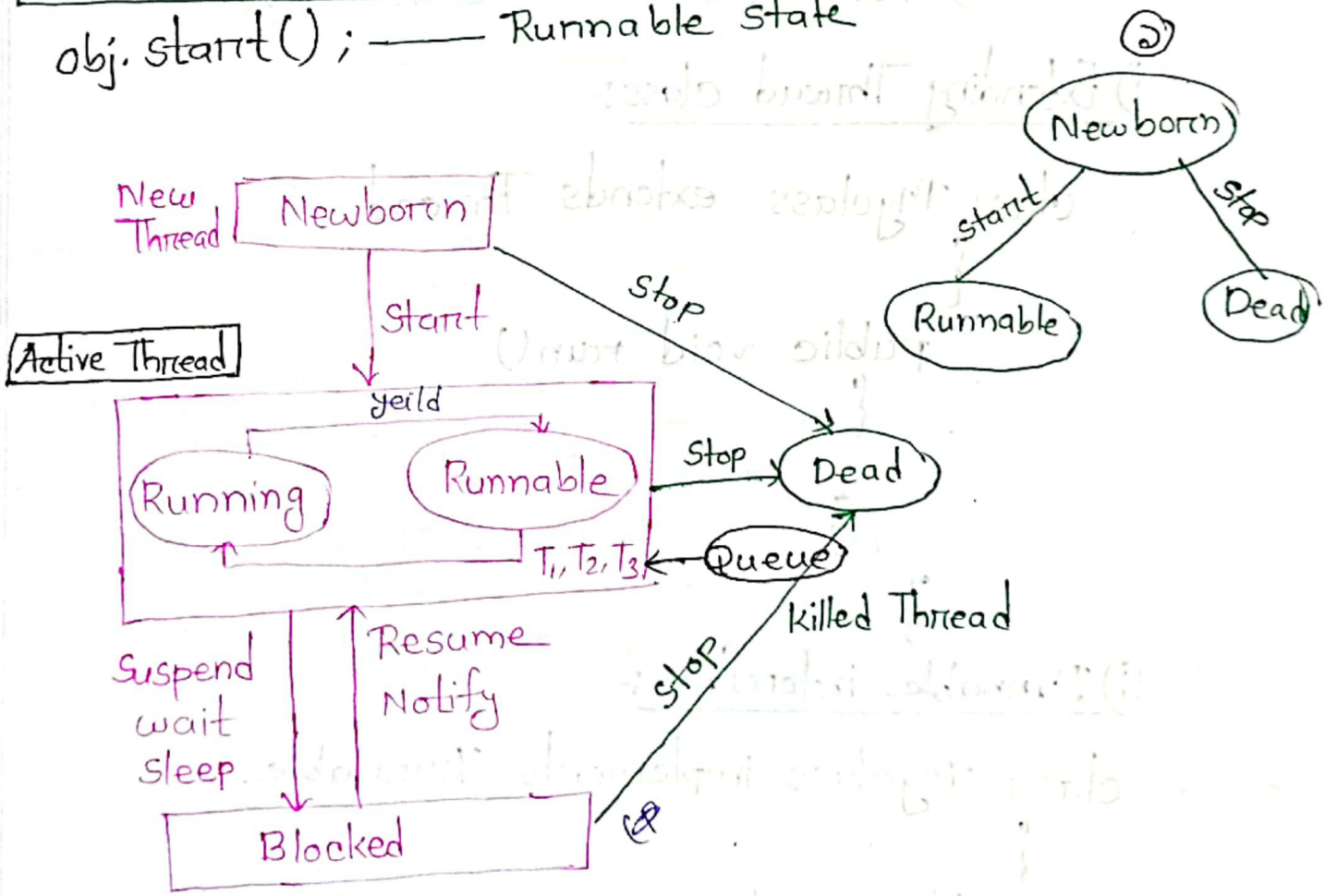
Life Cycle of a Thread:- State of Thread:

i) Newbron State (Thread তৈরি হয় কিন্তু run হয় না)
ii) Runnable state (Ready to run)
iii) Running State
iv) Blocked state
v) Dead State

```
class Myclass extends Thread
  {
   public void run()
     {
       :
     }
  }
```

Myclass obj = new myclass(); — Creating Thread (Newborn state)

obj.start(); —— Runnable state

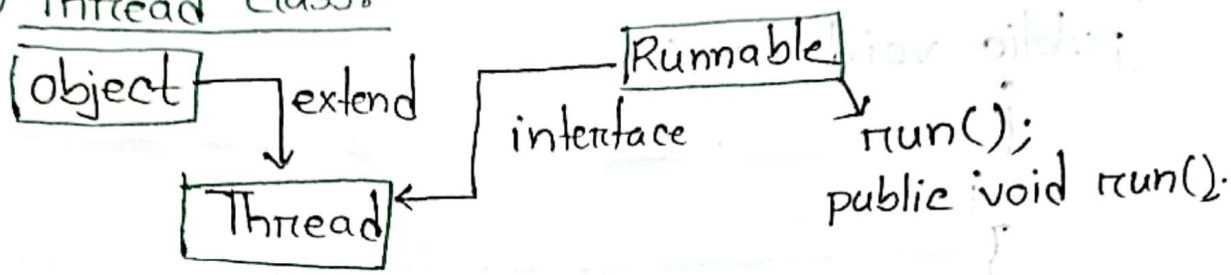


pre-matured dead :- Thread কাজ শেষ করার আগে dead হলে।

Natural dead :- Thread কাজ শেষ করার পর dead হয়

Creating Thread:-

i) Thread class:-



So, Thread 2 ভাবে create করা যায়। যথা:-

i) class এর মাধ্যমে (extending Thread class)
ii) Runnable interface এর মাধ্যমে

i) Extending Thread class:-

```
class Myclass extends Thread
    {
      public void run()
        {

        }
    }
```

ii) Runnable interface:-

```
class Myclass implements Runnable
   {
    public void run()
      {
      }
   }
```

• Runnable interface use করলে – পরবর্তী Thread start, stop, wait, sleep করাতে Thread class প্রয়োজন হয়।

Extending Threads :-

i) Declaring the class – class MyThread extends Thread;
ii) Implementing run() method
iii) Starting New Thread.

eg:-

```
class Mythread extends Thread ---- S1
  {
    public void run()  ---- S2
    {
     //code
    }
```

In main class:-

Newborn state

Myclass obj = new myclass ()

obj.start(); — Runnable — Multi-Thread হিসাবে কাজ করে

// obj.run(); — Multi-Thread হিসাবে execute হবে না। single Thread হিসাবে কাজ করে

Program:-

```
class A extends Thread
  {
  public void run()
     {
     for(int i=1; i<=5; i++);
        {
         System.out.println("From Thread A: "+i);
        }
       System.out.println("Exit from A");
     }
class B extends Thread
  {
  public void run()
     {
     for (i=1; i<=5; i++)
        {
         System.out.println("From Thread 'B' + i);
        }
       System.out.println("Exit from B");
     }
}
```

```
class Main
{
public void static main (String [] args)
  {
    A ob1 = new A();
    B ob2 = new B();

    ob1.start();
    ob2.start();
  }
}
```

Output:

```
From A : 1
From A : 2
From B : 1
From B : 2
From A : 3
From B : 3
From A : 4
From A : 5
From B : 4
From B : 5
Exit A
Exit B
```

(এখানে যেকোন sequence এর output আসতে পারে)

Exception handling :-

There are two types of error. They are :-

i) Compile time error. (Syntax error)

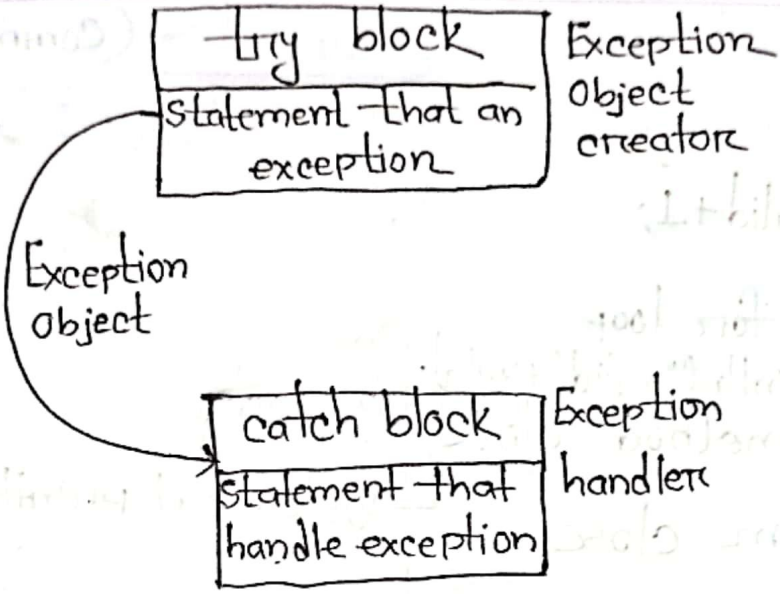
ii) Runtime error. (logical error, overflow)

. কোন Program এ logical error থাকলে compiler তো code run করবে but logic এ error এর কারণে run করার পর কোন এক সময় program terminate হয়ে যায়। এর সমাধান হলো exception handling. Program এর যে অংশে logical error আছে তা বাদে বাকি program স্বাভাবিকভাবে run করা যায় এই exception handling use করে। (Interpreter exception ও এর object create করে)

How to handle :-

i) Hit the exception
ii) Throw the exception
} Block-1

iii) Catch the exception
iv) Handle the exception
} Block-2




```
try
{
  statement;
}
catch(Argument exception)
{
  statement;
}
```

Catching invalid command Line Argument:-

```
class Test
{
 public static void main (String [] args)
   {
    int valid=0, invalid=0;
    int no;
   for (int i=0; i<args.length ; i++)
      {
        try
          {
            no = Integer.parseInt (args[i]);
          }
        catch (NumberFormalException e)
          {
            invalid = invalid + 1;
            System.out.println("Invalid :" + args[i]);
```

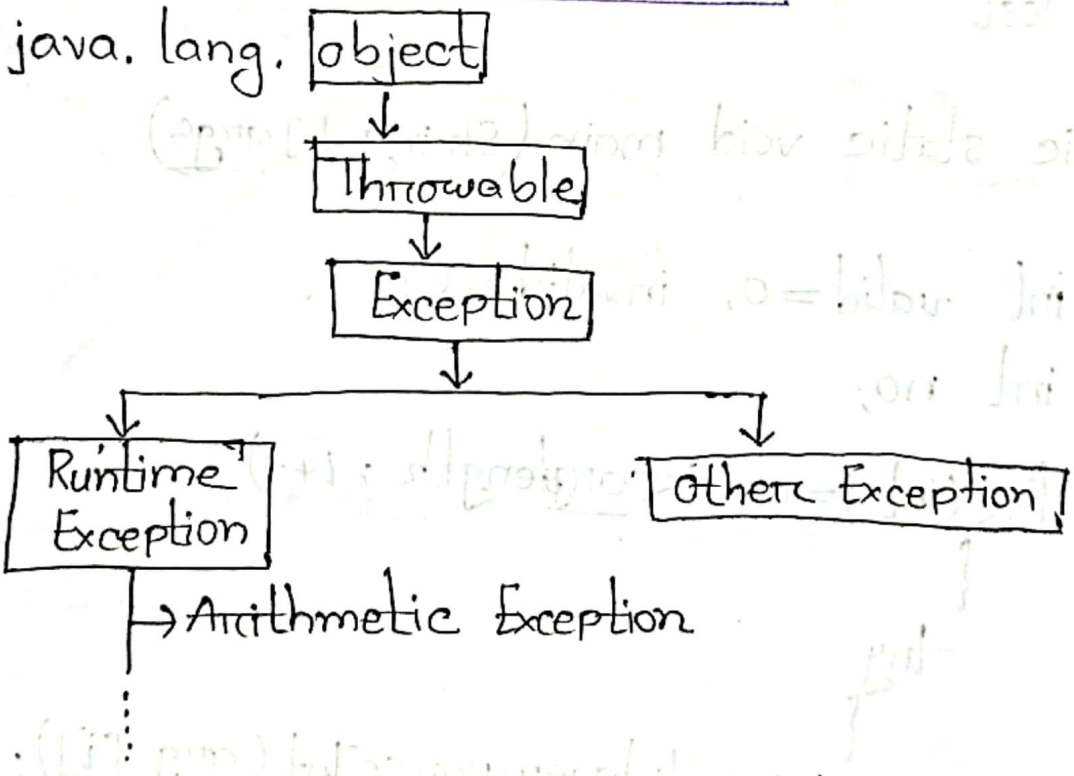
Static Method

```
        continue;
    }
    valid = valid+1;
} // close for loop
System.out.println("valid "+valid);
}//main method close
}// program close
```

Output :- (Command Line ↓)

```
Java Test 1020 Java 30
Invalid number Java
valid =
invalid =
System.out.println("Invalid "+invalid);
```

Example of exception handaling :-




```
catch (exception_type e)
catch (Exception e)                    → All of exception class
catch (ArithmeticException e)          → particular Exception
catch (ArrayIndexOutOfBound e)         → "
```

Program :-

```
class Test
 {public static void main (String [] args)
   {
    int a=20, b= 10,  c = 10;
    int x= a/(b-c);
    System.out.println("x :" + x);
    int y = a/(b+c);
    System.out.println("Y :" +y);
   }
 }
```

$\frac{20}{10-10} = \frac{20}{0}$ Exception

$\frac{20}{20} = 1$ valid

•Arithmetic Expression Reason by 0 (message show করবে)

To solve this type of program we need Exception Handling

```
class Test
 {
  public static void main(String [] args)
   {
    int a= 20, b= 10,  c =10;
    try
      {int x = a/ (b-c);
       System.out.println ("x :" +x);
      }
```

```
catch(ArithmeticException e)
  {
    System.out.println("Division by Zero");
  }
 int y = a/(b+c);
 System.out.println("Y :"+y);
 }
}
```

output :-

Division by Zero

Y : 1

Nested try block :-

User define exception :-

```
class MyException extends Exception
 {
  public MyException(String s)
    {
      super(s);
    }
  }
class Test
 {
  public static void main(String []args)
    {
     try {
```

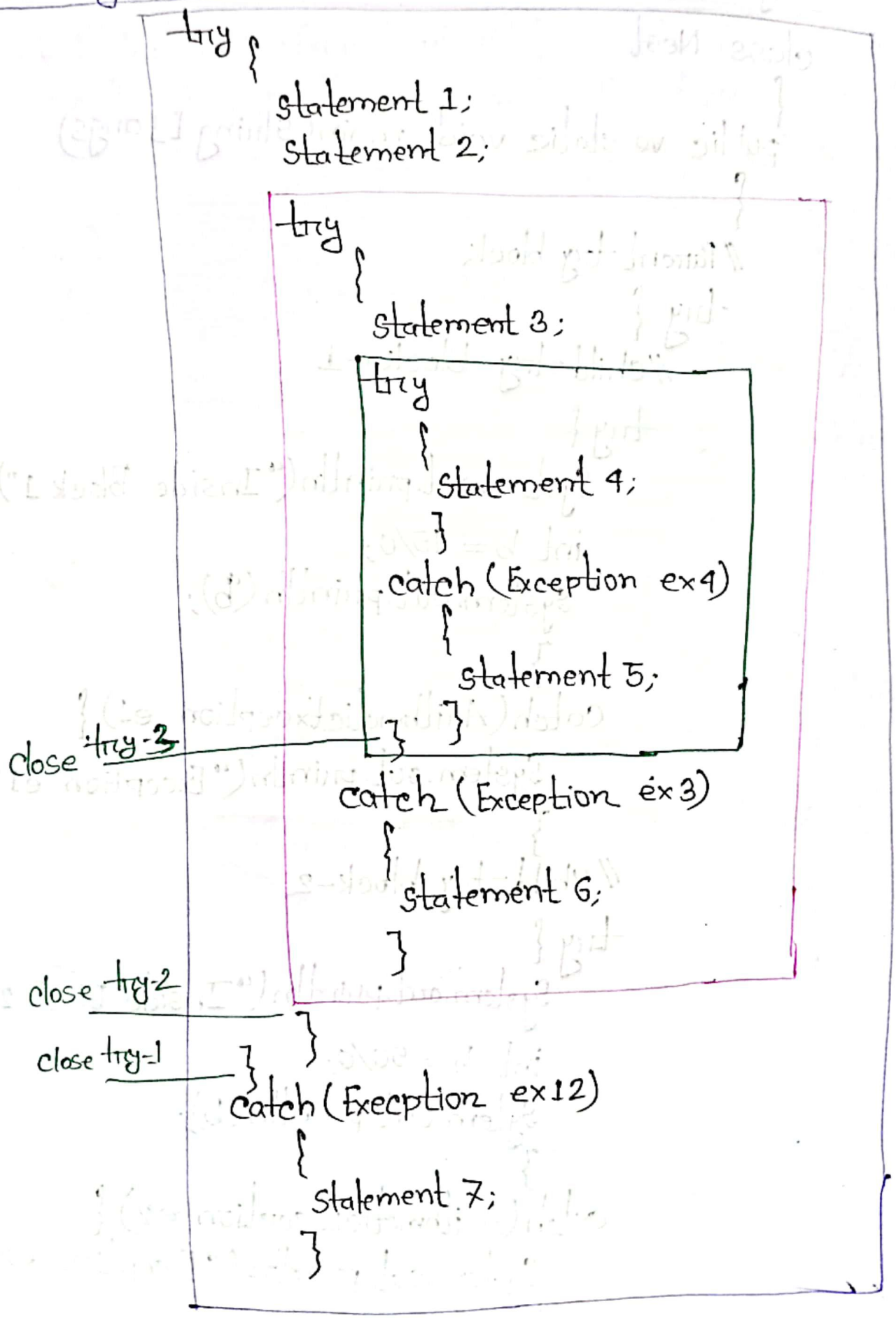
```
        throw new MyException(s);
        }
    Catch(MyException e)
        {
        System.out.println("e.getMessage());
        }
    }
}
```

*Write a program- if the value is less than 8 thats an exception. Otherwise execute the program.

```
package Exception;
import java.util.Scanner;

class MyException extends Exception
  {
  public MyException(String s)
      {
      super(s);
      }
  }
class Exception01
  {
  public static void main(String [] args)
      {
      Scanner sc = new Scanner(System.in);
      System.out.println("Enter a string");
```

```
String s;
s=sc.nextLine();
int l= s.length();
try if (s.length()<=8)
    {
     try
       {
         System.out.println("Too Small");
       }
try
    {
     if (s.length()<8)
       throw new MyException("Too Small");
     else
       System.out.println("Ok");
    }
catch(MyException ex)
    {
     System.out.println(ex.getMessage());
    }
  }
}
```

Nested Try... catch :-



Program :-

```
class Nest
 {
 public vo static void main(String [] args)
   {
   // Parent try block
    try {
      //child try block-1
       try {
         System.out.println("Inside Block 1");
          int b = 45/0;
           System.out.println(b);
            }
          catch(ArithmeticException e1) {
             System.out.println("Exception e1");
            }
         // Child try block-2
         try {
            System.out.println("Inside Block 2);
            int b = 50/0;
            Sytem.out.println(b);
            }
         catch(ArithmeticException e2) {
            System.out.println("Exception e2);
            }
```

```
            System.out.println("Just another statement");
        catch(ArithmeticException e3) {
            System.out.println("Arithmetic Exception");
            System.out.println("Inside parent try catch block");
            }
                                        e4
        catch(ArrayIndexOutOfBoundsException) {
            System.out.println("Array Index Out of Bound");
            System.out.println("Inside parent try catch block");
            }
        catch(Exception e5) {
            System.out.println("Exception");
            System.out.println("Inside parent try catch block");
            }
        System.out.println("Next Statement.....");
    }
}
```

Output :-

```
Inside block 1
Exception e1
Inside block 2
Exception e2
Inside block 3
Arithmetic Exception
Inside Parent try catch block
Next Statement.....
```

যে catch() এর exception আর্গ কার্যকর নয় সেগুলো default ভাবে print হবে এবং প্রয়োজন না হলে print হবে না।

Creating Thread by Runnable Interface:-

- Runnable interface ে শুধুমাত্র run() থাকে। তাই অন্য property কে access করতে Thread class এর object create করতে হবে।

Program:-

```
class Test implements Runnable
 {
  public void run()
    {
     for(int i=1; i<=5; i++)
        {
         System.out.println("Test Thread");
        }
    }
 }

class Main
 {
  public static void main(String[] args)
    {
     Test t = new Test();
     Thread obj = new Thread(t); //Pass object
      obj.start(); //As a Thread execute হবে
    }
    for(int i=1; i<=5, i++)
        {
```

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

output:
Test class
Test class
Main Thread
Test class
Main Thread
Main Thread

Simultaneous output

String in Java :-

. String sequence of character কে Represent করে

Java.lang → Package এ string class থাকে।

. Create string in two way.

i) By String Literal
ii) By New keyword

i) using Literal:-

```
String str = "Java";
System.out.println(str);
```

ii) Using New :-

```
String str;
str = new String("Java");
System.out.println(str);
```

Convert char array:-

```
char name[] = {'R', 'A', 'H', 'M', 'A', 'N'};
String str = new String(name);
```

Java তে String, character Array নয়। আবার NULL Termination ও নয়।

0	1	2	3	4	5
R	A	H	M	A	N

String str1 = new String(name, 2, 3);

2 → Starting Index, 3 → Range

[output :- HMA]

String length :-

- String class এ length() একটি function যা int value return করে।

```
String str = "Java";
System.out.println("Length : " + str.length());
```

String Concatination :-

- String str1 = "Java" + "Programming"

String Concatination occurs in two way. They are :-

i) Using + operator

ii) By concat() Method

- String str2 = str1 + "Exp"; //one string object
- String str3 = str1 + str2; // Both string object
- int no = 10;
  String str = "Java";
- String str4 = no + str; [10 Java]
- String str5 = str + no + 2; [Java 102]
- String str6 = 10 + 2; [ERROR]

Operator Precedence হিসাবে work করে।

- String str7 = str +(10+2); [Java12]
- String str8 = 10+2+ str; [12Java]
- String str9 = 10 + str +2; [10Java2]

ii) Concat() Method:-

Syntax:- public String concat (String s);
return a string ↳ input object

String str = "Java";
- String str1 = str.concat("Programming");
String s = "Language";
- String str2 = str1.concat(s); [Java Language]

Comparing two String using compareTo() Method:-

Syntax:- int compareTo (String str);

Value 0 হলে
$S_1 = S_2$
$S_1 > S_2 \rightarrow +1$
$S_1 < S_2 \rightarrow -1$

- Lexicographically compare করে।

String str1 = "Aman";
String str2 = "Aman";
- str1.compareTo(str2); [0]
String str3 = "Baman";
- String str1.compareTo(str3); [-1]
String str4 = "Champak";
str4.compareTo(str3); [+1]

- S1 = "Aman"
  S2 = "aman"
  S1.compareTo(S2); [-value] (65-97) $S_1 < S_2$

- S1 = "Aman"
  S2 = AmanNew"
  S1.compareTo(S2); [-3] $S_1 < S_2$
  S2.compareTo(S1); [+3] $S_2 > S_1$

- S1 = "Aman5"
  S2 = "Aman3"
  S1.compareTo(S2); [+2] $S_1 > S_2$

- S1 = "Aman";
  S2 = "Amam";
  S1.compareTo(S2); [+1] ~~$S_1 S_2$~~ 1 2 $S_1 > S_2$

Sorting Sting using compareTo():-

Bilaspur — 0	index
Raipur — 1	
Madras — 2	
Allahabad — 3	

```
class Test
{
  static string name[] = "Bilaspur", "Raipur", "Madras", "Alla
  public static void main (string [] args)
  {
```

```
int size = name.length;
String temp = null;
System.out.println("\n\t Before Sorting : ");
for(i=0; i<size; i++)
    {
     System.out.println("\n\t " + name[i]);
    }
for (i=0; i< size; i++)
    {
    for (int j=i+1; j<size; j++)
        {
         if (name[i].compareTo (name[j])>0)
             {
              temp = name [i];
              name [i] = name [j];
              name[j] = temp ;
             }
        }
    }
System.out.println("\n\t After Sorting : ");
for(int i=0; i<size; i++)
    {
     System.out.println("\n\t" + name[i]);
    }
}
}
```

Comparing String using equals() & equalIgnoreCase() :-

equals() :- Character, order, case-sensitivity check করে

```
boolean equals (object ob);
```

```
String str1 = "Hello";
String str2 = "Hello";
```

```
str1.equals (str2);
```
[True]

```
String str3 = "fasy";
```

```
str1. equals (str3);
```
[False]

```
String str4 = "hello";
```

```
str2. equals (str4);
```
[False]

equalIgnoreCase() :-

- Case-sensitivity ignore করে

```
boolean equalIgnoreCase(object ob);
```

```
Str2. equalsIgnoreCase (str4);
```
[True]

StringBuffer class :-

```
String str = new String ("Java");
```
→ Not changable

str → store

new String ("Java") → object create

Some construction while using StringBuffer:-

i) StringBuffer(); | object — empty & capacity — 16 character
& capacity increase automatic হয়।

ii) StringBuffer str = new StringBuffer();

ii) StringBuffer(int size or int capacity);

Capacity = Initally যতটুকু জায়গা নেয়। এটি default ভাবে 16 character তবে প্রয়োজনে এটি বাড়তে পারে।

StringBuffer str = new StringBuffer (50);

Initial capacity 50 তবে প্রয়োজনে এটি বাড়তে পারে।

iii) StringBuffer (String str)

StringBuffer str = new StringBuffer("Java");
value

Capacity = (4+16) ⇒ 20

Method - length() & capacity:-

int length(); ⟶ number of character

int capacity(); ⟶ Total Reserved Memory

```
StringBuffer str = new StringBuffer ("Java");
System.out.println("Length:" + str.length()); [L:4]
System.out.println(" Capacity :" + str.capacity()); [C:20
```

Program:-

```
class Test
{
 public static void main(String [] args)
    {
      StringBuffer S1= new StringBuffer();
      System.out.println("S1 Length :" + S1.length);
      System.out.println("S1 capacity :" + S1.capacity);

      StringBuffer S2 = new StringBuffer(10);
      System.out.println("S2 Length :" + S2.length());
      System.out.println("S2 capacity :" + S2.capacity());

      StringBuffer S3 = new .StringBuffer("Java");
      System.out.println("S3 Length :" + S3.length());
      System.out.println("S3 capacity :" + S3 capacity
    }
}
```

Output:-

```
S1. Length: 0
S1 Capacity: 16

S2 Length: 0
S2 Capacity: 10

S3 Length: 4
S3 Capacity: 20
```