26-02-19

Unit - IV

Inheritance

27-02-19

Inheritance:-

Inheritance is a Process of acquiring the Properties of one class to other class. The class which is inherited is called Super class (or) Base class (or) Parent Class. The class which is inhering is called sub class or derived class (or) child class.

→ In Java the sub class is inheriting superclass by using "extends" key word. It has the following syntax

Syntax:-

```
Class Sub'class Extends superclass
  {
  // methods / variables;
  }
```

→ With the inheritance we have the following goals

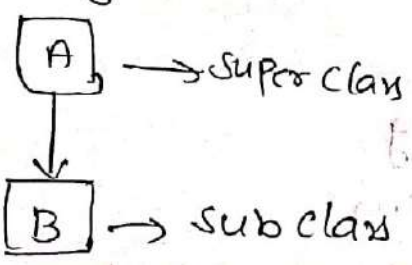
1) Method Overriding (To achieve runtime Polymorphism)
2) reusability

→ In Java we have the following types of inheritance

i) Simple or single inheritance
ii) Multilevel inheritance.
iii) Hierarichal inheritance

Simple or Single inheritance:-

In this type of inheritance the class can Extends from only one class.



Syntax:

```
Class A
{
---
}
Class B Extends A
{
----
}
```

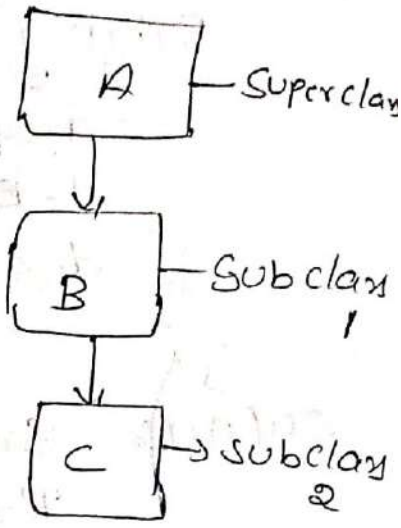
Example:

```
Class Animal
{
Void Eat()
{
System.out.Println("Eating....");
}
}
Class Dog Extends Animal
{
Void bark()
{
System.out.Println("Barking---");
}
}
Class Simple
{
Public static Void main(String args[])
{
Dog d = new Dog();
   d.eat();
   d.bark();
}
}
```

Output: Eating
Barking

Multilevel inheritance:-

In this type of inheritance we have two sub classes. The subclass 1 extends superclass and subclass 2 extends subclass 1



Syntax:-

```
Class A
{
- - - -
}
Class B Extends A
{
- - - -
}
Class C Extends B
{
- - -
}
```

Examples:-

```
Class Animal
{
Void eat()
{
System.out.Println("eating.....");
}
}
Class Dog Extends Animal
{
Void bark()
{
System.out.Println("Barking.....");
}
}
Class BayDog Extends Dog
{
Void weep()
{
System.out.Println("weeping.....");
}
}
```

```
Class Multilevel
{
Public static void main (String args[]).
{
BabyDog d=new BabyDog();
    d.Eat();
    d.Bark();
    d.Weep();
}
}
```

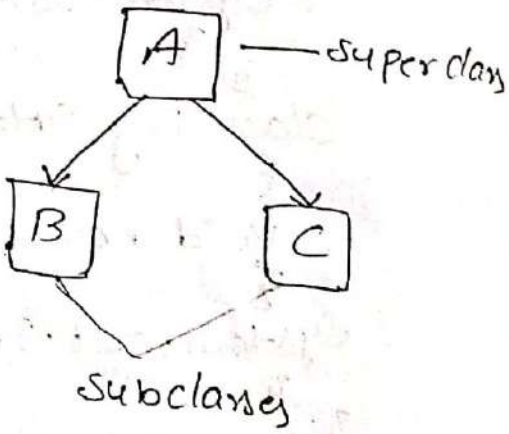
Output:- Eating
Barking
Weeping

Hierarchical Inheritance:-

In this type of inheritance we have one superclass and it is extended by two different subclasses.

Syntax:-

```
Class A
{
----
}
Class B Extends A
{
---
}
class C Extends A
{
---
}
```



Example:

```
Class Animal
{
void eat()
{
System.out.Println("Eating....");
}
}
class Dog Extends Animal
{
void bark()
{
System.out.Println("barking...");
}
}
class Cat Extends Animal
{
System. void meow()
{
System.out.Println("Meowing...");
}
}
Class Heirarichal
{
Public static void main(String args[])
{
Cat C = new Cat();
C.Eat();
C.meow();
Dog d = new Dog();
d.Eat();
d.bark();
}
}
```

Output: Eating
meowing
Eating
barking

Member Access Specifier:- extra

In Java we have the following three member access specifier. Those are

i) Public
ii) Private
iii) Protected.

Public:-

We can Create a Variable (or) method using Public access specifier. So that those Public Variable or method can accessed by anyone. If we don't have any access specifier before a Variable or Method by default it will consider as Public.

Private:-

We can Create the Private members using Private access specifier when we declared a member as a Private so that no other class members can access only its class members are can access.

Protected:-

It is similar to Private. Generally it is used in inheritance concepts. That means only the subclass Can access the Protected member

Syntax:

access specifier. return type Variable name;

Example:-

```
Public int a; (or) int a;
Private int f;
Protected int k;
```

Public int volume();

Example:-

```
Class A
{
Public int i;
private int j; // Public int j;
Public void set (int a, int b)
    {
    i=a;
    j=b;
    }
}
Class B Extends A
{
Public int k;
Public void sum()
{
k=i+j;
}S.O.P("sum="+k);
}
Class Test
{
Public static void main(String args[])
    {
    B objb=new B();
    objb.set(10,20);
    objb.sum();
    }
}
```

The above Program generates compilation errors because j can't be accessed by sub class because j can declared as Private.

Example:

```
Class Box
{
int width;
int height;
int depth;
Box()
{
width = -1;
height = -1;
depth = -1;
}
Box (int w, int h, int d)
{
width = w;
height = h;
depth = d;
}
Box (Box ob)
{
width = ob.width;
height = ob.height;
depth = ob.depth;
}
Box (int len)
{
width = height = depth = len;
}
int Volume()
{
return width * height * depth;
}
}
Class BoxWeight extends Box
{
int weight;
BoxWeight (int w, int h, int d, int wt)
```

```
    {
    width = w;
    height = h;
    depth = d;
    weight = wt;
    }
    Box weight()
    {
    width = height = depth = weight = -1;
    }
Box weight(Box weight ob)
    {
    width = ob.width;
    height = ob.height;
    depth = ob.depth;
    weight = do.weight;
    }
Box weight (int l, int wt)
    {
width = depth = height = l;
    weight = wt;
    }
}
Class Demo Box.
{
Public static void main (String args[])
    {
Box weight b1 = new Box weight();
Box weight b2 = new Box weight (10, 20, 30, 40);
Box weight b3 = new Box weight (20, 30);
Box weight b4 = new Box weight (b1);
System.out.println ("Box1 volume is :" + b1.volume());
S.O.P ("Box1 weight is : " + b1.weight);
S.O.P ("Box2 volume is :" + b2.volume());
S.O.P ("Box2 weight is :" + b2.weight);
```

```
S.O.P ("Box3 Volume is:" + b3.volume());
S.O.P ("Box3 weight is:" + b3.weight);
S.O.P ("Box4 Volume is:" + b4.volume());
S.O.P ("Box4 weight is:" + b4.weight);
  }
}
```

Output:

```
Box1 Volume is : -1
Box1 weight is : -1
Box2 Volume is : 6000
Box2 weight is : 40
Box3 Volume is : 8000
Box3 weight is : 30
Box4 Volume is : -1
Box4 weight is : -1
```

→ We are passing values to the Box class instance variables by using it child class constructor

Super Keyword: Extra

Generally a child class can know about properties of Parent class but if the Parent class also hides the details from the child class then the child class can use super keyword. we have the following two advantages with super key word.

1) To pass the parameters from child class constructor to Parent class constructor.

2) If same variables are using by both Parent & child class then we can't differentiate b/w them but using super keyword we can differentiate the superclass variable from subclass variable

→ We can pass the parameters from child class constructor to parent class by using the following syntax

Syntax:-

```
super (arg_list);
```

The above syntax is always used in child class constructor.

Example:

```
class A
{
int i, j;
A(int a, int b)
{
i=a;
j=b;
}
void showAB()
{
S.O.P("i and j are" + i + " " + j);
}
}
class B Extends A
{
int k;
B(int a, int b, int c)
{
super(a,b);
k=c;
}
void showC()
{
S.O.P("k=" + c);
}
}
class Demo
{
Public static void main(String args[])
{
B obj b = new B(10, 20, 30);
Obj b. showAB();
} Obj b. showC();
}
```

→ we can also use super keyword inorder to differentiate the variables which are same in both subclass and superclass.

it has the following syntax

```
super. variable_name;
```

Example:

```
class A
{
int i;
void showsuper()
 {
s.o.p("Super class i="+i);
 }
}
class B Extends A
 {
int i;
B(int a, int b)
 {
super. i=a;
   i=b;
 }
void showsub()
 {
 s.o.p ("subclass i="+i);
 }
}
class Demo
 {
Public static void main (string args[])
 {
 B. obj b = new B(10,20);
 Obj b.showsuper();
 Objb.showsub();
 }
}
```

27-02-19

Note:

→ Always the super keyword used as a first statement in subclass constructor.

Method overridings- Extra

When subclass as same method name, same type signature as superclass method then the subclass method is overriding the super class method.

(or)

A method overriding is a mechanism to override superclass methods by using its subclass methods.

→ Generally the method overriding is used to achieve runtime polymorphism.

Syntax

```
class superclass
{
------
return type method name()
  {
  ---
  }
-------
}
class-subclass Extends superclass
  {
  -----
  return type method-name()
    {
    ---
    }
  -----
  }
```

* The subclass method signature is same as super class signature. signature (Method name, return type and parameter)

Then the method is called overring method.

Syntax

```
class A
```

Eg:-

```
Class A
{
 int i,j;
A(int a, int b)
  {
   i=a;
   j=b;
 }
 void show()
  {
 System.out.Println("i and j are:"+i+" "+j);
  }
}
 Class B Extends A
{
int k;
B(int a,int b, int c)
 {
 super(a,b);
  k=c;
}
 void show()
 {
System.out.Println("k="+k);
 }
}
 Class Override
  {
 Public static void main (String[] args)
  {
  B obj =new B(10,20,30);      O/P: K=30
    obj.show();
  }
}
```

The above Program the sub class show method is overriding the super class show method. & we have create the object for subclass B. using the class B object we are accessing the show method

but it is always call the sub class show method Version it will not call the super class show method version.

→ If we want to call the super class show method Version using sub class object then we have to use the super key word inside the subClass show method version.

Eg:-

```
Class A
 {
 int i, j;
 A(int a, int b)
   {
   i=a;
   j=b;
 }
 void show()
 {
System.out.Println("i and j are:"+i+"  "+j);
 }
 Class B Extends A:
 {
 int k;
 B(int a, int b, int c)
  {
  super(a,b);
   k=c;
 }
  void show()
   {
   super.show()
  System.out.Println("k="+k);
   }
  }
   Class OVerride
  {
  Public static Void main (String [] args)
   {
     B obj = new B(10,20,30);
```

```
        obj.show()
    }
}
```

Output:- i and j are : 10 20
k = 30

Dynamic method dispatch:-

The Dynamic method dispatch is a mechanism by which to call a overriding method and resolved at run-time, but not at compile time in this mechanism sub class object are refered by super class reference

Syntax:

```
Super class reference = Superclass obj;
        reference.super.class.method();
            reference = sub-class-obj;
            reference.subclass.method();
```

Example:

```
class A
{
int i, j;
A(int a, int b)
{
  int i=a;
      j=b;
  }
  void show()
  {
  System.out.println("i and j are: "+i+" "+j);
}
class B extends A
  {
  int k;
  B(int c);
```

```
    }
     k=c;
    }
    void show()
     {
    system.out.Println("k="+k);
     }
    }
    class override
    {
   Public static void main (String[] args)
    {
    A objA = new A(10,20);
    B objA = new B(30);
    A ref;
     ref=objA;
     ref.show();
     ref=objb;
     ref.show();
    }
    }
```

01-03-19

Abstraction:-

An Abstraction is a method of hides the implementation details and show the functionality of the method.

→ Abstraction can be acheive by using following two ways.

i) Abstract class or using abstract keyword (0 to 100%)

ii) Interface (100%)

Abstract class:-

A class contains atleast one abstract method is called abstract class. Here the class and methods are specified by "abstract" keyword as follows

```
abstract class class-name // abstract class
{
abstract return-type method1(); // abstract method
  {
    - - - -
    - - - -

  return-type method2(); // concrete method
    {
    - - -
    - - -
  }
}
```

→ The method without any implemention and it is specified by an abstract keyword then those methods are called "abstract methods."

→ The method without an abstract keyword is called "concrete method."

→ All abstract methods are implemented by "class which is extending the abstract class.

→ If a class is already extending the abstract class but it doesn't want to implement the abstract method at this time class also become an abstract class.

→ An abstract class can't be instanciated that means we can't create an object for an abstract class

Eg:
```
abstract class Animal
    {
abstract void sound();
    }
    class Lion extends Animal
      {
        void sound()
      {
```

```
        System.out.Println("Roar.....");
    }
}
Class Dog Extends Animal
{
    void sound()
    {
    System.out.Println("Bark....");
    }
}
Class Test
{
Public static void main(String args[]);
{
    lion l=new lion();
    Dog d=new Dog();
      l.sound();
      d.sound();
}
}
```

final Keyword. Extra

In Java the final keyword is used as follows.

i) It is used to declare and initialize the constants are static.

Eg:- 1) final int filelength = 30;
2) final int Max = 50;

ii) It is used to avoid the inheritance in Java. for Example if a class is declared as final then no other class will Extends the final class.

Eg:-
```
final class A
{
.....
}
class B Extends A // Can not extend class A
{
...
}
```

iii) To is also used to Prevent from method Overriding. Suppose if a method is declare as final then no other methods are override

for Example

```
Class A
{
final void show()
{
---
}
}
Class B Extends A
{
Voidshow() // can not override show() method
{
----
}
}
```

Order of Constructor Execution: extra

In Inheritance Execution of Constructor starts as derived order that means first will Execute the Super class Constructor then will Executes the derived class Constructor.

Ex:-

```
Class A
{
A()
{
System.out.Println("A's Constructor");
}}
Class B Extends A
{
B()
{
System.out.Println("B's Constructor");
}
}
```

```
class C extends A
  {
  C()
  {
System.out.Println("C's constructor");
  }
}
  class Test
{
Public static void main(String args[])
  {
  C objC = new C();
  }
}
```

Output:- A's constructor
B's constructor
C's constructor.

Packages:-

02-03-19

Java Provides a mechanism that is used to Partitioning the whole class namespace into manageble chunks. (Parts). This mechanism is called Package. The Packages are containers which are used to Store classes, interfaces. That are used class namespace as compartments. we have the following advantages with Packages

1. It will categorize the classes and interfaces
2. It will remove name collisions (Class name)
3. It will Provide access Protection and visability

we have the following two types of Packages.

1) Built-in Packages.
2) User defined Packages.

Built in Packages:-

These Packages are already Predefined. This are also known as API Packages (Application Program interface) Every API Package has large no. of class, methods & inter-faces.

If we want these API Packages in to our program, we have the following syntax.

Syntax:-

~~Import API p~~ import Api Package;

Using import keyword a particular API Package in to our Program. we have some of the API Packages as follows

API Package	Description.
1) Java.lang	→ Language support Package. It is used to import the classes like String class, Exception class, Thread class etc. Eg:- import Java.lang.*; * → refers large no.of classes to interface
2) Java.util	→ Language utility support package. It is used to import the classes like vectors, hash tables, dictionary, data & time etc. Eg:- import Java.util.*;
3) Java.io	→ It is used to support ↑ to Perform input/output Operations. for Eg BufferedReader, InputStreamReader, PrintWriter etc. Eg: import Java.io.*;
4) Java.net	→ It is used to provide the network b/w two systems or clients. Eg: Socket Eg:- import Java.net.*;

5) Java.applet → It is used to create applet (browser enable Program)
Eg:- Applet class
import Java.Applet.*;

6) Java.awt (abstract window tooling kit) → It is used to create the graphical user interface. Eg:- windows, list, Menus, texts---
Eg:- import Java.awt.*;

7) Java.Swing → It is also used to create the graphical user interface. for eg windows, list, menus etc these are light weight compon
Eg:- import Java.swing.*;

8) Java.SqL (structure query language) → It is used to create data base for the application
Eg:- Result Set

User defined Packages:- 05-03-19

These type of Packages are created by user these are not Predefined Packages. we can create the user defined Packages by using following syntax.

Syntax:

Package Package-name;

where Package is a keyword and Package-name is a Package name.

In Every Java Programm if you are using Packages then the Package statement should be 1st statement in a Programm.

Eg: Package Pack;

The above statement creates the Package Pack. we can also create subPackages using the following statement.

Eg:- Package Pack.P1.class;

Compilation of Packages / Adding class Path:-

Generally Java file is compiled by javac file.java but the packages are compiled as follows

javac -d . filename.java

(spaces)

where -d refers directory, '.' refers current working directory (CMD)

Run:

we can run Package Program as follows

Java Package name . class-name

Eg:- ① Package Pack

```
import java.io.*;
class simple
{
public static void main(String args[])
{
 System.out.println("Hello");
}
}
```

Compile & run;

javac -d . simple.java

Java Pack.simple

O/P: Hello.

Program ②

```
Package Pack;
class Balance
{
string name;
int bal;
Balance (string nm, int b)
```

```
    {
    name =nm;
    bal =b;
  }
  void show()
    {
    if (bal<0)
    {
System.out.println (name+ "Rs." +bal)
  }
}
}
  class Account
  {
Public static void main(String args[])
  {
  Balance c[] =new Balance [3];
      c[0] = new Balance ("CCC", 200);
      c[1] = new Balance ("CCE", 150);
      c[2] = new Balance ("IT", -50);
      for (int i=0; i<3; i++)
        c[i].show();
  }
}
```

D:\Studentdata
↓
II year
↓
Java
- Simple.Java
- Pack
  - Simple.clas
  - Account.cla
  - Balance.clas
- Account.Java

O/P: IT Rs. -50

Importing Package:

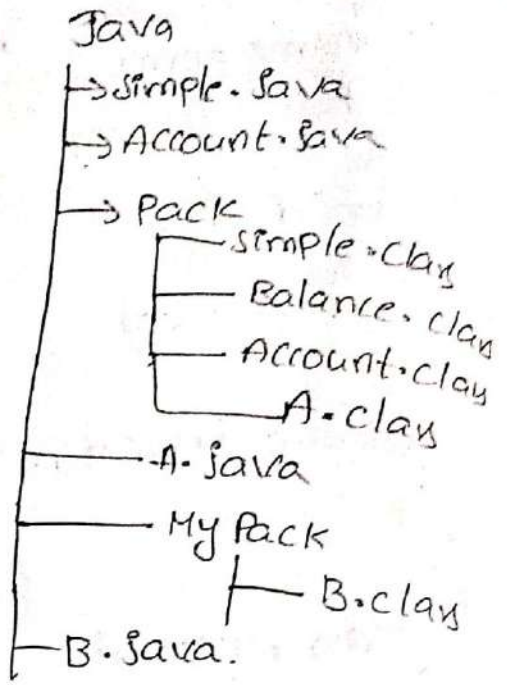
1) import Package_name.*;
2) import Package_name.class_name;
3) Using fully qualified name.

1) import Package_name.*;

In this method we can import all classes from on userdefined Package to another user defined Package.

Eg:

```
Package Pack;
Public Class A
  {
  Public void msg()
    {
    S.O.P("Hello");
    }
  }
```




```
Package myPack
 import Pack.*;
 Class B
 {
 Public static void main(String args[])
    {
    A obja = new A();
    obja.msg();
    }
  }
```

2. import Packagename.Class-name;

In this type we can import only the specific class from one user defined to another user defined type

Eg:

```
Package myPack;
  import Pack.A;
   Class B
  {
  PublicStatic Void main(String args[])
    {
    A obja = new A();
     obja.msg();
    }
  }
```

3. using fully qualified name:-

In this method we can directly use Package-name before the class name as follows

Eg: Package mypack;

```
class B.
{
public static void main (String args[])
{
Pack.A obja = new Pack.A();
  obja.msg();
}
}
```

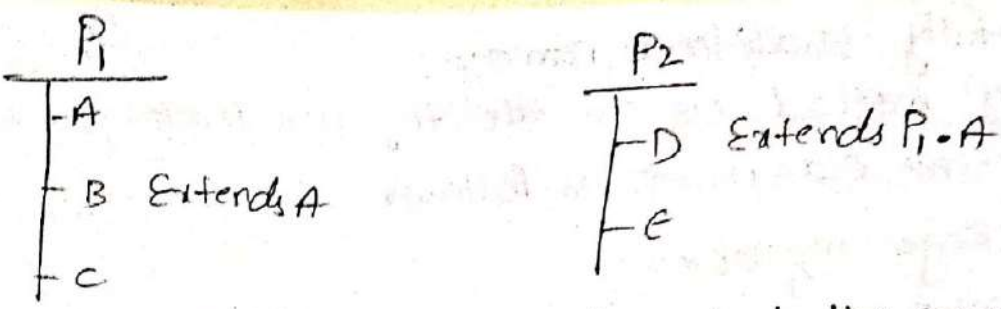
Access Protection.

In Java the Packages are Provides visibility and access Protection the visibility of a variable or member can be categorised as follows. Generally we have 3 types of access specifiers those are Public, Private & Protected.

If you don't have any explicit modifier than these members are accessed by only its package classes not the other package.

The following table shows the categories of visibility and access Protection.

	Private	No Modifier	Protected	Public
Same class	Yes	Yes	Yes	Yes
Same pkg & Sub class	NO	Yes	Yes	Yes
Same Package & not a subclass	NO	Yes	Yes	Yes
Other pkg & Sub class	NO	NO	Yes	Yes
Other Packages & not a subclass	NO	NO	NO	Yes.



write a Java Programm to demonstrate the working of access Protection using Packages?

Protection.java:

```
Package P1
Public class Protection
    {
    int n=1;
    Public int n_Pub=2;
    Private int n_pri=3;
    Protected int n_Pro=4;
Public Protection()
    {
    S.O.P("In Protection class");
    S.O.P("n="+n);
    S.O.P("n_Pub="+n_Pub);
    S.O.P("n_Pri"=+n_Pri);
    S.O.P("n_Pro"=+n_Pro);
    }
}
```

Derived.java:

```
Package P1
Class Derived Extends Protection
    {
    Derived()
    {
    S.O.P("In derived class");
    S.O.P("n="+n);
    S.O.P("n_Pub="+n_Pub);
// S.O.P("n_Pri="+n_Pri);
```

```
    S.o.p("n-Pro=" +n_Pro);
  }
}
```

Same Package.java

```
Package P1
Class Samepackage
{
Samepackage()
 {
Protection P= new Protection();
 S.O.P ("In same Package class");
 S.O.P ("n=" + P.n);
 S.O.P ("n-Pub=" + P.n_Pub);
// S.O.P ("n-Pri=" + P.n_Pri);
 S.O.P ("n_Pro=" + P.n_Pro);
 }
}
```

Demo.java

```
  Package P1
 Class Demo
{
Public static void main(String args[])
{
Protection P=new Protection();
Derived d = new Derived();
Same Package S= new Samepackage();
 }}
```

Protection2.java:

```
  Package P2;
 Class Protection2 Extends P1.Protection
 {
Protection2()
```

```
}
S.O.P ("In Protection 2 class");
//S.O.P ("n="+n);
 S.O.P("n_Pub="+n_Pub);
// S.O.P("n_Pri="+n_Pri);
  S.O.P ("n_Pro="+n_Pro);
 }
}
```

Other package.java:-

```
  Package P2
  class Other Package
   {
   Other Package()
   }
 S.O.P ("In other Package class");
  P1.Protection P= new P1.Protection();
 //S.O.P ("n="+n);
   S.O.P("n_Pub="+P.n_pub);
 //S.O.P ("n_Pri="+P.n_Pri);
 //S.O.P ("n_Pro="+P.n_Pro);
   }
 }
```

Demo2.java:-

```
  Package P2
  class Demo2
  {
  Public static void main (String.args[]).
   {
   Protection2 P = new Protection2();
   OtherPackage O = new OtherPackage();
  }
}
```

Run: Java P1.Demo, Java P2.Demo2

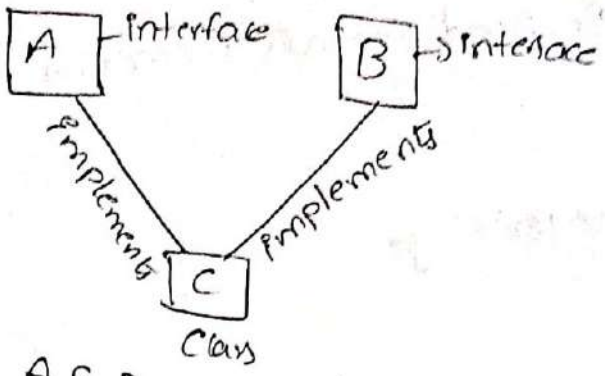
Skeleton:

```
Package P1;
Public clas Protection
    {
     - - - - -
    }
  clas Derived Extends Protection
   {
   }
 Clas Same Package
   {
   }
  Clas Demo
  {
  PSVm()
   {
   }
}
```

```
Package P2;
 Clas Protection2 Extends P1.Protection
   {
   }
 Clas OtherPackage
   {
   }
Clas Demo2
    {
    PSVm()
    {
    }
}
```

Interface:-

In Java interfaces are used to achieve Multiple inheritance and abstraction (100%). Generally a class can Extends only one class it can't Extends more than one class. But a class can implement more than one interface



In above fig A & B are interfaces and C is a class

→ We can define an interface using interface keyword it will specify what it does but not how it does it.

→ The interfaces are similar to class but in interface their is no instance type variables.

→ An interface contains only the abstract methods or static (or) final variables. All these members are by default public type.

→ The method without any implementation and it Ends with semicolon is called abstract Method.

→ In interface by default all methods are abstract type.

→ In interface all variables are final (or) static variable

→ An interface can't be instantiated that means we can't create object.

→ We can access the implementation methods by using interface reference.

→ A class can implement any no. of interfaces.

→ An interface can't implement another interfa

→ An interface can Extends another Interface
→ An interface can be implemented by any no.of classes.

Defining interface:-

An interface can be defined as follows.

Syntax:-

```
access interface <interface_name>
{
return-type method1 (parameter_list);
return-type method2 (parameters_list);
        .
        .
return-type methodN (parameters_list);
return type final_variable1;
return type static_variable2;
- - - -
}
```

Example:

```
interface Showable
{
Void show();
Void sound();
}
```

Implimentation of an interface:

We can implement an interface by using "implements" keyword. The interface methods are implemented by a class which implements an interface. We can implement as follows

Syntax:-

```
class class-name implements interface_name
{
Public return-type method1 (Parameter-list)
{
- - - -
}
}
```

Eg: write a Java Programm to demonstrate the working of interface.

```
Interface Showable
{
Void show();
Void sound();
```

```
}
class Dog implements showable
{
 Public void show()
  {
 System.out.Println("Dog");
  }
Public void sound()
  {
  System.out.Println("woof");
  }
class lion implements showable
  {
  Public void show()
   {
  System.out.Println("Lion");
   }
 Public void sound()
  {
 System.out.Println("Roar");
 }
  class Test
   {
   PSVM(String args[])
    {
     Dog d = new Dog();
     Lion l = new Lion();
      d.show(); Showable s;
     d.sound();  s=d;
      l.show();  s.show();
      l.sound();  s.sound();
    }  s=l;
   }  s.show();
      s.sound();
    }
```

by using interface reference

→ we can access implementation method by creating interface reference. The above example showable is a interface 's' is a reference of showable interface.

A class can implement more than one interface as follows. 08-03-19

Eg:-

```
interface showable
{
 void show();
 void sound();
}
interface Printable
{
 void Print();
}
class Dog implements showable, Printable.
{
 Public void show()
 {
S.O.P ("Dog");
 }
 Public void sound().
 {
 S.O.P("woof");
 }
 Public void Print()
 {
S.O.P ("Dog woof");
 }
}
class Lion implements showable, Printable
{
 Public void show()
 {
 S.O.P ("Lion");
 }
 Public void sound()
 {
 S.O.P ("Roar");
 }
 Public void Print()
 {
 S.O.P("Lion Roar");
 }
}
```

```
Class Demo
{
Public static void main (String args[])
  {
  Dog d = new Dog();
  Lion l = new Lion();
   d.show();
   d.sound();
   d.Print();
   l.show();
   l.sound();
   l.Print();
  }
}
```

Partial Implementation:-

If a class doesn't implement all methods in a interface then these type of implementation is called Partial implementation. when a class has a Partial implementation then the class is to be a abstract class. The remaining implementation can be done by the class which extends by the abstract class.

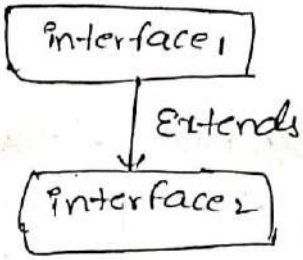
Programme:-

```
interface showable
 {
 void show();
 void sound();
 }
abstract class Dog implements showable
  {
  Public void show()
   {
   S.O.P("Dog");
  }
}
```

```
Class BabyDog Extends Dog
{
Public void sound ()
{
S.O.P ("BabyDog woof");
}
}
  Class Partial
  {
  Public static void main (string args[])
  {
  BabyDog b= new BabyDog ();
    b.show();
    b.sound();
  }
  }
```

Extending Interfaces:-

An interface can be extended another interface using extends keyword so that we have an inheritence blw interfaces.



Example:-

```
interface showable
{
  void show();
  void sound();
}
interface printable extends showable
  {
  void Print();
  }
  class cat implements Printable
    {
    public void show()
    {
    S.O.P ("cat");
    }
    }
```

```
Public void sound()
{
S.O.P ("meow");
}
Public void print()
{
S.O.P ("cat meow");
}
}
Class Extend interface
{
Public static void main (String args[])
{
Printable P= new Cat();
    P.show();
    P.sound();
}   P.Print();
}
```

Applying interface:-

The following Programm shows the implementation of stack. A stack has two operations those are Push and pop. Push operation is used to insert the element on the stack. POP operation is used to delete the element from stack. The following Example will Create interface called stack. In stack interface we have 2 methods. those are Push & POP using Stack interface we will implement fixed stack & dynamic stack. The fixed stack has only fixed size where as dynamic stack as dynamic size. so that we can store only fixed Elements in a fixed stack. but in dynamic stack we can store the dynamic no. of Elements.

```
interface Stack
{
 void push (int item);
  int pop();
}
class fixedStack implements Stack
 {
 Private  int stck[];
 Private  int top;
 FixedStack (int size)
 {
 stck = new int [size];
  top=-1;
 }
 public void Push (int item)
  {
  if (top= = stck. length-1)
  {
 S.O.P ("stack overflow");
  }
 else
  {
  stck [++top] = item;
  }
 public int pop()
  {
   if (top = = -1)
    {
     S.O.P ("stack underflow");
    return 0;
   }
  else
   {
   return stck [top--];
   }
}
class DynamicStack implements Stack
 {
  Private int stck[];
  Private  int top;
  Dyramic Stack (int size)
```

```
{
stck = new int [size];
top = -1;
}
Public void push(int item)
{
if(top == stck.length-1)
{
int temp[] = new int [stck.length*2];
for(int i=0; i<stck.length; i++)
temp[i] = stck[i];
stck = temp;
stck[++top] = item;
}
Else
{
stck [++top] = item;
}
Public int POP().
{
if(top == -1)
{
S.O.P ("stack underflow");
return 0;
}
Else
{
return stck [top--];
}
}
class StackDemo
{
Public static void main (String args[])
{
FixedStack f = new FixedStack (5);
DynamicStack d = new DynamicStack(5);
Stack s;
```

```
S.O.P ("Implementing fixed stack");
    S=f;
for (int i=0; i<5 ; i++)
    S. Push(i);
S.O.P ("The Popped elements are");
  for(int i=0; i<5 ; i++)
S.O.P ( S. POP());
S.O.P ("Implementing Dynamic stack");
    S=d;
for (int i=0; i<15 ; i++)
    S.push(i);
S.O.P ("The Popped Elements are");
  for(int i=0; i<15; i++).
S.O.P (S.POP());
}
}
```

13-03-19

Exception Handling:-

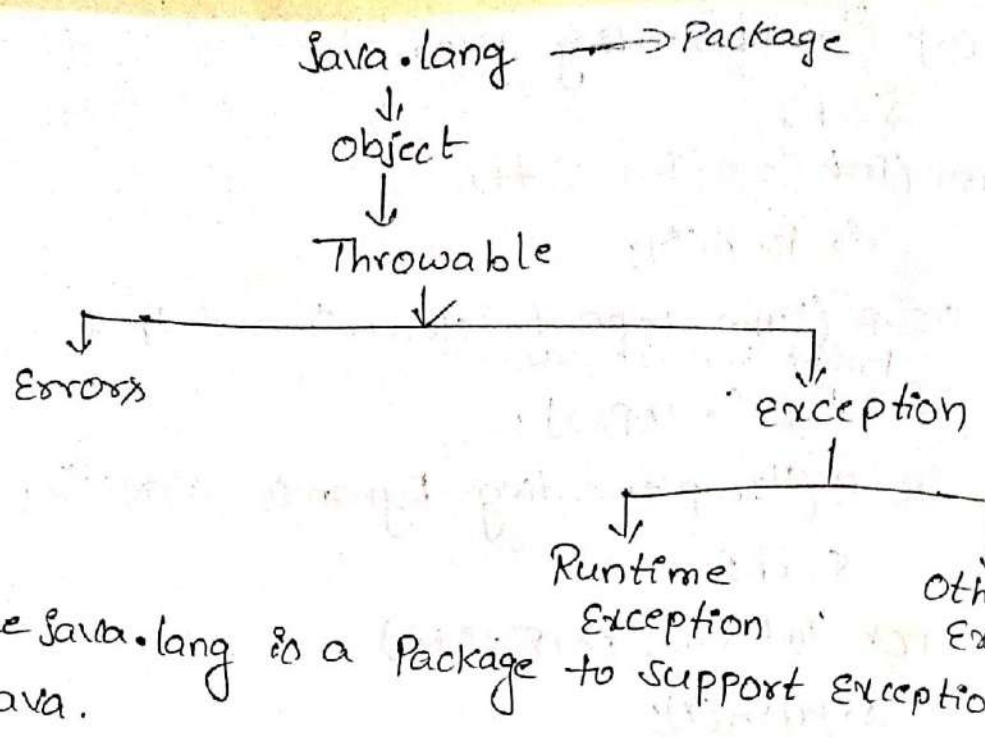
An Exception is an abnormal condition that will arrive in a piece of code at run time. An exceptions also called as run time errors.

[or]

Java exception is an object that describes an exceptional condition which occured in a piece of code at run time.

when an exception is occured a complier will not generate any errors at compile time but it will generate errors at runtime. These exceptions also terminate the flow of execution of our progra that means when an exception is occured immediatly it will stop the flow of execution at the same line where an exception is occured.

we have the following Java exception clasification



1. The java.lang is a Package to support Exceptions in java.
2. An object is a Parent class of all classes.
3. The throwable is Parent class of all Exceptions and errors
4. Exception is a Parent class of run time Exceptions and other exceptions in java.

In java we can handle the Exception using Exception handling keywords those are i) try ii) catch iii) throw iv) throws v) finally.

Ex: Program:-

```
class Exco
{
Public static void main (String args[3])
{
 int d=0
 int a=42/d;
S.O.P("I am in main method");
}
}
```

when we compile the above Program it will not generate any Errors it will compile successfully but while running it will generate an Exception as follows.

java.lang.ArithmeticException: / by zero at
Exco.main (Exco.java:6)

The above Programm is terminated at line 6 because in this line we are performing division by zero. So that it will generate on arthematic Exception. which is a sub-class of run-time Exception To handle Java Exceptions we have to use 5 exceptional keywords in our Programm.

1. try:

The try block always represents Program statements which you want to monitor for an Exception.

Syntax:
```
try
{
- - - -
- - - -
}
```

2. catch:

when an exception is occured in a try block then the Java run-time system will automatically throwon the generated exception, the catch block always catch the throwon exception and handle it.

syntax:
```
try
{
- - - -
}
catch (Exceptiontype Exception obj)
{
- - - -
}
```

3. Throw:

whenever an exception is occured the Java run time system will automatically thrown an exception but if we want to throwon an exception manually we have to use throw statement.

Syntax: throw throwable instance;

Eg: throw new IllegalAccessException();

4. throws:

If a method has a capable of causing an Exception and it doesn't handle it at this case the method will specify the behaviour so that callers of a method will protect against the exception. The throws specifies the list of Exceptions that are caused by a method. Generally the throws keyword will specifies all Exceptions except run time Exceptions and its sub classes.

Syntax:

```
type method_name (Par_list) throws Exception_list
{
// body of the method;
}
```

5. finally:-

finally block is an optional block. This block will be execute after completion of try, catch block it has the statements which to be execute compulsory even an Exception is occured.

14-03-19

```
try
{
-----
}
finally
{
-----
}
```

(or)

```
try
{
---
}
catch(---)
{
---
}
finally
{
---
}
```

try-

A try block has Programm statements which you want to monitor for an exception. when an Exception is occured then the further statement will not Execute try block. when an exception

occur it will be thrown by Java run-time system. the thrown Exception is catched by catch-block. It is a handle block to handling generator Exception

Ex:- Write a Java program to demonstrate the working of try-catch block?

```
Class Excd
{
Public static void main (String args[])
 {
 try
  {
   int d=0;
   int a = 42/d;
   S.O.P ("I am in try");
  }
 catch (ArithmeticException ae)
  {
 S.O.P ("Exception :" +ae);
  }
  S.O.P ("I am in main");
 }
}
```

Multi catchblocks:-

A try block can have more than one catch block but the Exception is handling by it matched Exception catch blocks that means only one catch block will be Executed based on the matched Exception.

** The order of catch block follows from specific to general that means 1st it will handle the specific Exception like Arithmetic Exception and next it will be handle the generic Exception like Exception.

```
class exc2
{
public static void main(String args[])
{
 try
 {
  int d=0,con arg.length;
  int a=42/d;
  int c[]={1};
   c[42]=90;
 }
catch (ArithmeticException ae)
  {
 S.O.P(ae);
  }
catch (ArrayIndexOutOfBoundsException e)
 {
 S.O.P (e);
 }
S.O.P ("I am in main");
}
```

runtime)
Java exc2 X
Java exc2 one Two

Nested try:-

A try block within another try block is called nested try. So these nested try will be use whenever an Exception is occured outside and Exception is occured inside. The syntax as follows

```
try
{
 try
 {
  ----
 }
 catch(---)
  {
  ---
  }
}
catch(---)
 {
 ---
}
```

Ex:-3

```
class Exc3
{
Public static void main (String args[])
  {
  try
    {
     int a = args.length;

     try
     {
       if(a==1)
         {
          a=a-a;
          int d=42/a;
         }
       else if (a>=2)
         {
          a=a-a;
          int d=42/a;
         }
       catch (ArithmeticException e)
         {
          System.out.println(e);
         }
       int c[]={1};
       s.o.p (c[40]);
     }
      catch (ArrayIndexOutOfBoundsException ae)
       {
        s.o.p (ae);
       }
      s.o.p ("rest of code");
    }
```

15-03-19

Ex:-4

```
class NestedTry
{
Public static void main (String args[])
  {
    try
    {
     int a = args.length;
```

```
int b = 42/a;
System.out.Println("a=" +a);
try
 {
  if(a==1)
  a = a/(a-a);
  if(a==2)
  {
  int c[] = {1};
   c[42] = 90;
  }
 catch(ArrayIndexOutOfBoundsException e)
   {
  System.out.Println("Caught." + e);
   }
}
 catch (ArithmeticException ae)
   {
   System.out.Println("Caught:" +ae);
   }
 }
}
```

Javac Nested try.Java

① Javac Nested try

② Javac Nested by one

③ Java Nested try one

Throw:

When an exception occur then Java run time system will automatically thrown an Exception But we want to thrown an exception manually we have to use throw statement.

Syntax: throw throwable instance;

Ex: /Class ThrowDemo

```
{
Static void validate (int age)
   {
   if(age<18)
   {
   throw new ArithmeticException("Not valid");
   }
```

```
else
{
S.o.P ("welcome to vote");
}
}
public static void main (string args[])
{
try
{
Validate (13);
}
Catch (ArithmeticException ae)
{
System.out.Println (ae);
}
System.out.Println ("rest of the code");
}
}
}
```

Throws:-

Throws keyword will be used along with the method signature because a method has a capable of generating an Exception but it doesn't handle by its own. So that it will specify its behaviour by using throws keyword. So that the caller of the method will take care by the generated Exception (by using try-catchblock.

Syntax:

```
type-name method (Par-list) throws Exception list
{
- - - - - -
}
```

Throws keyword will be list all Exceptions Except runtime Exceptions and its Sub classes (It will list only checked Exceptions).

Scanned by CamScanner

```
Class abc  → import java.lang.*;
           import java.io.*;
{
void method( ) throws IOException
{
throw new IOException ("device Error");
}
}
class ThrowsDemo
{
Public Static Void main (String args[])
{
abc a = new abc();
try
{
a.method();
}
Catch (IOException e)
{
System.out.println ("Exception handled");
}
System.out.println ("rest of code");
}
}
```

→ we have the following two types of Exceptions

* Unchecked Exceptions
* Checked Exceptions

Onchecked Exceptions:-

This Exceptions are not checked by the Compiler those Exceptions are called unchecked Exceptions

Ex:- ArrayIndexOutOfBoundsException, ArithmeticException etc.

Checked Exceptions:-

The Exception which are checked by the compiler those are called checked Exceptions.

Ex:- ClassNotFoundExceptions, InterruptedException.

Built-in Exceptions (or) Predefined Exceptions.

Exception name	Meaning
1. ArithmeticException	– Arithmetic errors like division by zero
2. ArrayIndexOutOfBoundsException	– when array index is exceed its bounds or limit or its index.
3. ArrayStroreException	– The invalid array Elements are store
4. IndexOutOfBoundsException	– The index is exceeded by its bound or limit
5. StringIndexOutOfBoundsException	– string index exceede by its bound.
6. ClassCastException	– The invalid class cast type.
7. NullPointerException	– when a Null is occured
8. NumberFormatException	– Mismatched number format.
9. NegativeArraySizeException	– when array size is -Ve
10. IllegelArgumentException	– when we pass an illegal arguments.
11. IllegalStateException	– An Illegel state is occured
12. IllegelThreadStateException	– missmatched thread state (or) running a mismatched state.
13. UnSupportedOperationException	– when unsupported Operation is execution
14. TypeNotfoundException	– An illegal type is used.
15. SecurityException	– Security is denied.

* The above all exceptions are unchecked run time exceptions and its sub classes.

1) ClassNotFoundException – If there is no such type of class.

2) NoSuchFieldException – If there is no such field

3) NoSuchMethodException – If there is no such method

4) InterruptedException – when a thread is running an interruption is occured

5) InstantiationException – Trying to create an object for abstract class

6) IllegalAccessException – trying to access an Illegal members (Private)

The above all exceptions are checked Exceptions.

User defined exceptions:

We can also create an user defined exception by Extending an Exception class.

Syntax:

```
class class-name extends Exception
{
- - - -
}
```

Example:

```
class MyException extends Exception
{
Private int detail;
MyException (int a)
{
detail = a;
}
Public String toString ()
{
return "MyException [" + detail + "] ";
}
}
```

```
class ExceptionDemo
{
static void compute (int a) throws MyExceptio n
{
System.out.println (" call compute (" +a+ ") ");
if (a > 20)
throw new MyException (a);
S.O.P ("Normal Exit");
}
Public static void main (String args[])
{
try
{
compute(1);
compute(25);
}
catch (MyException e)
{
S.O.P (e);
}
}
}
```

O/P:-
call compute(1)
Normal Exit
call compute (25)
my exception (25

finally:-

The finally block is a optional block. It is always executed after try or catch block completion of

If you want to keep the exit statements of our program then we have to keep all statements in finally block.

Syntax:-

```
try
{
---
}
finaly
{
---
}
```

```
try
{
---
}
catch
{
---
}
finally
{
---
}
```

Example:

```
Class finallyDemo
{
Public static void main (String args[])
{
try
{
int d=25/5; (or) int d=25/0;
}
catch (NullPointerException n)
{
System.out.println ("Exception handled");
}
finally
{
S.O.P ("I am in finally block");
}
S.O.P ("rest of code");
}
}
```

→ If an Exception is occured then their is no match catch block. The default handler will take the responsibility to handle the Exceptionand then it will Executed the finally block.

→ If there is a matched Exception catch block then it will be handle and again it will execute the finally bbck.

→ The finally block is executed irrespective of Exception Occurance.

Assignment - IV

1. Define an Exception. Explain all keywords in Exception Handling. How to create our own Exception with Example?

(throws)