Inheritance - Hierarchical Abstractions :

Reusability is another important feature of OOP. It is better if we could reuse something that already exists rather than creating the same again & again. Java supports this concept. Java classes can be reused in several ways. This is done by creating new classes, and reusing the properties of existing classes.

The process of deriving a new class from a old class is called as inheritance. The old class is known as super class or base class or parent class and the new class is known as sub class or derived class or child class. By using inheritance subclasses can inherit all the variables & methods of their ~~parent~~ super classes.

There are 5 types of inheritance -

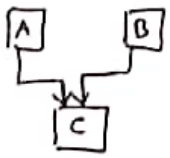
1. Single inheritance
2. multiple "
3. hierarchical "
4. multilevel "
5. hybrid "

1. Single inheritance :- here there is only one super class & one sub class

Super class

Sub class

2. multiple inheritance :- here a class is derived from several super classes ie there is only one sub class & many super classes.

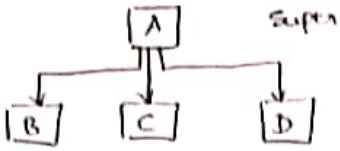


Super

Sub

Java doesnot support multiple inheritance, but it can be solved through interfaces. Interfaces helps to prevent some variables & methods from being accessed by a sub class by declaring them as 'private'.

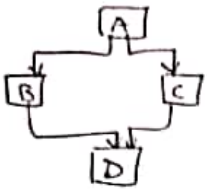
3. hierarchical inheritance :- here there is only one super class & sub classes. these subclasses are derived from only one super class.



4. multilevel inheritance :- here a class is derived from another sub class

here class 'c' is derived from B, which is derived from A. So, B acts as a subclass to A and super class to C.

5. hybrid inheritance :- it is a combination of multiple & hierarchical inheri.



Defining a sub class :-

A subclass is defined as -

```
class subclassname extends Superclassname
{
    variables declaration;
    methods declaration;
}
```

The keyword 'extends' means the properties of 'superclass' are extended to 'subclass'. Now, the subclass contains its own variables & methods and also the variables & methods of its superclass.

program :-

```
class Room
{
    int length, breadth;
    Room (int x, int y)
    {
        length = x;
        breadth = y;
    }
    int area()
    {
        return (length * breadth);
    }
}
```

(1)

```
class Room1 extends Room
{
    int height;
    Room1 (int x, int y, int z)
    {
        super (x, y);          // Pass values to superclass
        height = z;
    }
    int volume()
    {
        return (length * breadth * height);
    }
}

class one
{
    public static void main (String args[])
    {
        Room1 r = new Room1 (5, 6, 7);
        int area1 = r.area();          // super class method
        int vol = r.volume();          // sub  "  "
        System.out.println (" Area of room = " + area1);
        System.out.println (" volume of room = " + vol);
    }
}
```

Room1 r = new Room1 (5,6,7);

this statement first calls the 'Room1' constructor, which in turn calls the 'Room' constructor, by using 'super' keyword. Finally, the object 'r' of the subclass 'Room1' calls the method 'area' defined in the super class and also the method 'volume' defined in the subclass.

b. Combination – [illegible]

one parent class.

Benefits of inheritance :-

1. Increased Reliability – if a code is frequently executed then it will have very less number of errors. When the same components are used in two & more applications, the code will be exercised more than code that is developed for a single application. Thus, it is very easy to find the bugs present in this kind of code. Thus, the applications that use this component are more error free.

2. Software Reusability – properties of a parent class can be inherite by a child class. But, it does not require to recwrite the code of the inherited property in the child class. For example, searching of a string pattern, insertion of a new element into a table etc., are used in several applications and can thus be inherited from one single class.

3. Code sharing - code sharing can occur on several levels. on one level, many users & projects can use the same classes. Another form of sharing occurs when there are multiple classes developed by a single programmer as part of a project inherit from a single parent class.

4. Consistency of interface - when two or more classes inherit from from the same superclass, we are assured that the behavior they inherit will be the same in all cases.

5. Software components - Inheritance enables programmers to construct reusable software components. The goal is to permit the development of new & novel applications that require little or no actual coding.

6. Rapid prototyping - when a software system is constructed largely out of reusable components, developers can concentrate their time on understanding the new and unusual portion of the system. Thus, software systems can be generated more quickly & easily, leading to a style of programming known as rapid prototyping & exploratory programming.

7. Information hiding - when a software component is being reused by a programmer, he needs to understand only the interface and nature of that component. There is no need for him to know about the implementation of that component or the techniques used by that component

8. polymorphism - Softwares produced were written in bottom-up approach ie lower abstractions were placed at the bottom increasing gradually and the higher abstraction were placed at the top.

=

Member access rules :-

is same as "Access control" in unit - II

-

'Super' uses :-

'Super' is a keyword used by a subclass to refer to its immediate superclass. There are 2 uses of it. They are -

1. 'Super' keyword to call superclass constructor -

A subclass can call the Constructor of superclass by using 'Super' keyword. Its syntax is -

Super(parameter list);

where 'parameter list' are parameters needed by the superclass Constructor. The call to the superclass Constructor is made inside the subclass Constructor and so that the super() statement must always be the first statement inside a subclass Constructor.

program :-

```
class A
{
    private int width, height, depth;
    A (int w, int h, int d)
    {
        width = w;
        height = h;
        depth = d;
    }
    int volume()
    {
        return width * height * depth;
    }
}
class B extends A
{
    int weight;
    B (int w, int h, int d, int m)
    {
        super (w,h,d);          // calls superclass Constructor
        weight = m;
    }
}
class one
{
    public static void main (String args[])
    {
        B b = new B (2,3,4,5);
        System.out.println ("volume is = " + b.volume());
    }
}
```

Here, class B calls super() with w,h,d parameters. This causes the class A Constructor to be called where the values of width, height & depth are initialized using these values.

2. 'Super' keyword to access members :-

The 'super' keyword can be used to access the members of a superclass ie variables & methods of a superclass. Its syntax is

Super. member

This 'Super' usage is some what similar to 'this' except that it always refers to the superclass.

Program :-

```
class A
{
    int x;
    void show()
    {
        System.out.println("this is a superclass method");
    }
}
class B extends A
{
    int x;              // hides x in A
    B(int a, int b)
    {
        super.x = a;    // x in A
        x = b;          // x in B
    }
    void show()         // hides show() in A
    {
        super.show();
        System.out.println("this is a subclass method");
        "              ("x in superclass =" + super.x);
        "              ("x in subclass =" + x);
    }
}
class one
{
    public static void main(String args[])
    {
        B b = new B(10,20);
        b.show();
    }
}
```

op

```
this is a superclass meth
"      subclass "
x in superclass = 10
x in subclass = 20
```

program :- multilevel inheritance

```
class x
{
    void method1()
    {
        System.out.println("this is method1 of x");
    }
}
class y extends x
{
    void method1()
    {
        super.method1();
        System.out.println("this is method1 of y");
    }
}
class z extends y
{
    void method1()
    {
        super.method1();
        System.out.println("this is method1 of z");
    }
}
class mc
{
    public static void main(String args[])
    {
        z a = new z();
        a.method1();
    }
}
```

o/p

this is method1 of x
" " y
" " z

using 'final' with inheritance :-

1. using 'final' to prevent overriding -

A 'final' keyword is applied to a method to prevent method overriding. So, methods declared as final can't be overridden by subclasses. If a subclass tries to do it then it raises a compile time error.

```
eg:  class A
     {
       final void show()
       {
         =
       }
     }
     class B extends A
     {
       void show()          // compile time error occurs
       {
         =
       }
     }
```

'final' methods are used to increase the performance. As the compiler knows that 'final' methods can't be overridden, it calls them in-line. In other words when a small 'final' method is called, Java compile copies the byte code of that method with the compiled code of the calling method in-line.

2. using 'final' to prevent inheritance -

'final' keyword is also applied to a class to prevent other classes inheriting this class. The declaration of a class as final, declares all the methods of this ~~each~~ class as final. So, when a class tries to extend a final class then error occurs.

```
eg:  final class A
     {
       void display()
       { - }
     }
     class B extends A          // error
     {
       -
     }
```

3. 'final' with variables :-

A variable can be declared as final then we don't want the value to be be changed in the remaining part of the program. If we declare variables as 'final', then we can't modify its value. Values are initialized to final variables at the time of their declaration and they act as constant.

eg: final double PI = 3.14;
final int SNO = 5;

here the variables PI & SNO are declared as 'final', so their values cannot be changed. If the user tries to change their value, then compile time error occurs.

eg:
```
public class one
{
    public static void main (String args[])
    {
        int a = 100;
        final int b = 50;
        System.out.println ("value of a=" + a);
                 "          ("value of b=" + b);
        a = 150;
    // b = 200;        // error
        System.out.println ("a value after change =" + a);
    }
}
```

o/p

value of a = 100
" b = 50
a value after change = 150

=

polymorphism - method overriding :-

We know that a method defined in a super class is inherited by its subclass and it is used by the object created by the subclass. Method inheritance allows to use methods repeatedly in subclasses without define the methods again in sub class. In some situations, often we want an object to respond to the same method but having different behaviors when that method is called. That means, we have to override the definition of method. This is possible by defining a method in subclass that has same name, same arguments & same return type as a method in superclass. Then, when that method is called, the method present in subclass is executed, instead of method in superclass. This is known as method overriding.

program :-

```
class Super
{
  void display()
  {
     System.out.println("called display() of superclass");
  }
}
class Sub extends Super
{
  void display()          // overriding
  {
     System.out.println("called display() of subclass");
  }
}
class one
{
   public static void main(String args[])
   {
      Sub obj = new Sub();
          obj.display();        // calls display() in subclass
   }
}
```

o/p called display() of subclass

Differences :-

method overloading	method overriding
1. relationship is between methods of same class	1. relationship is between methods of superclass and subclass
2. different methods have the same name.	2. subclass method overrides the superclass method
3. does not block pole inheritance	3. blocks inheritance
4. different method signature	4. same method signature
5. Can have different return types	5. Can have matching return types

Abstract classes :-

Abstract method :-

An abstract method is a method that defines the features except its code. An abstract method declares the method name, parameters and return types, but does not have definition of the method. The abstract method is declared by 'abstract' keyword. An abstract method is a method that is declared, but no definition, but the subclasses have to give the definition of the method.

eg: abstract void display();

Abstract class :-

A class containing abstract methods is also deet called as abstract class and it must be declared with 'abstract' keyword.

If one of the methods in a class is abstract, then the class must be abstra

eg: abstract class one { }

A class is declared as abstract -

1. to prevent it from being instantiated by subclasses
2. when a class contains one or more abstract methods.
3. we cannot use abstract classes to instantiate objects directly

program :-

```
abstract class two
{
    Public abstract void display();      // a method without definition
}

Public class one extends two
{
  Public void display()
  {
     System.out.println("the definition of display() method in subclass");
  }
  Public static void main(String args[])
  {
     two obj = new two();      // error, we cannot create objects for
                               abstract class
     one a = new one();        // no error
         a.display();
  }
}
```

=

An abstract class may have,

1. only abstract methods.
2. no abstract methods
3. combination of abstract & non-abstract methods.

=

... and Interfaces

Packages:- A Package represents a directory that contains related group of classes and Interfaces. Packages are containers for classes that are used to keep the class name space compartmentalized.

Advantages of Packages:

Packages are useful for arranging the related classes & Interfaces into a group. This makes the classes & interface performing the same task to Put together in the same Package

Package hide the classes & interfaces in Separate subdirectory easy, so that accidental deletion of class & interface will not take place

A group of Packages is called library. The classes & interfaces in these libraries can be reused several times. This "reusability" nature of Package makes Programming easy.

There are two different types of Packages in Java. They are as follows

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

Built in Packages: These are already available in Java language known as Java API.

- applet
- awt
- io
- lang
- net
- util

.applet :- Applets are Programs which come from a Server into a client & get executed on the client machine on a network. Applet class of this Package is useful to create & use applets.

..awt :- awt stands for abstract window toolkit. This Package helps to develop GUI (Graphical user Interface) where Pgms with colorful screens, Painting & Images etc which is useful to provide action for create like components like. buttons, radio buttons, menus etc,

..io :- Io stands for I/P & O/P. This Package contains Streams. A Stream Represents flow of data from one place to another, and useful to store data in form of files. & to Perform I/O Related tasks.

.lang :- lang stands for language. This Package got Primary classes & Interfaces essential for developing a basic java Program. There are classes like String, StringBuffer, Mathematical functions, threads & exceptions

..net :- net stands for network. client-Server Programming can be done by using this Packages. ie., classes for communicating with local & remote system.

1. Util :- Util stands for utility. This Package contains useful classes & Interfaces like Stack, LinkedList, hash table, Vector, Array etc,

String a Package:-

To create a Package is easy. By using a "Package" command as the first statement in a java source file any classes declared within that file will belong to the specified package. The Package statement defines a namespace in which classes are stored.

The general form of the Package statement:

Package Package-name;

Here Package-name is name of the Package.

Eg: Package MyPackage;

Java uses file System directories to store Packages. Store .class files for any classes you declare to be Part of "Package" must be stored in a directory called MyPackage.

More than one file can include the same Package statement. We can create a hierarchy of Packages. To do so, Simply separate each Package name from the one and other. ie.,

Syntax: ~~Package Pkg1[.Pkg2[.Pkg3]];~~
Package Pkg1[.Pkg2[.Pkg3]];

Eg: Package java.awt.Image;
Package java.util.date;

It needs to be stored in java/awt/image, a/awt/image.

Creating and accessing Packages

1: Create a Package with the Next Simple code

Eg: Package P1;
class A
{
int i;
void set(int a);
{
i=a;
}
void display()
{
System.out.println("the value is :" +i);
}
}

```
class Packagedemo
{
Public Static void main(String args[])
 {
  A obj = new A();
  obj.set(10);
  obj.display();
 }
}
```

save this code by 'Packagedemo.java'. This file must be saved within the directory 'P1'

2:
compile the above Program.
> Make sure resulting .class file must be in the 'P1' ector. for this do as follows

E:\ document & settings\ KSRMCE\ CD P1 ↵

c:\ document & settings\ KSRMCE\ P1> javac Packagedemo.java ↵

3: Now execute the Program using the command to get an o/p. using the Packagename & dot Operator we can execute the Program..

c:\ document & settings\ KSRMCE> java P1.Packagedemo ↵

The value of i: 10 will be displayed as output.

his helps to find the CLASSPATH for the executable ile. If there is hierarchies of Packages then it can be entioned by names of Packages separated by dot.

y: P1.Packagedemo.another-Package

Package name (3) First Parent Package Program name

rting Packages:- Importing packages

All the standard classes in Java are stored in named ages. There is no standard class present in Java which is named. But it is always complicated to write the class s using a long sequence of packages containing dot operator.

Java includes the import statement to bring certain s, or entire Packages, into visibility. once imported, a can be referred to directly, using only its name. The t statement is a convenience to Programmer & is not ically needed to write a complete java program.

In java Source file, import statements occur immediately ing the 'package' statement and before any class definitions:

import Pkg1[.Pkg2](Classname/*);

Eg: import java.util.Date;

ere Pkg1 is the name of toplevel Package, & Pkg2 is the e of Subordinate Package inside the outer Package separated a dot (.). There is no Practical limit on the depth of a rage hierarchy. finally, we can specify either an explicit lassname or a star (*) which indicates that the java mpiler should import the entire Package.

```
import java.util.*;
class mydob extends Date
{
}
```

Interfaces:-

By using Interfaces, we can specify what a class must do, but not how it does it. i.e, we can form abstract a class. Interfaces are syntactically similar to classes, but lack of instance variables & methods are declared with out any body. Once it is defined, any number of classes can implement an interface. Also, one class can implement any number of interfaces.

To implement an Interface, a class must create the complete set of methods defined by the Interface. By providing the 'Interface' keyword, Java allows you to fully utilize the "one interface, multiple methods" as Polymorphism.

Interfaces are designed to support dynamic method resolution at runtime.

Using an Interface:-

Syntax:

```
access Interface name
{
  return_type method_name1(Parameter_list);
  return_type method_name2(Parameter_list);
  type final_variable1 = value;
  type final_variable2 = value;
  type final_variableN = value;
  return_type method_nameN(Parameter_list);
}
```

'access' is either public or not used. When no access specifier is included, then default access results and the interface is available to only other members of the package in which it is declared.

olic" the interface can be used by any other code.
me specifies name of the interface, and can be
ny valid identifier.
ethods are declared with no bodies.
ach class that includes all interfaces must implement all
f the methods.
finalvariables are declared as final & static
meaning they cannot be changed by the implementing
class. They must also be initialized with constant.

Eg: interface callback
{
Void callback(int Param);
}

Implementing Interfaces:-

Once an interface has been defined, one or more classes
implement that interfaces. To implement an interface,
ude the 'implements' clause in class definition.

tax:- access class classname [extends Superclass]
[implements 'interface1[,Interface2...]]
{
// class_body
}

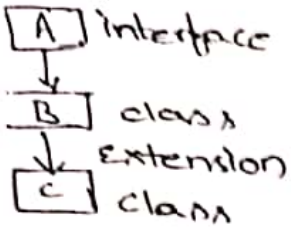
ccess specifies either public or not used. If a class
plements more than one interface, interfaces are separated
th a comma. The methods that implement an interface
ust be declared public, Signature of the implementing
ethod must match exactly the type signature specified
n the interface definition.

```
interface area
 {
  static final float Pi = 3.14;
  float value(float a, float b);
 }
class rect implements area
 {
  Public float value(float a, float b)
   {
    return( a*b);
   }
 }
class Circle implements area
 {
  Public float value(float a, float b)
   {return  Pi*x*x;
   }
 }
class test
 {
  Public static void main(String args[])
   {
    rect Obj1 = new rect();
   Circle obj2 = new Circle();
    sop("rea" + obj1.value(20,30);
    sop("rea" + obja.value(10,0);
   }
 }
```

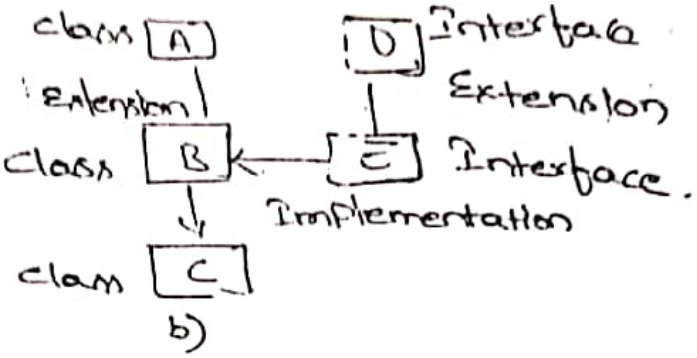
Any number of dissimilar classes can implement an interface, however, to implement the methods, we need to refer to class objects as types of the interface rather than types of respective classes.

e that if a class that implements an interface does not ...face implement all the methods of the interface, then class becomes an abstract class & cannot be instantiated

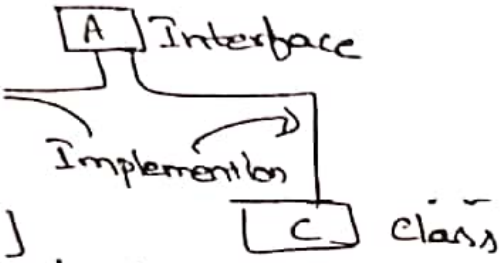
nplementation of interfaces as class type is as follows



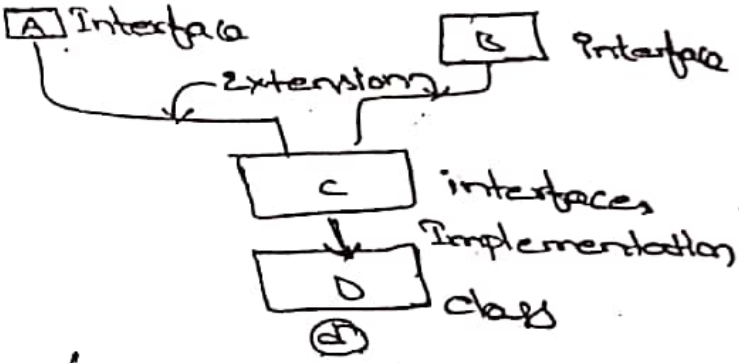
a)



b)



c)



d)

reating Interface Variables (Variables in interfaces)

Interface can be used to declare a set of constants can be used in different classes. This is similar to creating ...er files in C++ to contain a large number of constants

Such Interface does not contain methods, there is no need worry about implementing any methods. The constant values be available to any class that implements the interface. The ...es can be used in any method, as part of any variable ...ration, or anywhere where we can use a final value.

(Contd..,)

```
interface A
{
  int m=10;
  int n=50;
}
class B implements A
{
  int x = m;
  void methodB(int size)
  {
    ...
    if(size<n)
    .....
  }
}
```

Implementing multiple Inheritance

```
class Student
{
  int rollnumber;
  void getnumber(int n)
  {
    rollnumber = n;
  }
  void putnumber()
  {
    System.out.println("Roll No:" +rollnumber);
  }
}
class Test extends Student
{
  float p1, p2;
  void getmarks(float m1, float m2)
  {
    p1 = m1;
    p2 = m2;
  }
```

```
id PutMarks()

System.out.Println("Marks obtained");
System.out.Println("Part1 = " +P1);
System.out.Println("Part2 = " +P2);
}
terface Sports
{
 float SportWt=6.0F;
 void PutWt();
 }
ass Results extends Test implements Sports

float total;
Public Void PutWt()
  {
   System.out.Println(" Sports Wt = " + SportWt);
  }
 Void display()
   {
    total = P1+P2+ SportWt;
    PutNumber();
    PutMarks();
    PutWt();
    System.out.Println("total Score:" + total);
   }
}
lass Hybrid
{
Public Static Void main(String args[])
 {
   Results Students = new Results();
   Students.getNumber(1234);
   Students.getMarks(22.5F, 33.0F);
   Students.display();
  }
```

o/p:

Roll NO : 1234

Marks Obtained

P1 = 22.5

P2 = 33.0

Sports wt = 6

Total Score = 61.5

Extending interfaces

Like classes, interfaces can also be extended. i.e, an interface can be subinterfaced from other interfaces. The new interface will inherit all the members of superinterface in manner similar to subclasses.

Syntax:-

```
interface name2 extends name1
{
// body of name2.
}
```

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

// includes meth1() & meth2()
interface B extend A
{
void meth3();
}

class must implement all of A & B.
class MyClass implements B
{
public void meth1()
{
System.out.println("Implement meth1()");
}
public void meth2()
{
System.out.println("Implement meth2()");
}
public void meth3()
{
System.out.println("Implement meth3()");
}
```

(Contd---1)

```
class Ifextend
{
public static void main(String
                        args[])
{
Myclass O1 = new Myclass();
O1.meth1(); O1.meth2();
O1.meth3();
}
}
```

while interfaces are allowed to extend to other interfaces, terfaces cannot define the methods declared in the interfaces. After all, subinterfaces are still 'interfaces' not ses. Instead, it is the responsibility of any class that ements the derived interface to define all the methods. that when an interface extends two or more interfaces are separated by commas. It is important to ember that an interface cannot extend classes this would late the rule that an Interface can have only abstract thod & constants

erence between classes and Interfaces. - Differences

classes	Interfaces
It is denoted by a keyword 'class'	1) The Interface is denoted by a keyword 'Interface'.
class Contains variables & thods and also implementation the methods.	2) Interface contains variables & Methods, but implementation is not Present ie., the definition of method is not given in the Interface.
th the help of instance of ss, the variables can be essed.	3) We can not access the instance of Interface.
e class has access specifiers ike Public, Protected, Private.	4) Interface has only one access Specifier ie, Public.