JEPPIAAR INSTITUTE OF TECHNOLOGY

"Self-Belief | Self Discipline | Self Respect"

DEPARTMENT

OF

COMPUTER SCIENCE AND ENGINEERING

LECTURE NOTES

CS8392 – Object Oriented Programming

(Regulation 2017)

Year/Semester: II/03 CSE

2021 – 2022

Prepared by

Ms. R. Revathi

Assistant Professor/CSE

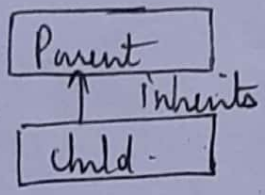
UNIT - II

Inheritance and Interfaces

Inheritance:-

Definition — It can be defined as the mechanism of deriving a Sub class from a Super class. It acquiring all the properties and behaviours of one class to another.

— It represents the Is-A relationship.



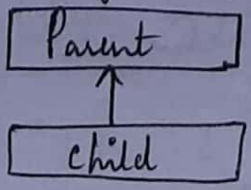
Types of Inheritance:-

The types of inheritance are

- ✓ Single Inheritance
- ✓ Multiple Inheritance
- ✓ Multilevel Inheritance.
- ✓ Hierarchical Inheritance
- ✓ Hybrid Inheritance.

Single Inheritance:-

Definition:- It is a Process of deriving a Sub class from a Single Super class.



When a child class inherits the Parent class the Child class will have it's own behaviour and also the behaviour of the Parent.

Syntax:-

```
Class Superclassname
{

}

Class Subclassname extends Superclassname
{

}
```


eg:-

```
Class A
{

}
Class B extends A
{

}
```

Progeam:-

```
Public class A
{
    Public void dispA()
    {
        S.o.P("disp() method of class A");
    }
}
```

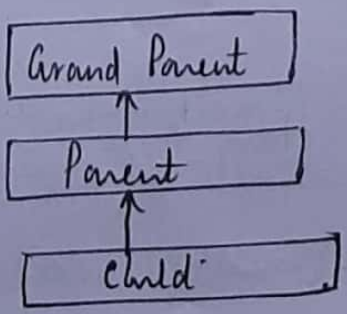
```
Class B extends A
{
    Public void dispB()
    {
        S.o.p("disp() method of class B");
    }
}
Class SingleInheritance
{
    Public static void main(String args[])
    {
        B b = new B();
        b.dispA();
        b.dispB();
    }
}
```


Output:-

disp() method of class A
disp() method of class B.

Multi-level Inheritance:-

Defintion:- It is a Process of deriving a Sub class from another derived class.



Syntax:-

```
class Superclassname
{

}
class Subclassname1 extends Superclassname
{

}
class Subclassname2 extends Subclassname1
{

}
```


eg:-

```
class A
{

}
class B extends A
{

}
class C extends B
{

}
```

Program:-

```
Public class A
{
    Public Void dispA()
    {
        S.o.P("disp() method of class A");
} }
```

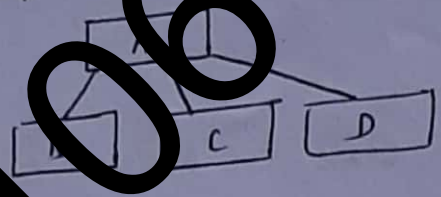
```
Public class B extends A
{
    Public void dispB()
    {
        S.O.P("disp() method of class B");
    }
}
Public class C extends B
 {
    Public void dispC()
    {
      S.O.P("disp() method of class C");
    }
 }
 Class Multilevel
 {
    Public static void main(String args[])
    {
       C ob = new C();
       ob. dispA();
       ob. dispB();
       ob. dispC();
    }
 }
```

Output:-

disp() method of class A
disp() method of class B
disp() method of class C.

Hierarchical Inheritance:-

Definition: It is a process of deriving a more than one sub class from a single base (or) super class.



Syntax:-

```
class superclass name
{
}
class subclass name1 extends superclass name
{
}
class subclass name2 extends superclass name
{
}
```

eg:-

```
class A
{
}
class B extends A
{
}
class C extends A
{
}
```

Progeam:-

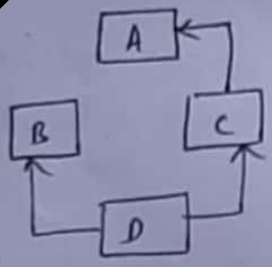
```
Public class A
{
    Public void dispA()
    {
      S.o.p("disp() method of class A");
    }
}
Public class B extends A
{
    Public void dispB()
    {
      S.o.p("disp() method of class B");
    }
}
Public class C extends A
{
    Public void disp()
    {
      S.o.p("disp() method of class C");
    }
} class Hierarchical {
Public static Void main(String args[])
{
    B b = new B();
    b.dispB();
    b.dispA();
    C c = new C();
    c.dispC();
    c.dispA();
} }
```

Output:-

disp() method of class B
disp() method of class A
disp() method of class C
disp() method of class A.

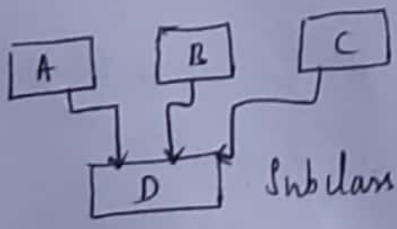
Hybrid Inheritance:-

Definition:- It is the Combination of both single and Multiple inheritance.



Multiple Inheritance:-

Defintion: It is a Proces of deriving a Sub class from more than One Super class. But in Java, multiple Inheritance is not supported directly. It is indirectly supported by a Concept called "Interfaces".



Super Keyword:-

Usuage of Super keyword are

① Super() invokes the Constructor of the base (or) Super class.

② . Super . Variable_name refers to the Variable in Super class.

③ . Super . method name refers to the method of the Super class.

① . Super() invokes the constructor of Super class :-

Progeam :-

```
class A
{
   A()
   {
      S.o.p("Super class default Constructor");
   }
}
class B extends A
{
   B()
   { Super(); // it calls super(A) class Constructor
   S.o.p("Sub class default Constructor");
   }
}
class Demo_Super
{
 Public static Void main (String args[])
 {
    B b = new B();
 }
}
```

Output :-
Super class default Constructor.
Sub class " "

② Super. Variable name refers to Variable in Super class :-

Program :-

```
class A
{
  int val = 99;
}
class B extends A
{ int Val = 123;
  Void disp()
  {
    S.o.p("Value is" + val);
  }
}
class DemoSuper2
{
 Public static Void main(String args[])
 {
   B ob = new B();
   ob.disp()
 }
}
```

Output :-

Value is 123.

- This will call only the Val of the Sub class Only. Without Super Keyword.

(Val) which is present in --
you cannot call the Value of the Super class.
For that, a change in the above Progeam.
as, { Void disp()
System.out.println("Value is" + Super.val);

③ Super.method name refer to the method of the Superclass:

```
Class A
{
  Void disp()
  {
    S.o.p("Super class method");
  }
}
Class B extends A
{
  Void disp()
  {
    S.o.p("Sub class method");
  }
  Void show()
  {
    disp();
    Super.disp();
  }
}
Class DemoSuper3
{ Public static Void main(String args[])
  { B b = new B();
    b.show();
  }
}
```

O/P:
Sub class method.
Super class method.

Abstract class:-

Definition: If a class contains atleast One abstract method, then the class is said to be an abstract class.

- It can have abstract and Concrete (non-abstract) methods.
- It cannot be instantiated, (ie) we can't use it to create objects.

Abstract method:-

If a method has no implementation (or) no body of the method, then it is called abstract method.

Syntax of abstract method:-

Returntype Methodname (arguments);

eg:-

Void Print ();

Example Program to Illustrate Abstract method:

```
class Shape
{
    Void draw();
}
class Rectangle extends Shape
{
    Void draw()
    {
      S.o.p("Drawing rectangle");
```

```
Class Circle extends Shape
{
    Void draw()
    {
        S.o.p("drawing circle");
    }
}
Class testabstract
{
    Public Static Void main(String args[])
    {
        Shape s = new Circle();
        s.draw();
    }
}
```


Output:-

drawing circle.

Interfaces in Java:-

Defintion:-

- An Interface is a Collection of only abstract static Constants (Variables) and methods. An Interface is similar to a class

- The interface in Java is a mechanism to achieve abstraction and Multiple Inheritance.

- Interface is declared using 'interface' keyword.

Syntax of Interface:-

```
interface interfacename
{
    // declare constant fields
    // declare methods (abstract)
}
```

eg:-

```
interface A        // Interface name
{
    void Print();
    void Show();
}
```

by default, abstract methods

Program:- to illustrate Interfaces:-

```
interface drawable
{
    void draw();
}
class Rectangle implements drawable
{
    public void draw()
    {
        S.o.p("Drawing Rectangle");
    }
}
class Circle implements drawable
{
    public void draw()
    {
```

```
        S.o.p(" Drawing Circle");
        }
    }
Class test_interface
{
    Public static Void main(String args[])
    {
        drawable d = new Circle();
        d.draw();
    }
}
```

Output:-

Drawing Circle.

Note:1: If a class need to inherit interfaces, then, we have to use 'implements' Keyword

Note:-2: A class can inherit more than One interfaces through 'implements' Keyword..

Multiple Inheritance in Java using Interfaces:-

In Java, Multiple inheritance is not supported directly. It is indirectly supported by a Concept called 'Interfaces'

Why Multiple Inheritance is not supported:

Justification:-

Consider, there is a Super class called 'A', which has One default

Constructor, and we have another Super class called 'B', which also have one default Constructor. Create a Sub class 'C' which also have one default Constructor. Trying to Create an object of Sub class 'C', will call Sub class Constructor, and in turn, the Sub class invokes 'Super' Keyword, then it will call Super class Constructors. But the Problem here is, it does not know to call which (Class Super) Constructor (i.e) A or B. It leads to Ambiguity. Therefore, A class Can inherit Only one Super class using 'extends' Keyword. This Problem Can be solved by using 'Interfaces'

Progeum to illustrate Multiple Inheritance using Interfaces:-

```
Interface A
{
    Void area();
}

class B
{
    void Print()
    {
        S.O.P("class B");
    }
}

class Rectangle extends B implements A
{
    Public Void area()
```

```
{
  S.O.P ("interface A");
}
}
Class Multiple_inheritance
{
Public static void main(String args[])
  {
     Rectangle r = new Rectangle();
     r.area();
     r.print();
  }
}
```


Output:-

interface A
Class B.

Rules for Interfaces:-

- ✓ Interface Cannot be declared as Private or Protected.
- ✓ All the interface methods are by default abstract and Public.
- ✓ Variables declared in interface are by default Public, Static and final
- ✓ A class can implement any no. of interfaces using.

eg: class A implements (B), (C) — Super classes

{
}

VB
bm (88)

Difference between Abstract class Vs Interface

Abstract class	Interface
01) Abstract class can have abstract and non-abstract methods	01) It can have only abstract methods. (In Java 8, can have static methods also)
02) Abstract class does not support multiple Inheritance	02) Supports Multiple Inheritance.
03) Abstract class can have final, non-final, static + non-static Variables.	02) It can have only static + final Variables
04) An abstract class can extend another Java class and implement multiple Interfaces.	04) Interface can extend another Java interface only.
05) It can be extended using the keyword "extends"	05) Can be implemented using keyword "implements"
06) It can have members like Private, Protected Public.	06) Members of Interface are Public by default

Final Keyword:

- Final Keyword can be used along with
  - ✓ Final Variable
  - ✓ Final Method
  - ✓ Final class

} three uses of final

① Final Variable:-

Def: A final Variable is a Variable whose Value Cannot be changed at anytime once assigned, it remains as a Constant forever.

Eg: (Progeam):-

```
Public class Travel
{
    final int speed = 60;
    Void increase speed()
    {
        speed = 70;
    }
}
class test
{
    Public static Void main (string args[])
    {
        Travel t = new Travel();
        t.increasespeed();
    }
}
```

Output: Exception in thread "main" java.lang.Error

- The above code will give you Compile time error, as we are trying to change the

Value of a final Variable 'Speed'.

② Final Methods :-

Def :- When you declare a method as final, then it is called as final method. A final method cannot be Overridden. (Prevents Method Overriding)

Progeam:

```
Class Parent
{
  Public final Void disp()
  {
    S.o.p("disp() of Parent class");
  }
}
Class child extends Parent
{
  Public Void disp()
  {
    S.o.p("disp() of child class");
  }
}
class final_method
{
  Public static Void main(String args[])
  {
    child c = new child();
```

Putting final in front of super class method, it prevents Overriding

```
c.disp();
}
}
```

③ Final class:-

Def: A final class cannot be extended.

It Prevents Inheritance.

Progeam:

```
final class Parent // final class
{
}
class child extends Parent
{
}
```

Prevents inheritance

Im ⊗⊗ vo

Object cloning:-

Defintion:- It is a way to Create exact Copy of an object. The clone() method of object class is used to clone the object

- The Java.lang. Cloneable interface must be implemented by the class whose object clone, we want to Create. If we don't implement cloneable interface, clone() methods generate `clone Not Supported Exception

Syntax of clone() method:-

Protected Object clone() throws CloneNotSupportedException

- The clone() method saves the extra Processing task for creating the exact copy of an object. If we perform it by using new keyword, it will take lot of Processing time to be Performed.

(ie) Why we use Object cloning.

Advantage of object-cloning:-

✓ No-need to write lengthy and repetitive codes.

✓ fastest way to copy array using clone()

Disadvantage:-

✓ To use Object.clone() method, we have to change lot of syntaxes to our code, like implementing a cloneable interface, defining clone() method and handling CloneNotSupportedException.

✓ doesn't invoke any Constructor. ∴ there is no control over object Constructio

Progeam:-

```
class Student implements Cloneable
{
    int rollno;
    String name;
  Student (int rollno, String name)
   {
       this.rollno = rollno;
       this.name = name;
   }
  Public Object clone() throws cloneNot
                              SupportedExeption
  {
   return super.clone();
   }
}

  class object_cloning
  {
    Public static Void main(String args[])
    { try {
     Student s1 = new Student (101, "mohan");
     Student s2 = (Student) s1.clone();
     S.o.p(s1.rollno + " " + s1.name);
     S.o.p(s2.rollno + " " + s2.name);
     } Catch(cloneNotsupportedExeption e)
          { }
  }
}
```

Output:- 101 mohan
101 mohan

Inner - classes:-

Definition:- Inner class means one class which is a member of another class.

4 types of Inner classes:-

1. Nested Inner class
2. Local Inner class
3. Anonymous Inner class
4. Static Inner class.

1. Nested Inner class:-

Def: It can access any Private Variable of Outer class.

Eg:

```
class Outer
{
  class Inner
  {
    Public void show()
    {
      s.o.p("In a nested class method");
    }
  }
}
class main
{
  Public static void main(String a[])
  {
    Outer.Inner in = new Outer().new Inner();
    in.show();
  }
}
```

Output:

In a nested class method.

② Local Inner class:-

Def: Inner class can be declared within a method of an Outer class.

Eg:

```
class Outer
{
  void Outer_method()
  {
    S.o.p("Inside Outer method");
    class Inner
    {
      void inner method()
      {
        S.o.p("inside inner method");
      }
    }
    Inner y = new Inner();
    y.inner Method();
  }
}
class method demo
{
  Public static void main(String args[])
  {
    Outer x = new Outer();
    x.outer_method();
  }
}
```

o/p: inside outer metho
inside inner metho

③ Static nested class:-

Static nested classes are not technically an inner class. They are like a static member of outer class.

Program:-

```
Class Outer
{
  private static void OuterMethod()
  {
    S.o.p("Inside Outer method");
  }
  Static class Inner
  {
    Public static void main(String args[])
    {
      S.o.p("Inside inner class method");
      OuterMethod();
    }
  }
}
```

Output:

- Inside inner class method
- Inside Outer Method

Strings in Java:-

Defintion:- In Java, string is basically an object that represents sequence of characters.

- In Java, string objects are immutable. Immutable means unmodifiable (or) Unchangeable.

Two Ways of creating object:

- ✓ By using String literal
- ✓ By using 'new' Keyword.

1. By using String literal :-

Java String literal is created by using double quotes.

Eg:- String s = "welcome";

2. By using 'new' Keyword :-

Eg: String s = new String ("welcome");

String methods :-

① int length() :-

Use:- It is used to find the length of the string.

② char charAt (int index) :-

Use: It is used to return char value for the Particular index.

③ String concat (String str):

Use: Concatenates the specified string.

④ boolean equals (Object):-

Use:- Checks the equality of string with the given object.

⑤ String replace (char old, char new)

Use:- Replaces all occurrences of the specified character value.

⑥ String toLowerCase():-

Use:- It returns a string in lowercase.

⑦ String toUpperCase():-

Use:- It returns a string in uppercase.

⑧ int indexOf (int ch):-

Use:- returns the specified char value index.

Example:- Progeam for strings:-

```
Public class strings
{
  Public static void main (String a[])
  {
    String s1 = new String ("hello");
    String s2 = new String ("hello");
    if (String1 == String2)
    {
      S.o.p ("String1 = " + String1 +
      "String2 = " + String2 + "are equal");
```

String1 = hello
String2 = hello

```
}
else
{
  S.o.p("Strings are not equal");
}       String b = "technology";
S.o.p("upper case of string is" + string1.toUpperCase());
S.o.p("Concatenation of object a and b is"
                    + string1.concat(b));
S.o.p("length of object is" + b.length());
S.o.p("The third character of object is" + b.charAt(2));
}
}
```

Java Arraylist:-

Defintion:- It uses a dynamic array for storing the elements.

Advantages (or) use of Arraylist (or) Rules:-

- ✓ It cannot contain duplicate elements
- ✓ It is non-synchronized
- ✓ It allows random access
- ✓ Manipulation is slow
- ✓ maintains insertion order

Eg Progeam:-

```
import java.util.*;
class Book {
    int id;
```

```
String name, author, Publisher;
int quantity;
Public Book ( int id, String name, String
                author, String Publisher)
{
    this.id = id;
    this.name = name;
    this.author = author;
    this.Publisher = Publisher;
    this.quantity = quantity;
}
}
Public class ArrayListExample
{
    Public static void main (String a[])
    {       // Creating list of books
    List <Book> list = new ArrayList
                                <Book>();
        // Creating Books.
    Book b1 = new Book (101, "letusc",
                "Yeshwant", "BPB", 8);
Book b2 = new Book (102, "OOPS", "Balu
                gurusamy", "TP", 9);
    // Adding books to list
    list.add(b1);
    list.add(b2);
```

```
// Traversing the list:
for (Book b : list)
        S.o.p (b.id + " " + b.name + " " +
        b. author + " " + b. Publisher + " " +
            b. quantity);
    }
}
}
```

Output:-

```
101  Let us C   Yeshwant  BPB  8
102  OOPS       Balagurusamy  9
```