Unit-II

Inheritance, Packages and interfaces.

Overloading methods - objects as parameters - returning object - static, Nested and Inner classes. Inheritance Basics - Types of inheritance - super keyword - method overriding - Dynamic method dispatch - Abstract classes - final with inheritance. Packages and interfaces: Packages - packages and member access - Importing packages - Interfaces.

Overloading method:

Overloading is a mechanism in which we can use many methods having the same function name but can pass different number of parameters or different types of parameter.

ex:

```
int sum(int a, int b);
double sum(double a, double b);
int sum(int a, int b, int c);
```

object as parameters :-

Call by value

Call by reference.

Call by value:

The value of the actual argument is assigned to the formal parameter.

```
public class ObjDemo
{
int height;
  int width;
  ObjDemo (int h, int w)
  {
  height = h; width = w;
  }
 void area(ObjDemo o)
 {
 int result = (height + o.height)*(width+
                                   o.width);
 System.out.println("The area is "+result);
 }
}
class Demo
{
 public static void main (String args[])
```

```
}
 objDemo obj1 = new objDemo (2,3);
 objDemo obj2 = new objDemo (10,20);
 obj1.accea (obj2);
}
}
```

O/P

The accea is 276

class

call by reference:

For passing the parameter by reference we pass the object of the class as a parameter.

ex:

```
public class objRetDemo{
int a;
objRetDemo (int val)
{
 a = val;
}
objRetDemo fn()
{
  objRetDemo temp = new objRetDemo (a+5);
  return temp;
}
}
```

```
class objRet {
 public static void main (String args[])
 {
  objRetDemo obj2 = new objRetDemo(20);
  objRetDemo obj1;
  obj1 = obj2.fn();
  System.out.println("The returned value is
                      ="+obj1.a);
 }
}
```

output:

The returned value is =25.

Static, Nested and Inner classes.

We can declare a class static by using the static keyword. A class can be declared static only if it is a nested class.

Inner class:

The classes that are non-static and nested are called inner classes.

Outer class:

The class in which nested class is defined is called outer class.

Nested class:

A class within a class is known as nested class.

```
public class Class1
{
 static class Class2
 {
  public void fn()
  {
   System.out.println("This a static nested
                       class");
  }
 }
public static void main (String args[])
{
Class1.Class2 obj = new Class1.Class2();
obj.fn();
}
}
```

o/p

This is a static nested class.

Inner classes:

Syntax:

```
{
 ....
Access_modifier class InnerClass
 {
   ....
  }
}
```

Types:

- → Static member classes
- → Member classes
- → Local classes
- → Anonymous classes.

ex:

```
class MyInnerClass implements Runnable
 {
 public void run()
 {
 System.out.println("Hello");
 }
class DemoClass
{
 public static void main (String [] args)
 {
 MyInnerClass my= new MyInnerClass();
Thread th = new Thread (my);
 my.start();
 }
}
```

Inheritance:

Inheritance is a mechanism in Java by which derived class can borrow the properties of base class.

Advantages:

* Reusability
* Extensibility
* Data hiding
* Overriding.

Types:

Single inheritance:

There is one parent per derived class.

Subclass can be defined as:

```
class name of Subclass extends Superclass
{
  Variable declaration
  method declaration
}
```

Multiple inheritance:

The derived class is derived from more than one base class.

Multilevel inheritance:

When a derived class is derived from a base class which itself is a derived class then it is multilevel inheritance.

Hybrid inheritance:

When two or more types of inheritance is combined together then it forms the hybrid inheritance.

Super Keyword:

It is used to access the immediate parent class from subclass.

There ways to use super keyword

* Variable invocation
* Method invocation
* Constructor invocation

ex:

```
class A
{
  int x=10;
}
class B extends A
{
  int x=20;
  void display()
```

```
    }
}
public void main (String args[])
{
    B obj = new B();
    obj.display();
  }
}
```

Method overriding:

Method overriding is a mechanism in which a subclass inherits the methods of superclass and sometimes the subclass modifies the implementation of a method defined in superclass.

Dynamic method dispatch:

It is run time polymorphism in which a call to overriden function is resolved at runtime instead of at compile time.

Syntax:

Super class ref variable = new subclass_obj();

ex: for method Overriding & dynamic method dispatch:

```
class x
{
    void show()
    {
        system.out.println("x class");
    }
}
class Y extends x
    void show()
    {
        System.out.println("Y class")
    }
}
class override
{
    Public static void main (String[] args)
    {
        Y obj = new Y();
        obj.show();
    }
```

O/p Y class

→ Now,

```
x ref = new y ()
ref. show ();
```

→ dynamic method dispatch

ract classes:

*Abstract class specifies only the member function.

Properties:

*Abstract method must be present in abstract class only.

*Abstract class cannot be instantiated using new operator.

*Abstract class must be parent class.

*Abstract method has no definition part.

Example:

```
abstract class A
{
   abstract void fun1(); ← This is an abstract method
   void fun2()
   {
      System.out.println("A: fun2");
   }
}
class B extends A
{
   void fun1()
   {
      System.out.println("B: fun1");
   }
}
class C extends A
{
   void fun1()
   {
      System.out.println("C: fun1");
   }
}
```

```
public class Demo
{
    public static void main(String
    {
        B b = new B();
        C c = new C();
        b.fun1(); // calls method of class B
        b.fun2();
        c.fun1(); // calls method of class C
        c.fun2();
    }
}
```

Output:

```
B: fun1
A: fun2
C: fun1
A: fun2
```

Final with Inheritance

Final keyword can be applied at 3 places.

* For declaring variables.
* For declaring the methods
* For declaring the classes.

Final variables and methods:

The final keyword can also be applied to method. On applying, method overriding is avoided. The method declared with final keyword cannot be

```
final void fun()
{
    System.out.println("Hello!");
}
}
```

Output :

1 error

cannot inherit from final test.

Packages :

Package is a mechanism in which variety of classes and interface can be group together.

syntax :

Package name_of_package.

Eg :

```
Package My_package
Public class A
{
    int a;
    public void set_val(int n)
    {
        a=n;
    }
    Public void display()
    {
        System.out.println(+a);
    }
}
```

overriden.

```
class Test
{
    final void fun()
    {
        System.out.println("Hello");
    }
}
class Test1 extends Test
{
    final void fun()
    {
        System.out.println("Hello1");
    }
}
```

Output:

1 error

fun() in Test1 cannot override fun() in Test.

Final classes to stop Inheritance:

If we declare class as final, no class can be derived from it.

```
final class Test
{
    void fun()
    {
        System.out.println("Hello");
    }
}
class Test1 extends Test
{
```

```
import My package A
class package demo
{
    Public static void main(string[] args)
    {
        A obj = new A();
        obj.set_val(10);
        obj.display();
    }
}
```

INTERFACES :

Java does not support more than one superclass. Java does not implement multiple inheritance diredly, but make use of this concept using interfaces.

Defining an interface :

Interface is a kind of class. like classes, interface contains methods and variables, but the method are not defined.

Interface define only abstract method and final fields.

Syntax :

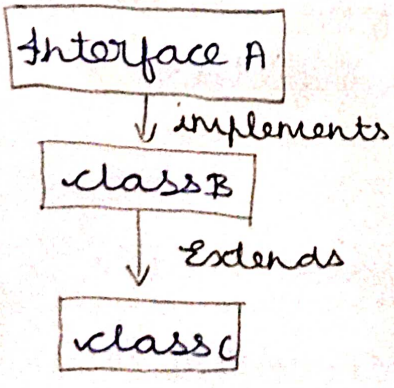
```
interface interface_name
{
    variable decleration;
    methods decleration;
}
```

Variables are declared as
static final type variable name
Eg: static final int X = 10;

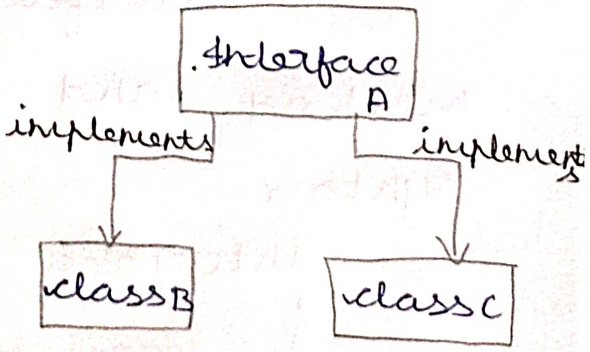
Difference between class and interface:

Class	Interface
*Class is denoted by keyword class.	*Interface is denoted by keyword interface.
*Instance of class can be created.	*Cannot create instance of class.
*Public, Private & Protected access specifiers can be used.	*Only public access specifiers is used.
*Class contains data member & methods. But methods are defined.	*Interface contain data members and methods. But methods are not defined.
*Data members can be constant or final	*Data members are always declared as final.

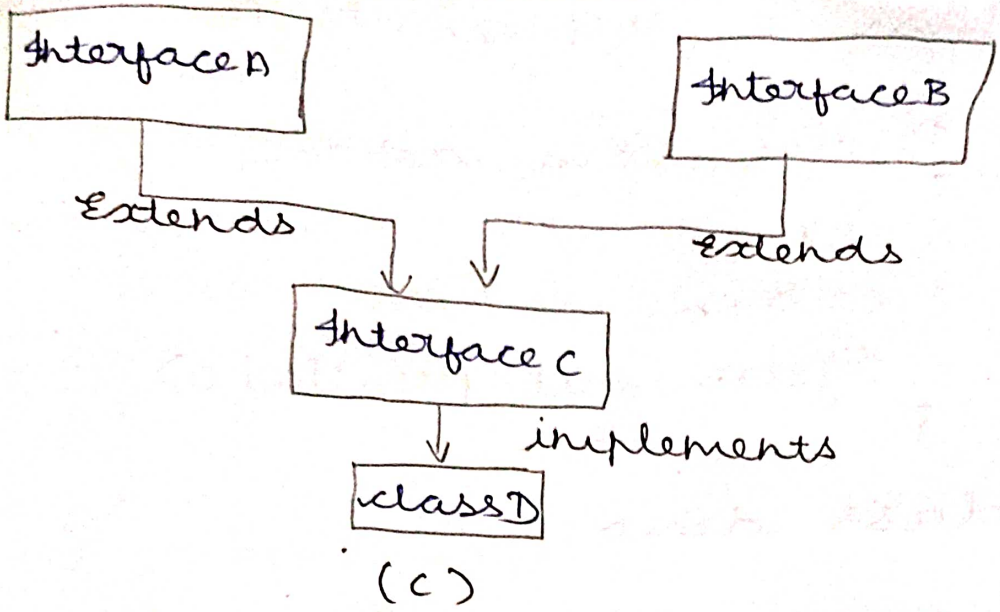
Implementing Interface:
Various way of interface implementation:



(a)



(b)



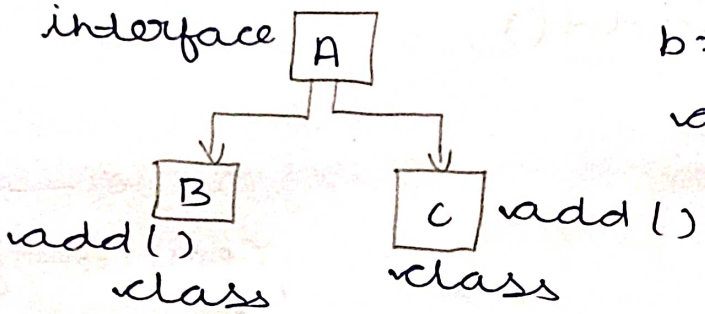
(c)

class - class → extends

interface - interface → extends

class - interface → implements

interface - class → implements.



Eg:

```
interface A
{
    void add();
    int a=5, b=2;
}
class B implements A
{
   Public void add()
   {
      int c;
      c=a+b;
      System.out.println(c);
   }
}
```

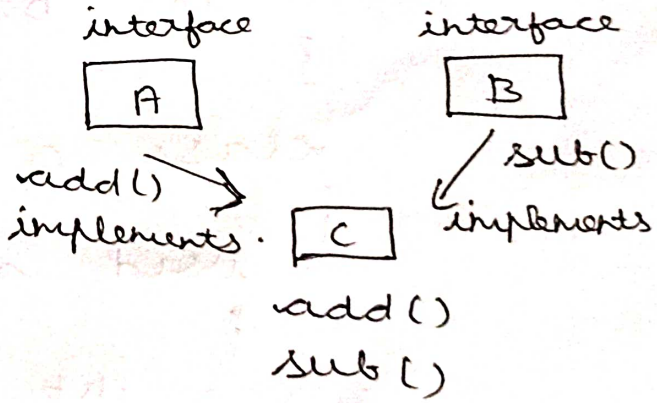
```
class C implements A
{
   Public void add()
   {
     int c;
     c=a+b;
     System.out.println(c);
   }
}
class demo
{
   Public static void main(String args[])
   {
      B b=new B();
      C c =new C();
      b.add(); ->7
      c.add(); ->7
   }
}
```

Save → demo.java
Compile → javac demo.java
Run → java demo

Multiple inheritance (using interface)

```
interface A
{
   void add();
   int a=5 , b=2;
}
interface B
{
   void sub();
   int a=5,b=2;
}
class C implements A,B
{
   Public void add()
   {
```

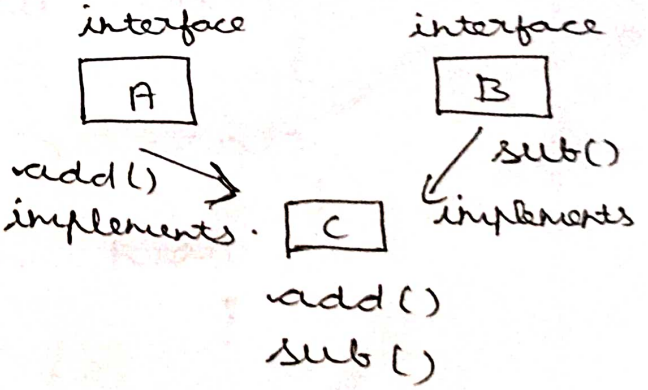
```
        int c ;
        c = a+b ;
        System.out.println(c);
    }
    Public void sub()
    {
        int c ;
        c = a-b ;
        System.out.println(c);
    }
}
class demo
{
    Public static void main(String args[])
    {
        C c = new c();
        c.add();
        c.sub();
    }
}
```



Extending Interface :

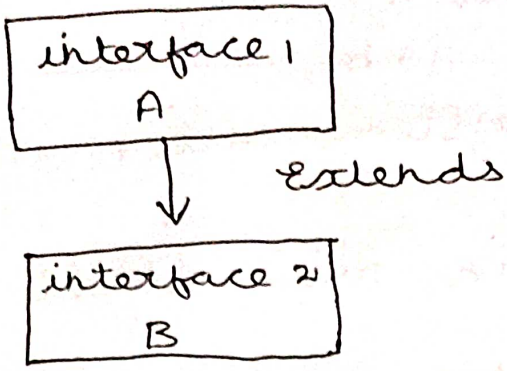
Interface are extended similar to classes. We can derive sub interfaces from main interfaces by using keyword extends.

```
        int c;
        c=a+b;
        System.out.println(c);
    }
    Public void sub()
    {
        int c;
        c=a-b;
        System.out.println(c);
    }
}
class demo
{
    Public static void main(String args[])
    {
        C c=new c();
        c.add();
        c.sub();
    }
}
```



Extending Interface:

Interface are extended similar to classes. We can derive sub interfaces from main interfaces by using keyword extends.



Syntax :

```
interface  interface_name2 extends interface_name1
{
    // body of interface
}
```

Example :

```
interface A
{
    int a=5;
}
interface B extends A
{
    int b=2;
}
class C implements B
{
    void add()
    {
        int c;
        c=a+b;
        System.out.println(c);
    }
}
class demo
{
    Public static void main(String
                            args[])
```

```
{
    C c = new C();
    c.add();
}
}
```