(28) Inner class and Introduction to Interface

↳ Navin & Nithin Sir
↳ 17 Nov Live class

→ Class → Bolue print from which individual objects are Created.

→ Object knows - Variables
→ Object does - Methods

→ Instance Variable :- Variable Created inside class and outside the Method

→ Local Variable :- Variable Created inside the Method.
↳ Scope of Variable over one the Method get Executed.

→ Object can only use Instance Variable..
→ Local Variable Can't Called using object
↳ They are Called using Methods then Method uses Variables.

→ Private is used for Instance Variables.
→ these accessed inside class.
→ to access outside the class we have to use getters and setters.

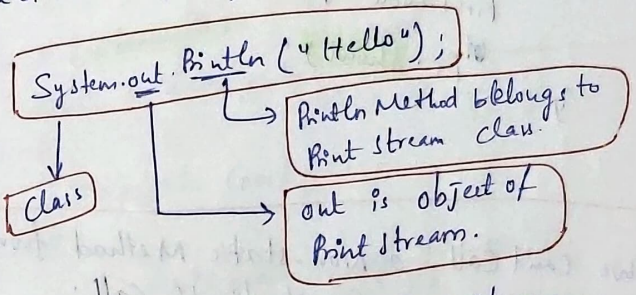
→ Class can have Variables, Methods & Constructors.

→ Class inside the class.

```
Class A {
    Variables;
    Methods;
    Classes; →
}
```

How & why to have class inside class.

→ Println Method belongs to Print stream class.



→ to println we need object of Print stream.
→ out is static Variable.
→ out is accessed by system class.
→ System class has in, out etc objects in it.
→ ctrl + Right click (Mouse) → give class info
→ you can't call a non-static Method from static Method. We Cannot directly Call it
→ we can call by making non-static Method as static Method and we can directly call.

Code:- Calling Method from static Method which is non static.

```
Public class FirstCode {
    int num;
    Public Void show() {
      s.o.p("in show" +num);
    }
    Public Static Void main(String[] args){
        FirstCode FC = new FirstCode();
        FC.show();
    }
}
```

→ We can't call a Non-static Method from Static Method. we Cannot directly call.
↳ by creating object we can call.
→ We can Call by Making non-static Method as static Method and we can directly call.

① Inner class :-

```
Public class FirstCode {
    int num;
    A obj2 = new A();        → Creating object of class A
    Public Void show()
    {
      System.out.Println("in show" + " "+num);
      obj2.config(); // Calling Config Method
    }
    class A { //inner class
        Public Void Config()
        {
          s.o.p("in Config");
        }
    }
    Public Static Void main(String[] args){
        FirstCode1 obj1 = new FirstCode1();
        obj1.show(); // Calling show Method.
    }
}
```

We cannot call Config Method outside class and outside Method.

→ we have 2 classes in Code
→ 2 class files are created when we compile above Code.
↳ FirstCode.class
↳ FirstCode$A.class [Class A belongs to FirstCode]

② Making Inner class Static and Access the Method inside it.

Code :- FirstCode2.java → refer git repo

```
Class A {
    Public Void show()
    {
        S.o.p("in show");
    }

    static class B {          → (Making inner class static)
        Public Void Config() {
            S.o.p("in Config");
    } }
}

Public class FirstCode2 {
    Public Static Void main (String [] args) {

        A obj1 = new A();   // Creating object of class A
        obj1.show();        // Calling show Method in Class A

        A.B obj2;
        obj2 = new A.B();
        obj2.Config();

    }
}
```

After Making Class b static we can access Constructor of B with the help of A.

→ Calling Config Method.

③ Inner class is non-static Member.

```
Class A {
    Public Void show() {
        S.o.p("in show");
    }
    Class B {
        Public Void Config() {
            S.o.p("in Config");
        }
    }
}

Public class FirstCode3 {
    Public Static Void main(String [] args) {
        // Creating object of class A
        A obj1 = new A();   →
        obj1.show();        → Calling show Method

    // we can access class B which inside class A from here
        A.B obj2;
        obj2 = obj1.new A.B();
        obj2.Config();      // Calling Config Method.
    }
}
```

→ To access the non-static Member we need to get object of class A

① if inner class is static we can use outer class Name for accessing Methods in inner class

```
A.B obj2 ;          → A.B irrespective of class B is static or not
obj2 = new A.B();   → using class A Name
obj2 = Config();
```

A.B irrespective of class B is static or Not.

② if inner class is non-static we have to use object of outer class to access the Methods inside inner class.

```
A.B obj2;
obj2 = obj1.newB1(); // using class A object (obj1)
obj2 . Config();
```

Inner class Example

we need chipset inside only in Computer:

```
class Computer {  //outer class
    Harddrive Hdd = new Harddrive();
    class Chipset  // inner class
    {

    }
}
```

Purpose of Inner class

→ if you have a class B which the scope of class should be only for class A.

Eg: only Computer class uses chipset class
only class A uses class B.

→ we don't need to maintain seperate documentation for class B and prefer to use in same class

important points:-

① we Can Make Inner class Static or Non-Static Member.

② we Can Create class inside Method as well but we Can use that class only inside Method not outside the Method.

Eg:

```
class A {
    Public Void show() {
        S.o.p("in show");
        class B {
            Public Void config()
            {
                S.o.p("in config");
            }
        }
    }
```

class B inside show() Method

④ Overriding Method generally

```
class Computer {
    Public Void Config() {
        S.o.p("in Computer Config");
    }
}
// overriding Config Method
Class AdvComputer Extends Computer
{
    Public Void Config() {  // overriden Method
        S.o.p("in Adv Computer");
    }
}
Public class FirstCode4 {
    Public Static Void main(String[] args)
    {
        // Parent type ref for child type object
        Computer obj = new AdvComputer();
        obj.Config();
    }
}
```

output: in Adv Computer

⑤ Overriding Method using Anonymous Inner class

↳ Providing implementation or defination at the time of Creating object.

↳ we are overriding Method at the time of Creating object of class.

```
Class Computer {
    Public Void Config() {
        S.o.p("in Computer Config");
    }
}
Public class FirstCode5 {
    Public static Void main(String[] args)
    {
// we are overriding Method at the time of Creating object
        Computer obj = new Computer()
        {
            Public Void Config() {
                S.o.p("new");
            }
        };
        obj.Config(); // Calling Config() Method
    }
}
```

→ it is inner class they are called anonymous inner class.

→ Anonymous class is coded between Constructor (Computer()) and semi colon (;).

⑥ accessing Abstract Method generally

```
abstract class Computer{
    Public abstract Void Config();
}
Class Laptop Extends Computer {
  // implementing body of abstract Method
  Public Void Config()
  {
    S.o.p("it's working");
  }
}
Public class First Code {
    Public Static Void main(String[] args)
    {
      Computer obj = new Laptop();
      obj. Config(); // output: its working.
    }
}
```

⑦ accessing Abstract Method by using Anonymous inner class :-

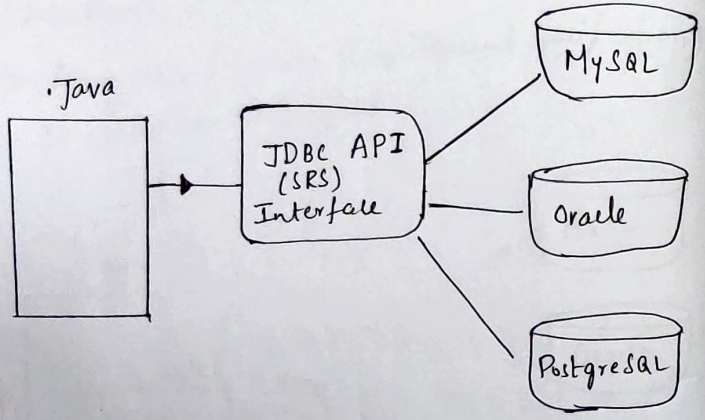
```
abstract class Computer{
  Public abstract Void Config();
}
Public class First Code {
    Public Static Void main(String[] args)
    {
      Computer obj = new Computer()
      {
        Public Void Config(){
          S.o.p("it's working fine");
        }
      };
      obj. Config(); // Calling Config Method.
    }
}
```

Anonymous → inner class

→ we can't reuse anonymous inner class.
it is used when the object is created.

② Interface In Java :- Definition - ①

① Any SRS (~~System~~ Service Requirement Specification) is called as SRS.



→ Java says write Code once and run anywhere.

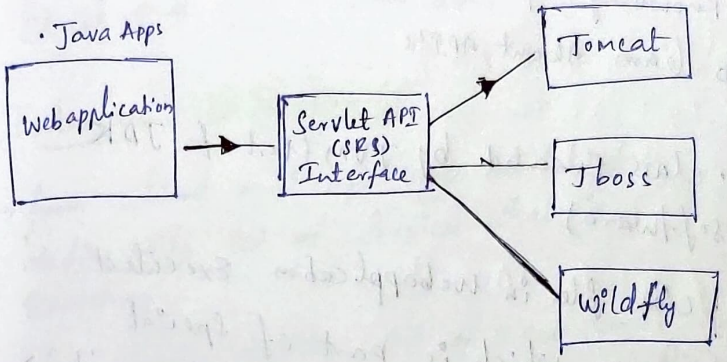
→ To Promote these Specifications ~~Practicesystem~~ Sun MicroSystem given API.

→ API → Collection of .class files.

→ Sun Microsystem gives API to make Java Connect to databases.

→ In Java to represent SRS (Service Requirement Specification) we use interface.

→



web application ②

→ These Application never run by JVM but they will run by Servers.

→ Special Software is required to run those Application.

→ Server Softwares → ① Tomcat
② Jboss (oracle)
③ Wildfly

→ write Code once and run on multiple platforms.

→ Here Server Softwares are platforms.

→ for promoting this Sun Microsystems introduced ~~Sunlet~~ Servlet API.

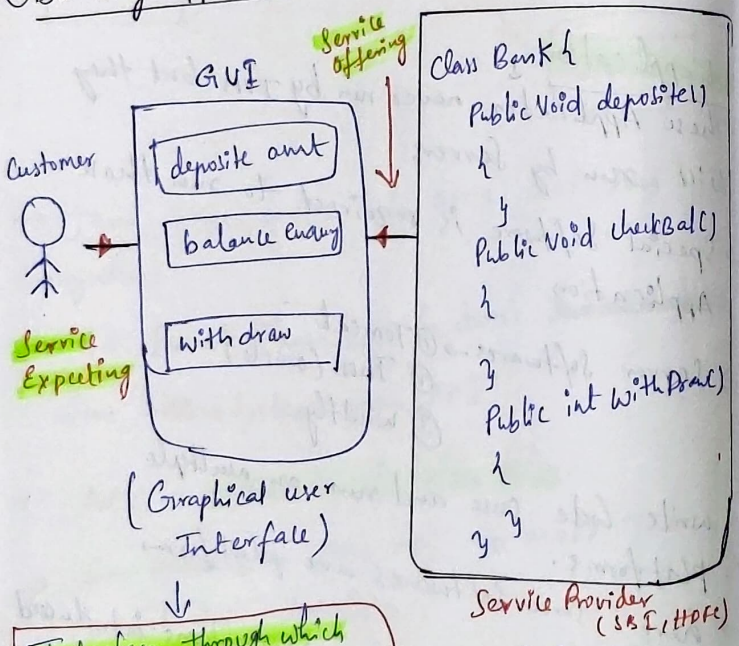
→ knowledge of interfaces is Mandatory to learn API.

→ Knowledge of interface is Mandatory to learn about API's.

→ .class Executed by JVM (part of JDK software)

→ .class file in webapplication Executed by JVM which is part of Spesial Softwares like Tomcat, glassfish and jboss.

③ Banking Application (Interface Example)



Interface through which interaction between Customer and Bank Application.

name of Services
ⓐ deposite
ⓑ check balance
ⓒ withDraw()

→ Interface acts like Contract between Client and Service provider.

Interface :- Defination - ④

→ from Client point of view an interface define the set of Services what is Expecting.

→ from Service provider point of view an interface define the set of Services what is "offering".

→ So interface acts as Contract between Client and Service provider.

Eg: GUI Screen of ATM defines the set of Services what the Customer is Expecting.

→ Bank people offered the same set of Services what the Customer is Expecting.

Customer → GUI → Bank.

→ SRS is represented through interface.

→ A Contract between Client, and servie provider is represented through Interface.

→ In order to promote 100% abstraction in Java we have go for Interface.

③ Interface - Defination ③

Inside Interface Every Method is always abstract Whether we are declaring (or) not hence interface is considered as 100% Pure abstract class.

Eg:-

```
Interface Account
{
    // it is 100% abstract class.
    // The Methods are abstract and
       Public by default.

    Void WithDraw();
    Void deposite();
    Void CheckBalance();
}

Public class TestApp
{
    Public static Void main(String[] args)
    {

    }
}
```

→ The access Modifier are public when declaring interface.

Summary:-

interface Corresponds to Service Requirement Specification (SRS) (or) Contract between client and Service provider (or) 100% Pure abstract class.

⇒ interface and classes follows Pascal Case Convention.

→ I followed by name → These Convention is used by many developers nowadays.

④ Declaration and Implementation of Interface

ⓐ Whenever we are implementing an interface Compulsorily for Every Method of that interface we should provide implementation otherwise we have to declare class as abstract class in that Case child is responsible to Provide implementation for remain Method

ⓑ Whenever we are implementing an interface Method Compulsorily it should be declared as public otherwise it would result in Compile Time Error.

Code:- Interface3.java → refer github.

```
Interface ISample
{

    Void m1();
    Void m2();

}
// SampleImpl gives implementation for Methods in ISample
// implements keyword is used.
class SampleImpl implements ISample
{

    @Override
    Public Void m1(){
        S.o.p("Hey Implementation is given for m1");
    }
    @ Override
    Public Void m2(){
        S.o.p("Implementation is given for m2");
    }
}
Public class Interface3{
        Public Static Void main(String[] args){

            ISample sample = new SampleImpl();
                                ↳ Lose Coupling.
            Sample.m1();
            Sample.m2();
}}
```

Rules to overriding a Method

1. Method Signature never be changed.
2. Retain Same access Modifier while overriding a Method.

   Eg: `Public Void m1(){ }`

→ Access Modifier is default in a class by default.

→ Method in interface is Public and abstract.

import points

1. I followed by name → indicate as interface
   ↳ developers using this Convention.
2. if a class Contains abstract Method then class should be Made abstract.
3. @Override - Indication to Compiler that these Methods are overriden Methods.
4. SampleImpl gives implementation for Methods in ISample Interface

→ implements keyword is used.

@ whenever we are implementing any interface Method compulsory for Every Method of the interface we should provide implementation otherwise we have to declare class as abstract class.

→ In this Case child is responsible to provide implementation.

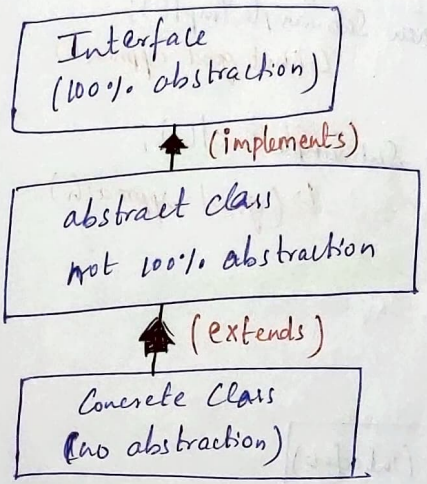
Eg:- Code:-

```
Interface Isample {
    Void m1()
    Void m2()
}
abstract class SampleImpl implements Isample
{
    @override
    Public Void m1(){
        sop("Implementation is given for m1");
    }
} // m2() Method implemented in child class

class SubSample Extends SampleImpl {

    @override
    Public Void m2(){
        S.o.p("implementation is given for m2");
    }
}
Public class Interface {
    Public static Void main(String[] args){
    SubSampleImpl Sample = new SubSampleImpl();
        Sample.m1();
    } } Sample.m2();
```

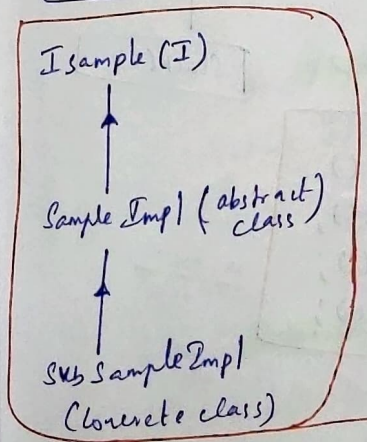
@ Hierarchy in Real time Coding (Developers approach)



→ Relationship b/w class & class is Extends

→ Relationship b/w class and interface is implements

→ from Code :-




```
SampleImpl ref = new SubsampleImpl();
Isample ref = new SubsampleImpl();
```

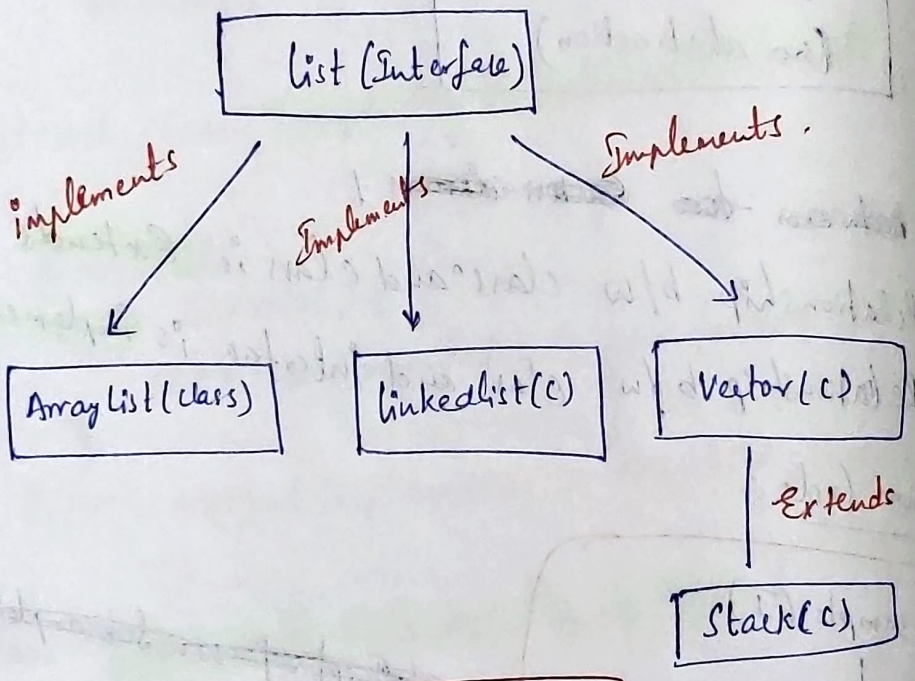
↳ not good approach.

↳ good approach

Loose Coupling

SampleImpl ref = new SubSampleImpl();
↳ (not good approach)

ISample ref = new SubSampleImpl();
↳ (good approach).




```
List ref = new ArrayList();
           new LinkedList();
           new Vector();
           new Stack();
```

↳ Polymorphism.