(30) Interface Conclusion and wrapper classes

↳ 21 Nov Live class
↳ Nithin Sir

① When to use interface, abstract class and Concrete class?

① interface → when we know about implementation

② abstract class → When we know partially about implementation.

③ Concrete class → Complement implementation and ready to provide service.

(or)

① interface :- it is preffered when we speak only about specification (no implementation)

② Abstract class :- it is when we speaks about partial implementation

③ Concrete class :- it is prefered when we speak about complete implementation and ready to provide service then we go for Concrete class.

② Difference between interface and abstract class?

interface	Abstract class
① 100% abstraction.	① not 100% abstraction.
② Methods in interface are by default Public and abstract.	② Methods need not be public and Abstract
③ Private, static, strictfp, Synchronized, native Methods not possible.	③ Since it is not Public and abstract Private, static, strictfp Synchronized, native methods possible.
④ Variables in interface are by default they are public static final.	④ Variables in Abstract class need not be Public static final.
⑤ Variable initialization should be at the time of declaration since Variables are Public static and final	⑤ Variable initialization can be at any place.
⑥ no need of constructor, instance block and static block.	⑥ We can have static, instance block and Constructor.

③ Static block, Instance Block and Constructor

(i) Static Block :- .class file loding happens and use to initialize static variable.

(ii) Instance block :- it is executed during the creation of an object just before the Constructor call.
→ it is used for initialization of instance Variable.

(iii) Constructor :- during the creation of an object, used for initialization of instance Variable.

Notes

Difference between interface and abstract class?

interface	Abstract Class
① if we don't know anything about implementation just we have requirement specification then we should go for interface	① if we are talking about implementation but not Completely Then we should go for Abstract class.
② Every Method present inside the interface is always public, ~~static~~ and abstract whether we are declaring (or) not	② Every Method present inside the ~~interface~~ abstract class need not be public & ~~static~~ abstract.
③ We can't declare interface Method's with modifiers like Private, Protected, final, static, ~~Sync~~ synchronized, native, strictfp.	③ there are no restrictions on abstract ~~class~~ class method modifiers.
④ Every interface variable is always public, static and final. whether we are declaring (or) not.	④ Every abstract ~~class~~ class Variable need not be public static final.
⑤ Every interface Variable is always public static final we can't declare with the following modifiers like Private, protected, transient, Volatile.	⑤ No restriction on access Modifiers.
⑥ for Every interface Variable Compulsorily we should perform initialization at the time of declaration, otherwise we get Compile time Error.	⑥ Not required to Perform initialization for abstract class Variables at the time of declaration
⑦ Inside interface we can't write static, instance block and Constructor.	⑦ Inside abstract class we can write static block instance block and Constructor.

Q) Can abstract class be instantiated / object be created?

A) Abstract class contains Concrete and abstract Methods. We cannot created object because it is abstract.

Q) ~~Can abstract class contains~~ ?

→ during child object creation only child class object will be created but no parent class object is created. Still constructor of Parent is called to bring the Properties of parent to child.

Code :-

```
Class Parent {
    Parent() {
        S.o.p("Parent class Constructor");
        S.o.p(this.hashCode");   → (1)
    }
}
class child {
    child() {
        S.o.p("child class Constructor");
        S.o.p(this.hashCode());   → (2)
    }
}
Public class interface {
    Public static void main(String() args){
        child C = new child();
        S.o.p(C.hashCode());   → (3)
    }
}
```

↳ gives the address of child obj

→ all hashcodes at (1) (2) & (3) then we conclude that only child class object is created.

Q) Can Abstract Class be instantiated / object be created?

A) we cannot create object because it is abstract

Q) ~~Can abstract class contain~~ ?

⑤ Can Abstract class Contains Constructor?

Ⓐ Abstract class will have a Constructor for initializing the instance Variables. this instance Variables needed by child class and child get those values by making class to Constructor of Parent class by Super Method.

Example:- interface19.java (refer git repo)

```
abstract class Person {
    String name;
    Integer age;
    float height;
Person (String name, Integer age, float height) {
        this.name = name;
        this.age = age;
        this.height = height;
    }
}
class students Extends Person {
    Integer sid;
    float marks;
Student (String name, Integer age, float height,
         Integer sid, float Marks) {
    super(name, age, height);
    this.sid = sid;
    this.marks = marks;
    }
}
```

↳ Constructor-II to initialize the instance Variables

↳ Constructor-II

Super Method Calls parent Constructor for getting values in Constructor

① Constructor-I in abstract class

↳ Constructor for initializing the instance Variables.

② Super Method in Constructor - Calls parent Constructor.

→ we can write abstract class without any Method.

⑥ can Interface object be instantiated (or) Can object for Interface be created?

Ⓐ No, Interface contain abstrat Methods. Interface does not contain implementation Methods. So object for Interface cannot be Created

⑦ Can interface contains constructor?

Ⓐ No Instance Variable in interface so Constructor is not required.

⑧ Can reference be created for abstract class?

Ⓐ we can create reference Variable for abstract class.

Eg: Person P = new Person ("ashish", 23, 5.8f);

↳ from above Code

⑨ Can ~~Datatype~~ reference be Created for interface?

⑩ ① an interface cannot be instantiated. Therefore, it cannot be referenced directly.

② However, an object of class type, which extends the interface, can be used to assign a reference of that interface type.

Eg:- `ISample Sample = null;`

Note:- Every Method present inside the interface is abstract, but in abstract class also we take only abstract Methods then

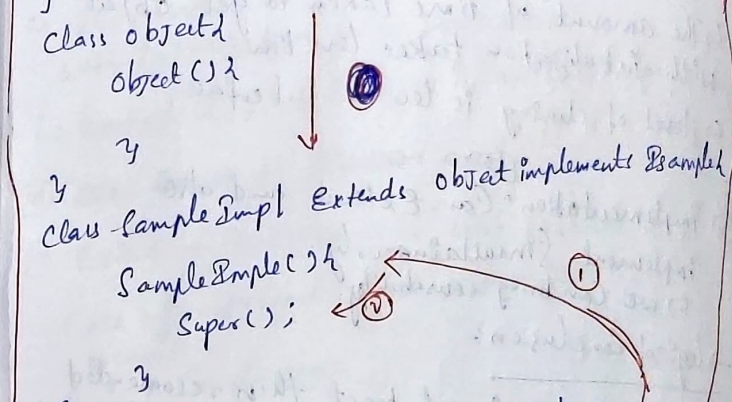
⑩ What is the need of interface ~~with abstract class~~ concept?

Code①: interface21.java

```
interface IDemo {
    Void m1();
    Void m2();
}
abstract class Demo {
    Public abstract Void m3();
    Public abstract Void m4();
}
Public class Interface21 {
    P.S.V.M(String a) {

    }
```

Why to go with interface?

```
interface ISample {

}
class object {
    object() {

    }
}
class SampleImpl Extends object implements ISample {
    SampleImpl() {
        super();
    }
}
ISample sample = new SampleImpl();
```



→ When object is created Control go SampleImpl() Constructor. ~~Interface~~ After that class object get Executed.

→ no Constructor in interface So Control goes to class object.

→ ~~Level of chaining~~ ②

→ Levels of chaing - 2 levels.

→ the amount time to get object with initialization take less time in interface

⑪ interface - Performance ~~is~~ high

→ level chaing in interface is less so Performance is high.

Conclusion on interface

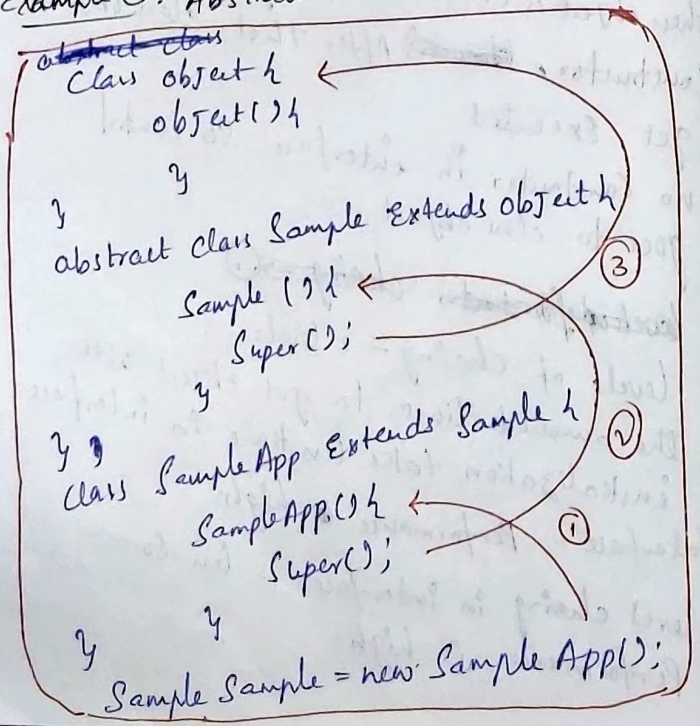
① Performance is high in case of interface
↳ The amount of time taken to get object with initialization takes less time.
↳ level of chaing is less in interface

② Implementation class can Extend and also implement simultaneously.
↳ we can bring reusability.

→ logical Conclusion :-

↳ if Everything is abstract then recomended to go with interface.

Example ②: Abstract class



Conclusion on Abstract class

① Performance is low in case of abstract class.
↳ the amount of time takeh to get object with initialization takes more time.
↳ level of chaining is more in abstract class
② ~~class cannot be extend and~~.
③ Extending two classes is not possible in case of abstract class. reusability is not brought.

→ logical Conclusion :-

↳ if Everything is abstract then recommended to go with interface.

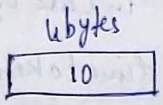
→ interface :- type in Command prompt
↳ java.util.Collection → Interface
↳ java.util.Array → Interface
↳ java.util.ArrayList
↳ java.util.AbstractList

→ always Constructor should be overloaded when we working at API level.

② Wrapper Class

ⓐ Primitive type

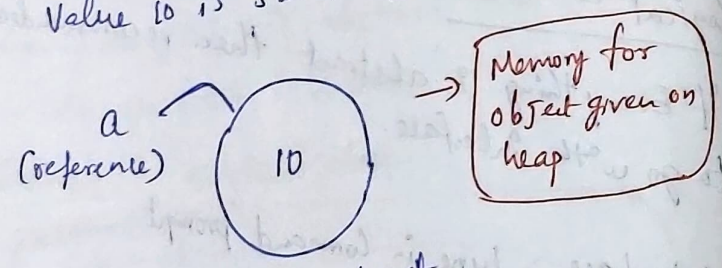
int a = 10;



→ JVM gives 4bytes of Memory on Ram for a.
→ then value is stored.

ⓑ Reference type:-

Integer a = 10;
→ here object is Created. In that object value 10 is stored. for that object



→ using object we can call Methods
→ type in cmd → Javap java.lang.Integer
→ we can call Methods like

ParseInt()
tostring()
Valueof()

→ they are called helper classes
→ Compiler will not store value but it replace the value in .class file.
→ JVM only save it.

→ Memory for Variables

① local Variable - Stack
② Instance Variable - heap
③ Static Variable - Method Area (heap)

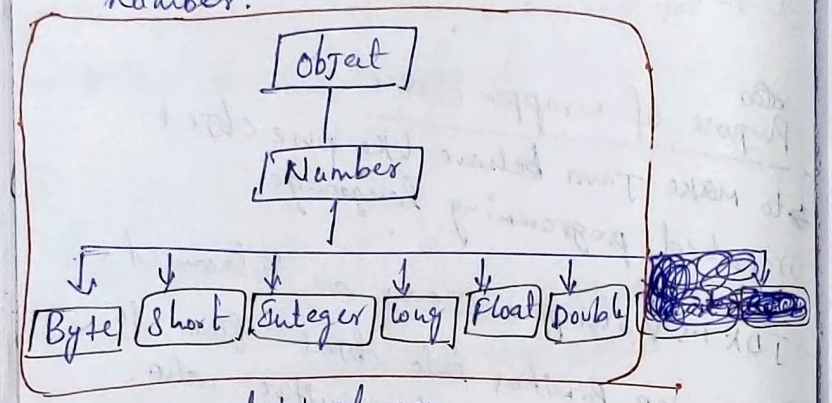
→ Purpose of wrapper classes

① to make Java behave like pure object oriented programming language
→ JDK 1.5 V → wrapper classes are introduced.
② To wrap Primities into object form so that we can handle primitive valo just like objects.
③ To define several utilities functions which are required for the primitives.

→ for Every class parent has to be object.
→ for all wrapper classes the parent is numbers.

→ for Every class Parent has to be object class.

→ The parent of all wrapper class is number.



→ All are child classes.

→ for object Memory is at heap level.

→ these classes are pre-defined. they are present in package.

→ they are present in java.lang Package.

→ Since they are pre-defined classes we have to create object for these classes.

→ cmd → type java.lang.Integer.

→ After Creating object Constructors are called.

Constructors

Almost all the wrapper classes have 2-Constructors

ⓐ One taking Primitive type

ⓑ One taking String type.

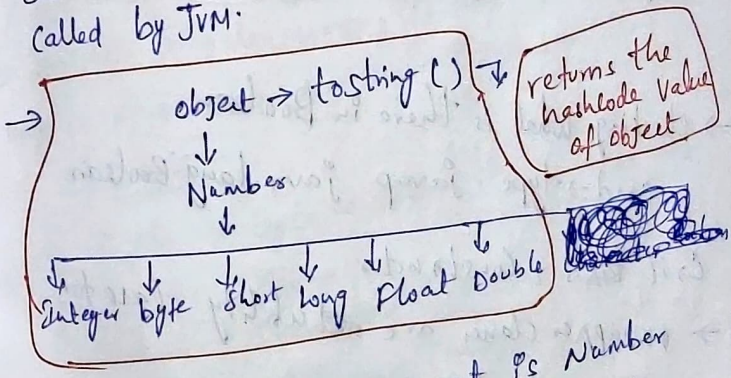
Eg: Integer i1 = new Integer(10.5);

Integer i2 = new Integer("10.5");

Double d1 = new Double(10.5);

Double d2 = new Double("10.5");

→ When we print any reference variable then automatically to String Method will be called by JVM.

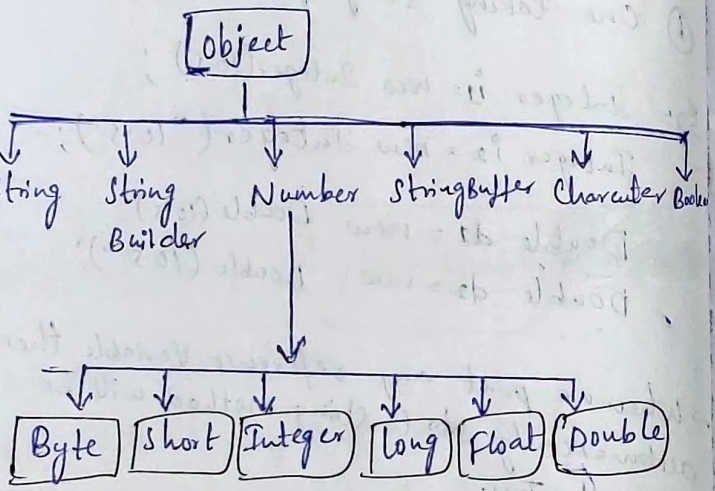


→ for Integer direct parent is Number

→ to String() Method present in object class.

→ implementation of tostring() Method is to Print hashcode.

→ for all wrapper class toString() is overridden to print data present in the object.



→ checking what is there in Boolean
cmd → type javap java.lang.Boolean.
↳ it takes 2 Constructor.

⇒ wrapper class are utility class for Java developers.

⇒ to reduce complexity when we use String and wrapper objects. They overriden object Methods

→ Eg: toString() and Equals() are overridden.

→ Package in "java.lang" package

In object class

① toString() – returns the hashcode value of the object.

② Equals() :- to compare reference.

In wrapper class

↳ this Methods are overriden for wrapper classes.

① toString() – overriden to print the data present in the object.

② Equals() :- overriden to compare the content present in the object.

⇒ Just like String class, wrapper classes are treated as immutable.

➡ wrapper class and its associated Constructor

Byte → Byte Constructor takes byte and String.
Short → Short and String
Integer → int and String
Long → long and String
Float → Float, String and double
Double → Double and String
Character → character
Boolean → boolean & String.

→ for code refer wrapper1.Java in git repo.

① // Integer Constructor takes Integer and String

```
Integer i1 = new Integer(10);
S.O.P(i1);
Integer i2 = new Integer("10");
S.O.P(i2);
Integer i3 = new Integer("Ten");
S.O.P(i3);
```

→ here ten is string
a Constructor which takes String as argument instead of integer type data it throws Exception.

NumberFormatException

② // Float Constructor takes float, double & String.

```
Float f1 = new Float(10.5f); // float
Float f2 = new Float("10.5f"); // String
Float f3 = new Float(10.5); // double
```

③ character takes only character.

```
Character c1 = new Character('a');
S.O.P(c1);
```

④ Boolean takes two have Constructor

```
Boolean b1 = new Boolean(true);
S.O.P(b1) // true
Boolean b2 = new Boolean(false);
S.O.P(b2) // false.
```

→ JVM don't know True, false, TRUE, FALSE

→ it gives Error.

```
Boolean b3 = new Boolean(True);
S.O.P(b3);
Boolean b4 = new Boolean(false);
S.O.P(b4);
Boolean b5 = new Boolean(TRUE);
S.O.P(b5);
Boolean b6 = new Boolean(FALSE);
S.O.P(b6);
```

gives Error

(4.1) Giving String Value for Boolean.

→ When you give String for boolean Case is not important but content is important..

→ In case of Boolean Constructor, boolean value be treated as true w.r.t to Case insensitive part of 'true', for all others it would be treated as 'false'.

Eg:

```
Boolean b6 = new Boolean("true"); // true
Boolean b7 = new Boolean("false"); // false
Boolean b8 = new Boolean("True"); // true
Boolean b9 = new Boolean("false"); // false
Boolean b10 = new Boolean("TRUE"); // true
Boolean b11 = new Boolean("FALSE"); // false
Boolean b12 = new Boolean("ashish"); // false
```

(4.2) Equals() and Equal operator.

```
Boolean b13 = new Boolean("Yes");
Boolean b14 = new Boolean("No");
S.o.p(b13); // false
S.o.p(b14); // false
S.o.p(b13.Equals(13)); // false.Equals(false) = True
```

↳ Equals Method Compares Content Inside it

```
S.o.p(b13 == b14);
```

↳ false (Compare reference).

(5) Equals Method on Integer

```
Integer I1 = new Integer(10);
Integer I2 = new Integer(10);
S.o.p(I2); // 10
S.o.p(I2); // 10
S.o.p(I1.Equals(I2)); // true.
```

⇒ **Immutable class :-**

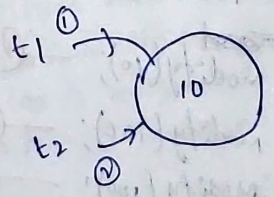
if we create an object and if we try to Make a change, with the change new object will be Created and those changes will not ~~reflect~~, reflected in the old copy.

① Can we Make our User defined class as immutable?

```
Test t1 = new Test(10);
Test t2 = t1.modify(10);
```

↳ user defined Method.

```
S.o.p(t1 == t2) // true.
```



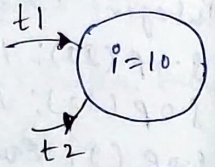
here Content is not modified so same reference will be reshared with t2.

→ Make class final no one should change the logic.

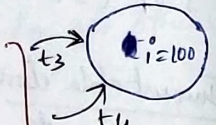
① Can we make our userdefined class a immutable?

Ⓐ Yes possible as shown below.

Code: refere wrapper2.java

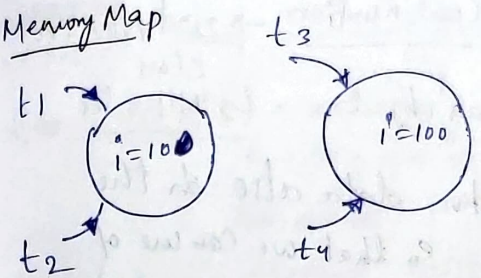



```
final Class Test {
    int i;
    Test(int i) {
        this.i = i;
    }
    Public Test Modify(int i) {
        if (this.i == i) {
            return this;
        }
        Else
            return new Test(i);
    }
    Public Static Void main (String[] args)
    {
        Test t1 = new Test(100);   —①
        Test t2 = t1.modify(10);   —②
        Test t3 = t2.modify(100);  —③
        Test t4 = t3.Modify(100);  —④
        S.o.p(t1==t2)  // true
        S.o.p(t1==t3)  // false
        S.o.p(t3==t4)  // false
    }   S.o.p(t3==t4)  // false.
```

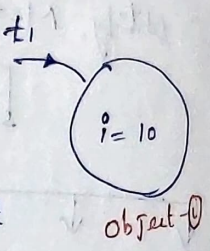


logic for user defined Method. (Modify())

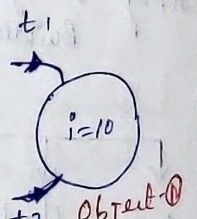
Memory Map



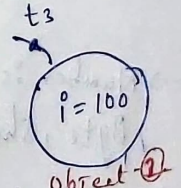
→ ① Test t1 = new Test(10)
↳ object is created
↳ 10 is assigned to i
↳ t1 is reference for object



② Test t2 = t1.Modify(10);
↳ here value of i is not Modified.
↳ Same reference is reshared with t2



③ Test t3 = t3.Modify(100);
↳ here value of i is Modified.
↳ a new object is created & 100 is assigned to i
↳ t3 is reference



④ Test t4 = t4.Modify(100);
↳ here value of i is not Modified.
↳ Same reference is reshared with t4
↳ t4 is reference