Interface

=======

 Def1::Any service requirement specification(SRS) is called interface. (OR)

Contact between client and service provider (OR)

100% pure abstract class.

eg:: SUN people responsible to define JDBC API and database vendor will provide  implementation for that

 SUN

 |

 JDBCAPI

 |

 Oracle MySQL PostgreSQL

===========================================

 SUN

 |

 ServletAPI

 |

 tomcat weblogic jboss

===========================================

Def2::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 a contract b/w 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.

 Cusomter => GUI => Bank

Def3:: 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 by default "abstract and public"

 void withDraw();

 void deposit();

 void checkBalance();

}

Summary::

Interface corresponds to Service Requirement Specification(SRS) or contract b/w  client and service provider or 100% pure abstract class.

Declaration and implementation of Interface

===========================================

a. Whenever we are implementing an interface means implemented class compulsorily for every method of that  interface  it provide implementation otherwise we have to declare class as abstract  class and in that case child class is responsible to provide  implementation for remaining methods.

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

eg:: interface ISample{

void methodOne();

void methodTwo();

 }

 abstract class Sample implements ISample{

 public void methodOne(){...}

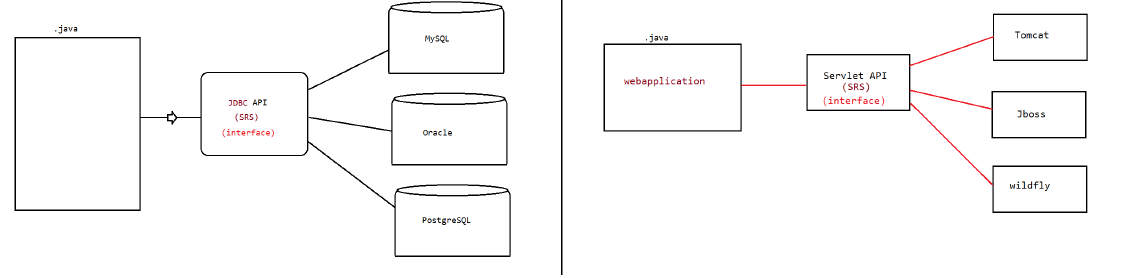
 }

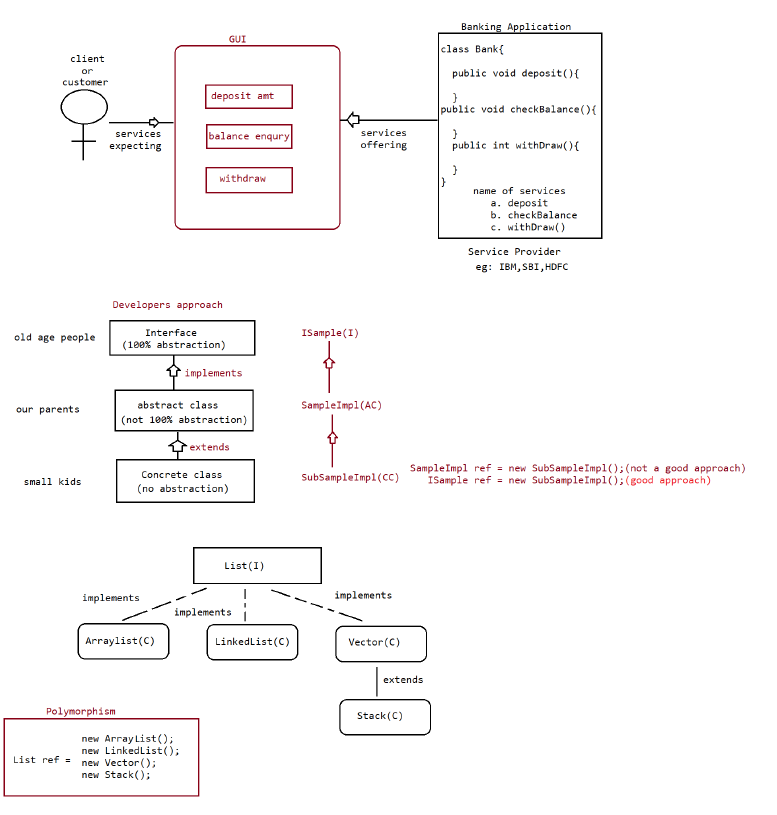
 class SubServiceProvider extends Sample{

 @Override

public void methodTwo(){...}

 }





18/11/2022

Difference b/w extends vs implements

case1:

extends :: One class can extends only class at a time  eg:: class One{}

 class Two extends One{}

implements:: One class can implements any no of interface at a time.  eg:: interface One{

 void methodOne();

 }

 interface Two{

 void methodTwo();

 }

 class Demo implements One,Two{

 public void methodOne(){}

 public void methodTwo(){}

 }

case2:

 A class can extend a class and can implements any no of interfaces  simultaneously.

 eg: interface One{

 public void methodOne();

 }

 class Two{

 public void methodTwo(){}

 }

 class Three extends Two implements One{

 @Override

 public void methodOne(){

....

 }

 @Override

 public void methodTwo(){

 ..

 }

 }

**case3:**

An interface can extend any no of interfaces at at time.  eg:: interface One{

 public void methodOne();

 }

 interface Two{

 public void methodTwo();

 }

 interface Three extends One,Two{

 public void methodOne();

 public void methodTwo();

 public void methodThree();

 }

Answer the following

==================

Which of the following is true?

 a. A class can extend any no of class at a time.

 b. An interface can extend only one interface at at time.

 c. A class can implement only one interface at at a time.

 d. A class can extend a class and can implement an interface but not both  simultaneously.

 e. An interface can implements any no of Interfaces at a time.  f. None of the above

answer: f

Consider the expression X extends Y which of the possiblity of X and Y expression  is true?

 1. Both x and y should be classes.

 2. Both x and y should be interfaces.

 3. Both x and y can be classes or can be interfaces.

 4. No restriction.

Equation : X extends Y == > true

class extends class => true

interface extends interface => true

Answer: 3

Q>

Predict X,Y,Z

 a. X extends Y,Z?

 b. X extends Y implements Z?

 c. X implements Y,Z?

 d. X implements Y extends Z?

a. X extends Y,Z?

X => interface

Y => interface

Z => interface

b. X extends Y implements Z?

X => class

Y => class

Z => interface

c. X implements Y,Z?

X => class

Y => interface

Z => interface

d. X implements Y extends Z

combination is illegal

**Interface Methods**

**=================**

 Every method present inside the interface is public.

 Every method present inside the inteface is abstract.

 How many declaration are valid?

 a. void methodOne();

 b. public void methodOne();

 c. abstract void methodOne();

 d. public abstract void methodOne();

answer: All are valid

public => To make the method available for every implementation class.

abstract => Implementation class is responsible for providing the implementation.

eg: jdbc api (java.sql.\*)

 |

 |

 implementation should be given by

a. mysql

b. oracle

c. postgresql

Since the methods present inside the interface is   => public,abstract methods

illegal combination of modifiers are  static,private,protected,strictfp,synchronized,native,final.

Interface variables

===================

 => Inside the interface we can define variables.

 => Inside the interface variables define requirement level constants.

=> Every variable present inside the interface is by default public static final.

eg:: interface ISample{

int x=10;

 }

 public :: To make it available for implementation class Object.

static :: To access it without using implementation class name.

final :: Implementation class can access the value without any  modification.

variable declaration inside interface

 a. int x=10;

 b. public int x=10;

 c. static int x=10;

 d. final int x=10;

 e. public static int x=10;

 f. public final int x=10;

 g. static final int x=10;

 h. public static final int x=10;

Answer: All are legal

since the variable defined in interface is public static final,we cannot use  modifiers like private,protected,transient,volatile.

since the variable is static and final,compulsorily it should be initialized at the time of declaration otherwise it would result in  compile time error.

eg:: interace IRemote{ int x;}// compile time error.

=> interface variables can be accessed from implementation class,but cannot modify  if we try to modify  it would result in compile time error.

eg:: interface Remote{

 int VOLUME = 100;

 }

 class Lg implements Remote{

 public static void main(String... args){

VOLUME=0;//CE:: cannot assign a value to final variable VOLUME

System.out.println("value of volume is ::"+VOLUME);

 }

 }

eg:: interface Remote{

 int VOLUME = 100;

 }

 class Lg implements Remote{

 public static void main(String... args){

int VOLUME=0;//local variable

System.out.println("value of volume is ::"+VOLUME);//0

 }

 }

Interface Naming Conflicts

==========================

 Case 1::

 If 2 interfaces contain a method with same signature and same return type in  the implementation class only   one method implementation is enough.

eg::

 interface Left{ public void methodOne();}

 interface Right{public void methodOne();}

 class Test implements Left,Right{

@Override

 public void methodOne(){

...

 }

 }

Case2:

 If 2 interfaces contain a method with same name but different arguments in the  implementation class we have to provide   implementation for both methods and these methods acts as a Overload methods.

eg::

 interface Left{ public void methodOne();}

 interface Right{public void methodOne(int i);}

 class Test implements Left,Right{

 @Override

 public void methodOne(){

 ...

 }

@Override

 public void methodOne(int i){

...

 }

 }

case3::

 If two interfaces contains a method with same signature but different return  types then it is not possible to implement   both interface simultaneously.

eg:: interface Left { public void methodOne(); }

 interface Right{ public int methodOne(); }

 class Test implements Left,Right{

 @Override

 public void methodOne(){

 ...

 }

@Override

 public int methodOne(){

...

 }

 }

Note:

Q> Can a java class implements 2 interfaces simultaneously?

 yes possible, except if two interfaces contains a method with same signature  but different return types.

Variable naming conflicts::

Two variables can contain a variable with same name and there may be a chance  variable naming  conflicts but we can resolve variable naming conflicts by using interface names.

example1:

 interface Left{ int x=888;}

 interface Right{ int x=999;}

 public class Test implements Left,Right{

public static void main(String... args){

System.out.println(Left.x);

System.out.println(Right.x);

}

 }

**Note:**

 inside interface the methods are by default "public and abstract".  inside interface the variables are by default " public static and final".  We can also write an interface without any variable or abstract methods.

interface Serializable{

}

class SampleImpl implements Serializable{

}

interface Cloneable{

}

class SampleImple implements Cloneable{

}

**MarkerInterface**

**================**

 => If an interface does not contain any methods and by implementing that  interface if our Object will get some ability such  type of interface are called "Marker Interface"/"Tag Interface"/"Ability  Interface".

 => example

Serializable,Cloneable,SingleThreadModel.

example1

By implementing Serializable interface we can send that object across the  network and we can save state of an   object into the file.

example2

 By implementing SingleThreadModel interfaace servlet can process only one client request at a time so that we can   get "Thread Safety".

example3

 By implementing Cloneable Interface our object is in a position to provide  exactly duplicate cloned object.

Without having any methods in marker interface how objects will get ability?  Ans.JVM is responsible to provide requried ability.

Why JVM is providing the required ability to Marker Interfaces?  Ans. To reduce the complexity of the programming.

Can we create our own marker interface?

 Yes, it is possible but we need to customize JVM.(hard for beginner)

=================================================================================== ===========

Adapter class(It is a design pattern allowed to solve the problem of direct  implementation of interface methods)

=================================================================================== ===========

It is a simple java class that implements an interface only with empty  implememtation for every method.

If we implement an interface compulsorily we should give the body for all the  methods whether it is required or not. This approach increases the length of the code and reduces  readability.

eg:: interface X{

void m1();

 void m2();

 void m3();

void m4();

 void m5();

 }

 class Test implements X{

public void m3(){

 System.out.println("I am from m3()");

 }

 public void m2(){}

 public void m3(){}

public void m4(){}

 public void m5(){}

 }

In the above approach, even though we want only m3(), still we need to give body  for all the abstract methods, which increase the  length of the code, to reduce this we need to use "Adapater class". Instead of implementing the interface directly we opt for "Adapter class".

Adapter class are such classes which implements the interface and gives dummy implementation for all the abstract methods of interface.

So if we extends Adapter classes then we can easily give body only for those  methods which are interested in giving the body.

eg::

interface X{

void m1();

 void m2();

 void m3();

void m4();

 void m5();

}

abstract class AdapaterX implements X{

public void m1(){}

 public void m2(){}

 public void m3(){}

public void m4(){}

 public void m5(){}

}

class TestApp extends AdapterX{

public void m3(){

System.out.println("I am from m3()");

}

}

eg:

 Servlet(I)

| implements

GenericServlet(abstract class)

| extends

HttpServlet(abstract class)

| extends

 MyServlet(class)

23/11/2022

Wrapper class utiltiy methods

=============================

1. valueOf() method.

2. XXXValue() method.

3. parseXxx() method.

4. toString() method.

 public static wrapper valueOf(String data, int radix) throws  java.lang.NumberFormatException;

 public static wrapper valueOf(String data) throws

java.lang.NumberFormatException;

 public static wrapper valueOf(int data);

valueOf() method

================  To create a wrapper object from primitive type or String we use valueOf().  It is alternative to constructor of Wrapper class, not suggestable to use.  Every Wrapper class,except character class contain static valueOf() to create a  Wrapper Object.

eg#1.

Integer i=Integer.valueOf("10");

Double d=Double.valueOf("10.5");

Boolean b=Boolean.valueOf("nitin");

 System.out.println(i);

 System.out.println(d);

 System.out.println(b);

eg#2.

 public static valueOf(String s,int radix)

 |=> binary : 2(0,1)

 |=> octal : 8(0-7)

 |=> decimal : 10(0-9)

 |=> hexadecimal : 16(0-9,a,b,c,d,e,f)

 |=> base : 36(0-9,a-z)

Integer i1=Integer.valueOf("1111");

 System.out.println(i1);//1111

Integer i2=Integer.valueOf("1111",2);

 System.out.println(i2);//15

Integer i3=Integer.valueOf("ten");

 System.out.println(i3);//RE:NumberFormatException

Integer i4=Integer.valueOf("1111",37);

 System.out.println(i4);//RE:NumberFormatException

eg#3.

 public static valueOf(primitivetype x)

Integer i1=Integer.valueOf(10);

Double d1=Double.valueOf(10.5);

Character c=Character.valueOf('a');

Boolean b=Boolean.valueOf(true);

 Primtive/String =>valueOf() => WrapperObject

2. xxxValue()

 We can use xxxValue() to get primitive type for the given Wrapper Object.  These methods are a part of every Number type Object.

 (Byte,Short,Integer,Long,Float,Double) all these classes have these 6 methods  which is

 Written as shown below.

Methods

=======

 public byte byteValue();

 public short shortValue();

 public int intValue();

 public long longValue();

 public float floatValue();

 public double doubleValue();

eg#1.

 Integer i=new Integer(130);

// result = minrange +(total -maxrange -1)

 System.out.println(i.byteValue());//-126

 System.out.println(i.shortValue());//130

 System.out.println(i.intValue());//130

 System.out.println(i.longValue());//130

 System.out.println(i.floatValue());//130.0

 System.out.println(i.doubleValue());//130.0

3. charValue()

 Character class contains charValue() to get Char primitive for the given  Character  Object.

public char charValue()

eg#1.

 Character c=new Character('c');

 char ch= c.charValue();

 System.out.println(ch);

4. booleanValue()

 Boolean class contains booleanValue() to get boolean primitive for the  given boolean  Object.

public boolean booleanValue()

eg#1.

 Boolean b=new Boolean("nitin");

 boolean b1=b.booleanValue();

 System.out.println(b1);//false

In total xxxValue() are 36 in number.

 => xxxValue() => convert the Wrapper Object => primitive.

parseXXXX()

===========

 We use parseXXXX() to convert String object into primitive type.

form-1

======

public static primitive parseXXX(String s)

 Every wrapper class,except Character class has parseXXX() to convert String into  primitive type.

eg: int i=Integer.parseInt("10");

 double d =Double.parseInt("10.5");

 boolean b=Boolean.parseBoolean("true");

usage of Wrapper class in realtime coding

==================================

//WAP to take inputs from the command line and perform arithemetic operations

class Test

{

public static void main(String[] args)

{

//valueOf() => Converts String/Primitive to Wrapper type

//xxxValue() => Converts Wrapper type to Primitive type

//parseXXX() => converts String to primitive type

//commandline arguments => String inputs = args[0],args[1]

int i1 = Integer.parseInt(args[0]);

int i2 = Integer.parseInt(args[1]);

System.out.println(i1+i2);

System.out.println(i1-i2);

System.out.println(i1\*i2);

System.out.println(i1/i2);

//args -> String, convert into primitive type and process

}

}

form-2

======

public static primitive parseXXXX(String s, int radix)

 |=> range is from 2 to 36

Every Integral type Wrapper class(Byte,Short,Integer,Long) contains the following  parseXXXX()

to convert Specified radix String to primitive type.

eg: int i=Integer.parseInt("1111",2);

 System.out.println(i);//15

Note: String => parseXXX() => primitive type

toString()

=========

 To convert the Wrapper Object or primitive to String.

 Every Wrapper class contain toString()

form1

=====

 public String toString()

1. Every wrapper class (including Character class) contains the above toString()  method to convert wrapper object to String.

2. It is the overriding version of Object class toString() method. 3. Whenever we are trying to print wrapper object reference internally this   toString() method only executed

eg: Integer i=Integer.valueOf("10");

 System.out.println(i);//internally it calls toString() and prints the Data.

form2

=====

 public static String toString(primitivetype)

1. Every wrapper class contains a static toString() method to convert primitive to  String.

String s=Integer.toString(10);

 |=> primitive type int.

eg:

 String s=Integer.toString(10);

 String s=Boolean.toString(true);

 String s=Character.toString('a');

form3

=====

Integer and Long classes contains the following static toString() method to convert the  primitive to specified radix String form.

 public static String toString(primitive p,int radix)

 |=> 2 to 36

eg: String s=Integer.toString(15,2)

 System.out.println(s); // 1111

form4

=====

Integer and Long classes contains the following toXxxString() methods. public static String toBinaryString(primitive p);

public static String toOctalString(primitive p);

public static String toHexString(primitive p);

Example:

class WrapperClassDemo {

 public static void main(String[] args) {

String s1=Integer.toBinaryString(7);

String s2=Integer.toOctalString(10);

String s3=Integer.toHexString(20);

String s4=Integer.toHexString(10);

System.out.println(s1);//111

System.out.println(s2);//12

System.out.println(s3);//14

System.out.println(s4);//a

 }

}

Note:

String class

 public static String valueOf(boolean);

 public static String valueOf(char);

 public static String valueOf(int);

 public static String valueOf(long);

 public static String valueOf(float);

 public static String valueOf(double);

String data = String.valueOf('a');//static factory methods (We are calling valueof() method on String class)

String data = "sachin".toUpperCase();//instance factory methods ( we are calling valueOf(); method on String Object)

AutoBoxing and AutoUnBoxing

=========================

untill 1.4Version, we can't provide wrapper class objects in place of primitive and primitive in place of wrapper object all

the required conversions should be done by the programmer.

But from jdk1.5 Version onwards,we can provide primtive in place of wrapper and in  place of wrapper we can keep primitive also.All the requried conversion will be done by the compiler automatically, this  mechanism is called as "AutoBoxing" and "AutoUnBoxing".

eg#1.

Boolean b1 = Boolean.valueOf(true);

if (b1)

 System.out.println("hello");

eg#2.

ArrayList al = new ArrayList();

 al.add(10);

Autoboxing

=========  Automatic conversion of primtive type to wrapper object by the compiler is called "AutoBoxing".

Integer i1 = 10;

 |

|After compilation the code would be

|

|

 Integer i1 = Integer.valueOf(10);

Note: Autoboxing is done by the compiler using a method called "valueOf()".

AutoUnBoxing

===========Automatic conversion of wrapper object to primtive type by compiler is called  "AutoUnBoxing".

Integer i1 = new Integer(10);

 int i2 = i1;

|

|compiler converts Integer to int type using intValue()

|

 int i2 = i1.intValue();

Note: AutoUnboxing is done by the compiler using a method called "xxxValue()"

Case1:

=====

class Test

{

static Integer i1 = 10;// AutoBoxing

public static void main(String[] args)

{

int i2 = i1;//AutoUnBoxing

m1(i2);

}

public static void m1(Integer i2){//AutoBoxing

int k = i2;//AutoUnBoxing

System.out.println(k);//10

}

}

Compiler is responsible for conversion of primitive to wrapper and wrapper to  primitive using the concept of  "AutoBoxing and AutoUnBoxing".

case2:

class Test

{

static Integer i1;//i1 = null

public static void main(String[] args)

{

int i2 = i1;// int i2 = i1.intValue() :: NullPointerException System.out.println(i2);

}

}

Case3 :

Integer i1 = 10;//AutoBoxing

Integer i2 = i1;

i1++; = > i1 = i1+1

System.out.println(i1);

System.out.println(i2);

System.out.println(i1==i2);

Case4:

Integer x = new Integer(10);

Integer y = new Integer(10);

System.out.println(x == y);//false

Case5:

Integer x = new Integer(10);//memory from heap area

Integer y = 10;//AutoBoxing ===> Integer y = Integer.valueOf(10); System.out.println(x == y);//false

Case6:

Integer x = new Integer(10);

Integer y = x; ===> reference is reused so pointing to same object System.out.println(x == y);//true

Case7:

Integer x = 10;

Integer y = 10;

System.out.println(x == y);

Integer a = 100;

Integer b = 100;

System.out.println(a == b);

Integer i = 1000;

Integer j = 1000;

System.out.println(i == j);

In the remaining cases new object will be created.

// String/primtive to wrapper => valueOf()

// Wrapper type to primitive => xxxValue()

class Test

{

public static void main(String[] args)

{

Integer x = 128;

Integer y = 128;

System.out.println(x == y);//false

Integer a = 127;

Integer b = 127;

System.out.println(a == b);//true

Boolean b1 = true;

Boolean b2 = true;

System.out.println(b1==b2);//true

Double d1 = 10.0;

Double d2 = 10.0;

System.out.println(d1==d2);//false

}

}

valueOf()

String Primitive

public static Character valueOf(char);

String

Wrapper Object

parseXXX()

Wrapper Object

Wrapper Object

xxxValue()

Integer Byte Short

=>6 xxxValue()

Long

Float

Double

booleanValue()

charValue()

toString()

Primtivetype

Wrapper Object primtive

type

primitive

type

String

String

toString()

toString()

parseXXX()

Compiler will do the conversions automtically from JDK1.5Version

valueOf()

AutoBoxing(valueOf())

Primitive

xxxValue()

type

Wrapper

type

Primitive

type

AutoUnBoxing (xxxValue())

valueOf()

Integer il 10;

Integer 12 = il;

i1++;

System.out.println(i1); //11

System.out.println(12); //10

System.out.println(i1==i2);//false

Integer x = 10;

Integer (Immutable)

il

10

11

i2

Compiler uses "valueOf()" for AutoBoxing.

↑

Implemented in intelligent way in Wrapper classes

Buffer of Objects

At the time of loading the .class file jvm will create buffer of object to be used during AutoBoxing(range: -128 to +127)

Integer y = 10;

System.out.println(x == y); //true

Integer a 100;

Integer b= 100;

System.out.println(a == b);//true

Integer i 1000;

1000

Integer j = 1000;

System.out.println(i

== j);//false

-128

;

1000

-128 to +127

a (100 +127