***-:Interface and Abstract class:-***

1. Introduction

2. Interface declaration & implementation

3. Interface methods

4. Interface variables

5. Interface naming conflicts

i. method naming conflicts

ii. variable naming conflicts

7. Marker Interface

8. Adapter class

9. Interface vs abstract class vs Concrete class

10. Difference between interfaces and abstract class

11. Conclusion

**Introduction:**

Definition – 1:

Any service requirement specification (SRS) is considered as an interface.

Ex: JDBC API acts as requirement specification to develop data base driver.

Data base vender is responsible to implement this JDBC API.

Ex: Servlet API acts as requirement specification to develop web server. Webserver vender is responsible to implement servlet API.

Definition – 2:

From client point of view an interface defines the set of services what he is expecting. From service point of view an interface defines the set of services what he is offering. Hence any contract and service provider as an interface.

Ex: Through bank atm guy screen bank people are highlighting the set of services what they are providing at the same the same guy screen represents the set of services what customer is expecting. Hence this guy screen acts as contract between customer and bank people.

Definition – 3:

Inside interface every method is always abstract whether we declare or not. Hence interface is considered as 100% pure abstract class.

**Definition:**

Any service requirement specification or any contract between client and service provider or 100% pure abstract class in nothing but an interface.

**Interface declaration and implementation:**

Whenever we are implementing an interface for each and every method of that interface we have too provide implementation. Otherwise we have to declare class as abstract then next level child class is responsible to provide implementation.

Every interface is always public and abstract whether we are declaring or not. Hence whenever we are implementing an interface method compulsory we should declare as public otherwise we will get compiling time error.

Ex:

interface Intf{

void m1();

void m2();

}

**abstract** class Serviceprovider implements Intf{

**public** void m1(){

}

}

Class subservice extends Serviceprovider{

**public** void m2(){

}

}

**Extends vs implementation:**

A class can extend only one class at a time.

An interface can extend any number of interface at a time.

Ex:

interface A{

}

interface B{

}

interface C extends A , B{

}

A class can implement any number of interface simultaneously.

A class can extend another class and can implement any number of interface simultaneously.

Ex: class A extends B implements C , D , E{

}

Q. Which of the following is valid?

1. A class can extend any number of classes at a time.
2. A class can implement only one interface at a time.
3. An interface can extends only one interface at a time.
4. An interface can implement any number of interface simultaneously.
5. A class can extends another class or can implement an interface but not both simultaneously
6. Non of the above.

Answer : 6

Q. consider the following expression x extends y. for which of the following possibilities of x and y the above expression is valid?

1. Both x and y should be classes

2. Both x and y should be interfaces

3. Both x and y can be either class or interface // Correct

4. No restriction

Answer : 3

Q. x extends y, z

**🡪** x, y , z should be interface.

Q. x implements y, z

**🡪**  x should be class and y, z should be interface

Q. x extends z implements y

**🡪**  x, z should be class and Y should be interface

Q. X implements Y extends Z

**🡪** compile time error : we have to take extends first followed by interface.

**Interface methods:**

Every method present inside interface is always public and abstract whether we are declaring or not.

interface intf{

void m1();

}

The method is public abstract void m1();

Why public ?

To make the method available to every implementation class.

Why abstract?

Implementation class is responsible to provide implementation.

* Hence inside interface the following method declaration are equal.

void m1();

abstract void m1();

public void m1();

public abstract void m1();

As every interface method is always public and abstract we can’t declare interface method with the following modifiers.

private

protected

static

final

synchronized

strictfp

native

Q. Which of the following method declaration are allowed inside interface?

1. public void m1(){} // Invalid
2. private void m1(); // Invalid
3. protected void m1(); // Invalid
4. static void m1(); // Invalid
5. public abstract native void m1(); // Invalid
6. abstract public void m1(); // Valid

**Interface variables:**

An interface can contain variable. The main purpose of interface variable is to define requirement level constants.

Every interface variable is always public static final whether we are declaring or not.

public : To make this variable available to every implementation class.

static : Without existing object implementation class can access this variable.

final : If one implantation class change value then remaining implementation class will be affected. To restrict this every interface variable is always final.

Hence within the interface the following variable declaration are equal.

1. int x = 10
2. public int x = 10
3. static int x = 10
4. final int x = 10
5. public static int x = 10
6. public final int x = 10
7. static final int x = 10
8. public static final int x = 10;

As every interface is always public static final we can’t declare with the following modifier.

* private
* protected
* transient
* volatile

For interface variable compulsory we should preform initialization at the time of declaration otherwise we will get compile tome error.

Interface intf{

int x; // CE : = expected

}

Inside implementation class we can access interface variable but we can’t modify values.

Ex: Interface Intf{

int x = 10;

}

class Test implements Intf{

public static void main(String[] args){

x = 45; // CE: can’t assign a value to final variable

Sop(x);

}

}

But here we will not get any error

class Test implements Intf{

public static void main(String[] args){

int x = 45; // here x is local variable

Sop(x);

}

}

O/p: 45

**Interface naming conflicts:**

**Interface method naming conflicts:**

case:1

If two interface contains methods with same signature and same return type. Then in the implementation class we have to provide implementation for only one method.

Ex: interface Left{ interface Right{

public void m1(); public void m1();

} }

class Test implements Left , Right{

public void m1(){

}

}

Case 2:

If two interface contains a method with same name but different argument types then in the implantation class we haver to provide implementation for both method. These methods acts as overloaded methods.

Ex: interface Left{ interface Right{

public void m1(); public void m1(int i);

} }

class Test implements Left , Right{

public void m1(){

}

public void m1(int i){

}

}

Case 3:

If two interfaces contains a method with same signature but different return types then it is impossible to implement both interfaces simultaneously (if return types are not covariant ).

Ex: : interface Left{ interface Right{

public void m1(); public int m1();

} }

We can’t write any java class which implements both interfaces simultaneously.

**Interface variable naming conflicts:**

Two interfaces can contain a variable with same name. And there may be a chance of variable naming conflicts. But we can solve this problem by using interface names.

Ex: interface Left{ interface Right{

int x = 777; int x = 888;

} }

class Test implements Left , Right{

public static void main(String[] args){

// Sop(x); CE: reference to x is ambiguous

Sop(Left.x); // 777

Sop(Right.x); // 888

}

}

**Marker interface:**

The interface which doesn’t contain any method and by implementing that interface if our object will get some ability then it is called marker interface or ability interface or tag interface.

1. Serializable (I)
2. Cloneable (I)
3. RandomAccess (I)
4. SingleThreadModel (I)

These are marked for some ability.

Ex: By implementing serializable interface our object can be saved to the file and can travel across the network.

Ex: By implementing cloneable interface our objects in a position to produce exactly duplicate cloned objects.

Q. Without having any method how the objects will get ability in marker interface?

Ans: internally JVM is responsible to provide required ability.

Q. Why JVM is providing required ability in marker interface?

Ans: To reduce the complexity of the programming and to make java language as simple.

Q. Is it possible to create our own marker interface?

Ans: Yes, it is possible to create our own marker interface but customisation of JVM is required.

**Adapter classes:**

Adapter class is a simple java class that implements only empty implementation.

Ex:

interface x{

m1();

m2();

.

.

.

M1000();

}

Abstract class AdapterX{

m1(){ }

m2(){ }

….

m1000(){ }

}

If we implement an interface for each and every method of that interface we should provide implementation whether it is required or not.

Class Test implements x{

m3(){

-----

}

m1(){ }

m2(){ }

…

m1000() { }

}

The problem in this approach is it increases the length of the code and reduces readability we can solve this problem by using adapter classes.

Instead of implementing adapter classes we can extend adapter class we have to provide implementation only for required method and we are not responsible to provide implementation of every method of interface so that length of the code is going to reduce.

Class Test2 extends AdapterX{ **(** **x** 🡪 **AdapterX 🡪 Test2 )**

m3(){

…….

}

}

**Note:**

Marker interface and adapter class simplify complexity of programming and these are best utility to the programmer and programmer life will become simple.

**Interface vs Abstract class vs Concrete class :**

If we don’t know anything about implementation just we have requirement specification then we should go for interface.

Ex: Servlet(I)

If we are talking about implementation but not completely (partial implementation) Then we should go for abstract class.

Ex: Generic servlet, http servlet

If we are talking about complete implementation and ready to provide service then we should go for concrete class.

Ex: myOwnServlet

**Difference between interface and abstract class:**

|  |  |
| --- | --- |
| interface | abstract class |
| 1. If we don’t know anything about implementation and just we have requirement specification then we should go for interface. 2. Inside interface every method is always public and abstract whether we are declaring or not hence interface is considered as 100% pure abstract class. 3. As every interface method is always public and abstract and hence we can’t declare with the following modifiers: private, protected, final, static, native, synchronized and strictfp. 4. Every variable present inside interface is always public static final whether we are declaring or not. 5. As every interface variable is always public static final we can’t declare with the following modifiers: private, protected, volatile and transient. 6. For interface variable compulsory we should perform initialization at the time of declaration only otherwise we will get compile time error. 7. Inside interface we can’t declare static and instance block. 8. Inside interface we can’t declare constructor. | 1. If we are talking about implementation but not completely(partial implementation) then we should go for abstract class. 2. Every method present inside abstract class need not be public and abstract and we can concrete methods also. 3. There are no restriction on abstract class methods modifiers. 4. Every variable present inside abstract need not be public static final. 5. There are no restriction on abstract class variable modifier. 6. For abstract variable we are not required to perform initialization at the time of declaration. 7. Inside abstract we can declare static and instance block. 8. Inside abstract class we can declare constructor. |

Q. Anyway we can’t create object for abstract class but abstract class can contain constructor. What is the need?

Ans: Abstract class constructor will be executed whenever we are creating objects for child class for child class object initialization only.

Q. Any way we can’t create object for abstract class and interfaces but abstract can contain constructor but interface doesn’t contain constructor. What is the reason?

Ans: The main purpose of constructor is to perform initialization for the instance variable. Abstract class can contain instance variable which are required for child objects. To preform initialization of those instance variables constructor is required for abstract class.

But every variable present inside interface is always public static final whether we are declaring or not and there is no chance of existing instance variable inside interface. Hence constructor concept is not required for interface.

Whenever we are creating child class object parent object won’t be created just parent class constructor will be executed for the child object purpose only.

Q. Inside interface every method is always abstract and we can take only abstract method in abstract class also then what is the differences between interface and abstract class i.e. is it possible or not?

Ans: We can replace interface with abstract class but it is not a good programming practice. This something like recruiting IAS officer for swiping activity.

If everything is abstract it is highly recommended to go for interface but not for abstract class.

Abstract class: While extending abstract class it is not possible to extend any other class and hence we are missing inheritance benefits. In this case objects creation is costly.

Interfaces: while implementing interface we can extend other class and hence we won’t miss any inheritance benefits. Here object creation is not costly.