**Declarations and Access Modifiers – Part-14- Interfaces-04**

* 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.

Example: Servlet

* + If we are talking about implementation but not completely (partial implementation). Then we should go for Abstract Class.

Example: GenericServlet, HttpServlet.

* + If we are talking about implementation completely and ready to provide service, then we should go for concrete class.

Example: Myown servlet.

* + Diagram:

Interface Servlet Plan

Abstract Class GenericServlet Partially

HttpServlet completed

building

Concrete Class MyOwnSerlvet Fully

Completed

Building

* Differences between Interface and Abstract Class.

|  |  |  |
| --- | --- | --- |
| S.No | Interface | Abstract Class |
| 1 | If we don’t know anything about implementation and just we have requirement specification. Then we should go for interface. | If we are talking about implementation but not completely (partial implementation). Then we should go for Abstract Class. |
| 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. | Every method present inside abstract class need not be public and abstract and we can take concrete methods also. |
| 3 | As every interface method is always public and abstract and hence we can’t declare with the following modifiers (private, default, protected, final, static, synchronized, native and strictfp). | There are no restrictions on abstract class method modifiers. |
| 4 | Every variable present inside interface is always public static final, whether we are declaring or not. | Every variable present inside abstract class need not be public static final. |
| 5 | As every interface variable is always public static final. We can’t declare with the following modifiers. (private, protected, volatile, transient) | There are no restrictions on abstract class variable modifiers. |
| 6 | For interface variables compulsory we should perform initialization at the time of declaration only. Otherwise we will get compile time error. | For abstract class variables we are not required to perform initialization at the time of declaration. |
| 7 | Inside interface we can’t declare static and instance blocks | Inside abstract class we can declare static and instance blocks. |
| 8 | Inside interface we can’t declare constructors. | Inside abstract class we can declare constructor. |

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

Abstract class constructor will be executed whenever we are creating child class object to perform initialization of child class object.

Approach\_1 without having constructor in Abstract class.

abstract class Person{

String name;

Int age;

Total 100 properties;

}

class Student extends Person{

int rollNo;

Student(String name, int age, … 100 properties){

this.name=name

this.age=age

.

.

.

this.rollNo=rollNo;

}

}

Student student = new Student(101 properties);

class Teacher extends Person{

String subject;

Teacher (String name, int age, … 100 properties){

this.name=name

this.age=age

.

.

.

this. subject = subject;

}

}

Teacher teacher = new Teacher(101 properties);

Note: Problem with this code is more code and code is redundancy.

Approach\_2 with constructor inside abstract class.

abstract class Person{

String name;

Int age;

Total 100 properties;

Person(String name, int age, ….){

this.name=name;

this.name=age;

100 properties

}

}

Note: Above class will work for every child object creation.

class Student extends Person{

int rollNo;

Student(String name, int age, … 100 properties){

super(100 properties of parent class);

this.rollNo=rollNo;

}

}

Student student = new Student(101 properties);

class Teacher extends Person{

String subject;

Teacher (String name, int age, … 100 properties){

super(100 properties of parent class);

this. subject = subject;

}

}

Teacher teacher = new Teacher(101 properties);

Note: This concept clearly concludes that constructor is just for initializing the properties not to create an object. If its used for object creation, the code will fail whenever we call the super inside the child class.

Note:

Either directly or indirectly we can’t create object for abstract class).

* Anyway we can’t create objects for Abstract class and interface, but abstract class can contain constructor but interface doesn’t contain constructor. What is the reason?

The main purpose of constructor is to perform initialization of instance variables.

Abstract class can contain instance variables which are required for child object. To perform 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.

Example:

class P{

P(){

System.out.println(this.hashCode());

}

}

class C extends P{

C(){

System.out.println(this.hashCode());

}

}

class Test(){

public static void main(String[] args){

C c = new C();

System.out.println(c.hashCode());

}

}

Output: whenever we execut this class, it will always return the same hashcode at all the three sysouts. This clears and confirms that whenever we create a child object, the parent constructor is called but it doesn’t create an object for parent class.

* Inside interface every method is always abstract and we can take only abstract methods in abstract class also, then what is the difference between interface and abstract class, that is, is it possible to replace interface with abstract class?

We can replace interface with Abstract class, but it is not a good programming practice.

This is something like recruiting IAS officer for sweeping activity.

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

Approach\_1: Approach\_2:

abstract class X{ interface X{

} }

class Test extends X{ class Test implements X{

} }

While extending abstract class it is not possible to extend any other class and hence we are missing inheritance benefit.

While implementing interface we can extend some other class and hence we won’t miss any inheritance benefit.

In the approach\_1 object creation is costly. As it has to call/execute the parent constructors.

Example:

Test t = new Test();

2mins

In the approach\_2 object creation is not costly.

Test t = new Test();

2 secs