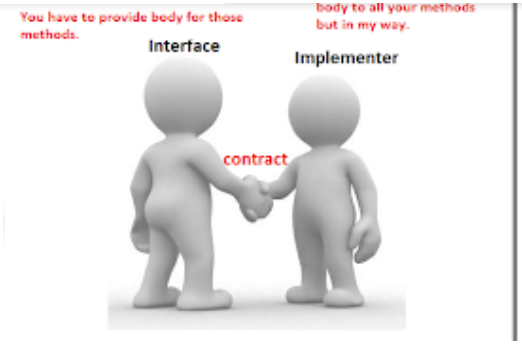
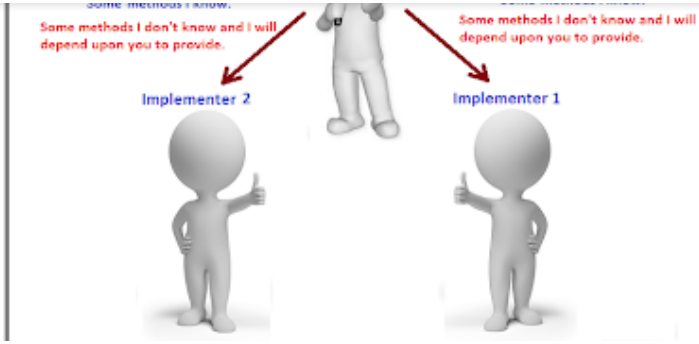


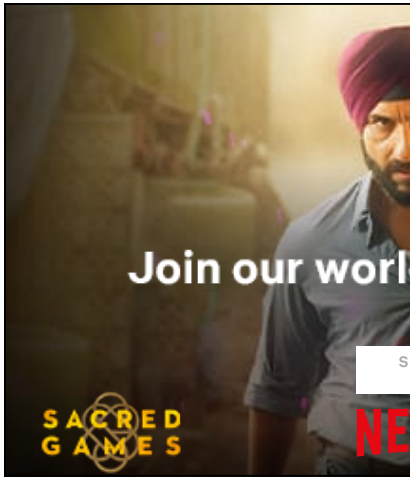


interface Vs abstract class in Java.

Interface:

Visit us:

> *Interface is used when you want to define a contract and you don't know anything about implementation. (here it is total abstraction as you don't know anything.)*



Abstract class:

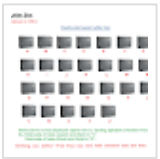
> *Abstract class is used when you know something and rely on others for what you don't know.(here it is partial abstraction as some of the things you know and some you don't know.)*

Any questions/feedback, Please drop a mail at jayeshmaheshpatel@gmail.com

CATEGORIES

Algorithm Angular2 AngularJS Array Backtracking Binary Search Binary Search Tree Binary Tree Bit Manipulation Database Datastructure Design Pattern Graph Heap Interviews Java Java8 Javascript Linked List Matrix Miscellaneous Multithreading OOPS Puzzle Quartz Queue Sorting Spring Stack Stream Strings Typescript Web Service Windows

POPULAR POSTS



How time complexity of Hashmap get() and put() operation is O(1)? Is it O(1) in any condition?

How time complexity of Hashmap get() and put() operation...

What is Load factor and Rehashing in Hashmap?

Now, Let's understand above difference between Interface and Abstract class with real world project example.

When to use Interface

Scenario,

Consider we want to start a service like "makemytrip.com" or "expedia.com", where we are responsible for displaying the flights from various flight service company and place an order from customer.

Lets keep our service as simple as,

1. Displaying flights available from vendors like "airasia", "british airways" and "emirates".
2. Place and order for seat to respective vendor.

How should we design our application considering interfaces and abstract class? In this scenario, interface is useful or abstract class?

Visit us:

Remember, In this application, we don't own any flight. we are just a middle man/aggregator and our task is to first enquire "airasia", then enquire "british airways" and at last enquire "emirates" about the list of flights available and later if customer opts for booking then inform the respective flight vendor to do booking.

For this, first we need to tell "airasia", "british airways" and "emirates" to give us list of flights, internally how they are giving the list that we don't care.

1. **This means I only care for method "getAllAvailableFlights()"**

"getAllAvailableFlights()" from "airasia" may have used SOAP service to return list of flights.
"getAllAvailableFlights()" from "british airways" may have used REST service to return list of flights.
"getAllAvailableFlights()" from "emirates" may have used CORBA service to return list of flights.

Convert integers to roman numerals equivalent in Java

Converting Integers to Roman Numerals equivalent in Java ...

How ConcurrentHashMap works and ConcurrentHashMap interview questions.

ConcurrentHashMap Interview Questions In Java. ...

Find all subsets of a set (Power Set)

Print power set of any given set OR Print all subsets of...

RECENT COMMENTS

2. Similarly, for booking I only care for method "**booking()**" that all vendors should have, internally how this vendors are doing booking that I don't care.

To conclude: We know contract.

So we can say that we know the contract that irrespective of who the Flight vendor is, we need "**getAllAvailableFlights()**" and "**booking()**" method from them to run our aggregator service.

> *In this situation, Interface is useful because we are not aware of the implementation of all the 2 methods required, and what we know is the contract methods that vendor(implementer) should provide. so due to this total abstraction and for defining the contract, interface is useful in this place.*

Visit us:

Technically, we need to design our interface somewhat like below,

FlightOpeartions.java(Contract)

```
interface FlightOpeartions{
 void getAllAvailableFlights();
 void booking(BookingObject bookingObj);
}
```

BookingObject.java

```
class BookingObject{}
```

BritishAirways.java (Vendor 1)

```
class BritishAirways implements FlightOpeartions{

```

```
    }

    public void booking(BookingObject flightDetails){
            //place booking order in a way British airways
            //told us to place order for seat.
    }

}
```

Emirates.java (Vendor 2)

```
class Emirates implements FlightOpeartions{

    public void getAllAvailableFlights(){
            //get Emirates flights in the way
            //they told us to fetch flight details.
    }

    public void booking(BookingObject flightDetails){
            //place booking order in a way Emirates airways
            //told us to place order for seat.
    }
}
```

Visit us:

When to use Abstract class

Scenario,

Consider we want to start a service like Bulk SMS sender, where we take orders from various telecom vendors like Airtel, France Telecom, Vodafone etc.

For this, we don't have to setup our own infrastructure for sending SMS like Mobile towers but we need to take care of government rules like after 9PM, we should not send promotional SMS, we should also not send SMS to users registered under Do Not Disturb(DND) service etc. Remember, we need to take care of government rules for all the countries where we are sending SMS.

Note: *for infrastructure like towers, we will be relying on vendor who is going to give us order.*

Example, In case of,

Vodafone request us for bulk messaging, in that case we will use Vodafine towers to send SMS.

So what all methods we require would be somewhat like below,

```
1  public void eastablishConnectionWithYourTower(){
2     //connect using vendor way.
3     //we don't know how, candidate for abstract method
4  }
5
6  public void sendSMS(){
7     eastablishConnectionWithYourTower();
8     checkForDND();
9     checkForTelecomRules();
10    //sending SMS to numbers...numbers.
11    destroyConnectionWithYourTower()
12 }
13
14 public void destroyConnectionWithYourTower(){
15    //disconnect using vendor way.
16    //we don't know how, candidate for abstract method
17 }
18
19 public void checkForDND(){
20    //check for number present in DND.
21 }
22
23 public void checkForTelecomRules(){
24    //Check for telecom rules.
25 }
26
```


Out of above 5 methods,

1. Methods we know is "sendSMS()", "checkForDND()", "checkForTelecomRules()".
2. Methods we don't know is "eastablishConnectionWithYourTower()", "destroyConnectionWithYourTower()".

we know how to check government rules for sending SMS as that is what our job is but
we don't how to eastablish connection with tower and how to destroy connection with tower because this is purely customer specific, airtel has its own way, vodafone has its own way etc.

So in the given scenario, we know some methods but there also exist some methods which are unknown and depends

In this case, what will be helpful, abstarct class or interface?

> *In this case, Abstract class will be helpful, because you know partial things like "checkForDND()", "checkForTelecomRules()" for sending sms to users but we don't know how to eastablishConnectionWithTower() and destroyConnectionWithTower() and need to depend on vendor specific way to connect and destroy connection from their towers.*

Let's see how our class will look like,


```
abstract class SMSSender{

 abstract public void eastablishConnectionWithYourTower();

 public void sendSMS(){
  /*eastablishConnectionWithYourTower();
  checkForDND();
  checkForTelecomRules();

  sending SMS to numbers...numbers.*/
 }

 abstract public void destroyConnectionWithYourTower();

 public void checkForDND(){
  //check for number present in DND.
 }
 public void checkForTelecomRules(){
  //Check for telecom rules
 }
}


class Vodafone extends SMSSender{

 @Override
 public void eastablishConnectionWithYourTower() {
```

```
31    @Override
32    public void destroyConnectionWithYourTower() {
33     //destroying connection using Vodafone way
34    }
35
36   }
37
38   class Airtel extends SMSSender{
39
40    @Override
41    public void eastablishConnectionWithYourTower() {
42     //connecting using Airtel way
43    }
44
45    @Override
46    public void destroyConnectionWithYourTower() {
47     //destroying connection using Airtel way
48    }
49
50   }
```


So to summarize,

For Interface:

Interface is used when you don't know anything about implementation but know the contract that implementer should have to accomplish the task.

For Abstract class:

Abstract class is used when you know partial implementation, where say out of 5 methods, you know implementation of 3 methods and don't know implemenatation of 2 methods in that case 2 methods will be abstract and you need to rely on implementer as a contract to must provide body of abstract methods to accomplish the task.

You may also like to see