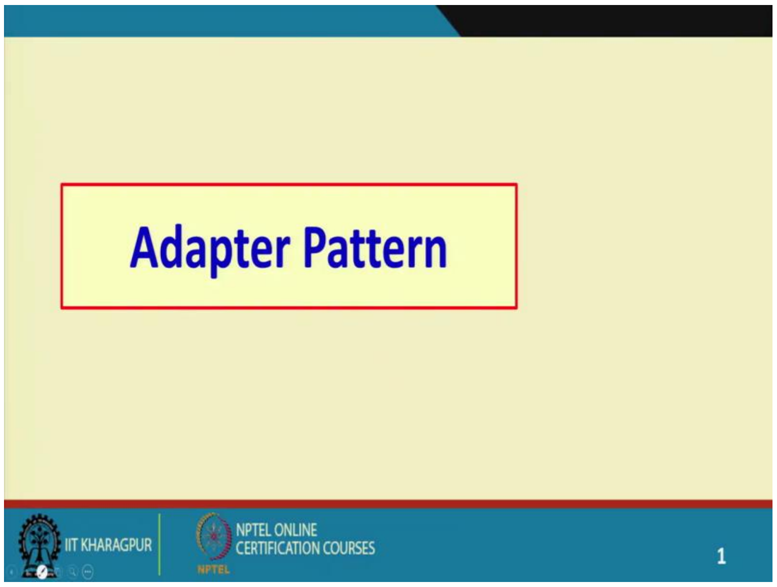
Object-Oriented System Development Using UML, Java and Patterns

Professor. Rajib Mall
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur
Lecture No. 51
Adapter Pattern - 1

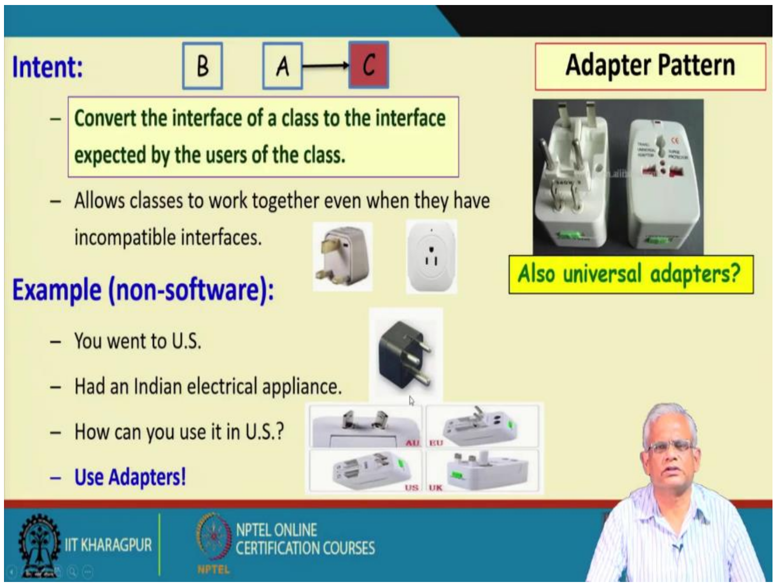
Welcome to this session! In the last session, if you remember, we had discussed about the Adapter Pattern. We just started to discuss the adapter pattern and we look at the details of the adapter pattern in this lecture.

(Refer Slide Time: 00:35)



Adapter Pattern

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
1



Intent:

- Convert the interface of a class to the interface expected by the users of the class.
- Allows classes to work together even when they have incompatible interfaces.

Example (non-software):

- You went to U.S.
- Had an Indian electrical appliance.
- How can you use it in U.S.?
- Use Adapters!

Adapter Pattern

Also universal adapters?

B
A
C
AU
EU
US
UK
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES

The intent of the adapter pattern is to, as is written here, convert the interface of a class to the interface expected by the users of the class. Let me just explain the meaning of this. Let us say we had a Class A, which was invoking the services of class B. Now, the class B had some methods, let us say P, Q, R etc. So, A used to invoke services of B by calling the methods P, Q and R. But then we had another class C, which we wanted to use, we choose a better class in terms of performance, functionality and so on. But then the method available here are different. B had P, Q, R, but C had L, M, N and so on.

Now, we want A to work with C, without changing the code of A. How do we achieve this? We need to use an adapter. So, we had A, Class A using the services of class B. But now we want to use the services of class C and as it is it would be difficult to use the services of class C, because we do not want to change the code of class A and C has methods L, M, N and A was calling the methods P, Q, R.

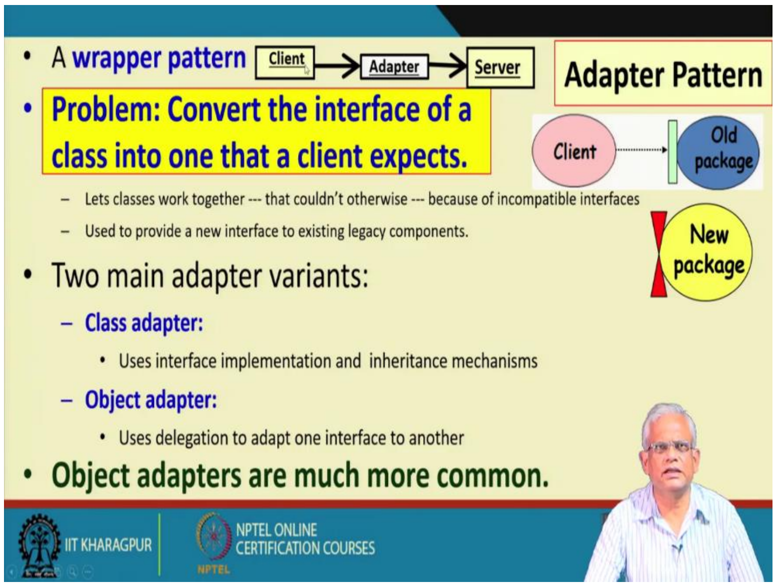
In a first glance they look like incompatible classes, A is calling P, Q, R and C is providing L, M, N. The adapter pattern, the intent is to make these two incompatible classes to work together meaningfully. Let me just give a non-software example to make the adapter class, role of the adapter class clear. Let us say we visited a country and we had a mobile charger, which we were using in India. This is the charger as shown in the figure above and we were using the charger in India, where the socket was different.

Now how do we make this work with the country that we visited? Let us say USA has this kind of a socket as shown in the figure above. How do we make our mobile charger which needs a socket of different type to work with the socket here in USA? We need an adapter. The adapter fits in here and this can sit under adapter and we can make it work. But what about we visited different countries like let us say Australia, UK, European Union and so on. They all have different sockets and we have this charger, mobile charger.

Now how do we get our mobile charged, we need different types of sockets one for Australia, another for European Union, one for the United States, another for UK but those are too many. If you visited dozen country, you would have to carry dozen adapters. It is too cumbersome. But can we have a universal adapter that will simplify the problem. In fact, that is what everybody does, when going abroad, they carrier universal adapter. It can fit into the socket of any country and here, your appliance can use this socket.

The role of the adapter class is very similar to what happens here in the non-software situation, where we are using an electrical device and trying to plug it into the socket of a country.

(Refer Slide Time: 06:04)


Adapter Pattern

- A wrapper pattern
- **Problem: Convert the interface of a class into one that a client expects.**
  - Lets classes work together --- that couldn't otherwise --- because of incompatible interfaces
  - Used to provide a new interface to existing legacy components.
- Two main adapter variants:
  - Class adapter:
    - Uses interface implementation and inheritance mechanisms
  - Object adapter:
    - Uses delegation to adapt one interface to another
- **Object adapters are much more common.**

The adapter pattern is actually a wrapper pattern; we call it is as a wrapper pattern. Because here, the adapter is an object which sits around the class that we are trying to use, the object that we are trying to use, it is wrapper pattern. There are many wrapper patterns where an object, the client uses the object like adapter and the adapter internally passes on the request to another object, that is a wrapper pattern and here the problem that the adapter pattern tries to address is convert the interface of a class into one that the client expects.

The client class uses the interface P, Q, R and the server is having the interface L, M, N and the adapter will convert P, Q, R into element. When P is called, the adapter will internally make a call to L, method L makes classes work together; without adapter they cannot work, like you went to USA, did not have an adapter and you cannot really charge your phone, your phone gets discharged, but you cannot help, because of the incompatible interfaces and also, this pattern is very useful to provide a new interface to existing legacy components. That is we have some old packages, classes that make call to some other classes. Now we have got some new components available and making these old components work with that we need adapters.

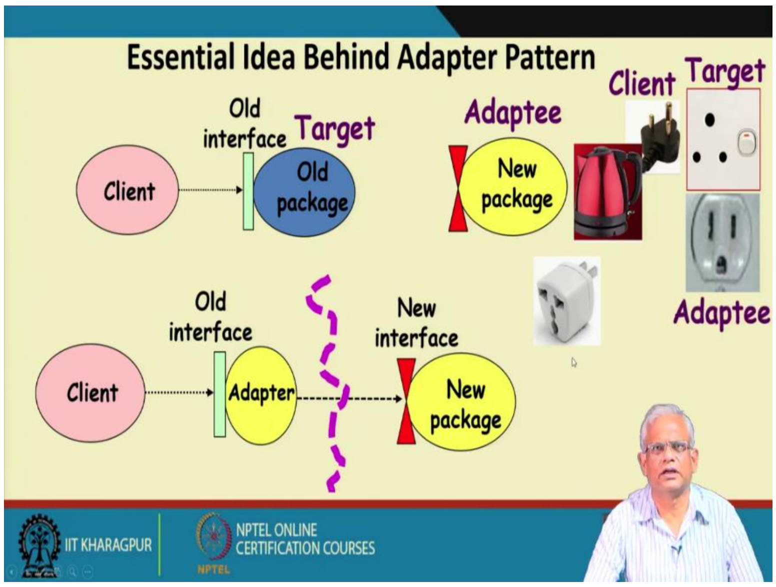
So, it is a very important pattern useful in software maintenance, extensively used and also whenever we want to work to classes having different interfaces. If we look at the nitty gritty of the adapter pattern, we will see that there are two variants of the adapter pattern we will discuss about both the variants of the adapter pattern, one is called as the class adapter and the other is called as the object adapter.

In the class adapter, the main mechanism that is used is inheritance and interface implementation. There are two main mechanisms used; implementing an interface and using inheritance. On the other hand, the object adapter here the main idea used is delegation and also implementation of interface. As we will see that the object adapters are much more common, request you to listen carefully from now on and identify what is the advantage of object adapters, we will discuss about both these. Please look at the structure of the class adapter, object adapter, their working and so on and please identify why the object adapters are much more common. We will address that question at the end of this lecture.

As we have already said, the client used to use some package. This is the interface of the package and that used to support methods like P, Q, R. But then we have a new package available, which is much more efficient, better and so on. But then the methods supported are L M N, we want to make the client work with a new package without changing the code of the client. That is, whenever the client makes a call to P, or Q or R, a call has to be made to the new package with L M N and that is what will the adapter do.

We can think of the adapter is a wrap around on the server. Whenever the client makes a call to the adapter, using P, the adapter makes a call to the server using L, if the client makes a call using Q, adapter makes call to the server with M and so on. So basically, it translates the client requests into the one expected by the server.

(Refer Slide Time: 11:50)



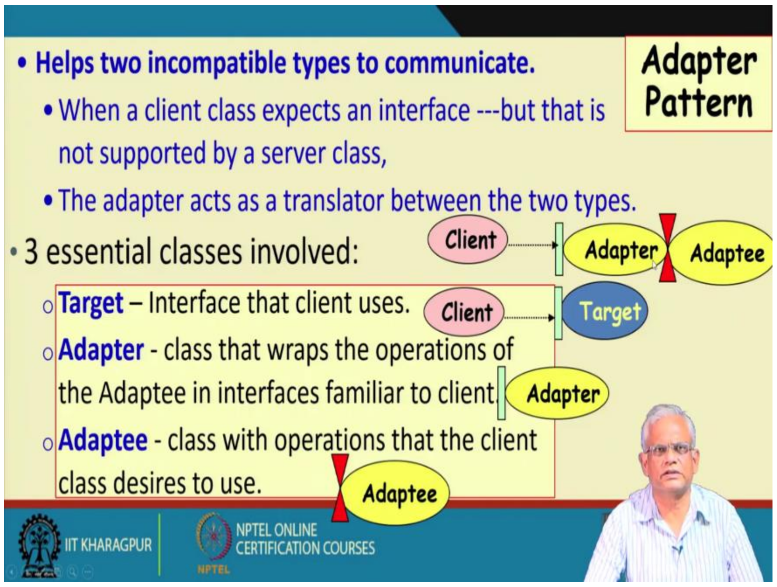
Essential Idea Behind Adapter Pattern

Now let us understand the main idea behind the adapter pattern and also let us get used to the terminology used. The interface that the client is used to which is accustomed to the interface, we call it as the target and then recall the new package as adaptee and then we have the adapter which supports or it has the old interface, the targets interface, but then it makes call to the new interface.

So, let us understand these terms target, which is the interface that the client used earlier. The adaptee is the interface of the new package and the adapter has the interface of the old package, but it makes call to the new interface, that is the adaptee's interface. Now, let us try to visualise that in terms of the non-software example. So, please identify what do we call this? Is it a target adaptee or adapter?

This is the old interface, you have gone to USA with this tea kettle and trying to us there and we call this as the target, this is the target interface and then we went to USA found that this is the kind of socket available. So, what do we call it, we call it as the adaptee and then we have the adapter which has the interface, target interface, but it makes call to the adaptee interface. It sits on the adaptee interface and then the tea kettle. The plug here fits in, so this is the terminology the target, the adaptee and the adapter. So, the client uses the target interface and then this is the adaptee and this is the adapter.

(Refer Slide Time: 14:48)



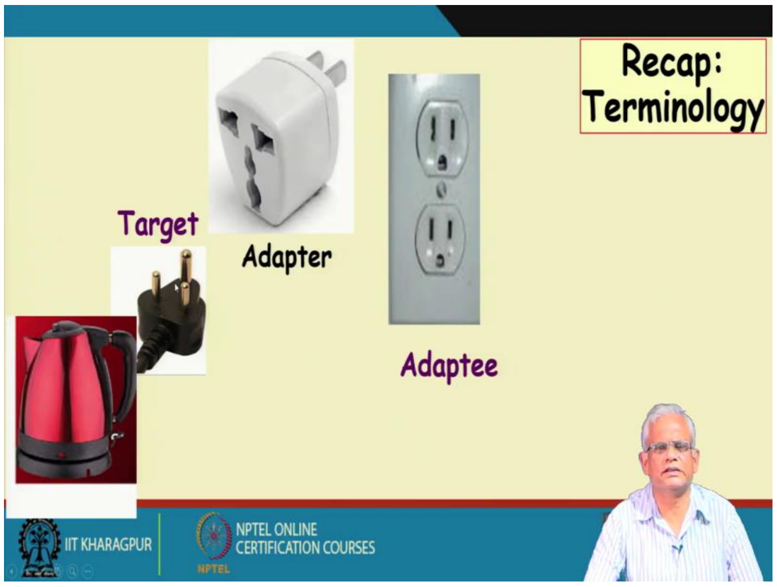
Adapter Pattern

- **Helps two incompatible types to communicate.**
  - When a client class expects an interface ---but that is not supported by a server class,
  - The adapter acts as a translator between the two types.
- 3 essential classes involved:
  - **Target** – Interface that client uses.
  - **Adapter** - class that wraps the operations of the Adaptee in interfaces familiar to client.
  - **Adaptee** - class with operations that the client class desires to use.

The adapter pattern as we have already mentioned, helps two incompatible types to communicate when the client expects some interface, and that is not supported by the server class, the adapter comes into picture. It acts as the translator between the two types and 3 main classes involved in the class structure of the adapter pattern. One is the target. This is the interface class that the client uses.

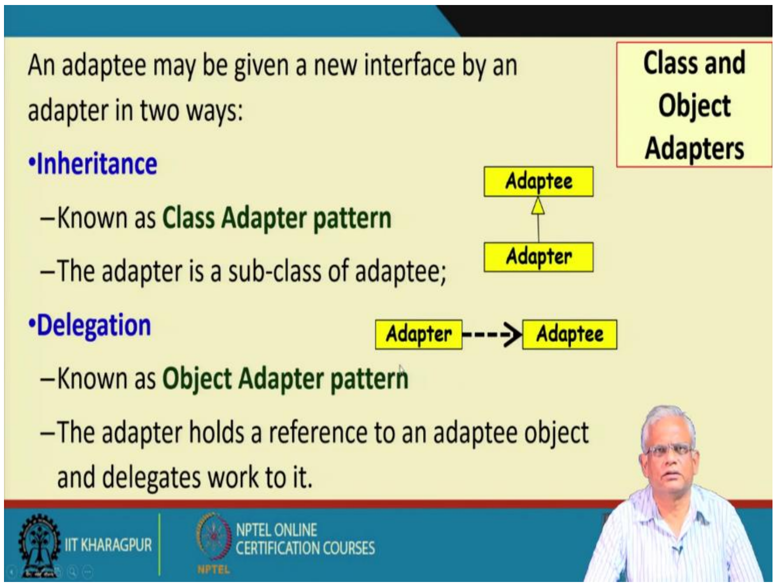
So, this is the target interface and then the adapter is the class that has the, that implements the target interface and makes call on the adaptee interface and adaptee is the class with operations that the client wants to use. So, the client makes call to the adapter, the adapter has the target interface and then it makes call to the adaptees interface.

(Refer Slide Time: 16:07)



Now please recap the terminology. What do we call this, the non-software example, this is the client expecting the target interface. But then we have the adaptee here and we need the adapter which has the target interface, but then can make call or sits on the adaptee interface. So, this is the adapter, this is the adaptee and this is the target. Now with this knowledge, let us look at the exact working of the adapter pattern.

(Refer Slide Time: 17:07)



Class and Object Adapters

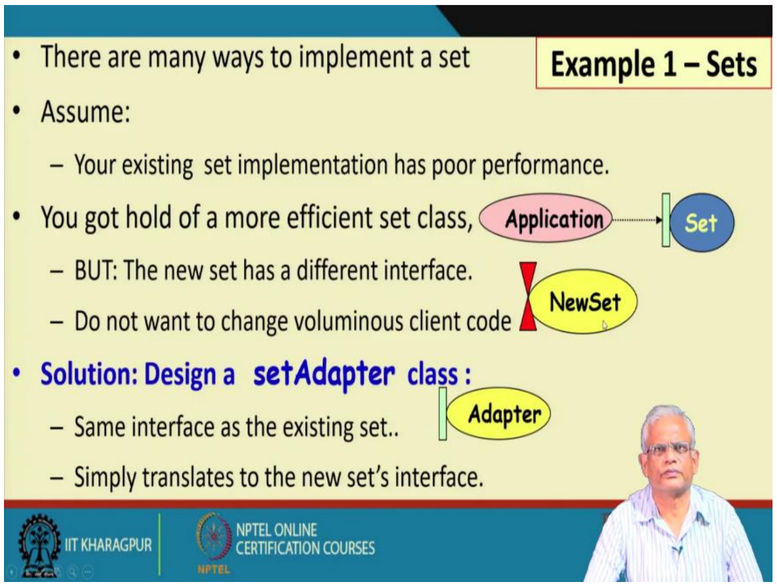
An adaptee may be given a new interface by an adapter in two ways:

- **Inheritance**
  - Known as **Class Adapter pattern**
  - The adapter is a sub-class of adaptee;
- **Delegation**
  - Known as **Object Adapter pattern**
  - The adapter holds a reference to an adaptee object and delegates work to it.

Adaptee
Adapter
Adapter
Adaptee
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES

As you were saying that there are two main types of adapter pattern, one is the class adapter and the other is object adapter. In the class adapter, inheritance is the main mechanism used. Here, the adapter is a subclass of the adaptee and also the adapter implements the target interface. So, this is the class adapter pattern and in the object adapter pattern delegation is the main idea here and the adapter holds a reference to the adaptee and makes a call to the adaptee based on the reference that it has and the adapter implements the target interface. So, this the object adapter pattern or the main idea of the object adapter pattern and this is the main idea behind the class adapter pattern.

(Refer Slide Time: 18:19)



Example 1 – Sets

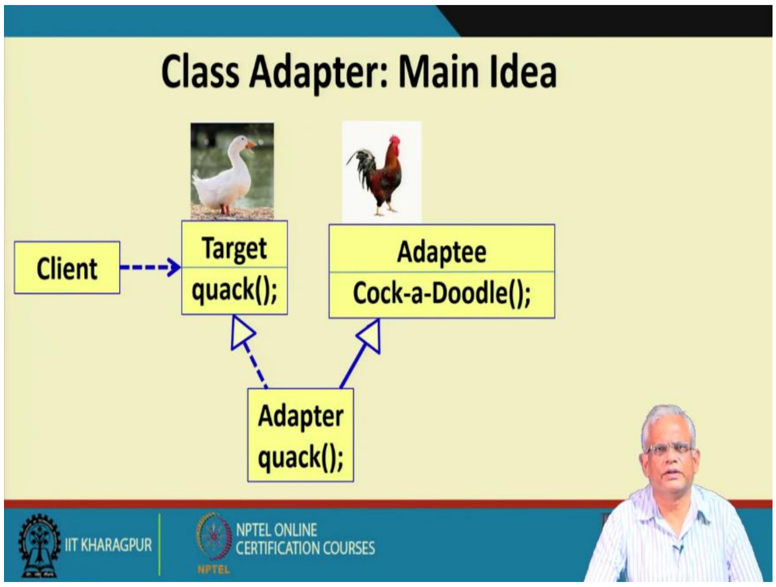
- There are many ways to implement a set
- Assume:
  - Your existing set implementation has poor performance.
- You got hold of a more efficient set class,
  - BUT: The new set has a different interface.
  - Do not want to change voluminous client code
- **Solution: Design a setAdapter class :**
  - Same interface as the existing set..
  - Simply translates to the new set's interface.

Application
Set
NewSet
Adapter
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES

Now let us try to explain the working of this pattern. Using a simple example. Let us say we have a class set. It supports various operations on a set, add, delete, etcetera. Now, let us say our class for set implementation has poor performance. So, this is the set class that we are using. It had some interest. But then the set in class was not really very efficient and we got hold of a more efficient set class, but it has a different interface. So, this is the new set class having a different interface.

Now, we do not want to change the application code, but we want to make it work with the new set. How do we do it? We use the set adapter class. It has the same interface as the target. But then it translates any call to made on the target interface into the new sets interface.

(Refer Slide Time: 19:40)

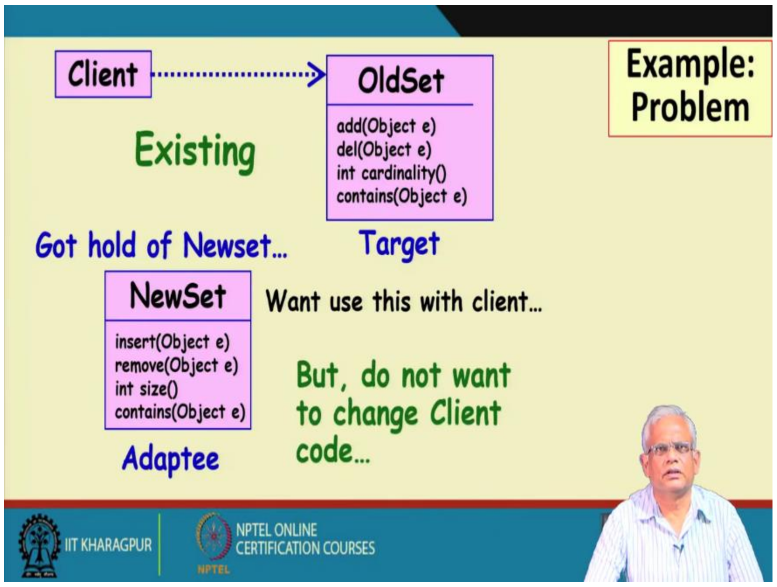


Class Adapter: Main Idea

Just to explain the class adapter pattern, let us say we have a target which use to support the method quack. The client class used to call the method quack. But then we have another different class we want to use, which is the adaptee and the method here supported is Cock-a-Doodle. The solution to this is that we have to write this adapter class. The adapter class implements the target interface and is a subclass of the adaptee. So, by being a subclass of the adaptee, it has the method Cock-a-Doodle available here, internally and it interface, it implements the target interface.

So, for the quack method here, whenever the client calls the quack method on the adapter, it will internally call the Cock-a-doodle method of the adaptee. In the object adapter, on the other hand, the adapter implements the target interface that is quack. But then it holds a reference that is this adaptee and the adapter are associated, the adapter holds a reference to the adaptee object and whenever the client calls the quack on the adapter, it calls the Cock-a-Doodle on the reference of the adaptee that it internally stores. So, these are the two main ideas in the adapter pattern, the class adapter and the object adapter.

(Refer Slide Time: 21:56)



Example: Problem

Client

OldSet
add(Object e)
del(Object e)
int cardinality()
contains(Object e)

Existing

Target

Got hold of Newset...

NewSet
insert(Object e)
remove(Object e)
int size()
contains(Object e)

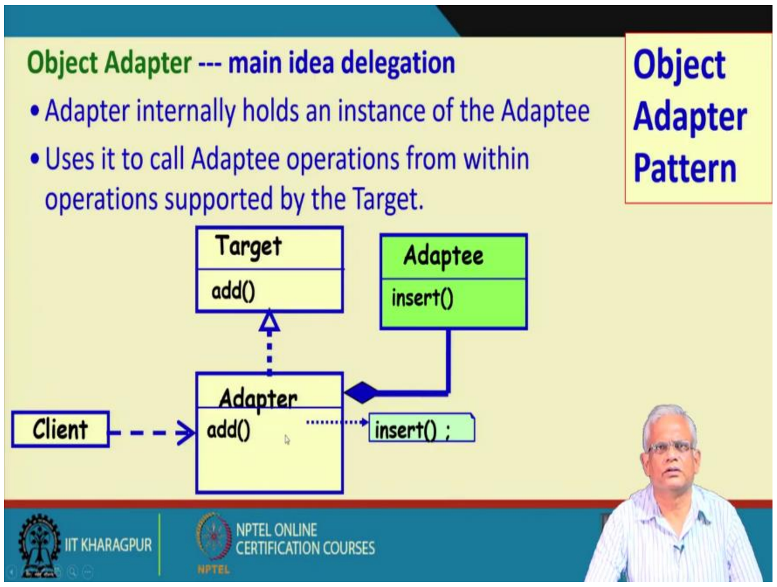
Adaptee

Want use this with client...

But, do not want to change Client code...

Now let us proceed with our example that we are discussing about the old set, the old set class had, the methods add, delete, cardinality of the set and whether it contains an object. So, that is the target, this is the target interface, add, delete, cardinality and contains. But then we got hold of the new set, which is much more efficient. But instead of add, delete, cardinality and contains, it has the methods insert, remove, size and contains. We do not want to change the code of the client and we want the client to make to work with the new set and therefore, we will have to write the adapter.

(Refer Slide Time: 23:01)



Object Adapter Pattern

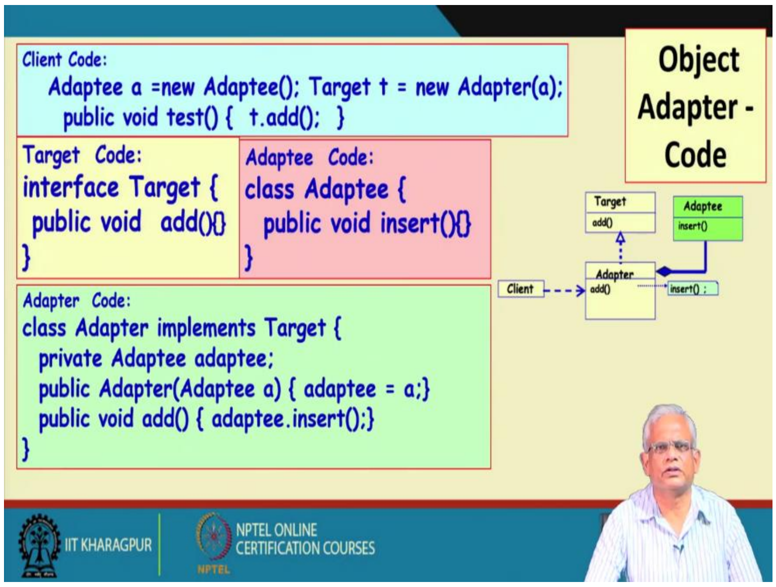
Object Adapter --- main idea delegation

- Adapter internally holds an instance of the Adaptee
- Uses it to call Adaptee operations from within operations supported by the Target.

In the object adapter pattern, we just store a reference to an instance of the adaptee class, we first create an adaptee object and store the reference in the adapter class and whenever there is a call to the adapter, it just calls the corresponding method of the adaptee object. So, this is the class diagram. The adapter implements the target interface. So, for the add, we need to provide an implementation and the implementation of add, we will just write insert on the adaptee reference, it holds a reference to the adaptee.

We can, show this as the association or aggregation. Whenever there is a call to add by the client in the implementation of the adapter will make the call insert on the reference of the adaptee class that internally stored.

(Refer Slide Time: 24:18)



Object Adapter - Code

Client Code:

```
Adaptee a =new Adaptee(); Target t = new Adapter(a);
  public void test() {  t.add();  }
```

Target Code:

```
interface Target {
 public void  add(){}
}
```

Adaptee Code:

```
class Adaptee {
  public void insert(){}
}
```

Adapter Code:

```
class Adapter implements Target {
  private Adaptee adaptee;
  public Adapter(Adaptee a) { adaptee = a;}
  public void add() { adaptee.insert();}
}
```

Let us see the code here. The client side we first create the adaptee object. So, A is the adaptee object and then we create the adapter and we pass the reference of an adapter, so that internally the adapter can store the reference and then in the test method, we call t.add and check whether the corresponding insert method of the adaptee is called or not. The target code is just interface target and public void add.

The adaptee code is class adaptee and public word insert and not shown the exact code, but just an overall idea of how it works. The target interface as add and the adaptee has the method insert and the adapter implements target and it has a reference to the adaptee object and in the constructor, it takes a reference to the adaptee object and store it internally and then, whenever the client calls the add, it internally calls adaptee.insert.

If we have understood this example, then we know more or less everything about the object adapter pattern. We are almost at the end of this lecture. In the next lecture, we will see the same example the, set and we will try to have the class diagram for the class adapter and also we will see the Java code for that. Thank you.