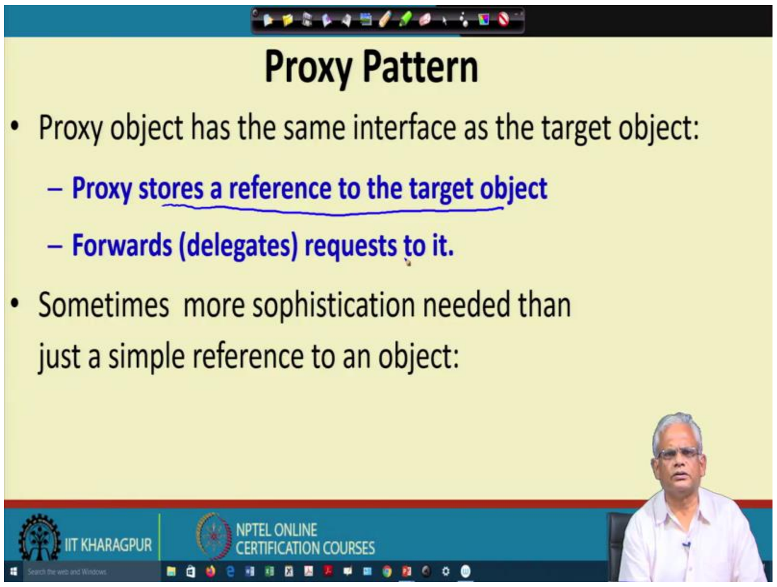
Object Oriented System Development Using UML, JAVA and Patterns

Professor Rajib Mall
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur
Lecture 55
Proxy Pattern I

Welcome to this lecture! In the last lecture, we had just started the Proxy Pattern, we had said that the proxy design pattern is a very important design pattern and very widely used in almost every application will have opportunity to use the proxy pattern. Now, let us look at the details of the proxy pattern last time. The last lecture looked at only the basic overview of the pattern. Let us now look at the details of this pattern.

(Refer Slide Time: 00:51)

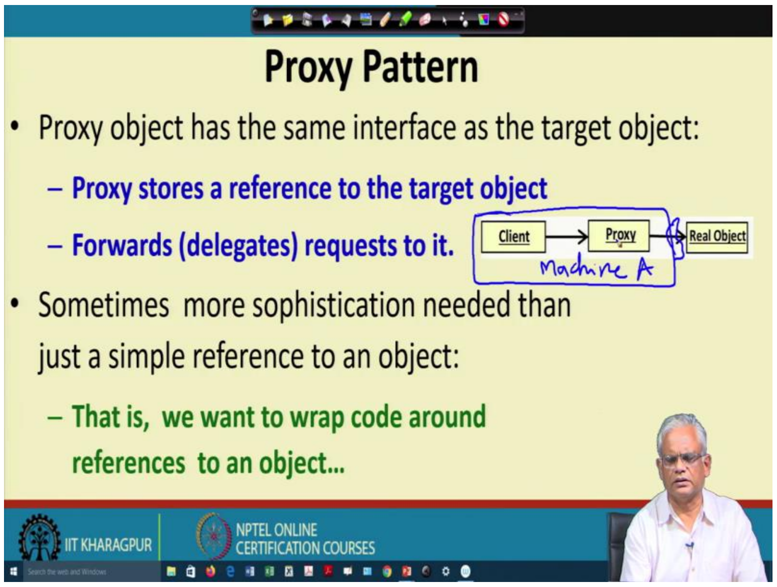


Proxy Pattern

- Proxy object has the same interface as the target object:
  - Proxy stores a reference to the target object
  - Forwards (delegates) requests to it.
- Sometimes more sophistication needed than just a simple reference to an object:

The proxy pattern, here the proxy has the same interface as the target object. The client uses the proxy and the proxy stores a reference to the target object. The client creates the proxy and interacts with the proxy. For the client, the proxy is actually as good as the target object. The client does not interact with the target object; it interacts with the proxy. But the proxy stores a reference to the target object. And any method invocation by the client and the proxy, the proxy handles it if it can otherwise it forwards the delegates to the target object or the server. Sometimes, it does not just forward it or just make the method invocation and the server object, but also has its own wrap around the code. It executes some code before calling the method of the target object. And also, after the target object method completes, it might have some code to execute. In that way, the proxy might enhance the capability of the server object.

(Refer Slide Time: 02:50)



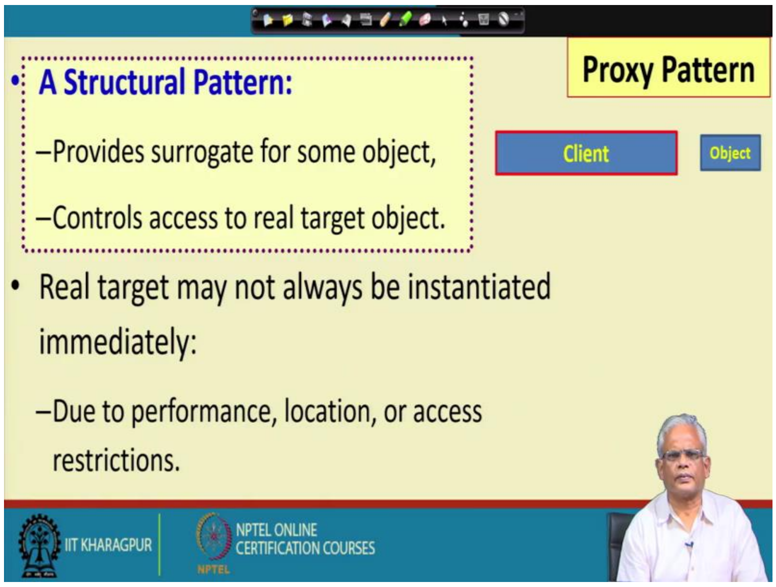
Proxy Pattern

- Proxy object has the same interface as the target object:
  - Proxy stores a reference to the target object
  - Forwards (delegates) requests to it.
- Sometimes more sophistication needed than just a simple reference to an object:
  - That is, we want to wrap code around references to an object...

So here, the proxy might have some wraparound code under method of the target object, you can see in this diagram that the client interacts with a proxy. For the client, the proxy is as good as the client, the client has no idea or no knowledge of the real object. It interacts with a proxy thinking that, that is the real object, the proxy in turn, for some simple queries that it can answer, it responds to the client.

For the queries that it cannot answer, it forwards it to the real object. But one thing is that in many times in a network situation, the proxy is in the same machine as the client. And the real object is across the internet in some other machine. So, just drawing a rectangle over here, saying that it is on the same machine, on the same machine both client and proxy objects reside. The real object is on a different machine and possibly across the network. For this reason, we call the proxy object as the local sit in for the real object just like in a classroom scenario, a student who does not come to the class, he sends a sit in, who sits in the class and behaves just like the real student, and then contacts the real students, if there is some important announcement, we can consider this as an analogy to that the proxies the local sit in for the real object and contacts the real object when required.

(Refer Slide Time: 05:40)



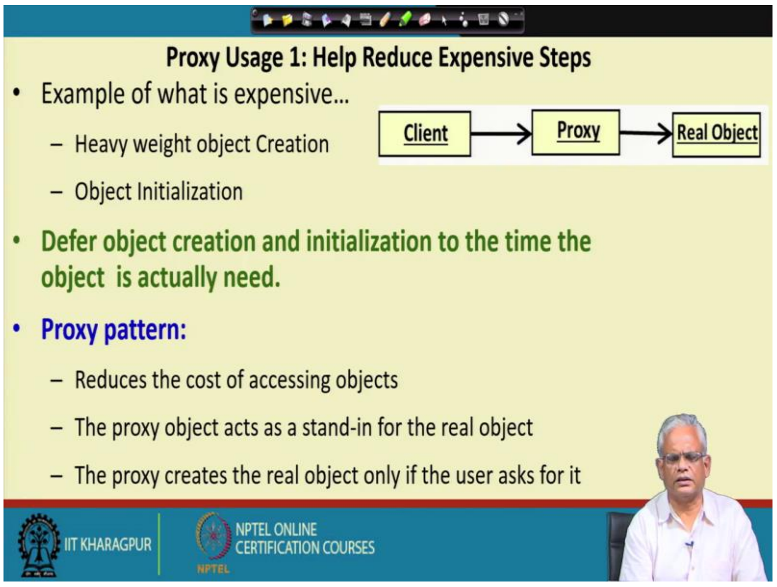
Proxy Pattern

- A Structural Pattern:
  - Provides surrogate for some object,
  - Controls access to real target object.
- Real target may not always be instantiated immediately:
  - Due to performance, location, or access restrictions.

Client
Object
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES

It is just structural pattern. And the proxy is a surrogate for the server surrogate we can consider as a duplicate for some object, and in some way, it controls the access of the client to the real object, it tries to answer some of the queries and only if required, under some conditions it forwards to the real object, otherwise, it does not forward. And also, many times the server object may not have been instantiated. Only the proxy exists, the client interacts with the proxy, and only when the proxy cannot respond to the client method call. It needs to delegate the method to the real object then possibly it might create the real object. That helps in many ways, because the real object need not be created, unless the client gives some specific requests for which the real object needs to be involved. And that helps in performance.

(Refer Slide Time: 07:15)



Proxy Usage 1: Help Reduce Expensive Steps

- Example of what is expensive…
  - Heavy weight object Creation
  - Object Initialization
- Defer object creation and initialization to the time the object is actually need.
- Proxy pattern:
  - Reduces the cost of accessing objects
  - The proxy object acts as a stand-in for the real object
  - The proxy creates the real object only if the user asks for it

$$\begin{tikzcd} \boxed{\text{Client}} \arrow[r] & \boxed{\text{Proxy}} \arrow[r] & \boxed{\text{Real Object}} \end{tikzcd}$$

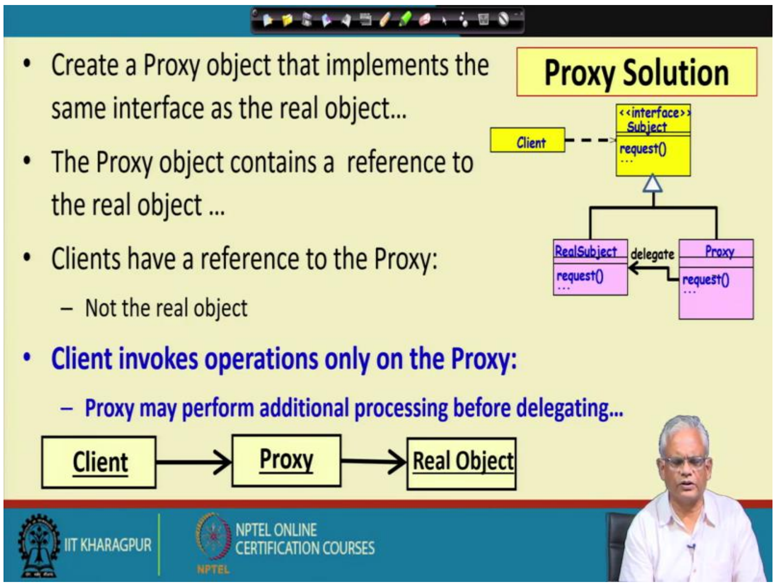
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

The proxy as we were discussing, has many uses, almost a dozen or more uses. The first use is that we discuss here is that it speeds up an application by reducing some expensive steps. By expensive, we mean creation of objects with a lot of data and initialization of those objects. The proxy creates these expensive objects only when it is absolutely necessary. And many of these expensive objects do not get created at all. If we did not use the (proxy objects) proxy pattern, we would have to create all those objects and slowing down the application considerably.

With the help of some examples, we will see how it does what are the steps involved. But the proxy differs object creation and initialization to the time the object is actually needed. The server objects are not created at the start of the application. But these are created and initialised only when these are actually needed. And therefore, we can say the proxy pattern reduces the cost of accessing the object. And the proxy acts as a stand in for the real object.

So, this is the same diagram where the client interacts only with the proxy. And the real object gets created only when it is absolutely necessary and if the proxy can handle the method call by itself. It does not need to delegate any responsibility to the real object and did not even create the real object.

(Refer Slide Time: 09:43)



Proxy Solution

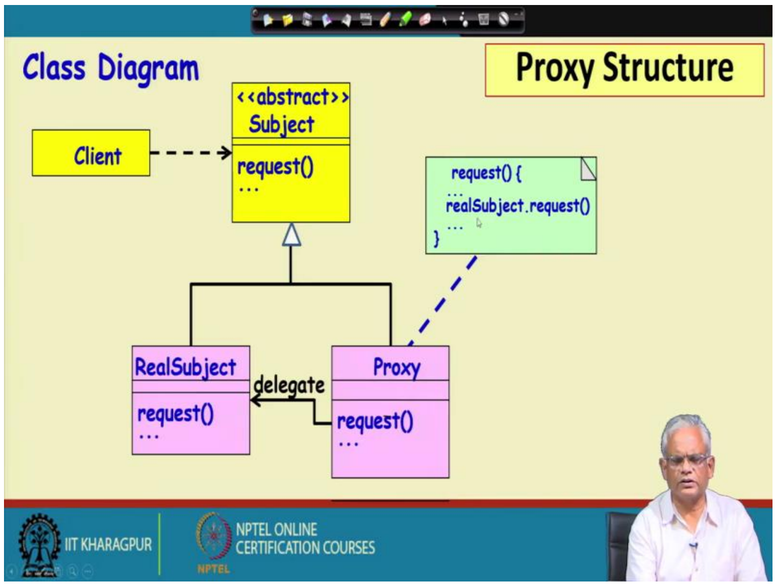
- Create a Proxy object that implements the same interface as the real object...
- The Proxy object contains a reference to the real object ...
- Clients have a reference to the Proxy:
  - Not the real object
- **Client invokes operations only on the Proxy:**
  - **Proxy may perform additional processing before delegating...**

In the proxy pattern, the client creates the proxy object and the proxy object has the same interface as the real object. As far as the client is concerned, there is no difference between the proxy and the real object, they are the same. And the proxy contains a reference to the real object. The clients interact only with the proxy and not the real object. If we draw the class structure, it would look like this that the client interacts only with the proxy.

The client interacts only with the proxy and both the real subject and the proxy they implement the same interface they have written here the subject interface. And the client interacts with the proxy, which handles some of the requests. And if it needs to delegates, it has the local reference of the real subject anyway. And it will delegate the call to the real subject.

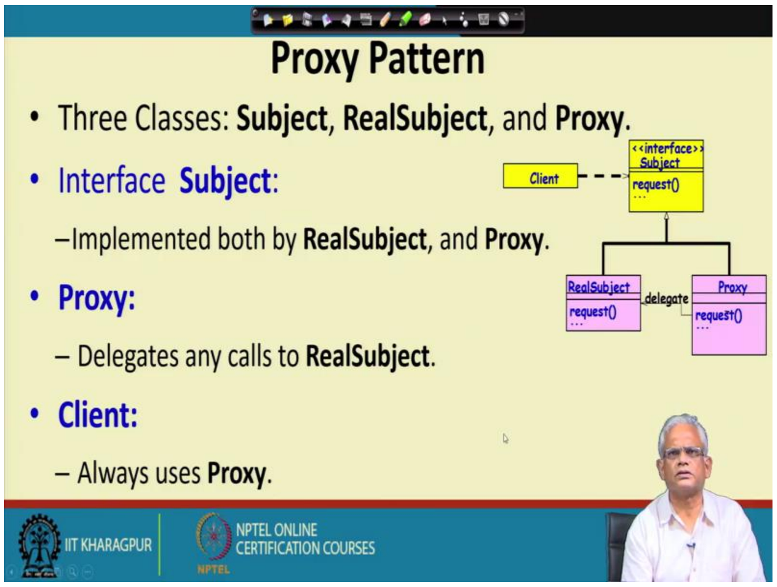
So, this is basically simple class diagram for the proxy pattern. As we had already mentioned, the proxy may perform some steps, some processing before it forwards a request to the real object.

(Refer Slide Time: 11:42)



The same diagram simple class structure of the proxy pattern. The client interacts with the proxy, the proxy and the real subject. Both implement the abstract subject interface. And whenever the client invokes a method call, the proxy may delegate to the real subject.

(Refer Slide Time: 12:20)



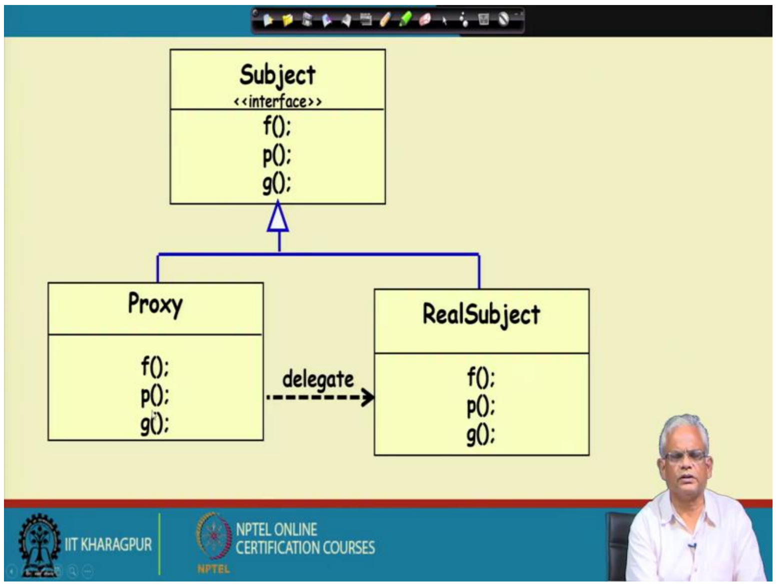
Proxy Pattern

- Three Classes: **Subject**, **RealSubject**, and **Proxy**.
- Interface **Subject**:
  - –Implemented both by **RealSubject**, and **Proxy**.
- **Proxy:**
  - – Delegates any calls to **RealSubject**.
- **Client:**
  - – Always uses **Proxy**.

Client
<<interface>>
Subject
request()
RealSubject
request()
delegate
Proxy
request()
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES

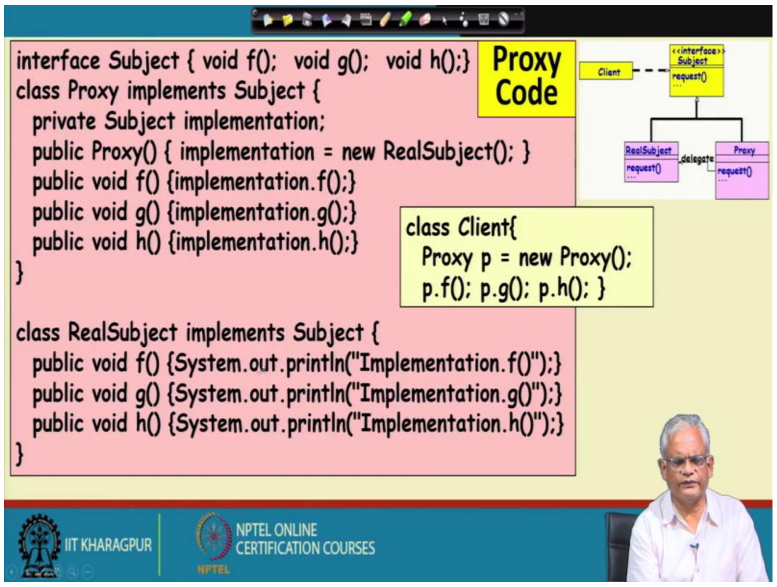
Only three classes involved here; the subject, the real subject, and the proxy that is the pattern. And the subject is an interface. And that interface is implemented by both real subject and proxy. The proxy keeps a reference of the real subject and delegates any calls to the real subject if needed. And the client interacts only with the proxy that is the is one of the very simplest patterns we have discussed.

(Refer Slide Time: 13:02)



If the subject has the methods f, p, g, then the proxy and the real subject implement the same methods is the subject is an interface f, p, g, are the methods here, and the proxy and the real subject they implement these methods f, p, g. And the proxy also stores a reference to the real subject and delegates any method invocation that it does not handle by itself.

(Refer Slide Time: 13:44)



Proxy Code

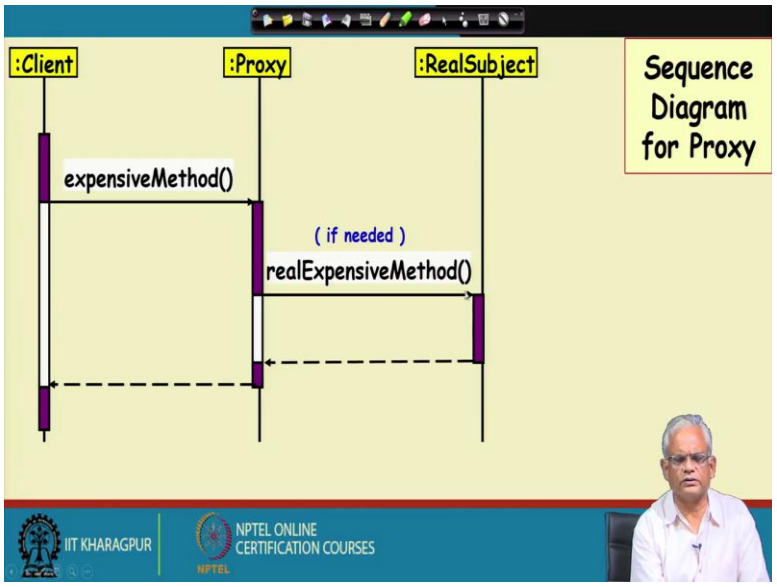
```
interface Subject { void f();  void g();  void h();}
class Proxy implements Subject {
  private Subject implementation;
  public Proxy() { implementation = new RealSubject(); }
  public void f() {implementation.f();}
  public void g() {implementation.g();}
  public void h() {implementation.h();}
}

class RealSubject implements Subject {
  public void f() {System.out.println("Implementation.f()");}
  public void g() {System.out.println("Implementation.g()");}
  public void h() {System.out.println("Implementation.h()");}
}
```

```
class Client{
  Proxy p = new Proxy();
  p.f(); p.g(); p.h(); }
```

Now, let us look at the code. The client, this is the client code. The client interacts with the proxy and the client creates the proxy. And then invokes all its operations on the proxy. The interface subject has the methods f, g, h. The proxy implements the subject interface. And also, it keeps a reference of the real subject written your implementation, but that is the real subject. And during the creation of the proxy on the constructor itself the real subject is created. But if the real subject is a heavies object, computationally expensive, then it may not be created here, it will be created only when it is required. We will see those examples a little later. But here for simplicity, in the constructor of the proxy, the real subject is created and the reference is stored in the variable implementation.

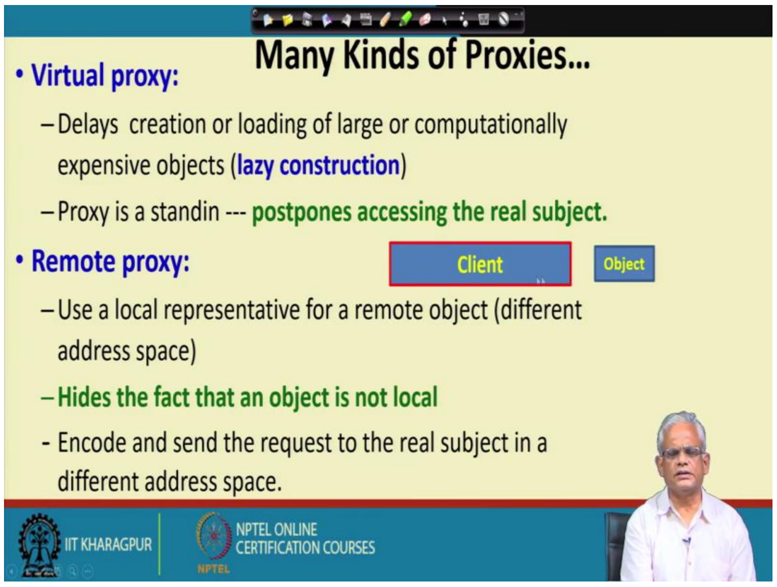
And it implements the methods f, g, h and the implementation as you can see, it just invokes the server the corresponding method on the server. The server ID is stored in the local variable implementation. And the real subject also implements the subject interface. Here, we have the definition for f, g, h and here is just a simple definition here, just print f, g, h.

(Refer Slide Time: 16:06)



The sequence diagram, the client makes invocations and proxy for simple methods, the proxy handles and for the methods for which it cannot handle it delegates to the real subject.

(Refer Slide Time: 16:36)



Many Kinds of Proxies...

- Virtual proxy:
  - Delays creation or loading of large or computationally expensive objects (**lazy construction**)
  - Proxy is a standin --- **postpones accessing the real subject.**
- Remote proxy:
  - Use a local representative for a remote object (different address space)
  - **Hides the fact that an object is not local**
  - Encode and send the request to the real subject in a different address space.

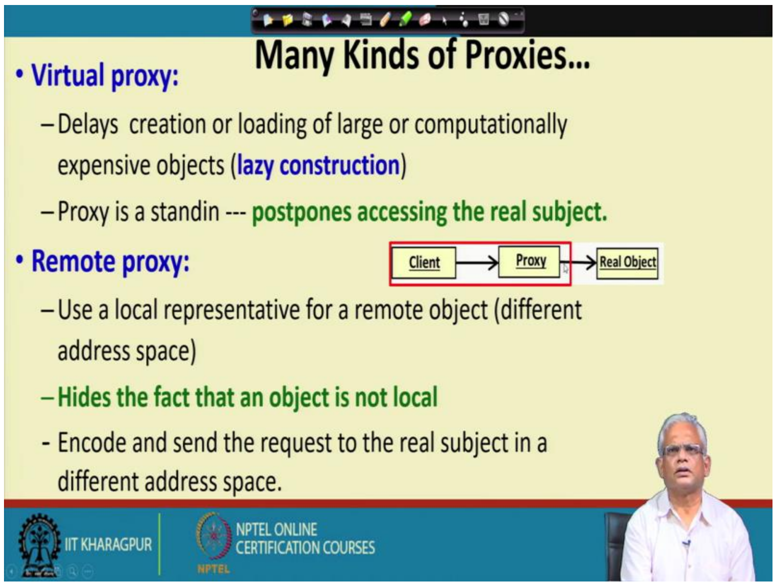
Now, let us see the different types of proxies. The proxies, as we were saying has many different types of application. I was saying that more than a dozen applications will discuss few important types. But that does not restrict all the applications. There are many other applications. The first application that we will discuss is the virtual proxy.

And then we look at the remote proxy. The virtual proxy, the main idea here is lazy construction of the server object. The virtual proxy helps to speed up an application. The computationally heavy objects are not created, unless they are actually needed. And when they are needed, due to some client interaction at that point they are created.

So, the main idea behind the virtual proxy is to speed up the application through lazy construction. And the idea is the same proxies the local stand in, and it postpones successing the real subject. Unless it is absolutely required, it does not invoke the method on the real subject, the remote proxy here the client and the proxy they reside on the same machine.

And the proxy is the local stand in for the real object. The client does not feel that the server is on another machine it behaves, it interacts with the proxy, thinking that it is on the same machine. And therefore, use of a remote proxy hides the fact that the real object is not a local object, but the client, it gives a feeling to the client that it is on the local machine. And whenever a client invokes a method under proxy, the proxy encodes creates a request packets and sends across the network to the real object.

(Refer Slide Time: 19:15)



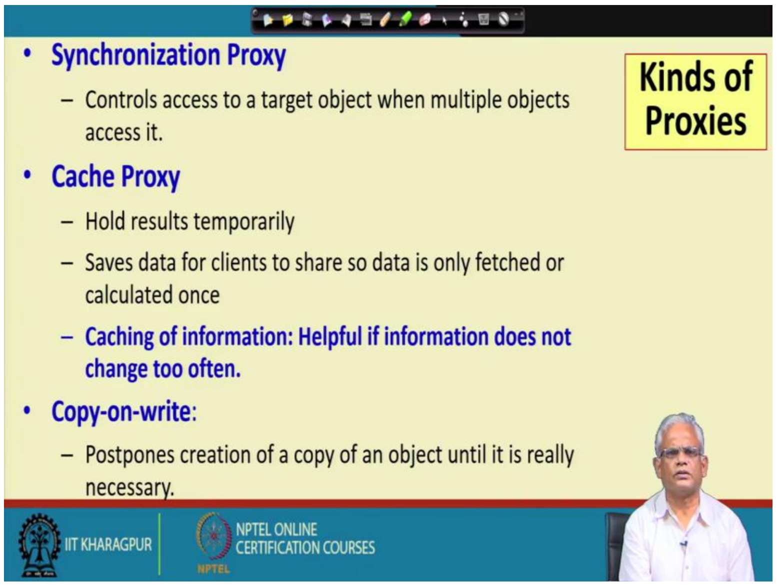
Many Kinds of Proxies...

- **Virtual proxy:**
  - Delays creation or loading of large or computationally expensive objects (**lazy construction**)
  - Proxy is a standin --- **postpones accessing the real subject.**
- **Remote proxy:**
  - Use a local representative for a remote object (different address space)
  - **Hides the fact that an object is not local**
  - Encode and send the request to the real subject in a different address space.

So, the client, as you can see, interacts only with the proxy they are on the same machine. Proxy is the local stand in for the real object. The client does not even know that the real object is somewhere in the network thinks that the real object is on the same machine is a programming convenience.

The client interacts with the proxy, the proxy, in turn interacts with the real object across the network by following the basic network protocols, encoding the request contacting the real object wherever whenever required.

(Refer Slide Time: 20:07)



Kinds of Proxies

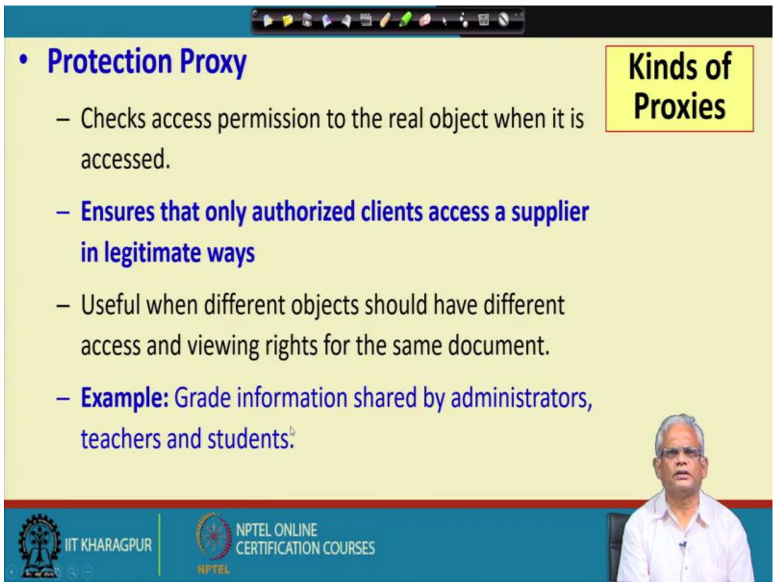
- **Synchronization Proxy**
  - Controls access to a target object when multiple objects access it.
- **Cache Proxy**
  - Hold results temporarily
  - Saves data for clients to share so data is only fetched or calculated once
  - **Caching of information: Helpful if information does not change too often.**
- **Copy-on-write:**
  - Postpones creation of a copy of an object until it is really necessary.

There are other types of proxy, the synchronization proxy. In the synchronization proxy, when there are multiple objects that can access server object. The proxy controls the access of the server object that is who is next going to access. Basically, here, the synchronization proxy sequentializes concurrent access by multiple clients to a server object.

The cache proxy, this is another application of the proxy pattern here the role of the proxy that it holds the results temporarily i.e., it caches information and therefore, if multiple clients need the same information, the proxy fetches the data only once just like in a network proxy. The proxy pattern, the proxy object fetches the data once and performs the necessary calculation only once and as different objects request it shares this data. As you can imagine the cache proxy application here the traffic on the network is reduced. And also the response time of the client is improved.

Copy-on-write this is another application of the proxy. Here if there is an object, a server object, it holds some information which many clients need. Then the same server object is shared among the multiple clients. But if one of the client tries to change the data on the server object, then it is given a separate server object a separate server object is created and given to it. And therefore, it postpones creation of an object until it is really necessary.

(Refer Slide Time: 23:01)



Kinds of Proxies

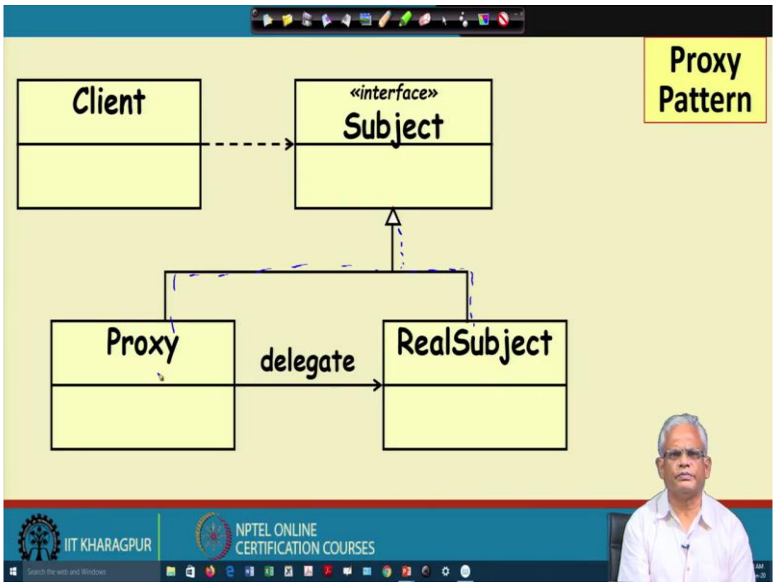
- **Protection Proxy**
  - Checks access permission to the real object when it is accessed.
  - **Ensures that only authorized clients access a supplier in legitimate ways**
  - Useful when different objects should have different access and viewing rights for the same document.
  - **Example:** Grade information shared by administrators, teachers and students.

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES

The protection proxy this is another application. Here when multiple clients access to a server object, the client permission is checked. And only those methods of the server which they can access, they are allowed to access. So, the role of the proxy here that it checks access permission of the clients, and then it allows them to invoke those methods for which they are permitted.

It ensures that only the authorised clients access a supplier in legitimate ways. The protection proxy is useful when there are different client objects who have different access and viewing rights. One example of the protection proxy is let us say the student grade information is accessible by different clients, the administrator, teacher and students. The students have only read permission under grade information they can only look at their grade. The teacher can post the grades change the grades and so on. And the administrator also can change and view the grades.

(Refer Slide Time: 24:46)



Let us look at this diagram carefully. This is basically the proxy pattern. If we understand this diagram well, we can apply this pattern to one of the many occasions on which we can apply. We just saw that some 7-8 different types of proxies or different contexts in which the proxy is used.

The diagram itself is very simple that the client interacts only with the proxy, the proxy and the real subject they implement the subject interface, okay I should have used the implementation notation here rather than the inheritance notation, I should have used the dotted because subject is the interface.

The proxy and the real subject implement the subject interface. The proxy stores the reference to the real subject, and when the client requests the method which it cannot serve by itself it delegates that to the real subject, and provides the answer to the client. We are almost at the end of this lecture. We will stop here. And we will look at some more examples of proxy pattern in the next lecture. Thank you.