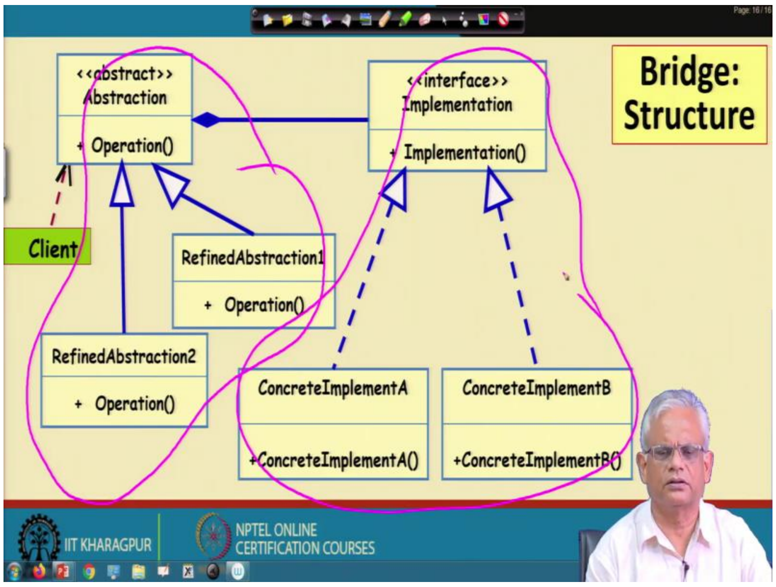
Object Oriented System Development Using UML, JAVA and Patterns
Professor Rajib Mall
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur
Lecture 54
Bridge Pattern II

Welcome to this lecture! In the last lecture, we were discussing about the Bridge Pattern very frequently used. The programmers doing object oriented application development very frequently it was the bridge pattern, very popular pattern solves an important problem the mistake that the novice programmers do wherever there is any change request they just use the inheritance to define a new class has its problems on the unless we are careful.

It can bind the abstraction concepts to the implementation concepts and make the application extremely complex and there will be a rapid increase in the number of classes in the application. And therefore, everybody who is concerned with object-oriented software development must know the bridge pattern well. It helps to avoid some of the major problems in object orientation in specific violation to the single responsibility principle or the SRP. Now, let us recollect the class structure of the bridge pattern.

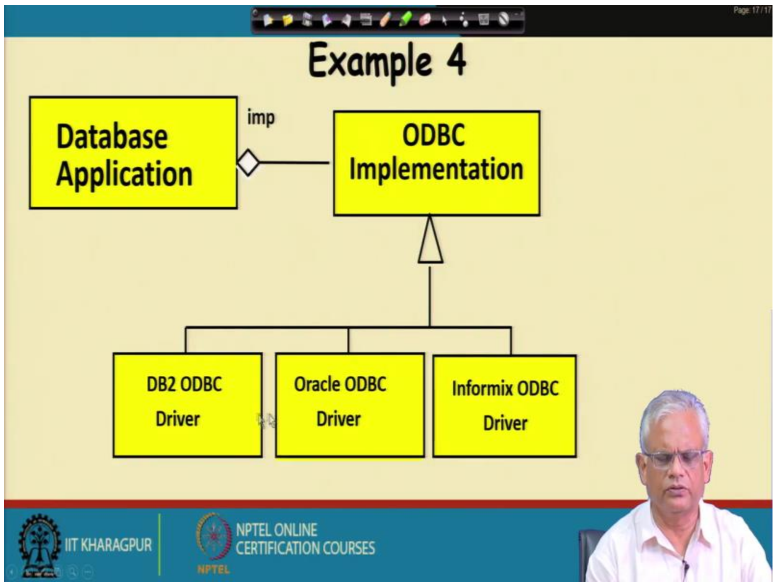
(Refer Slide Time: 02:04)



The class structure of the bridge pattern, we have the abstraction hierarchy where we have the abstraction as abstract class and then we have the concrete abstractions and these are typically named from the application domain terms. The client invokes services and the abstraction and the operations may be fulfilled by this abstraction class, concrete abstraction class itself.

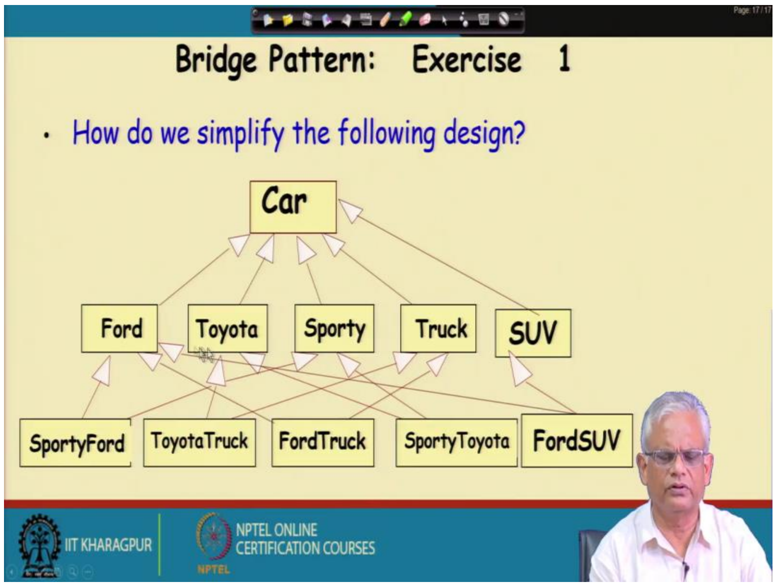
And if necessary, it will delegate to the concrete implementation class and this the implementation hierarchy and this is the general structure of the bridge pattern. Now, let us try to solve one more problem and see how we can apply the bridge pattern.

(Refer Slide Time: 03:02)



This is already there in most of the applications, if you try to understand the application code, you will see that every database application where we have the application level classes, they have a bridge to the ODBC implementation, the ODBC implementation is sub-classed into various types of databases. The DB2, Oracle, Informix and so on. And this helps us to separate out the implementation hierarchy from the application hierarchy.

(Refer Slide Time: 03:46)


Bridge Pattern: Exercise 1

- How do we simplify the following design?

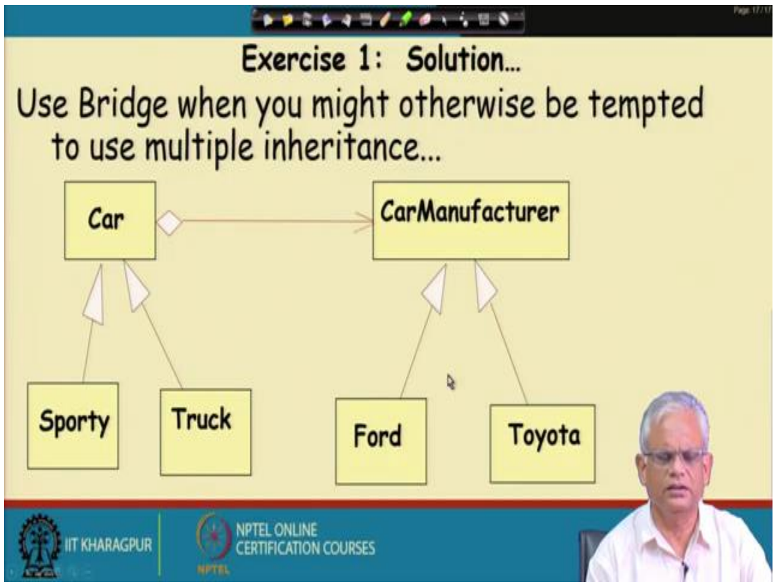
Now, please try to do one exercise; let us say a programmer was employed in a vehicle sales showroom and then he was trying to have this application developed to keep track of various vehicles that the showroom deals with and then how many units are sold and so on. His main class hierarchy that was used in the application looked like this, that the car are different types; Ford car, Toyota car, sports car and a truck is a special type of car. The SUV is also a special type of car. And then we have the Ford and the Sporty model. The Toyota truck which is subclass from the truck and the Toyota, the Ford truck, Ford and truck and the Sporty Toyota which is just a subclass of the Toyota and the sporty; the Ford SUV which is a subclass of SUV and Ford.

And as the number of models increased and became much messier, very complex violates some principles, object-oriented principles make the design extremely complex to understand and maintain. Now, this is the candidate for application of the bridge pattern to simplify the design. But the problem that we are posing before you is that how do we exactly apply the bridge pattern here? We know that we will have to apply bridge pattern.

But how exactly do we apply the bridge pattern? Where is the abstraction hierarchy? And what are the implementation hierarchy? If you think enough, we will find that the things that are on the abstraction side at the different types of cars may be the sporty truck SUV. And then these are the ones that are manufactured by different vendors like Ford, Toyota, Maruti and so on.

So, on one side, we have the abstraction the same sporty maybe manufactured by Toyota, by Ford, by Maruti and so on. So, we need to separate out the abstraction hierarchy from the implementation hierarchy.

(Refer Slide Time: 07:00)

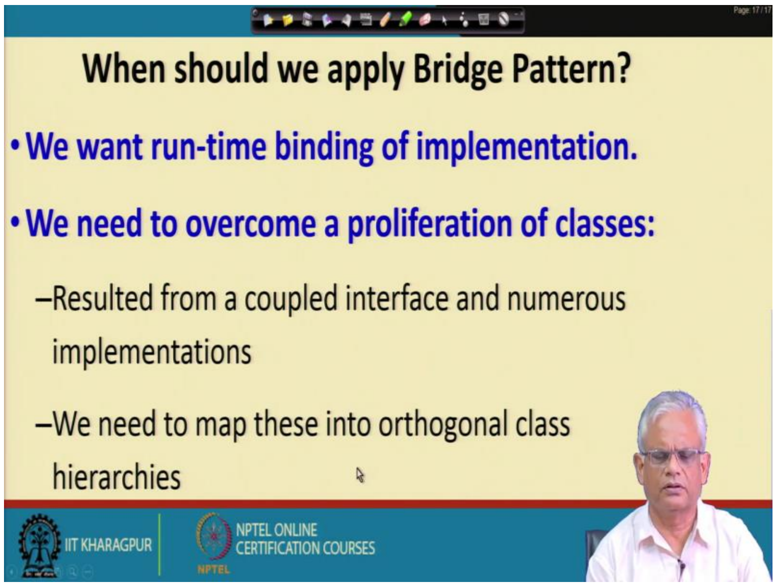


Exercise 1: Solution...

Use Bridge when you might otherwise be tempted to use multiple inheritance...

And that is what we have done here; applied the bridge pattern. We have the car models, sporty, truck, SUV and so on. This is abstraction hierarchy and on the implementation hierarchy we have the car manufacturer, the different companies Ford, Toyota, Maruti and so on.

(Refer Slide Time: 07:25)



When should we apply Bridge Pattern?

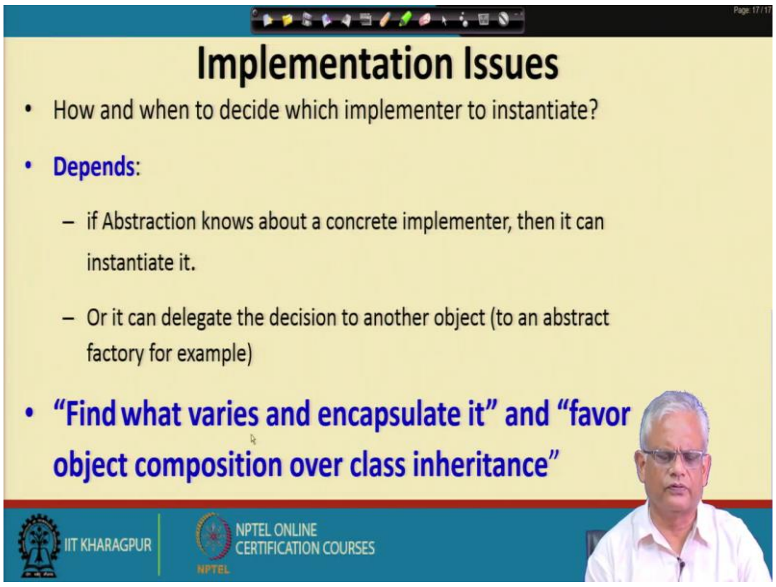
- We want run-time binding of implementation.
- We need to overcome a proliferation of classes:
  - Resulted from a coupled interface and numerous implementations
  - We need to map these into orthogonal class hierarchies

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES

Now few concluding remarks on this pattern. We apply the bridge pattern when we want a runtime binding with the implementation through an association relation and delegation of responsibility. If we do not use the bridge pattern, we will have a permanent binding through inheritance. It is not a runtime binding, but a permanent binding.

And the bridge pattern helps to overcome proliferation of classes. And to apply the bridge pattern we must clearly understand the two orthogonal class hierarchy water on the abstraction hierarchy and water on the implementation hierarchy that requires some understanding of the problem. And we can see that what exactly is the abstraction hierarchy and the implementation hierarchy.

(Refer Slide Time: 08:32)



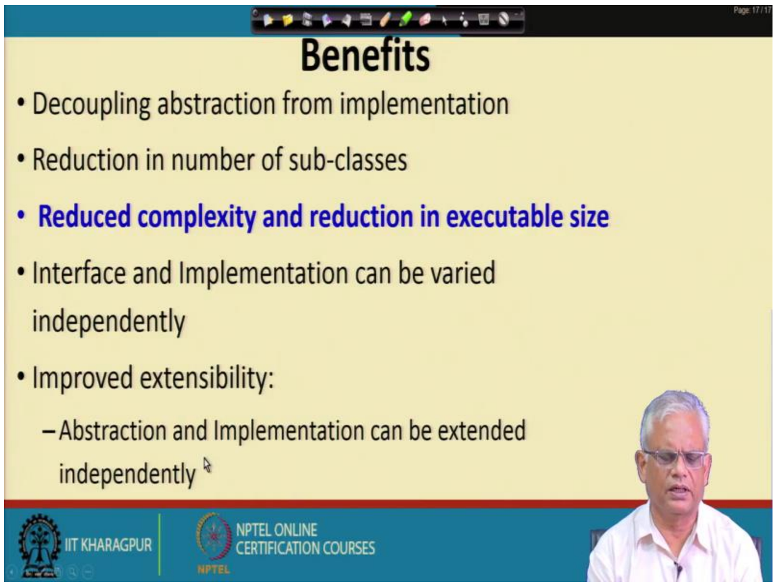
Implementation Issues

- How and when to decide which implementer to instantiate?
- **Depends:**
  - if Abstraction knows about a concrete implementer, then it can instantiate it.
  - Or it can delegate the decision to another object (to an abstract factory for example)
- **"Find what varies and encapsulate it" and "favor object composition over class inheritance"**

One of the issues is that when a request comes to abstraction class, it needs to delegate to a concrete implementer. But then there are many types of concrete implementers on the implementation hierarchy. So, which one does it delegate. In our code example if you see that we had stored the specific or the required implementation class, when we created the abstraction class, through the constructor of that, we had given the reference of the implementation class.

So that is a simple thing to do. A simple way to identify which is the implementer class method, which will be invoked and the bridge pattern helps us to find what varies and encapsulate it and it helps us to favour composition over class inheritance.

(Refer Slide Time: 09:50)

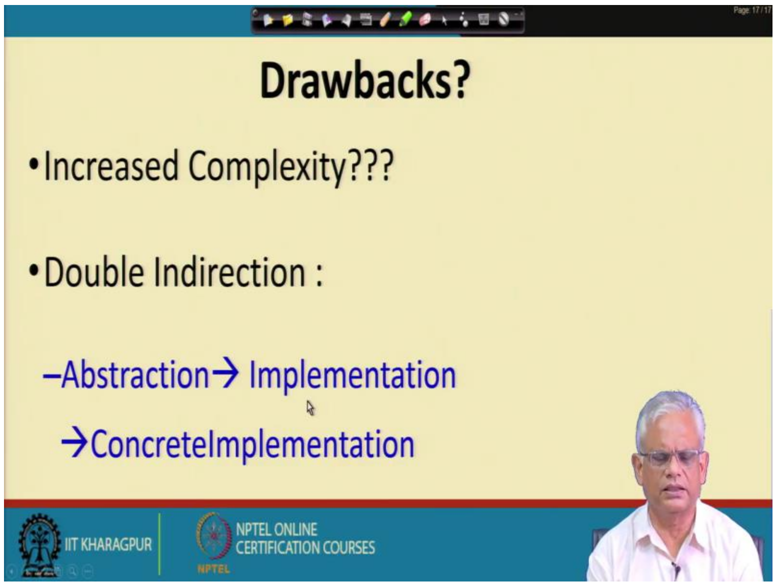

Benefits

- Decoupling abstraction from implementation
- Reduction in number of sub-classes
- **Reduced complexity and reduction in executable size**
- Interface and Implementation can be varied independently
- Improved extensibility:
  - Abstraction and Implementation can be extended independently

It has many benefits; decouples abstraction from the implementation, reduces the number of subclasses, reduces complexity and the code size. The different concepts in the interface and implementation can be varied independently without affecting the two hierarchies together. So, if we vary the interface or the abstraction hierarchy, only the abstraction hierarchy will vary; it should not affect the implementation hierarchy.

And similarly, if we have a new implementation, it should only change the implementation hierarchy, it should not affect the abstraction hierarchy. And it makes it more maintainable.

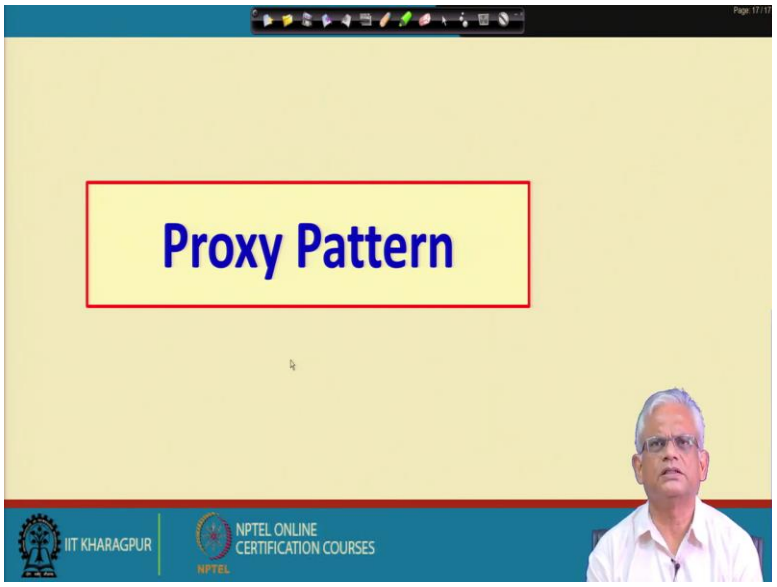
(Refer Slide Time: 10:41)


Drawbacks?

- Increased Complexity???
- Double Indirection :
  - Abstraction→ Implementation
    →ConcreteImplementation

But, are there any drawbacks of this pattern? Is there anything or any consideration on which using this pattern we are a loser. We sacrifice some benefits; we really think about it. Okay, increased complexity do not agree to that, I think it reduces the complexity. But possibly indirection that we have to store a reference to the implementation class and invoke its method. If we really think of it as a drawback, possibly that can be a drawback that is an indirection. But I do not think that these are really big problems.

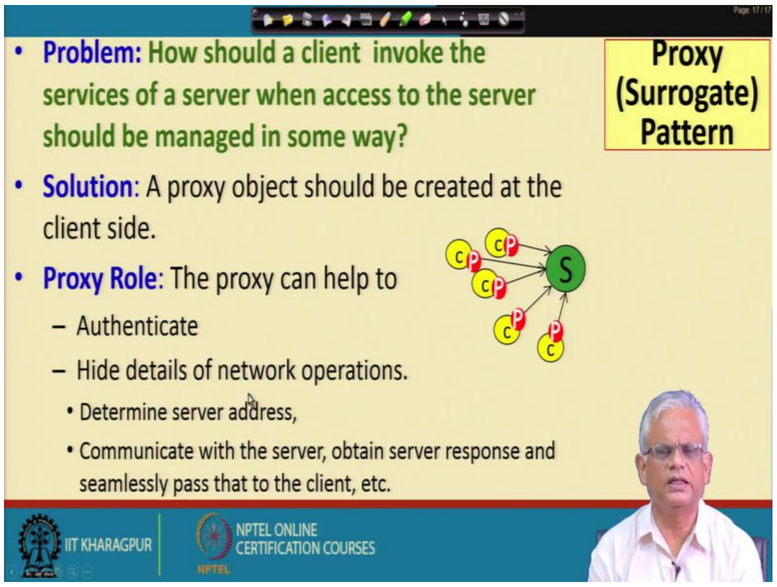
(Refer Slide Time: 11:42)



Proxy Pattern

Now, let us look at the proxy pattern. The proxy pattern is also a frequently used pattern. And one of the important things of this pattern is that this same pattern we used for various purposes will outline about a dozen purposes for which the proxy pattern becomes useful. And no wonder that this is also one of the very frequently used pattern any programmer is developing an application of some sides would have an opportunity to use a proxy pattern in his application development.

(Refer Slide Time: 12:31)



Proxy (Surrogate) Pattern

- **Problem:** How should a client invoke the services of a server when access to the server should be managed in some way?
- **Solution:** A proxy object should be created at the client side.
- **Proxy Role:** The proxy can help to
  - Authenticate
  - Hide details of network operations.
    - Determine server address,
    - Communicate with the server, obtain server response and seamlessly pass that to the client, etc.

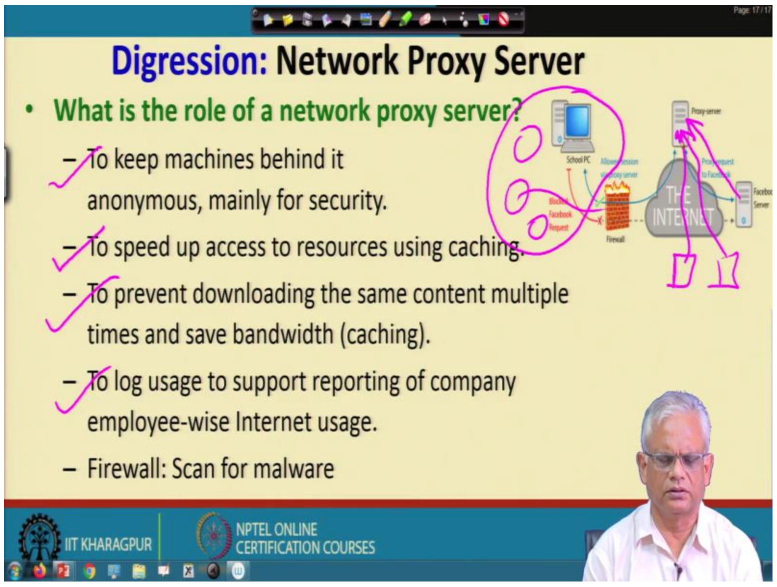
Now let us try to understand the basic idea of this pattern. This is called as a proxy or a surrogate pattern. Here, we have an application where there is a server class and whose methods are invoked by various client classes. The server provides service to many clients. But the problem that we are trying to address in the proxy pattern is that how should the clients invoke the services of the server when they access to the server, when the access to the server should be managed in some way.

For example, only some clients should be able to execute some methods. Some clients should be able to change data, others should only read data and so on. The solution provided by this pattern is that we have to create a proxy object on the client side. See here, we have several objects created on the client side, the proxy objects; these proxy objects can help to do various things.

For example, they can help to authenticate only a client who can invoke some specific methods will be allowed to invoke that method otherwise you will be denied lack of permission. The proxy can hide network operations such as determining the server address, communicating with the server, establishing handshake, obtain server response, seamlessly pass that to the client etc.

Basically, simplifies the client side design. It makes many things; it takes the responsibility of many things. And the client just delegates those responsibilities and the client seamlessly contact the server as if it is under local machine. They hide the details of the network operations. And there are many other role of the proxy as you will find out now.

(Refer Slide Time: 15:10)



Digression: Network Proxy Server

- What is the role of a network proxy server?
  - To keep machines behind it anonymous, mainly for security.
  - To speed up access to resources using caching.
  - To prevent downloading the same content multiple times and save bandwidth (caching).
  - To log usage to support reporting of company employee-wise Internet usage.
  - Firewall: Scan for malware

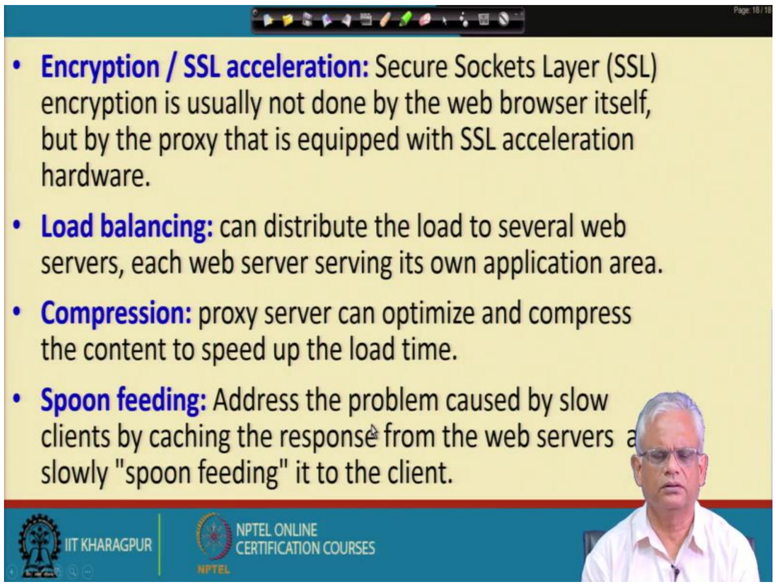
Before we discuss the nitty gritty of the proxy design pattern, let us just try to understand what is the role of a network proxy server? Anybody using an internet will definitely be aware of the network proxy server. Why do we use a network proxy server? In what ways does it help? If we can answer these questions, then we will see that this had some analogue with the proxy design pattern that we are going to discuss the proxy in the proxy design pattern and also had similar responsibilities, like the responsibility of the proxy server in a network.

Let us just identify the responsibility of a network proxy server even though it appears as a digression because that is not our focus in this course to discuss about networking issues. But then if we can know the responsibility of a proxy server, that will help us to understand our proxy design pattern better. A network proxy server has various responsibilities. The first is to keep machines behind anonymous. They have private IPs. The machines behind the proxy have private IPs. And they are anonymous, mainly for the security and also it helps to handle the problem of providing enough number of IP addresses. It helps the machines behind anonymous and the outside world. They basically need to contact the proxy over the network. And then we will have firewall and so on. And therefore, the computers behind the proxy are safe. Another important role of the proxy is that they cache resources, as the different computers and the network they access resources on the internet, various servers, they get resources, these are cached here.

For example, you as a user accessed news item that is cached here, another user trying to access the same news item do not need to really go over the internet and again fetch it is available fresh news item and that will be given without really having a multiple access over the internet. It prevents downloading the same content multiple times and saves bandwidth this basically the caching that because of the caching, we provide fast access.

If it is cached here, we not only provide fast access but also we save bandwidth. It can log uses, what are the users and what sort of uses and how much bandwidth they have consumed there can be a log of that. And this can be provided to the company management on employee wised internet eaters. We can have a firewall associated with the proxy server to scan for malware.

(Refer Slide Time: 20:02)



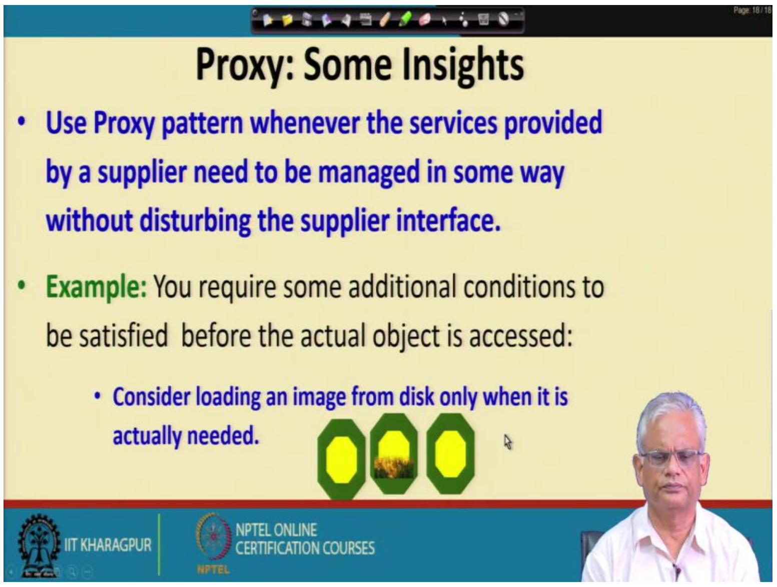
- **Encryption / SSL acceleration:** Secure Sockets Layer (SSL) encryption is usually not done by the web browser itself, but by the proxy that is equipped with SSL acceleration hardware.
- **Load balancing:** can distribute the load to several web servers, each web server serving its own application area.
- **Compression:** proxy server can optimize and compress the content to speed up the load time.
- **Spoon feeding:** Address the problem caused by slow clients by caching the response from the web servers a slowly "spoon feeding" it to the client.

Also, the encryption and SSL acceleration, the secure socket layer, encryption is typically not done by the web server, the web browser, otherwise it will really slow up if we do the SSL encryption on the browser itself. In software, rather, we will have a hardware acceleration installed in the proxy server, because all requests to the internet goes through the proxy. And therefore, the hardware acceleration, hardware, SSL acceleration, etc. can be provided on the proxy.

Load balancing, as an external request comes when you have multiple servers behind the proxy, the proxy can do some kind of load balancing and find out the server that is less loaded can forward the request to that. Compression; proxy server can optimise and compress the contents before transmitting it under network.

Spoon feeding; sometimes some of the clients are too slow. And if the server dumps the response on them, they will be overwhelmed. And the proxy, basically, spoon feeds them as it determines the slow clients. As you can see that a network proxy server has many responsibilities.

(Refer Slide Time: 21:50)



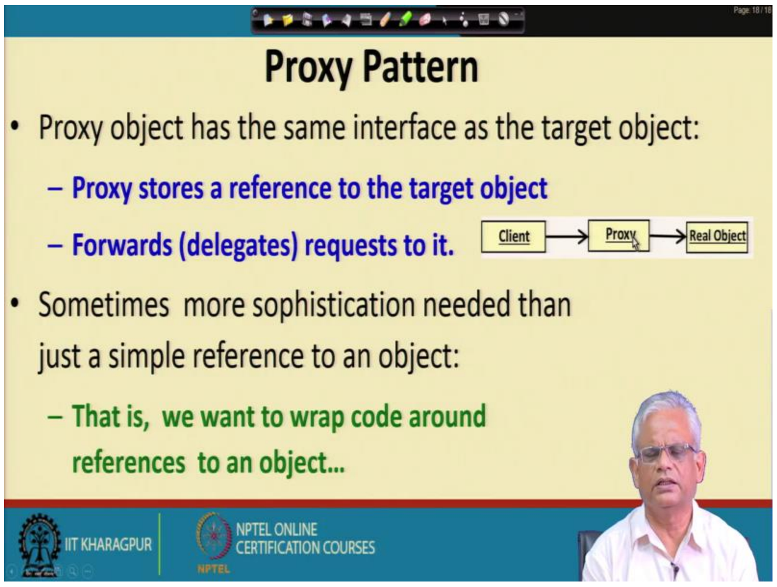
Proxy: Some Insights

- **Use Proxy pattern whenever the services provided by a supplier need to be managed in some way without disturbing the supplier interface.**
- **Example:** You require some additional conditions to be satisfied before the actual object is accessed:
  - **Consider loading an image from disk only when it is actually needed.**

And similar is the case with the proxy, the proxy design pattern. Now let us look at some of the ways in which proxy can be used. We use a proxy when the services provided by a supplier needs to be managed in some ways without disturbing the supplier interface. So, we have a supplier or the server and we want to manage it someway. Maybe authentication, maybe unless you really can have to invoke the server will not allow it and so on.

These basically the proxy helps to implement some additional conditions to be satisfied because before the server object is accessed. Let us say we have various pictures displayed here. These basically add the proxies. If you click on a picture, it will show the picture it is not showing now as you click, it will load only that picture at that time once you request it because there may be thousands of days. And if you load all of them, it will be very slow. So, load on demand, the proxy takes the responsibility that when a client is interested in a specific picture it will only show that.

(Refer Slide Time: 23:46)



Proxy Pattern

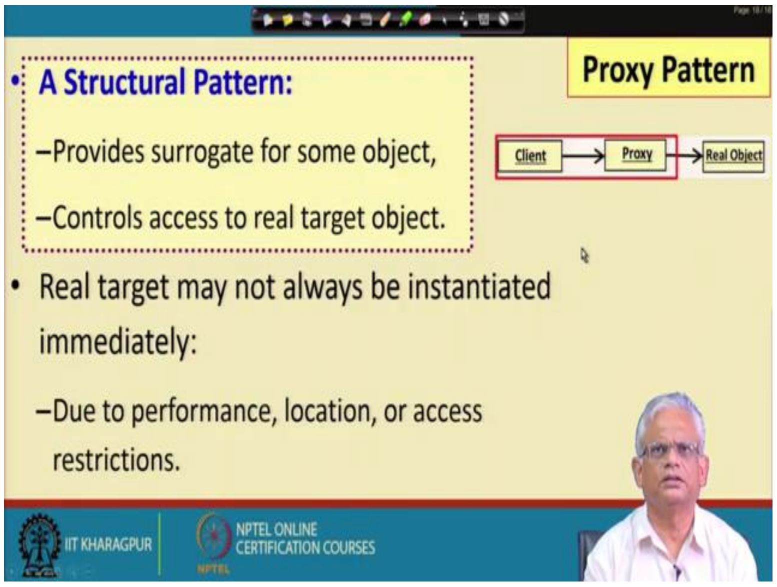
- Proxy object has the same interface as the target object:
  - Proxy stores a reference to the target object
  - Forwards (delegates) requests to it.
- Sometimes more sophistication needed than just a simple reference to an object:
  - That is, we want to wrap code around references to an object...

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

Now, let us look at the pattern. Here we will have a proxy object created, that will have the same interface as the target object or the server, so that the client would have no way to know whether it is dealing with a proxy or the target. The client may be under the impression that it is directly invoking the target. But it might be dealing with only the proxy and the proxy should store a reference to the target object or the server. And whenever required a client request comes in the proxy cannot handle it. It will forward it to the target object of the server. But sometimes it just does not forward it as it comes but it does some pre-processing then invokes the server and then maybe doing some post-processing. So, basically the proxy can have help us to wrap code around the reference to a server.

The client would interact with the proxy. The proxy can answer many of the queries of the client or it can provide service many of the service requests it can handle itself. And some it may forward to the real object or the server. But then it may wrap some code around it may do some pre-processing and post processing steps. So, those are the roles of the proxy.

(Refer Slide Time: 25:41)



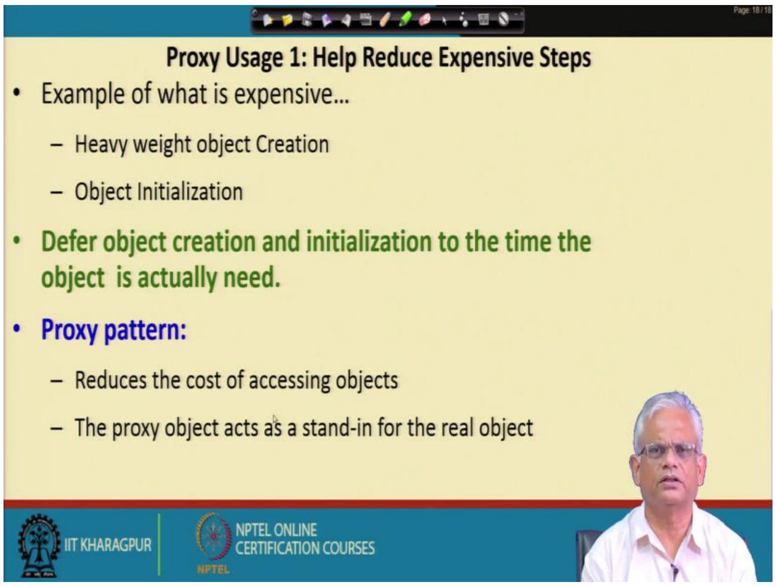
Proxy Pattern

- **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
Proxy
Real Object
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES

It is a structural pattern, it provides a surrogate for some object, so the object actual object or the server, maybe in a different machine or across the network somewhere, but the client would not know it because the proxy will be co-located with the client and the client would be under the impression that it is interfacing with the real object even though it is only interfacing with the proxy. And the real object or the server may not be instantiated. The proxy will determine when to instantiate the real object or the server. And that will help us to speed up the application give improved performance, and also provide location and access restrictions.

(Refer Slide Time: 26:50)



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

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES

Now, let us look at the first use as a proxy. Help reduce expensive steps in application so that the application will get speeded up. Now, if we analyse the application and find out what is expensive, we will see that it takes time to create heavyweight objects. For example, if you have to read 50 megabytes of data to create an object, a picture object or something it is definitely time consuming.

Similarly, object initialization for the heavyweight objects, these are expensive. And to speed up the application we should create this and initialize this only when needed because there may be thousands of such objects and maybe in a typical run maybe just couple of them need to be created. Why should we create thousands of them if we can determine which one or two are to be created and we just create only those.

So, it helps us to differ object creation and initialization to the time the object is actually created, actually needed. We do not have to, at the beginning of the application run, we need not create all these heavyweight objects. The proxy will help us to create and initialize at the runtime only those which are actually needed. This will help to make the application work very fast. This will help as if the application is working very fast. And many of the applications actually do use this technique. The proxy pattern reduces the cost of accessing objects. And the proxy objects act as a stand in for the real object. We will see with the help of some concrete examples. And we will name the different types of proxies that are used in typical applications. At least we will identify half a dozen or so. Ofcourse there are more uses of the proxy pattern.

But we will identify some of the important uses of proxy pattern will give the class diagram; the class structure for the proxy pattern so that given a situation, we should be able to create the proxy pattern solution to handle the problem. We are already at the end of this lecture and we will stop here. We will continue with the proxy pattern in the next lecture. Thank you.