Object-Oriented System Development using UML, JAVA and Patterns

Professor Rajib Mall
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture 47

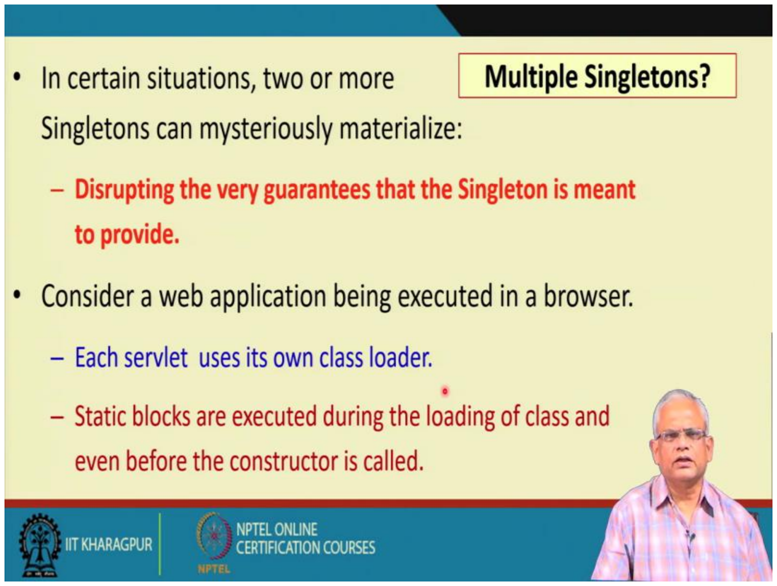
State Pattern - I

Welcome to this session! In the last session, we were discussing the singleton pattern, very useful pattern. But then it is also very simple. We saw that singleton has a static getInstance method through which all access to the singleton occurs, all instance access and the constructor is private. So the new operation on singleton class does not work.

An instance reference has to be obtained through the getInstance method. And the first time it is accessed, typically the instance is created. And then later, when the getInstance is called, just returns the reference to the single instance. We had looked at simple examples of singleton pattern. But then we were trying to identify a problem in the simple solution that we discussed, namely can multiple singletons appear under some circumstance.

And if in the simple code that you had seen multiple singleton objects can be instantiated, and the very purpose of the singleton will be lost. And therefore, we have to be careful about the context in which multiple singleton can appear, we have to be aware. And if that context exists, then we will have to take the precaution. Let us see what is the context in which multiple singletons appear and what precaution we need to take to prevent the creation of multiple singletons.

(Refer Slide Time: 2:17)



Multiple Singletons?

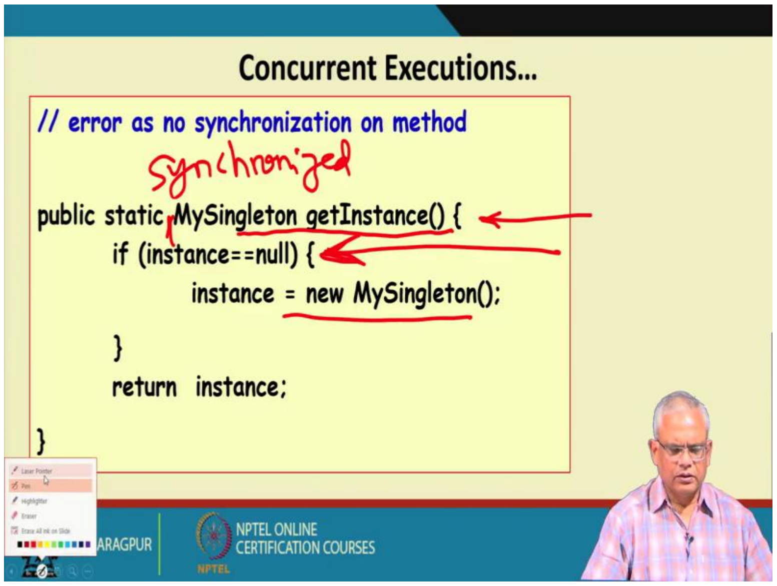
- In certain situations, two or more Singletons can mysteriously materialize:
  - Disrupting the very guarantees that the Singleton is meant to provide.
- Consider a web application being executed in a browser.
  - Each servlet uses its own class loader.
  - Static blocks are executed during the loading of class and even before the constructor is called.

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
NPTEL

We will see that in some situation, two or more singletons can appear. And this will disrupt the very guarantee that the singleton is supposed to provide. Let us see one instance where multiple singletons can appear. Let us say we have a web application, on the client side the web application is executed through a browser and on the server side, we have the servlet which uses its own class loader. Now, the static blocks are executed during the loading of class, all static blocks and we had seen that getInstance is a starting block.

And therefore, as multiple clients connect to the server multiple instances of the getInstance will get called and we will see variation in the next slide, how multiple singletons can appear in this situation, where concurrent access through a web browser occur to singleton. It may not be from web browser only; it may be any concurrent access to the getInstance method of a singleton.

(Refer Slide Time: 3:59)



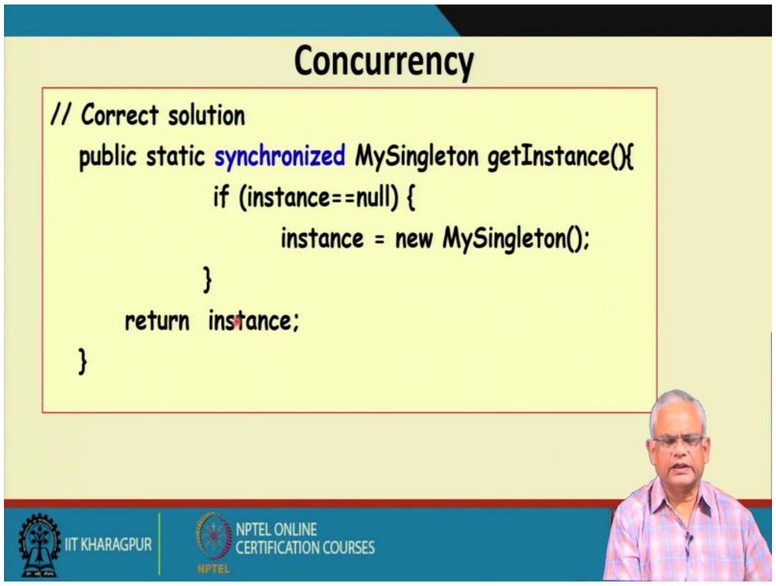
Concurrent Executions...

```
// error as no synchronization on method
                synchronized
public static MySingleton getInstance() {
        if (instance==null) {
                instance = new MySingleton();
        }
        return  instance;
}
```

Let us just recollect the code where we had this public static getInstance which is supposed to return a singleton object. And if the instance is null, then instance is created. But if the already singleton has been created, it just returns the instance. But what if two calls to the getInstance method occurs through concurrent invocations, two different threads or maybe two different class loaders they execute the getInstance it is a static block and therefore it will be executed in a class loader.

But even otherwise, in a concurrent execution scenario, different threads might invoke the getInstance method. Now as the getInstance method is invoked, as the getInstance method is invoked. Now, let us say one of the first instance let us say the getInstance is called here and just after the invocation we know that as a method is called some operations occur, for example, the parameters are pushed on to stack, program counter is changed and so on. At that instance, let us say there is a context switch and the other concurrent execution the getInstance call is passes through here, the first instance, let us say the instance = = null was also checked. So, at this point, the other instance started running and there is no singleton has been created, because the other one has started running and then it just finds instances is null, creates a new singleton. But the other one has stopped after checking the null, it had found it null and created the next singleton. So two singletons can occur in that situation. And of course, it can happen then more than two singletons can occur if there are too many concurrent method calls. So, if we analyze the problem here, the problem is because these are not synchronized, we should have added the synchronization, synchronized key word here, synchronized or so that a method once it starts executing, the other methods are prevented only until the method completes.

(Refer Slide Time: 7:55)



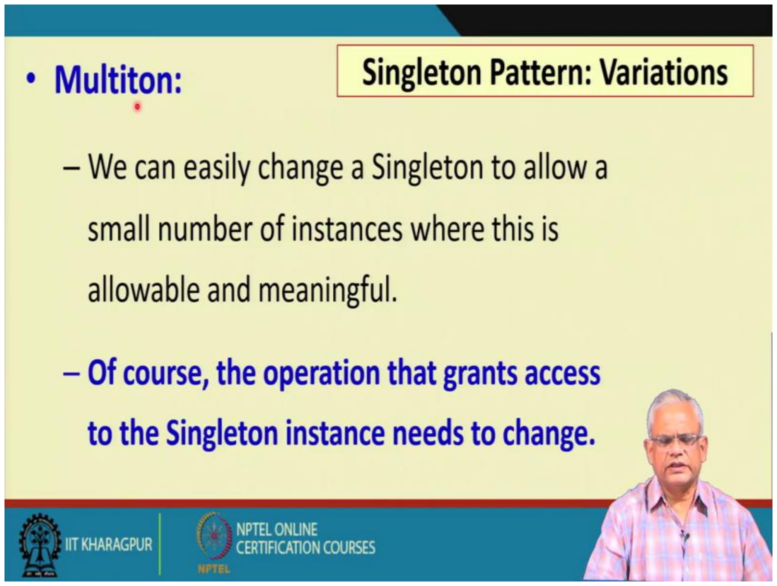
Concurrency

```
// Correct solution
   public static synchronized MySingleton getInstance(){
                  if (instance==null) {
                           instance = new MySingleton();
                  }
         return  instance;
   }
```

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
NPTEL

And therefore, we need to do that we have added the synchronized keyword here. And in this case, even during concurrent execution, only a single thread can check that it is null and create it, multiple creation of the singleton will be prevented.

(Refer Slide Time: 8:19)


Singleton Pattern: Variations

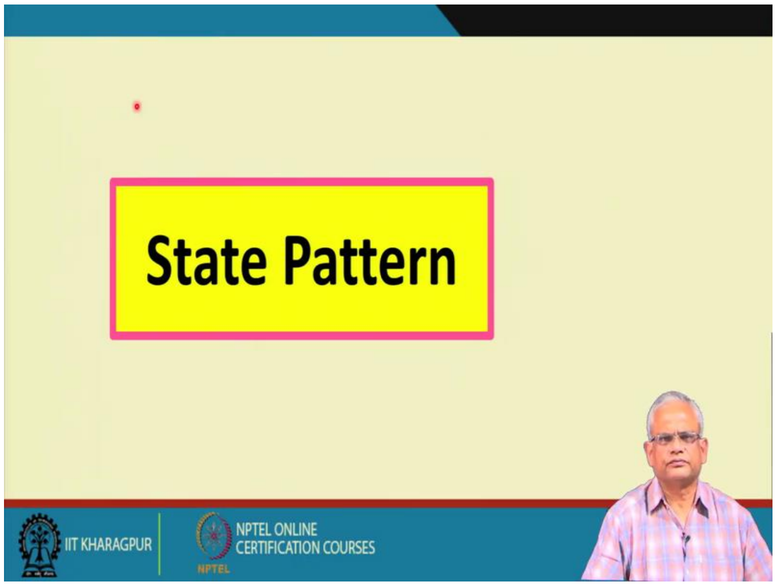
- **Multiton:**
  - We can easily change a Singleton to allow a small number of instances where this is allowable and meaningful.
  - **Of course, the operation that grants access to the Singleton instance needs to change.**

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
NPTEL

There are some variations of the singleton pattern, one is called as the multiton. In multiton, we want 2 or 3 objects to be created, exactly 2 or 3 objects not more than that, up to 2 or 3 objects we need to create and then we need to little bit relax the creation of the singleton we need to have an instance count and then as the count reaches the specified number, we need to prevent further creation, but then we have to be careful about returning the instances.

So, the getInstance method needs to change because we should not keep on sending the same instance, we need to keep track which instance is under use and then send an instance on request such that all instances that shared among the different requests. So, the operation to grant access to the singleton instance that is getInstance needs to change.

(Refer Slide Time: 9:41)

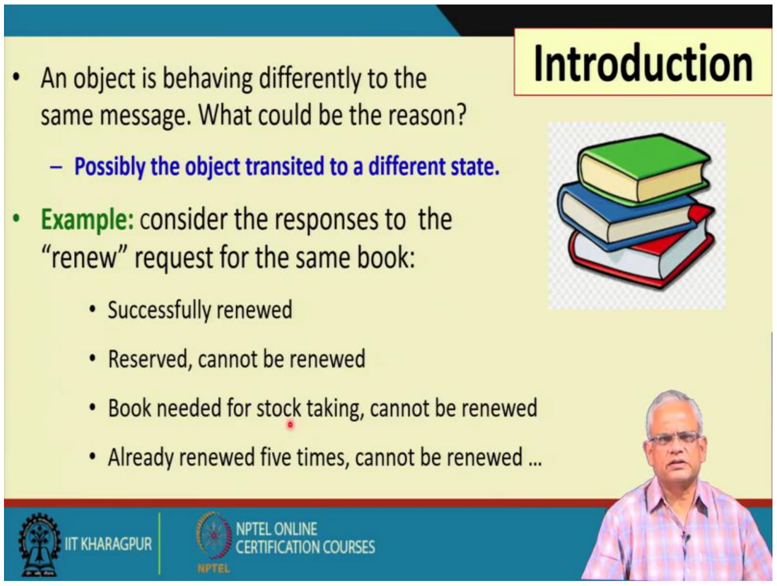


State Pattern

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES

We have discussed more or less the simple pattern singleton. But then there are some issues which we did not really focus on, for example, sub classing the singleton, we did not spend much time because we have many other patterns to look at. But then then those who are interested can look further. For example, on sub classing, the singleton in some situations, we need different types of singletons. And in those cases, we need to subclass the singleton such that the different objects might get different types of singleton. Now, let us start discussing about a new pattern, which is the state pattern.

(Refer Slide Time: 10:37)



Introduction

- An object is behaving differently to the same message. What could be the reason?
  - Possibly the object transited to a different state.
- **Example:** consider the responses to the "renew" request for the same book:
  - Successfully renewed
  - Reserved, cannot be renewed
  - Book needed for stock taking, cannot be renewed
  - Already renewed five times, cannot be renewed ...

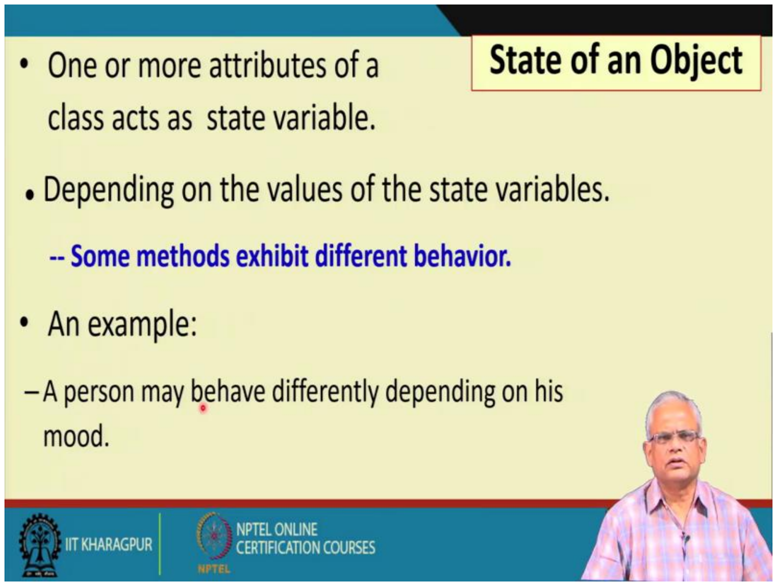
IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES

The state pattern is a behavioral pattern. Here an object sometimes is required to behave differently to the same message, the same message call, the object will respond differently under different situations, but why should this occur? The reason is that the state of the object has changed, the behavior of an object depends on the state in which it is and as the state of the object changes the behavior of the object changes.

Just to give an example, let us say we have a book object and then a user gives the request renew book. The renew book method of the book object is invoked based on the request, but then the renew method will react differently. So, these are the book objects for a specific book object. The renew method is called but then it so happens that sometimes the book gets renewed, sometimes says the book is reserved it cannot be renewed, sometimes the book is out for stock taking it cannot be renewed. Already renewed five times cannot be renewed and so on.

A convenient way to think of it is that the state of the book has changed and therefore, the responses are different. The book transits to different states, based on some events, and in different states of the book, different responses to the same method occurs, it is a rather simple and trivial example, we will see more meaningful examples where a book can have very meaningful states. Here this is more like an artificial example. But then, we can have examples where we have very meaningful states and transitions among the objects, among the object states.

(Refer Slide Time: 13:20)



State of an Object

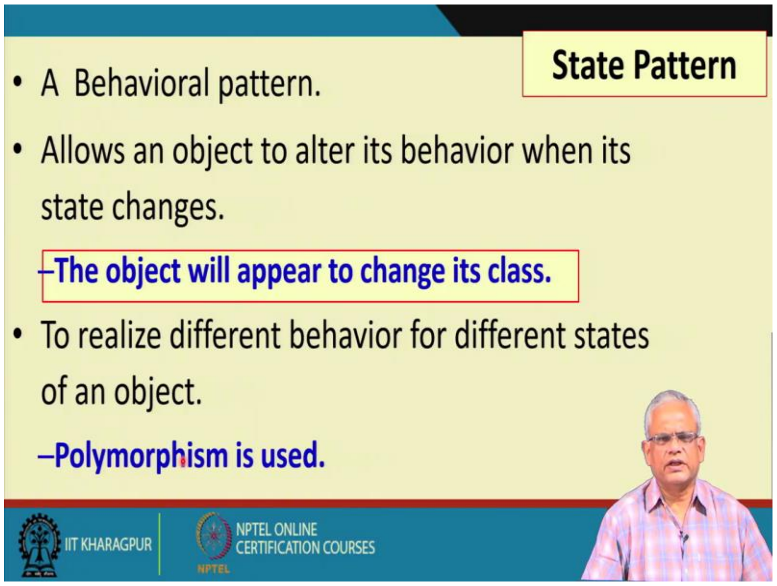
- One or more attributes of a class acts as state variable.
- Depending on the values of the state variables.
  - -- Some methods exhibit different behavior.
- An example:
  - – A person may behave differently depending on his mood.

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES

Now, let us look at the state of an object. We had already seen during our discussion on UML state machine that in an object some instance variables of the object, they start, they act as the state variable. And depending on the state variable value, the same method can exhibit different behavior. For example, in the case of a book, if the book is available, then the renew; if the book has not been reserved, then the renew method may be complete successfully. But if the state of the book is reserved, then the renew method will say that it cannot be renewed because the book has been resolved.

Another example may be that a person has a mood, the mood is a state variable of a person. And depending on the mood, the person reacts differently to an event. The same person you ask for help, says I will provide help necessary help. But then if the person mood is angry, and you request help, he will say I will not help and so on. So, an object that is a person and the state is the mood, mood can be a variable of the person class. And then depending on what has been said in the mood variable, the behavior will be different.

(Refer Slide Time: 15:25)

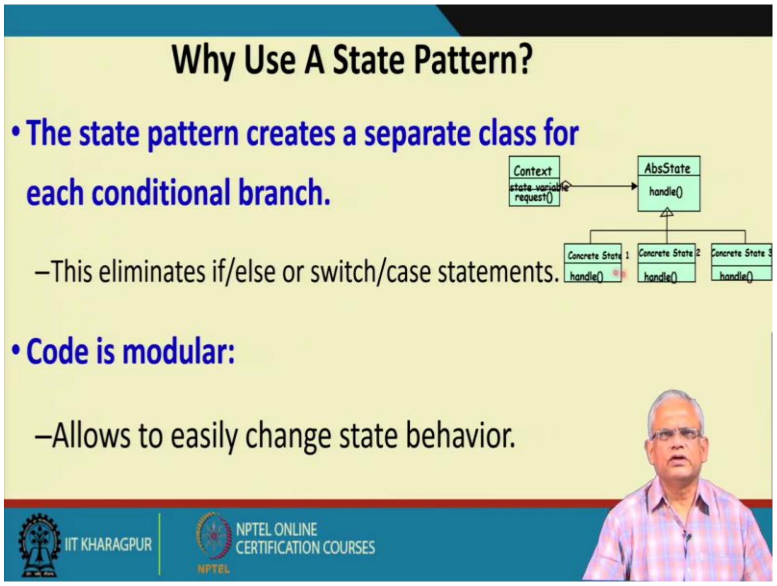


State Pattern

- A Behavioral pattern.
- Allows an object to alter its behavior when its state changes.
  - –The object will appear to change its class.
- To realize different behavior for different states of an object.
  - –Polymorphism is used.

The state pattern, it is a behavioral pattern and it allows an object to change the behavior when the state changes. But the way the pattern is implemented, is that we will find that the object, thus appears to change its state, change its class, when the state change occurs, the object will appear to change its class, we will see how it is done. As state change occurs, the object will appear to change its class. And here, we will have different objects for different states. And as the state change occurs, we will use polymorphism to replace one object with another.

(Refer Slide Time: 16:35)



Why Use A State Pattern?

- The state pattern creates a separate class for each conditional branch.
  - –This eliminates if/else or switch/case statements.
- Code is modular:
  - –Allows to easily change state behavior.

Context
state variable
request()
AbsState
handle()
Concrete State 1
handle()
Concrete State 2
handle()
Concrete State 3
handle()
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES

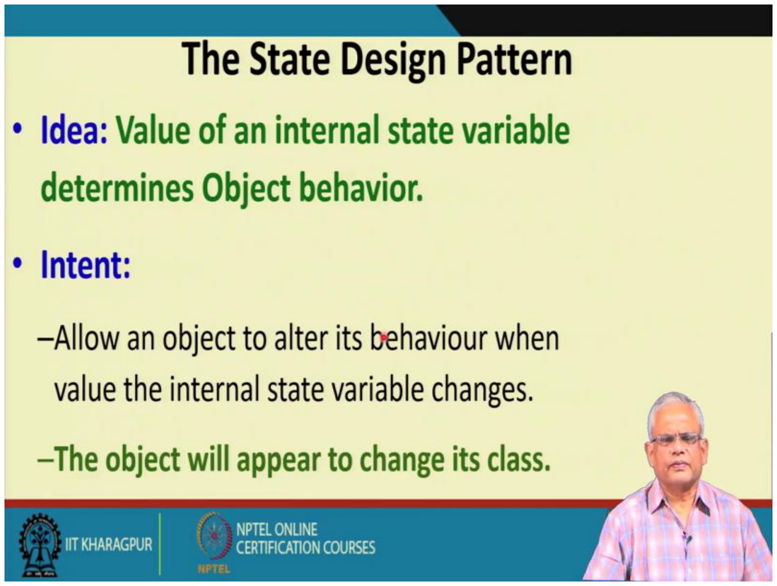
Now, let us see how it is implemented and then what are its advantages. We had seen during the UML state machine discussion, that a state machine can be easily encoded in C, Java or C++ language very easily, almost mechanically, we had the state variable and then we had the nested loop. Depending on the state, we switch the first switch statement and the inner nested switch looks at the event. And at a specific state, the type of event we can have the specific action and also a change of state.

But then, let us see why we need a state pattern because we could always code the way that we did. Here, we will see that each state behavior is encoded in one of the object just look at here that the context is actually the object whose state changes and the state of the object is determined by a state variable which is object here, is the state variable. And the state variable keeps an instance of the state here, this is the abstract state and then there are several types of extensions of the abstract state here. Concrete state 1, concrete state 2, concrete state 3 and so on. And at any time one of the state object will be pointed to by the state variable. And whenever a request comes, the context class which is the stateful object, it just delegates the request to the concrete state object that is present being pointed to by the state variable.

And therefore, we can think of that compared to our previous solution, each of those inner switches, that is depending on state, the specific events in that state are encoded into an object here, instead of having multiple branches, or a switch with multiple cases, we have just an object here. Then you will say what will be the advantage if we have, it eliminates the if/else switch. If the number of events are let us say 20. And then each time let us say on the average we traverse half of the if then, then there is a significant overhead.

But here, it is just encoded into an object and we invoke the object method. The code becomes modular. The different states are naturally encrypted into a state object, it becomes easy to add additional behavior, we can just add another object without disturbing the context class you may find that we need to modify our application. And the context class need additional state and additional transitions that can be possible here without changing the context class.

(Refer Slide Time: 20:30)



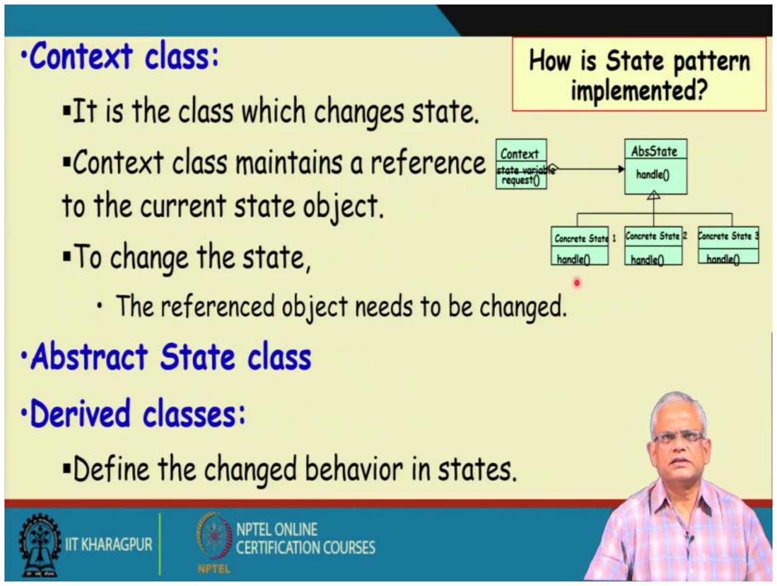
The State Design Pattern

- **Idea:** Value of an internal state variable determines Object behavior.
- **Intent:**
  - –Allow an object to alter its behaviour when value the internal state variable changes.
  - –The object will appear to change its class.

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
NPTEL

Now, let us look at the nitty gritty of the state design pattern. As in our code examples that we had seen, the state is an internal variable but then here, it is an object, the state object, and this object determines the behavior of the stateful object. Any request to the stateful object is delegated to this internal state object variable. This allows the object to alter its behavior when the state changes because the delegation will occur to a different state object. And the result is that the object will appear to change its class.

(Refer Slide Time: 21:33)



How is State pattern implemented?

- Context class:
  - It is the class which changes state.
  - Context class maintains a reference to the current state object.
  - To change the state,
    - The referenced object needs to be changed.
- Abstract State class
- Derived classes:
  - Define the changed behavior in states.

Context
state variable
request()
AbsState
handle()
Concrete State 1
handle()
Concrete State 2
handle()
Concrete State 3
handle()
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES

Let us see how it occurs. This is the model state pattern; this is a generalized pattern solution. The objects that are there, here are the classes that interact here is the context class, which stores an instance of the concrete object. The concrete object can be various types; concrete state 1, concrete state 2, concrete state 3 and they extend the abstract state.

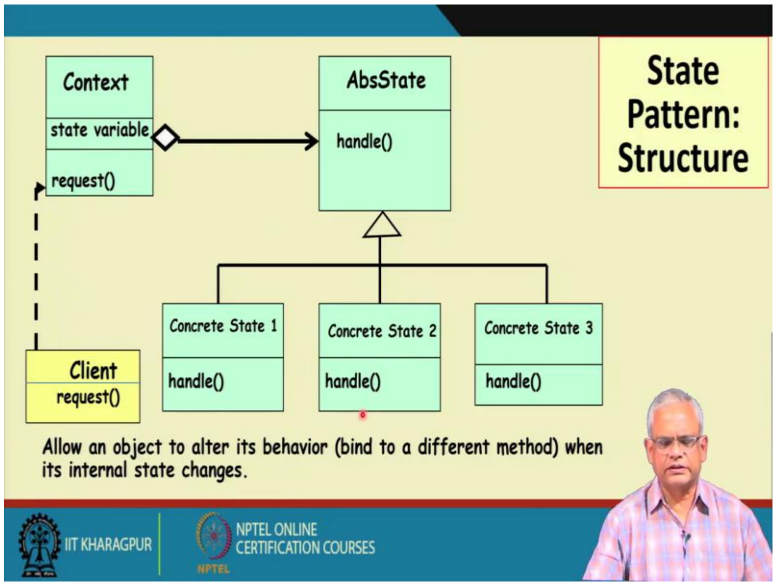
So far, we know why we need the abstract state or interface state, I hope it is clear to everybody that use of an interface or abstract class removes the dependency that would occur otherwise, it breaks the dependency and makes the context class independent. The concrete classes can change independently, we might add another concrete class and so on, without really affecting the context class recompilation and the code will not change at all.

This is one of the main principles in object orientation that we need to break dependencies and that we do by using the abstract classes or interface classes. The context class is the one which changes state. We were trying to implement the context class and we will take help of this abstract class and the different extensions of the abstract class; the concrete state 1, concrete state 2 and concrete state 3.

The context class maintains a reference to the state object. And to change the state, we just replace the state object with another state object, maybe initially the state variable was pointing to the concrete state object 1. And after a state change, we just replace the object and put in its place a concrete state object 2.

And therefore, any request to the context class, since it delegates that request to the state object, depending on the state object that is being pointed to the state variable, the behavior will be different. And the other participants in this pattern are the abstract state class and the derived classes. The derived classes are the ones which actually encode the behavior.

(Refer Slide Time: 24:34)



Allow an object to alter its behavior (bind to a different method) when its internal state changes.

So this is an enlarged view of the pattern, take a good look at the pattern, because we will solve several problems. And we will more or less adapt this solution to the specific problem that we have at hand. The context class is the one which is stateful and we were trying to implement the context class, for example, a book; a person having different mood and so on.

The clients they request to the context class and depending on the state of the context class, the response that the client gets becomes different internally. The context class delegates the request to the state class that is pointed to by the internal state variable. And the state variable, they are extension of the abstract state, these are derived classes from the abstract state and they encode the behavior that should occur in specific states.

For example, the state may be that the book is available. and then here handles once a request for renew comes, then the context class that is the book class delegates the request to this, request of the concrete state object which is handled and it may get successfully renewed. When it is on loan, it will not be renewed. We are at the end of this session. We will continue from this in the next session.