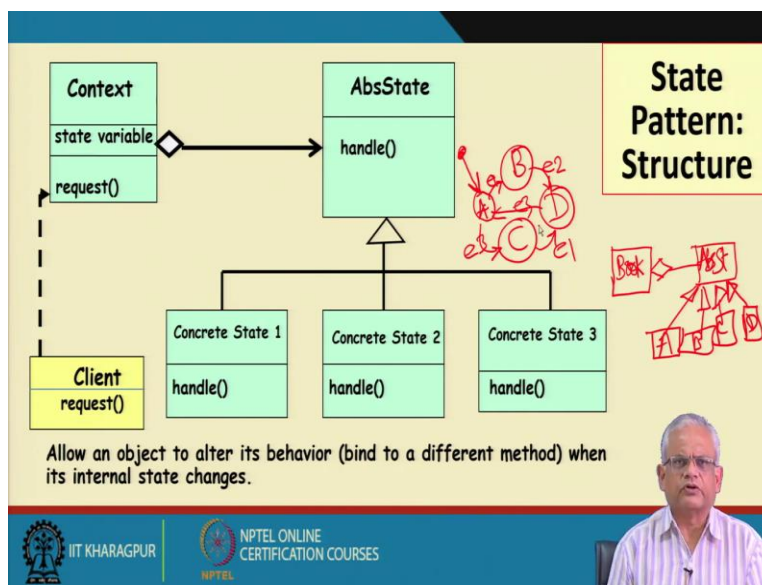


**Object-Oriented System Development using UML, JAVA and Patterns**  
**Professor Rajib Mall**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Kharagpur**  
**Lecture 48**  
**State Pattern - II**

Welcome to this session! In the last session, we were discussing about the State Design Pattern. The State Design Pattern is an important pattern often used in many application developments. The main problem addressed here is that if an object changes state, we know that the state behavior is given in the form of a state machine diagram. If we can represent the behavior in the form of a state machine diagram, how are we going to implement the behavior of the object in terms of code.

(Refer Slide Time: 1:13)



Now, let us look at the state pattern which we have been discussing in the last class. The context class is the one who changes state. And the state behavior of the context class is given in the form of a state machine diagram. The main idea behind the state pattern is that we have a state variable and the state variable points to an object, the state object; the state object implements the abstract state. So, if we have 4 states in the state machine diagram, then we will have 4 concrete states, which are derived from the abstract state. And now, the context class which is the one which implements the state behavior, it points to the present state which is a state object basically, which encodes the behavior of the context object in that state. Whenever a request

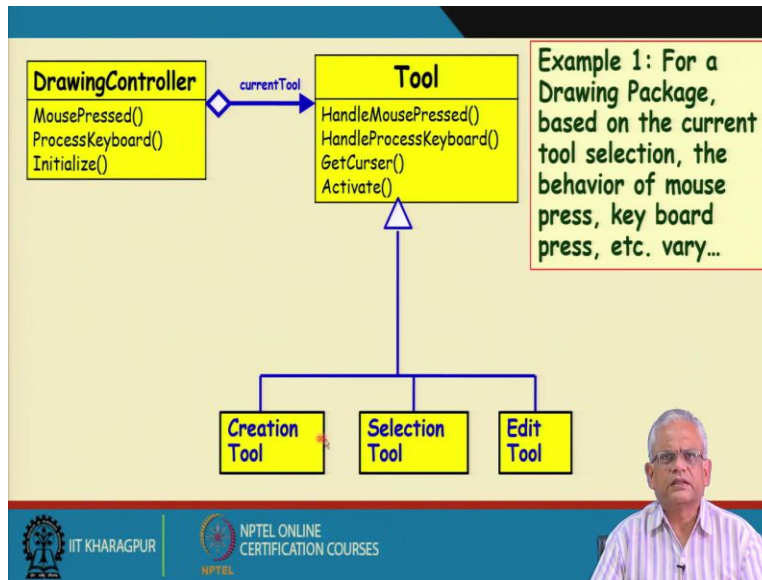
from client comes to the context class, it just delegates it to the state object which is currently being pointed to this by the state variable.

And whenever there is a state change, a new object, let us say it changes to concrete stage 2, then the state variable points to the concrete state 2. And the request coming will be delegated to the concrete state 2, the handle method, the request method is invoked under context it and that in turn delegates to the handle method of the concrete state object.

Now, let us just take one example. Let us say we have a simple state machine diagram given as there are let us say 4 states; state A, B, C, D. And let us say the initial state is A and then on event e1, it transits to B state and from B state it transits based on event e2 and from A state it transits to state C states based on e3 event, and, and C to D transits on e1 event, and D to A let us say transits on e3 event.

Now, this is a state machine diagram, not very trivial diagram. Now, to implement this in the state design pattern, we will have, let us say this is the behavior of a book class. Now, we will have the book class and then the book class contains state variable called a state and that is derived from an abstract state. The abstract state class, there are 4 concrete states, let us say A state, B state, C state and D state. Now, to start with the state variable here, we will be initialized to point to the object of A class. And then whenever an event occurs, it is delegated to the object that the current state variable of the book class points and therefore it will invoke the state behavior under A object and the A object on different events, it will determine how the book object behaves. And whenever there is a state change, a new object of the corresponding class will be pointed to by the state variable. So that is basically the state pattern implementation of the simple state machine diagram. Now let us see the details. And also how the code will look like we will just discuss about that.

(Refer Slide Time: 6:40)



Now, before we do that, before we discuss about the code, let us look at a very simple example. The example is about a drawing package, that the drawing package is having state behavior because depending on the current state, which is defined by the tool selection, if the tool selection is create state of the drawing package, then if we click, it creates an object.

If the state of the drawing package is select, then if we click the mouse, then it will be a selection of the object being pointed to by the mouse. If it is an edit state, then the keyboard whatever we press is entered, for example, the name of a diagram can be changed, but if it is in the selection state pressing keyboard has no effect and so on.

What will be the state pattern solution for the drawing package? The drawing package has state behavior, it is determined by the current state of the drawing package which is either drawing mode, selection mode, edit mode and so on. Now, the drawing package we have represented that in the form of a drawing controller, because any key press is reported to the controller and we say that the drawing controller is the one which is having, which implements the state behavior from the drawing package.

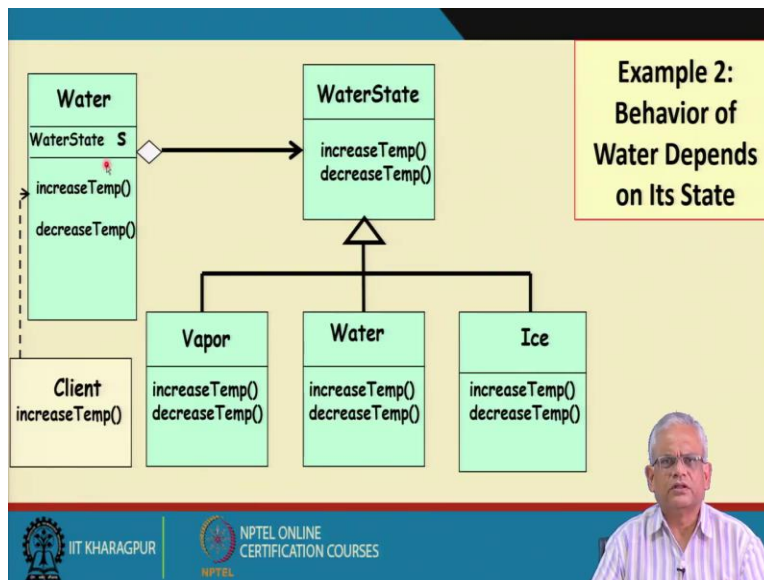
We could also have written here drawing package but here we will consider the drawing controller object to be providing behavior to the drawing package and the different events to which it reacts are the mouse press, keyboard, process initialize, etc. Now, the drawing controller has a state variable which points to the current tool selection. The tool is an abstract class and

there are 4 methods here; handle mouse pressed handle, handle process keyboard, get cursor, activate, etc.

Now the concrete classes you are at the creation state, selection state and edit state. Now, depending on the object that is being pointed to by the current state object of the drawing controller, the corresponding behavior of the drawing controller will be determined by that object. If the mouse pressed event occurs, then that will be delegated to the current state object being pointed to as soon as a mouse pressed object event occurs then the corresponding event on the state object will be called.

So mouse pressed, the handle mouse pressed event of the corresponding object, if it is a creation tool, then the creation tool handle mouse pressed event will be called. If the current state is a selection tool state, then the mouse pressed event of the selection tool will be called and so on. Now, let us look at another diagram, this is very simple, the drawing package, only 3 states and these are derived from the concrete tool state.

(Refer Slide Time: 10:52)

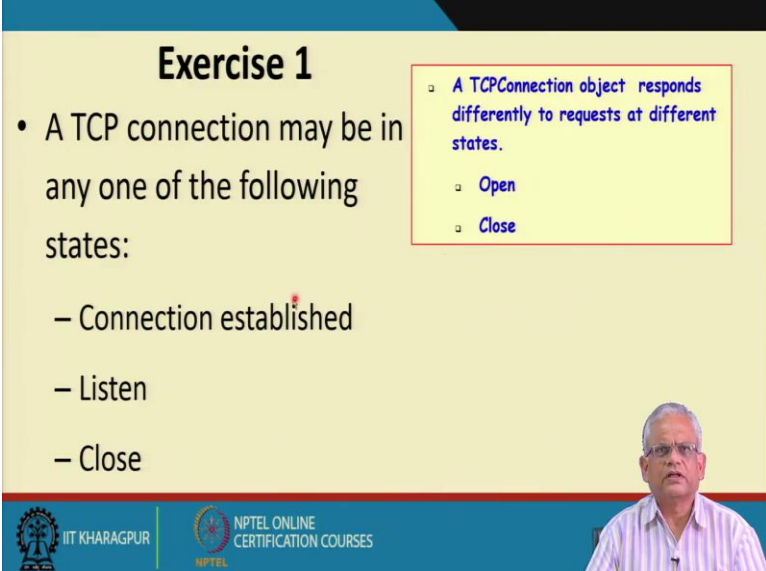


Now, let us say the behavior of water depends on its state. If in the water state, if the temperature is increased, then the temperature of the water increases until it becomes 100 degree centigrade and the water becomes a vapor state. And from the vapor state if the temperature is decreased, it reduces until it becomes a water and then water state if the temperature is decreased, then at certain point of time, 0 degree centigrade, it becomes a frozen state and it goes to the ice.

And from ice if the temperature is reduced, it remains as ice, if the temperature is increased, it becomes liquid and so on. Now, how do we encode the behavior of the water in the form of a Java program and how do we have the corresponding class design? The class design can be given in the form of a state pattern. Water is the object whose state we are representing. So that is the context class and that has the method increased temperature and decreased temperature. And that has a state variable called as the water state, S is the name of the state variable. To start with and water object will be pointed to by the water state variable S and if the increased temperature is called then the corresponding increased temperature in the water will be called. On a decreased temperature, the decreased temperature of water will be called and at certain point of time it will become ice and the water object will be de allocated and the ice object will be created and the water state is will point to the ice state and so on.

So, the state pattern solution here is that we have this water which is having the state dependent behavior, it is having a state variable S which points to an object of a class which is derived from the abstract class water state. So, this is again a very simple solution. And as the client of the water increases or decreases the temperature calls the method increase or decrease temperature the corresponding method of the state object will be called.

(Refer Slide Time: 14:07)



**Exercise 1**

- A TCP connection may be in any one of the following states:
  - Connection established
  - Listen
  - Close

A TCPConnection object responds differently to requests at different states.

- Open
- Close

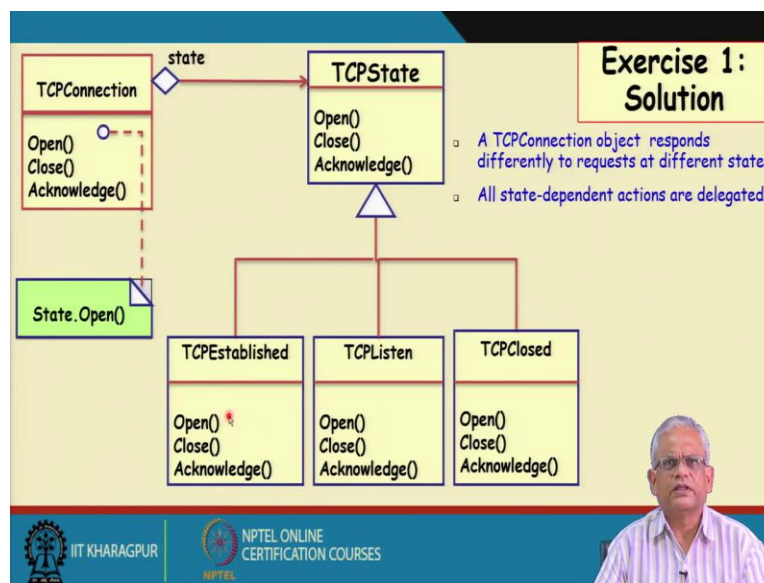
The slide includes logos for IIT KHARAGPUR and NPTEL ONLINE CERTIFICATION COURSES at the bottom. A small video inset of a man is visible in the bottom right corner of the slide.

Now let us do one exercise. We want to design a TCP connection. The TCP connection is having a state behavior and depending on the state that the TCP connection the different methods

invoked under TCP, invocation under TCP connection will be different. For example, if the connection is established, this is the state then we can transmit and we can have the close. Either TCP state is closed, then we can open the connection but we cannot transmit and so on.

How do we represent the state behavior of the TCP connection class? This is again we will have the concrete states. So, the TCP connection will point to the concrete state and the different actions that are take place open, close, etc., the behavior will be different depending on what state it is.

(Refer Slide Time: 15:31)



So we will have the state behavior represented here in this class diagram, the TCP connection is the one which is having the state behavior, there are 3 states; TCP established, TCP listen and TCP closed. The TCP state is the abstract class and these are the concrete classes. And depending on which object is being pointed to by the state object in the TCP class, the open, close and acknowledge will be invoked in the corresponding object and the behavior will be defined by the current object that the state object points to.

(Refer Slide Time: 16:21)

### State Pattern: Exercise 2

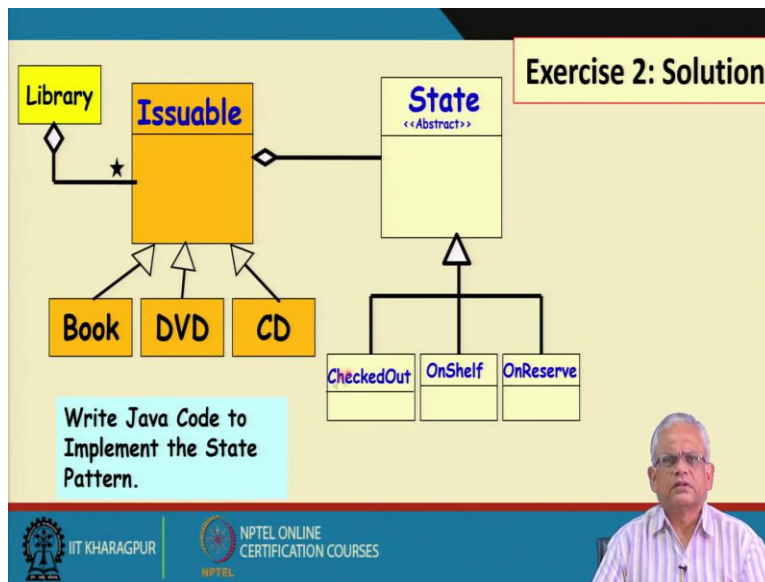
- A Library has a large number of Issuables.
  - An issuable is either a book, a DVD, or Music CD.
- Each issuable can either be on shelf, issued out, or on reserve.
  - Response to request for issue and renew an issuable is different in different states.

IT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES

Now, let us look at another problem. A library has a large number of issuable. The different issuable of the library are either a book, a DVD or a music CD, these the members can issue. But each issuable has a state, the different states are, the issuable is on the shelf, it is issued out or it is put on reserve. So, there are 3 states of the issuable, the library has many variables and each variable has 3 states. The request to issue and renew is determined by the state, if it is on the shelf and issue request comes then it is issued out. If it is already issued out, then the issue request is not honored. And the renew request, if it is put on reserve then it is not honored. Otherwise, the renew request, the renew request is honored if it is in the issued out state and also on shelf the renew request is not honored and the on shelf request, on the on shelf state, only the issue request is honored in the book is issued out.

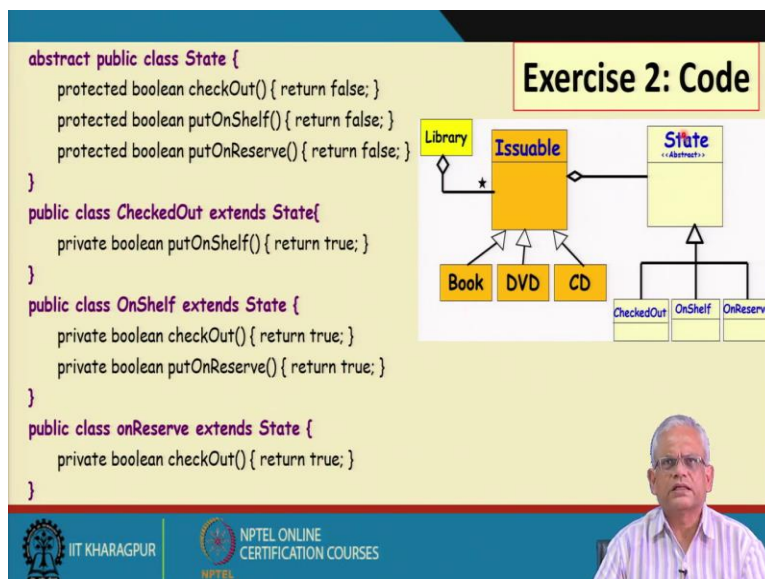
Now, how do we draw the class diagram according to the state pattern? The first thing is that we have the library consists of many issuable, we can draw an aggregation so we will have the library and it has many issuales and each issuable has different states and the different states will represent the form of a state object and this abstract state and then there are different concrete states that is on shelf issued out and put on reserve, this the basically the state diagram for the, this exercise. We will just draw out nicely.

(Refer Slide Time: 19:30)



This is the one, that the library has many issuable. The issuable can be a book, DVD or CD that is also given in the problem statement and each issuable is a neither of the state checked out, on shelf and on reserve and the concrete states of the issuable our derived from the abstract state. Now, how do we write Java code for this? If we can write Java code for this, we will be able to understand how a method called under issuable is implemented through the state pattern. Now, let us write the Java code for this design solution for the library.

(Refer Slide Time: 20:38)





So, this is the overall design solution, we have the abstract state and then the events that can occur are check out, put on shelf and put on reserve. These are the 3 events that can be called under issuable. And the issuable in turn calls those methods of the concrete state class. And therefore each of these concrete state classes checked out on shelf and on reserve, they implement these three methods of the abstract state class and in the abstract class, each of them returns false.

That is the implementation for the state class and for the specific implementation in the derived classes, these methods are overridden. Let us say when in the checked out state, the checked out state extends the state and in checked out, if the put on shelf occurs, the book has already been checked out and it is put on shelf then return true.

The on shelf class extends the state and here there are two events to which it reacts one is checkout, it is on shelf and the checkout method is called and returns true and the put on reserve event occurs and the on shelf state then it returns true and on reserve state then the checkout is true. But how about the issuable class? How is the issuable class code look like? These are the code for the abstract state and checked out, on shelf and on reserved classes.

(Refer Slide Time: 23:00)

```

abstract public class Issuable{
    State state=new onShelf();

    Public boolean checkOut(){
    if(state.checkOut() == true) {
    changeState(new CheckedOut());
    return true;}
    else return false;
}
}
    
```

```

Public boolean putOnShelf() { ... }
public boolean putOnReserve() { ... }

public void changeState(State newState){
    state = newState;
}
    
```

**Exercise 2: State Pattern Code**

UML Class Diagram:

- Library** (Class) has an aggregation relationship with **Issuable** (Class).
- Issuable** (Class) is an abstract class with subclasses **Book**, **DVD**, and **CD**.
- Issuable** (Class) has an aggregation relationship with **State** (Class).
- State** (Class) is an abstract class with subclasses **CheckedOut**, **OnShelf**, and **OnReserve**.

So this is our code for the issuable, here we have a state variable. To start with the state variable points to on shelf and then we have these different events on the issuable, the different method calls on the issuable; one is checkout. If the state is checkout, now the checkout method is called

in the corresponding state and only if it is in the on shelf state this returns true and then the change state occurs to the checked out state, otherwise it returns false, similarly, the put on shelf and put on reserve. And for each of these whenever there is a change state, we just, the change state is called here and the change state just changes to the new state.

So that is the simple implementation of the issuable class. Observe that the state changes are implemented by the issuable class, but the behavior in the specific state is given by invoking the corresponding method under state class. Similarly, we can write the put on shelf and put on reserve classes, the methods we can complete please try to do that. Simple basically will look something like this. Please complete the methods put on shelf and put on reserve.

(Refer Slide Time: 24:41)

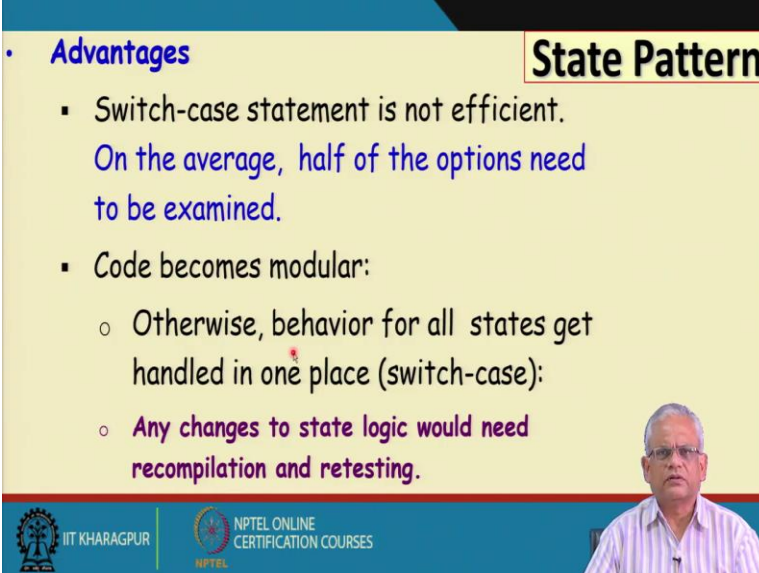
Advantages	Disadvantages	Trade-offs
<ul style="list-style-type: none"><li>• Encapsulates behavior of a state into an object</li><li>• Eliminates long lines of code involving if/else or switch/case statements with state transition logic</li><li>• State changes occur using one object, therefore avoids inconsistent states.</li></ul>	<ul style="list-style-type: none"><li>• Increased number of objects.</li></ul>	

As we could see so far, the state pattern provides a simple and elegant solution to design an object which has significant state behavior. The main advantage is that it encapsulates the state behavior into an object. The different types of behavior are given by different states of the object and each state is encoded into a state object. Remember, the code that we had written from a direct translation to from the state machine diagram to a C program or a Java implementation, there we had a long list of if then else.

First we had a switch state and in the (switch), in the specific state, depending on the state, the behavior is provided. And there we had a long if then else, or another switch, a switch inside switch. And that was not a very good solution, because we might have different actions,

inconsistent actions take place in different if then else clauses. And the state changes occur using one variable in the state pattern solution, and therefore, it avoids inconsistent states. But the chances of inconsistent states in a straightforward translation to the state machine to C code or Java code can create inconsistent states. If we can think of disadvantages, the main problem is that there is increased number of objects because the states are now objects. It is not really a big disadvantage, but if we have to think of a disadvantage, then this will be the discernment.

(Refer Slide Time: 26:49)

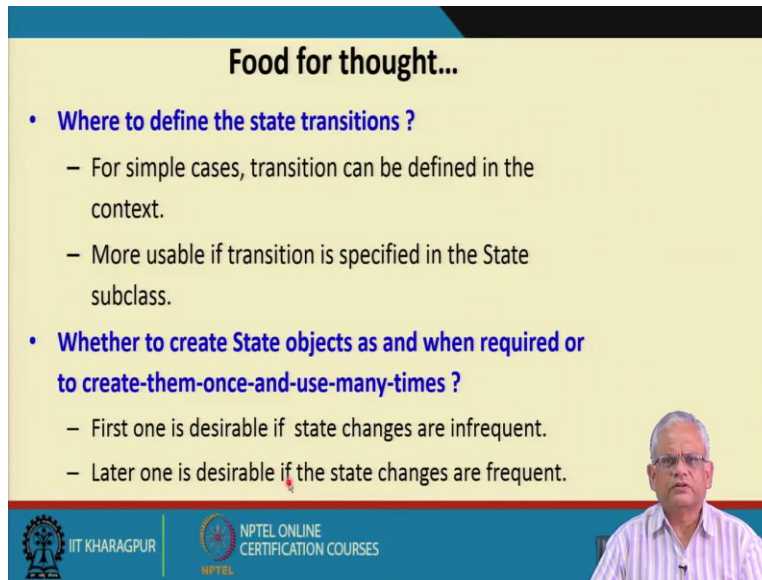


The slide is titled "State Pattern" in a red-bordered box at the top right. It lists two main advantages of the State Pattern over switch-case statements. The first advantage is efficiency, stating that switch-case is not efficient and that on average, half of the options need to be examined. The second advantage is modularity, stating that code becomes modular, whereas otherwise, behavior for all states is handled in one place (switch-case), and any changes to state logic would require recompilation and retesting. The slide also features logos for IIT KHARAGPUR and NPTEL ONLINE CERTIFICATION COURSES at the bottom left, and a small video inset of a speaker at the bottom right.

- **Advantages**
  - Switch-case statement is not efficient.  
On the average, half of the options need to be examined.
  - Code becomes modular:
    - Otherwise, behavior for all states get handled in one place (switch-case):
    - Any changes to state logic would need recompilation and retesting.

In the straightforward translation of the state machine diagram into code, we can have a long list of if then else, and on the average half of the options we will have to examined, if we have let us say 20 different case statements; if that 20 if then else statements, then on the average 10 statements would have to be examined. And that is overhead, which does not occur in the state pattern, because we are only invoking the code and the corresponding state. The code becomes modular. So, there are two advantages one is efficiency, runtime efficiency and the other is that the code becomes modular, all the state behavior is present in one state object, and therefore, we cannot have an inconsistent state if we solve using the state pattern.

(Refer Slide Time: 27:52)



**Food for thought...**

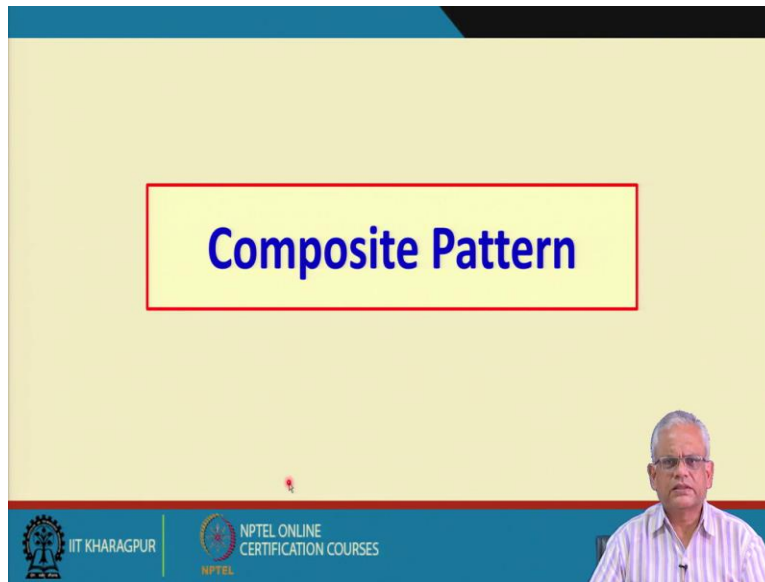
- **Where to define the state transitions ?**
  - For simple cases, transition can be defined in the context.
  - More usable if transition is specified in the State subclass.
- **Whether to create State objects as and when required or to create-them-once-and-use-many-times ?**
  - First one is desirable if state changes are infrequent.
  - Later one is desirable if the state changes are frequent.

IT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

But then, the questions that we have implicitly assumed so far and also given in our Java code that we examined, that where does the state transitions defined? In the example we had given the state transitions were defined by the context classes. The specific example, in the issuable class, the state transitions were implemented. But in more complicated cases, the state transitions can be implemented in the state classes themselves.

But then, is it that we have to create new objects each time and dispose of the object when a state change occurs? If the state changes are infrequent, then it is not a problem you can create object each time and dispose of. But then if the state changes are frequent, then we may recycle the object, that is we do not dispose of, we keep, we create the initial state variables and then we change the state variable.

(Refer Slide Time: 29:10)



Now, the next session, we will start discussing about the composite pattern. We are already at the end of this session. We will continue in the next session with the composite pattern. Thank you.