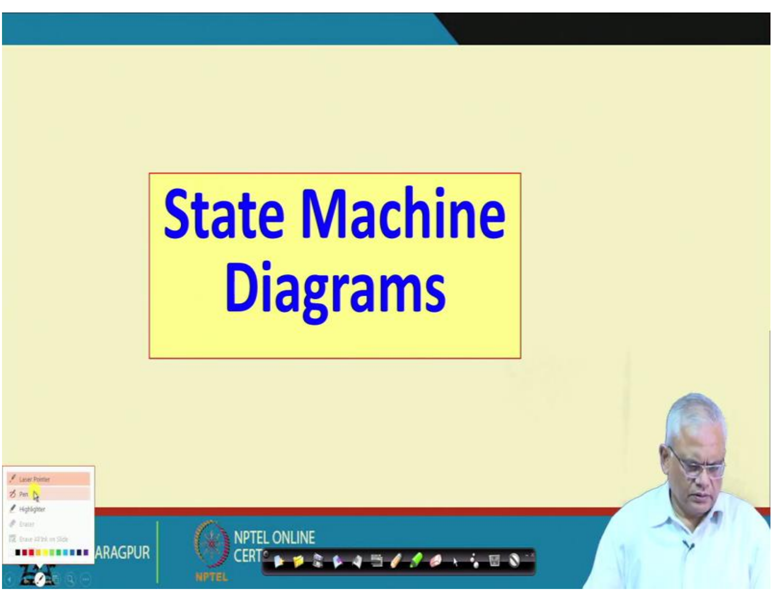
Object- Oriented System Development Using UML, Java and Patterns
Professor. Rajib Mall
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur
Lecture 19
State Machine Diagrams

Welcome to this lecture.

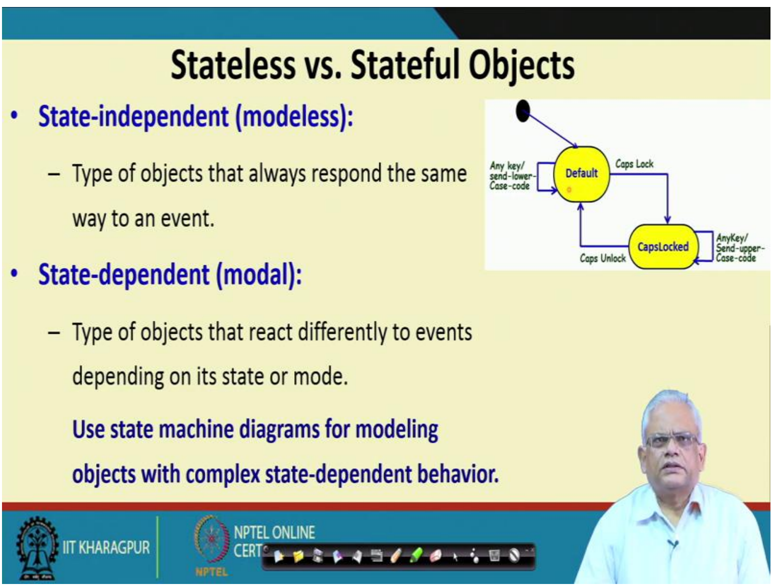
In the last session, we had completed discussing the class model. We looked at class relations and how to represent them in the UML, how to identify from a text description, and so on. Then, we had just started discussing about the State Machine Diagram.

(Refer Slide Time: 00:37)



Now let’s continue discussing the state machine diagram. This is an important diagram in UML and is used to represent significant state models of a class.

(Refer Slide Time: 01:00)



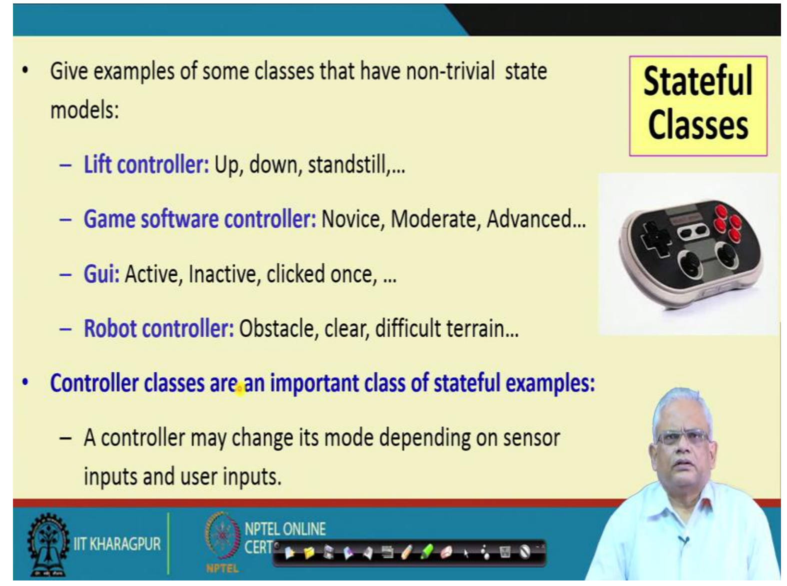
Here, if we look at the objects in an application, we find that there are two types of objects. One is the state-independent objects, which are also called as the modeless objects. The modeless objects always remain in the same state. We can say that, the object always responds to the same way to an event, that is whenever there is a method call, the response of the object is similar. On the other hand, we have the state-dependent or the modal object. Here the object can remain in one of several states and the behavior of the object will depend on which state it is there.

Here the object reacts differently to events depending on the state or the mode in which it is in and this kind of objects, we need to represent them on a state machine diagrams. We will see that if we are able to represent such objects using a state machine diagram, not only that it becomes easy to understand, the behavior of the object, but also, we can automatically generate code based on the state machine diagram as we have been doing for class diagram.

Specially for objects with complex state dependent behavior, the state machine diagrams are essential to represent them. Without that it becomes very difficult to understand the behavior. We might leave out some behavior and it will appear as a bug in the final code. This is our first state machine diagram that we are seeing in the above slide. This models a keyboard. The different states of the object are written as rounded circles. Here, Default and CapsLocked are two states.

In the keyboard, as it starts, it is in the default mode and any keypress transmits the lowercase characters to the computer. But then if some events like Caps Lock keypress occurs, then it moves to a CapsLocked mode. In CapsLocked mode any keypress it runs, it transmits to the computer in the capital key and if the Caps Unlock key is pressed in this state, then it goes to the default state.

(Refer Slide Time: 04:46)



Now, let's see further details about the state model of objects. There are some classes in every application which are modal objects or stateful objects. You have developed several applications using Java, C++, etc. Can you identify any object which are significant states?

If you are not able to identify, then one thing you may notice is that all controller objects are stateful.

For example, a lift controller in a lift application, the mode of the controller may be up, down, and standstill and all events are reported to the lift controller. If a up key is pressed, if it is in the up state then it will go up in the desired floor. But then, if there is a down request on a floor through which it is moving and it is in the up mode, it will not stop. But if it is in standstill and either up or down is pressed somewhere, it will move and will stop in the desired floor. Similarly, a game controller can be in novice, moderate, and advanced. Depending on which

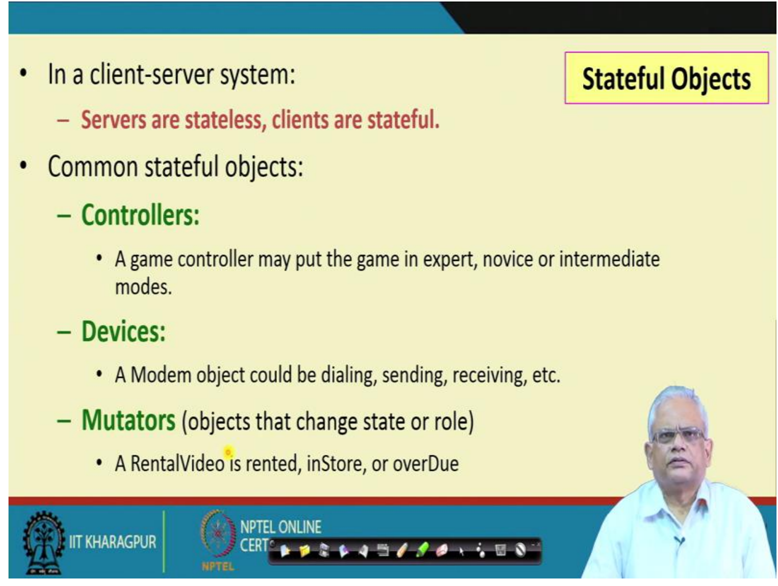
mode it is, novice mode, moderate mode or advance mode, in response to a move by a human player, its move will be different, whether it is novice, moderate or advanced.

Similarly, a graphical user interface, the mode of a menu item may be active or inactive. In active, it responds to a press or a key click of the user. If it is inactive, then any keypress it does not react.

Similarly, a robo controller, depending on whether it is in the state of encountering an obstacle, path is clear, differ in terrain, etc. The move command to the robo will get different responses from the robo.

We can say that in every application there are several controller classes. We will see as we proceed in this course, that whatever application that you write there will be several controller classes in that and these are an important example of stateful objects.

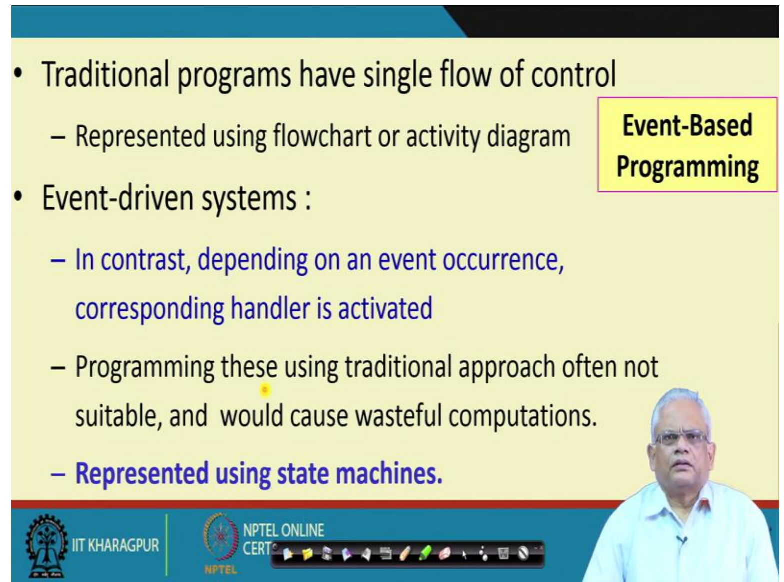
(Refer Slide Time: 08:15)



There are other stateful objects (in the above slide). For example, in a client-server system, the clients are stateful, the servers are not stateful. A modem maybe stateful. A modem object could be dialing, sending, receiving and depending on which state it is the response to the modem to transmit will be different. There are other objects also which are stateful. Like, mutator objects. Here the objects change state or role. For example, a book object. Depending on what happened, for example, whether the book is on shelf, whether it is issued out, whether it is gone for binding

or it is lost, the response to the issue book will be different depending on the state of the book object.

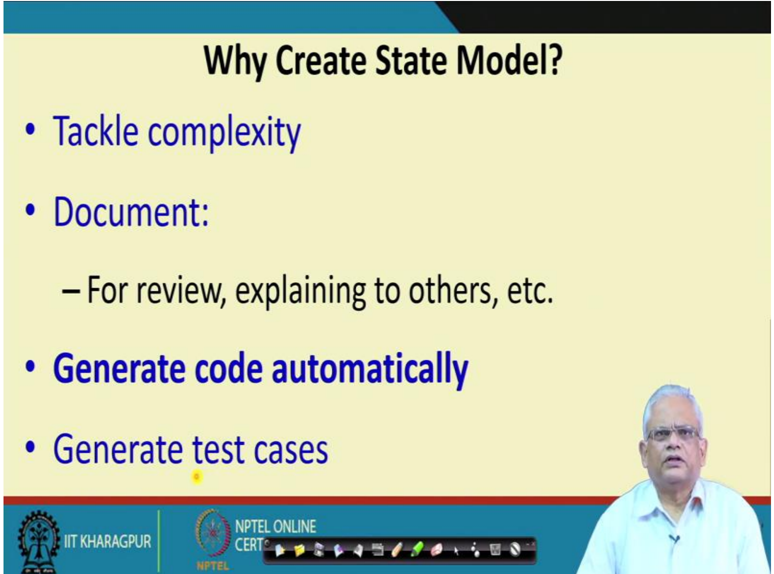
(Refer Slide Time: 09:29)



One thing we must point out at this point, during the class diagram, the code we generated, is very similar to the code of these stateful objects. For example, in a traditional C programming, Java, or C++ programming, we had a single flow of control and we can represent the behavior using a flow chart or an activity diagram. But the state-based behavior, it cannot be programmed in a traditional way, here we need event-based programming. In the traditional programming, the programs continue execution and depending on the user response it may do different things, but at any time it just continues execution and waits for a specific user event. Let say in a ATM cash withdraw use case, it request the user to enter the password and depending on whether the user enters correct password or wrong password, it reacts differently and then it asks for the amount to be withdrawn. At any time, the user is given one choice to enter, one of several choices and then depending on the choice entered by the user, it processes that. But here, at any time very different events can occur. This is not like a procedural programming where the ATM machine ask to enter the amount to withdraw or whether to withdraw from savings account or checking account. Here the events may be widely different and therefore, here depending on the event the response will occur and the system waits for an event typically and then we will have an event

handler, which will handle specific events. As we proceed, we will see that how this event-based programming is different than the traditional flow-based programming.

(Refer Slide Time: 12:30)

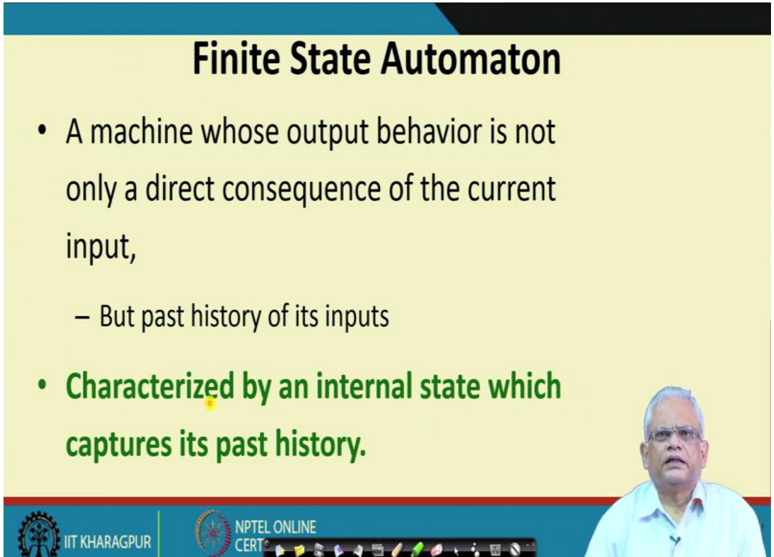


Now, before we look at the state model, let's have the motivation for creating the state model. The first motivation is that if we develop the state model of an object which has significant state behavior, then it becomes easy to understand the behavior and we tackle the complexity. Without describing the behavior of a state model, it will be extremely difficult to understanding the behavior.

The state model is an important documentation. We document the behavior for review, explaining to others, etc. But possibly one of the most important use of creating the state model is that if we create the state model correctly then we can automatically generate the code using case tool and even ourselves, we can look manually at the state model and almost mechanically write the code.

We will see that as we proceed in this course, given a state model how to mechanically or automatically generate the code and of course, once the system is developed, we need to test the behavior of the system and if we have the state model, we can automatically generate the test cases.

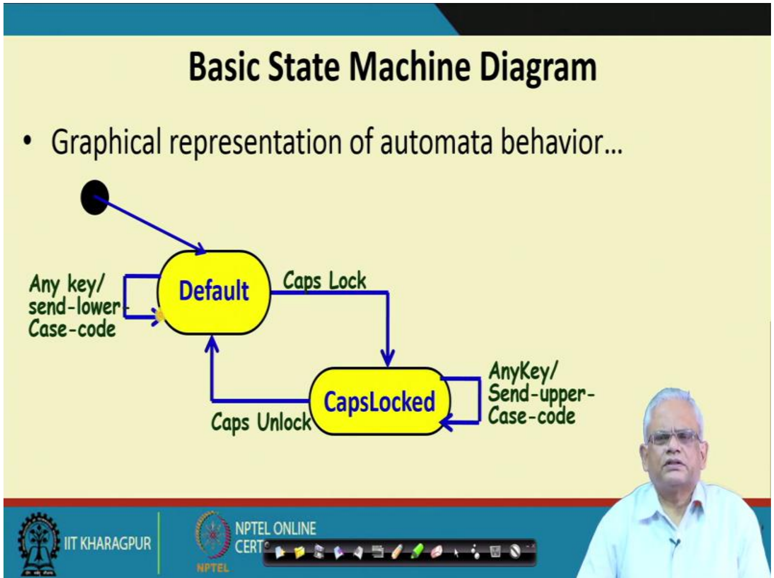
(Refer Slide Time: 14:21)



The state machine model is an extension of the finite state automaton. All of you must have used finite state automaton in various ways. It's used across different subjects. For example, it's used in the computer architecture, to model systems, it is used in operating systems to describe the behavior of a process we use a finite state automaton, in theory of computation we use finite state automaton, and so on.

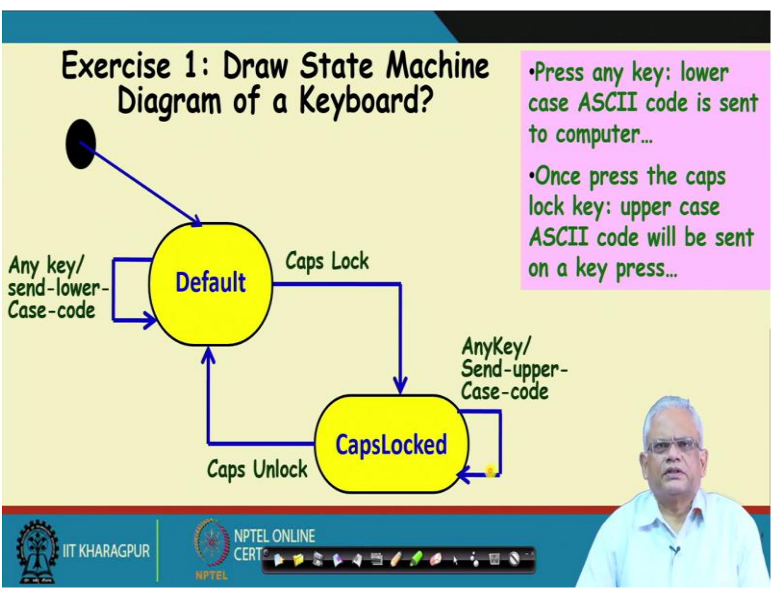
So, it's used across very various subject areas. So, finite state automation is a machine whose output behavior is not only a function of its current input but also the past history of inputs. What it means the output of the state machine is not really or completely determined by the current input, but also the past history of inputs because it might change its state and different state's behavior may be different and this finite state automaton has an internal state which captures the past history (what events have occurred), and that determines the current state.

(Refer Slide Time: 16:22)



The basic state machine diagram is a graphical representation of the state-based behavior. It represents how the state changes on different event. For example, in the Caps Lock state, if a Caps Unlock key is pressed, goes to the default state where the lowercase key is transmitted to the computer when a keypress occurs and so on.

(Refer Slide Time: 16:57)



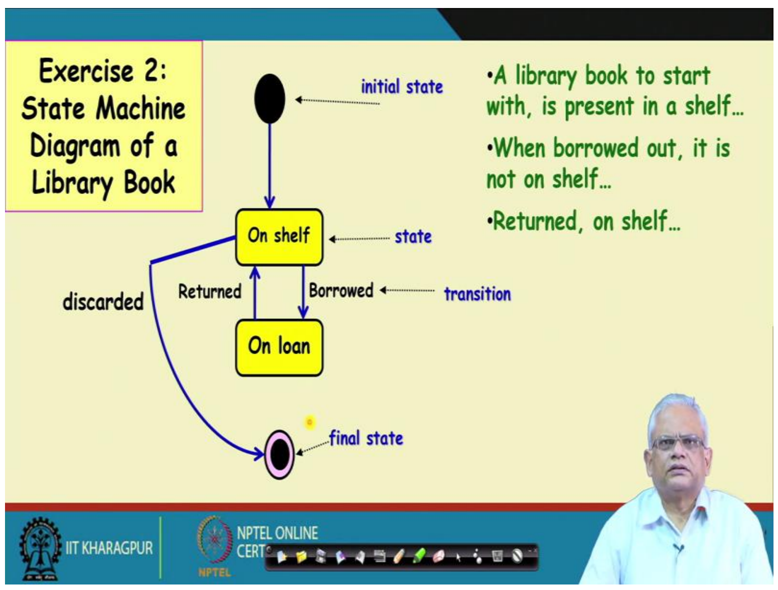
Now, let’s try to draw a state machine diagram of a keyboard (in the above slide). In the Default state, when we press any key, lowercase ASCII code is sent and once the Caps Lock key is

pressed, the uppercase ASCII code is sent to the computer. It is the same example which we have seen so far. Any key is pressed, lowercase ASCII key code is sent. If you press the Caps Lock key, uppercase code is sent and if you press the Caps Unlock key, then the lowercase key is sent from that time onwards and so on.

This is the one of the simplest finite state automaton. Here the default state is represented as a darkened circle. The open arrow is a transition and the state are represented in rectangle with circular edges. The model start with the keys is in default state where any keypress occurs, it sends lowercase code. But when the Caps Lock key is pressed it goes to CapsLocked state. In this state, any key is pressed, it sends the uppercase code. This keeps on doing that until the Caps Unlock key is pressed and so on.

It is a straightforward diagram. We could easily identify that there are two states, one in which lowercase ASCII code is sent and one in which uppercase ASCII code is sent and we have the transition defined among them and the action in each state when any key pressed.

(Refer Slide Time: 19:34)

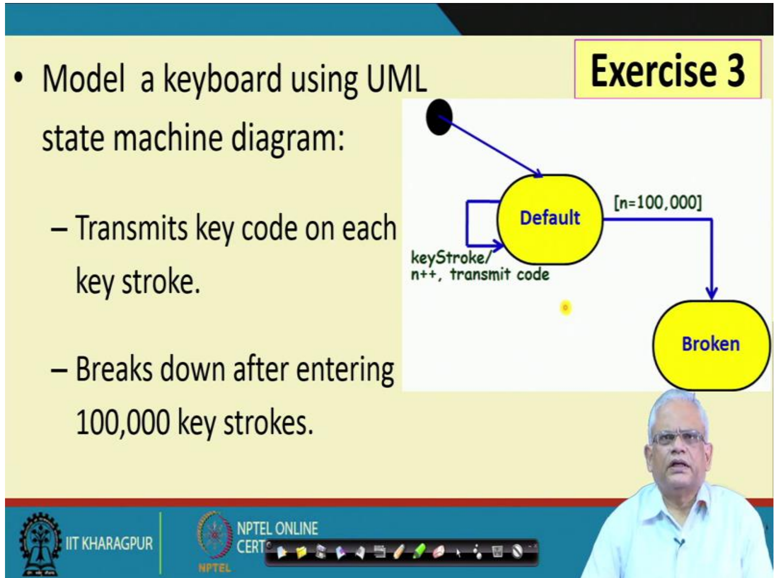


Now let’s do a slightly more complicated diagram. This is about a library book (in the above diagram). In this model, a library book to start with is present on the shelf. Now, when borrowed out, it is not on the shelf and when returned, it is on the shelf. How do we draw the state machine diagram for a book? Initial state represents with a filled circle and then it start with present on the

shelf, that is the first state and when the borrowed event occurs, it is not on the shelf and when it is returned, it is on the shelf and if it is discarded, it is ends. End is represent using final state.

To start with which is the initial state, darkened circle and then a pseudo transition and move to shelf state and keeps on staying on the state, on the shelf until the borrowed event occurs. The name of the event is annotated on the transition. Once the borrowed event occurs it is on loan or it is not on the shelf and when it is returned, it goes back on to the shelf and if it is discarded by the librarian, then the behavior ends and we show this in the form of a final state. We have two circles with the inside one solid circle to represent final state.

(Refer Slide Time: 21:54)



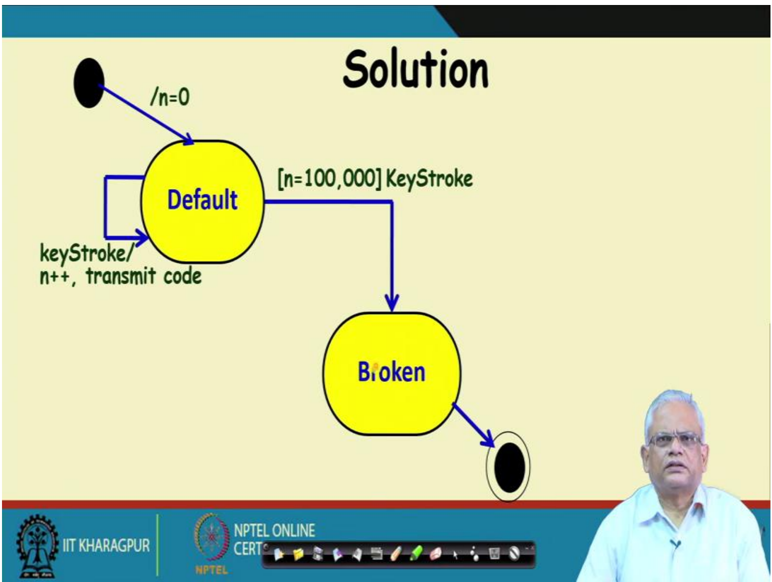
Now, let's do one more exercise (on the above slide). Which is again about a keyboard which transmits key code on each keystroke. Breaks down after entering 100,000 keystrokes. We are trying to draw simple state machine diagrams based on our past experience in drawing state machine diagrams and here it is just slightly more complicated problem than what we have been doing. In the last example about the keyboard we saw, there the behavior was with respect to the Caps Lock key and Unlock key. It used to transmit lowercase key and then as the Caps Lock is pressed, it used to transmit the uppercase key code. The state model was simple but it is slightly more complicated here that we have gained the keyboard which we are trying to model the behavior of the keyboard.

On each keystroke it transmits a key code, the ASCII key code to the computer, but then after entering 100,000 keystrokes, the keyboard breaks. How do we model this?

Actually, we cannot model with the type of FSM. We need slightly powerful way of doing it. We need a variable (‘n’) to represent it. In each time a keystroke occurs, ‘n’ is incremented and also the code is transmitted.

This kind of state machine is slightly more powerful, where we have a state variable that is annotated on the state machine and here, we have the event as the keystroke and on the event the transition occurs from one state to another state at the same time, the variable is incremented. If ‘n = 100,000’ then it just goes to the broken state. So, we have a more powerful state model here where we use a state variable and this is called as the extended state machine.

(Refer Slide Time: 25:20)



So, here is the complete solution (in the above slide). To start with the variable ‘n’ is set to ‘0’ and on each keystroke, n’’ is incremented and the code is transmitted and when n = 100,000, then it goes to the broken state.

So far, in this session we have seen that the state machine model is a powerful modeling tool and every application has two types of objects, one is modeless, where the state behavior is very trivial, it just continues to stay in the same state. We do not need state machine diagram for the such objects. But then for the stateful objects, which of significant state behavior, we need state machine diagrams and we are trying to develop some diagrams based on our past knowledge on finite state machines.

We have looked at three examples. The first two examples are very straightforward and the third example about the keyboard where the key after 100,000 keypresses, the keyboard breaks. For this, we needed a slightly more powerful mechanism that is an extended state machine where we have an explicit state variable which keeps track of some information.

We are almost at the end of this lecture. We will stop here and we will continue in the next session.

Thank you.