

Software Engineering
Prof. Rajib Mall
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

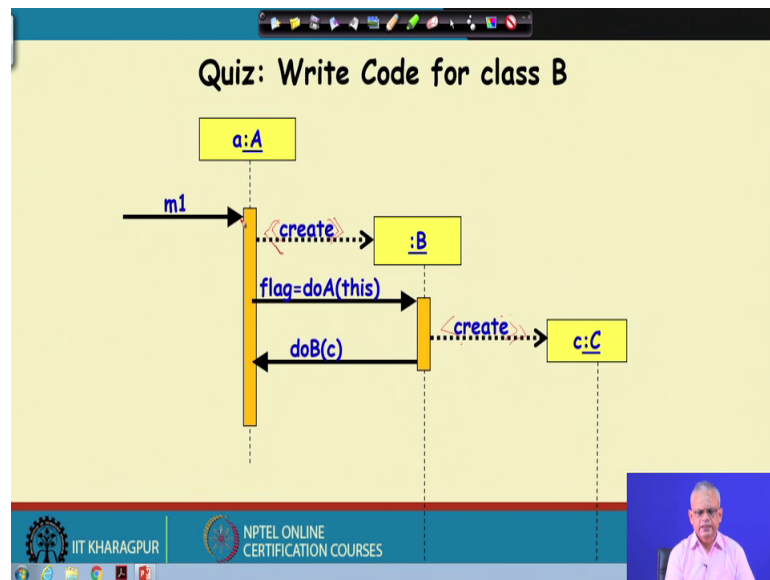
Lecture - 39
State – Machine Diagram

Welcome to this lecture in the last lecture, we had discussed about the interaction diagrams. We said that there are two main types of interaction diagrams, the sequence diagram and the collaboration diagram. The sequence diagram captures how the objects interact during execution of a use case? And, we also said that the sequence diagram plays a very important role in a design process any design process, sequence diagram as a very important role, the role is that it helps in identifying which classes have which methods that we call as method population in the classes.

To start with the design process we identify the classes that exist and later we identify the methods and then we identify the attributes. So, the sequence diagram has a very important role of identifying, which classes should support which methods. And, then we saw the collaboration diagram is automatically derivable from a sequence diagram, and most case tools they just display the collaboration diagram based on just a press of a button. The main purpose of the collaboration diagram is that it shows the association relation between different classes.

Now, we know that based on the sequence diagram we can also write the code for a method, because we know that when the method is invoked on a class what it needs to do? It might call a self method or call another method some argument etcetera and therefore, some code also gets developed for different classes not only that the method prototypes are developed for every class, but also the method could gets developed skeleton code. Now, let us do it through a small exercise and that is simple and of given a diagram would be able to develop the code, because it is very very simple let us see that let us see that we have this sequence diagram.

(Refer Slide Time: 02:59)



There are 3 objects which participate in the sequence diagram, method m 1 is first invoked under object a the class capital A. And, then it invokes the create I should have put the stereotype here; which I can put now just put the stereotypes yet to be correct. It invokes the create on the B class and object anonymous object gets created here and then it calls the do A method on the B class sorry anonymous object of the B class with argument this and remember this is the small a itself. And, then it returns a flag and in turn the object of the B class it creates an object of C class called as c name of the object that is created is C, and then it calls the do B method of the A class with argument C.

So, what will be the methods for different classes and can we write some of the code here. We know that m 1 is a method of a class, do A is a method of the B class of course, create that is constructed for the C class. Now, the question is that can we write the code for class B. So, that is for do a method. So, we know that class B contains do A method it takes an object of class A as argument and then inside the method it will create the class C. So, it will call the new operator in java for the class C and then call do B method of the class A.

(Refer Slide Time: 05:33)

```
Quiz: Ans

public class B {
    ...
    int doA(A a) {
        int flag;
        C c = new C();
        a.doB(c); ...
        return flag; }
}
```

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, if we write the code neatly we will have class B and then it has the do A method, it returns a flag integer and then the first it creates a C object and the name of the object created is small c. And, it calls the a object the do B method of the a object with c as the parameter and then it returns a flag.

(Refer Slide Time: 06:01)

State Machine Diagrams

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, not only that based on the sequence diagram, you can populate the methods, but some skeletal code for the different methods can also be created. Now, let us look at the state machine diagram this is also an important diagram, even though it is not developed

for every class in every problem, but then if the classes do have significant states then we so, that in the form of a state machine. And, also the state machine helps us helps to generate some code, because our idea is that once we do the design, we should have some code written for us. Most, case tools do that good case tool generate lot of code and we just need to fill few of the code there, and we saw that how the interaction diagrams help us to generate the code.

Now, let us look at the state machine diagram and see that if the classes some classes have significant states, then we can develop the state machine diagram and at the same time the code for that class will get generated. We will see the nitty gritty of the state machine class sorry the state machine diagram and also see the code that a state machine diagram can generate.

(Refer Slide Time: 07:47)

The slide is titled "State Chart Diagram" with "Cont..." to its right. It contains the following text and diagrams:

- State chart avoids two problems of FSM:
 - State explosion
 - Lack of support for representing concurrency
- A hierarchical state model:
 - Can have **composite states** --- OR and AND states.

Handwritten diagrams in red ink illustrate these concepts. The top diagram shows two states, S1 and S2, with transitions between them, representing state explosion. The middle diagram shows two separate state machines side-by-side, representing concurrency. The bottom diagram shows a composite state containing two sub-states, representing OR and AND states.

The slide footer includes the IIT KHARAGPUR logo, the NPTEL ONLINE CERTIFICATION COURSES logo, and a small video inset of a man in a white shirt.

The state machine diagram as it is called in UML 2.0 is based on the state chart diagram. The state chart diagram is a refinement of the finite state machine we all know the finite state machine. And, the state chart diagram was proposed by David Haril long back maybe 30 years back, to handle two problems of the finite state machine. Even though we use finite state machine for small problems, but if we use it in object orientation in modelling classes systems and so on and in many other application it suffers from two main problems.

The FSM suffers from two main problems one is called as the state explosion, that is the number of states becomes too many 100 1000s, and it becomes very difficult to draw the diagram on a page or a on a screen and also to understand the diagram. The, second problem is that the state machine is just a sequential diagram; states get activated one after other you do not have concurrent states. And, the state chart diagram over comes these to basic problems of the efficient and we will see that it is an elegant formalism, and UML use the state chart diagram and call state as the state machine diagram.

The main improvements to the finite state machine is a hierarchical state, a finite state machine is a flat state machine, if you draw an efficient let us say we draw it like this that we have 2 states and then there are some transitions. It is a flat state machine let us say State S 1 S 2 and then some transition based on some event, some transition takes place and we have some action e 1 causes this transition e 2 causes this transition.

But, in a state chart diagram we have hierarchical state's that is a state here can have further states. So, even though the top level we have this state machine, but we can look at any state and that also contains the state machine. And, this may contain state machine, but we can look at any state and that also contains a state machine and this may contain state machine. So, there is a hierarchy and if you look at this state in the second level it might also contain a state machine.

And, this is the one mechanism hierarchical state to handle the problem of state explosion, earlier as the number of state variable increase the state efficient became extremely complex, number of states increased exponentially with the number of state variables. We will just see that with an example that how would the state's increase exponentially with state variable.

But, here we address the problem of state explosion by having a very simple diagram with few number of states and transition, but then we use the abstraction mechanism, we said the beginning of our discussion on software engineer that wherever things become very complex we use either abstraction or the decomposition to handle that. And, here we use the abstraction mechanism and that is done through hierarchical state model.

So, instead of making the diagram flat and very complex we have a hierarchical diagram and the top level is simple and the complexity are slowly added over the hierarchies. And, here for representing the concurrency we will have composite state, that is within

one state we can have a concurrency that is the object can exist in two different states within one state let us we will look at that.

(Refer Slide Time: 13:20)

Robot: State Variables

- Movement: On, OFF 2
- Direction: Forward, Backward, left, Right
- Left hand: Raised, Down
- Right hand: Raised, down
- Head: Straight, turned left, turned right
- Headlight: On, Off
- Turn: Left, Right, Straight

How many states in the state machine model?

$2 \times 4 \times 2 \times 2 \times 3$
 $\times 2 \times 3$
 $= \underline{384}$

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES

Let us first look at an example and understand the state explosion problem in a efficient. Let us look at this robo and this robo can make movements the first state variable. Let us try to compute the number of states in modeling a robo is a behavior. Let us assume that there is a switch to have the movement of the robo on or off, the robo can move if the movement is on movement can be off also. So, there is a state can be that movement on and movement off.

Now, there is another state variable called as a direction and the direction can be forward backward left and right. So, this state variable direction can take 4 different values and these are 4 states. So, based on this two we will see that the state for movement there are only 2 states that is the robo movement on, robo moment off. But, now with the second state variable we have 8 states that is robo movement on with direction forward, robo movement on with direction backward, robo movement on with direction left, robo movement on with direction right, and similarly robo movement of with forward backward etcetera. So, with those 2 state variables we have 8 states.

Now, that can be further states. For example, the left hand raised or down. So, these are 2 states. Now, the number of state variable becomes 2 into 4 into 2, because for each one here each state here we can have 2 cases that left hand raised the down. Similarly, the

right hand raised the down. Similarly, will have head straight or turned left the turned right, you can have the head light turn on and off and we can have the turn left right or straight.

So, it may be proceeding straight or it may be turning to left or right. So, to model this thus robo, how many states we need in a simple finite state machine model, we will need to that is 2 into 4 into 2 into 2 into 3 into 2 into 3. So, this becomes 8 16 32 96 192 and this becomes 384. So, just imagine efficient you draw to model this and it has 382 states. If, you even if you spend a month trying to understand the behavior it becomes because very complex. But using the state chart notation we can draw a very simple diagram to represent this the state machine for the robo, because you will use the concurrent states and the number of states will drastically come down to only a few.

(Refer Slide Time: 17:39)

The slide is titled "Features of Statecharts" and lists two major features: nested states and concurrent states. It also lists other features like history state, broadcast messages, actions on state entry/exit, and more. Two diagrams illustrate these features: a nested state diagram with state S1 containing state S2, and a concurrent state diagram with two parallel state machines (Move ON/Move OFF and Move Forward/Move Backward) separated by a dashed line.

Features of Statecharts

- Two major features are introduced :
 - nested states
 - concurrent states
- Many other features have also been added:
 - History state
 - broadcast messages
 - actions on state entry, exit
 - ...

Nested State Diagrams

Concurrent State Diagrams

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, let us look at the features of the state chart. So, this is the concurrent state machine for the robo we have one thing is that movement on and off, and at the same time we can also set the movement to forward to backward.

Similarly, we can set the state 2 hand left hand raise left hand down and so on. Just see that the state machines are becoming independent and therefore, the combinational explosion that was occurring there is a stopped here in the state chat, because these are independent. The state variables are shown the concurrent state that is the same time it can have the state change occurring from forward to backward and on to off and so on.

The other one is the hierarchy is the nested states the state S 1 contains another state machine S 2 and so on. And of course, there are few other features on a state chart like history state, broadcast message, actions on state entry, state exit etcetera these part will not discussion this a chart.

(Refer Slide Time: 19:08)

State Chart Diagram

- Elements of state chart diagram
- **Initial State:** A filled circle
- **Final State:** A filled circle inside a larger circle
- **State:** Rectangle with rounded corners
- **Transitions:** Arrow between states, also boolean logic condition (**guard**)
- UML extended state chart is called state machine

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, to draw state the diagram with draw an initial state, which is a filled circle will draw an initial state, to start with it will come to this state let us write S 1 and then so, by default to start with it will start in S 1, maybe go to S 2 based on some event e 2. And, there is a final state which is a filled circle inside another circle. Is a rectangle with rounded corners these are the state representation and arrows between states these are the transition. And we can have Boolean logic here the show that based on some condition the transition occurs even if by event e 2 occurs transition will not occur unless the condition holds true.

(Refer Slide Time: 20:24)

The slide is titled "How to Encode an FSM?". It lists three main approaches: "Doubly nested switch in loop", "State table", and "State Design Pattern". Handwritten notes include "while() {", "switch() {", "case 1: switch() {", "e1", "e2", "case 2: switch() {", and "3". A diagram shows a table with three columns labeled "S1", "S2", and "S3" and one row labeled "e1".

How to Encode an FSM?

- Three main approaches:
 - Doubly nested switch in loop
 - State table
 - State Design Pattern

Handwritten notes: while() {, switch() {, case 1: switch() {, e1, e2, case 2: switch() {, 3, 3.

	S1	S2	S3
e1			

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, let us address this question, that if we developed the state machine representation let us first look and efficient. And, you can do for state chart that given an efficient how can code be generated from that? Because, an efficient or a state chart is used to represent the state model of a class now, can we generate some code for the class? Actually, it is very simple if you understand it let us look at that, there are 3 main approaches one is called as doubly nested switch in loop.

So, the meaning of this is that we have one switch in C and inside switch we have let us a case 1 and inside that we have another switch event and then we have e 1 e 2 etcetera. And, similarly case 2 that will also have a switch so, there is a doubly nested switch. So, there is a switch here inside another switch many switches here between nested switches and then this is within a loop while some condition.

We will get a code like this is the first approach, we will see that it is extremely simple give an FSM, you can easily draw this develop this code. We can actually automatically generate this code. The second one is state table this is also straight forward just use the table and write the states and for each S 1 S 2 S 3. And based on the different events if it is in some state what occurs etcetera we can also write that this very simple will discuss that, but the state design pattern will not be discussing in this lecture.

(Refer Slide Time: 22:46)

- **Doubly nested switch in loop:**
 - Global variables store state --- Used as switch
 - Event type is discriminator in second level switch
 - Hard to handle concurrent states, composite state, history, etc.
- **State table:** Straightforward table lookup
- **Design Pattern:**
 - States are represented by classes
 - Transitions as methods in classes

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, there are 3 principal base the doubly nested switch in a loop, we here we use a global variable to store the state. And, switch on the state and the event type is the discriminator in the second level switch, but here it is bit harder to handle concurrent states composite state history etcetera. And, the state table is a straight forward table lookup, the in the design pattern state design pattern to have a as many classes at the states and the transitions are methods in the class will not be discussing this.

(Refer Slide Time: 23:30)

```
int state, event; /* state and event are variables */
while(TRUE){
    switch (state){
        Case state1: switch(event){
            case event1:
                state=state2; etc...; break;
            case event2: ... state=state1, ...
            default:
        }
        Case state2: switch(event){...
        ...
    }
}
```

Doubly Nested Switch Approach

Diagram: A circular state transition diagram with nodes labeled state1, state2, state3, state4. Transitions are labeled event1, event2, event3, event4.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

First let us see the doubly nested switch approach here we have the states as a global variables; the states as a global variables here and then we switch on the state. This is a loop here keep on doing this and as an event occurs first switch on the state depends on which state the same event made have different actions. If it is in state 1, again we switch on the event here and depending on the event to let say, you do something that state becomes current state becomes state 3 and print f something and break. And, if a event 2 occurs when it is in state 2 maybe we have state becomes event 2 event 2 maybe state becomes state 1 get some input from the user do something and then break.

So, this is a simple translation we will just look at the state machine and we see the state variable that is used to write that here the form of the specific type of the state whether it is a integer and so on. And, then we look at the events here e 1 e 2 e 3 and you look at the states' S 1 S 2 S 3, and then switch that case S 1 you the event e 1 occurs then transit S 2. And this is a simple state machine and that is it, and if it is an S 2 if the only switches based on e 2 and their the state is S 2 S 3 and in S 3 it is transistor S 1. So, this is a very simple code and the finest sorry the state table is also very simple you can also look at that.

(Refer Slide Time: 26:07)

```
int state, event; /* state and event are variables */
while(TRUE){
    switch (state){
        Case state1: switch(event){
            case event1:
                state=state2; etc...; break;
            case event2:...
            default:
        }
        Case state2: switch(event){...
        ....
    }
}
```



So, here is a doubly nested switch the first one is switch on the state, the second one is switch on event and depending on the state the same event may have different actions.

(Refer Slide Time: 26:23)

State Table Approach

- From the state machine, we can set up a **state transition table** with a column for the actions associated with each transition

Present state	Event	Next state	Actions
Light_off	e1	Light_off	none
	e2	Light_on	set red LED flashing
Light_on	e1	Light_on	none
	e2	Light_off	reset red LED flashing

 IIT KHARAGPUR  NPTEL ONLINE CERTIFICATION COURSES

The state table approach is also extremely straight forward here, we have the present state and the events that are recognized here and then what action takes place. So, which will be the next state, if light is off this is the present state and it reacts to e 1 and e 2 if e one occurs when light is off.

So, there is no action basically for e 1 e 2 than light becomes on and set the red light flashing. Similarly, for every state we write the specific events and then what are the state change and what are the actions? So, it becomes easy to write the program whenever there is a it is in a state we just look up and the state table and set the state next state based on this and then we perform the action and that is it the code becomes very straight forward.

So, we are almost at the end of this lecture.