Object-Oriented System Development Using UML, Java and Patterns
Professor Rajib Mall
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur
Lecture 27
Sequence Diagram - II

Welcome to this session.

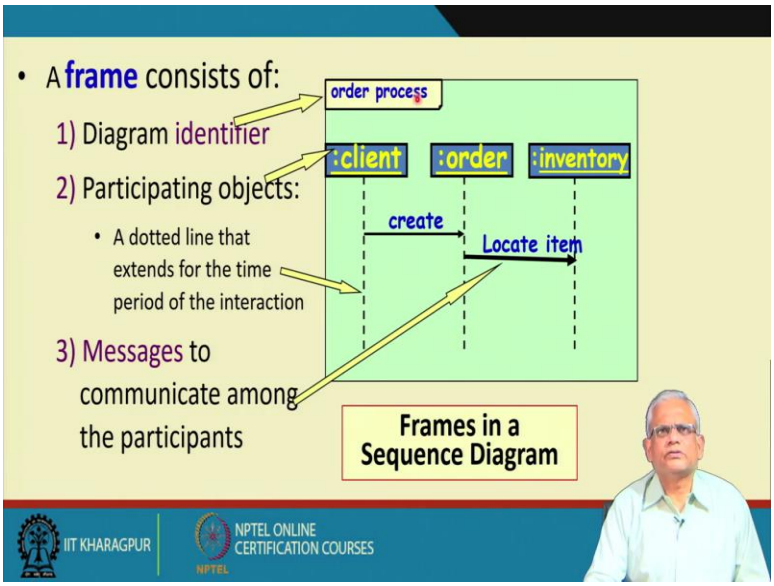
In the last session, we had remarks that sequence diagrams are very important. They help to populate the methods in the classes automatically and if you are doing it manually, based on the inspection of the sequence diagram, we can write down which classes has which methods and then the sequence diagram can help us to generate substantial amount of code also. The case tools automatically do that once you draw the sequence diagrams. The case tools not only allocate the methods to the classes, but also good amount of code can get generated.

And of course, using the case tools we did not get the complete code for a method, but then we can fill in the remaining code and refine it a little bit. And the last session towards the end, we are just remarking that since UML 2.0 the frames have been introduced, the frames improve the expressive power of the sequence diagrams and also make it much more understandable and elegant.

Some aspects which you could not express at all in UML 1.x sequence diagram were able to express that in the UML 2.0 using the frames, and also it makes the different interactions that occur during a use case explicit which make easy to understand.

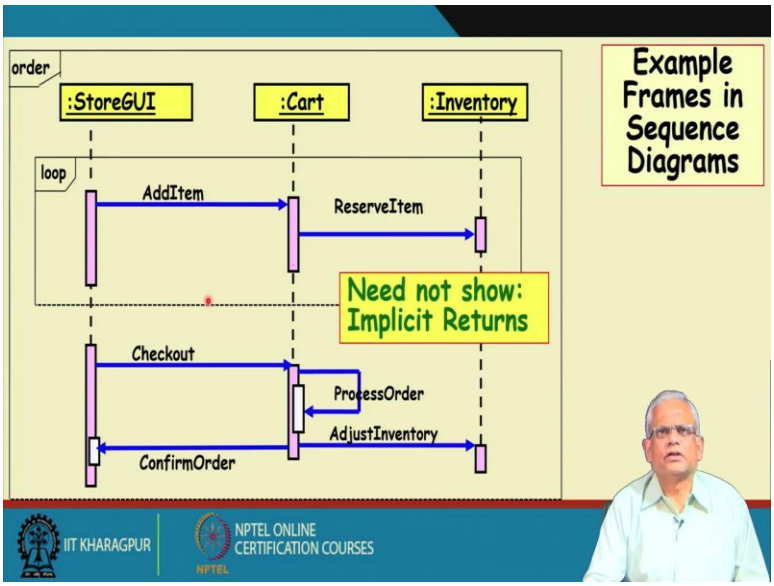
Now, let's proceed with that we will discuss about the frames and a few issues with the sequence diagram. We will look at briefly at the collaboration diagram and then we will look at the activity diagram.

(Refer Slide Time: 2:30)



In the sequence diagram, the new concept which was introduced in UML 2.0 are the frames. A frame is a rectangular region and at the top left corner. We have an annotation here, which is the diagram identifier (shown in the above slide) for the top-level diagram, but then the top-level diagram can contain several other frames and these will contain the operators and their operands. Due to the frames, it becomes easy to associate which use case it captures or represents the interactions in which use case.

(Refer Slide Time: 3:24)



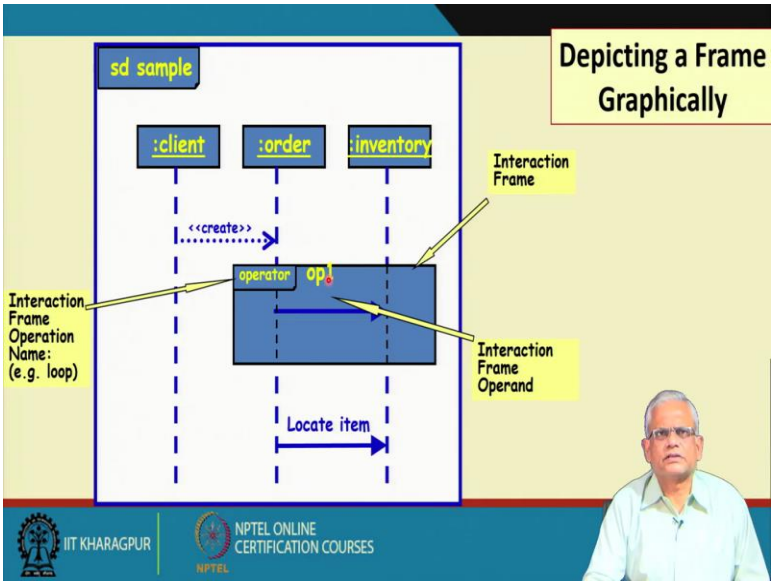
Now, this is a frame (in the above slide) and just observe that it embeds another frame. And here, this is the operator loop and we might write an argument here saying that for all items AddItem method call on Cart object. So, here the 'order' is the use case which is the top level frame (represent by big rectangle in the above slide), it contains other frames and here initially we keep on adding items because this is in loop. We keep on adding item and reserving item, which gets reserved in the inventory for the specific buyer in the e-commerce site.

The store GUI is used by the customer to select an add items, as he adds items to the cart and confirms the reservation of the item, then in the inventory the item is reserved for him. And just observe here that he might select multiple items. And then he might reserve some of them or all of them, and that is represented using the loop. And also, observe that once an item is added to the cart, it calls the reserve item on the inventory. And that occurs again and again due to the loop each time add item and reserve item. But in UML 1.x, we could not do this using loop. In UML 1.x we write *AddItem and *ReserveItem, then is it first all items are added to the cart and then all the items are reserved in the inventory? No, that is not the way we intended, we intended that each time an item is added, it should be reserved. And this simple thing could not be expressed in UML 1.x which you can represent using the loop. So, that each time one item is added and then as it is added, it is also reserved in the inventory.

So, the loop frame is a powerful construct which increases the expressive power of the sequence diagram. And here, after some time, the customer checks out and then ProcessOrder is called and then it adjusts the inventory and confirms that the order has been shipped. But one thing we need to mention here, that even though we can use dotted arrows it's a valid syntax, it unlocks the object at that time, the activation box ends once the dotted arrow is received.

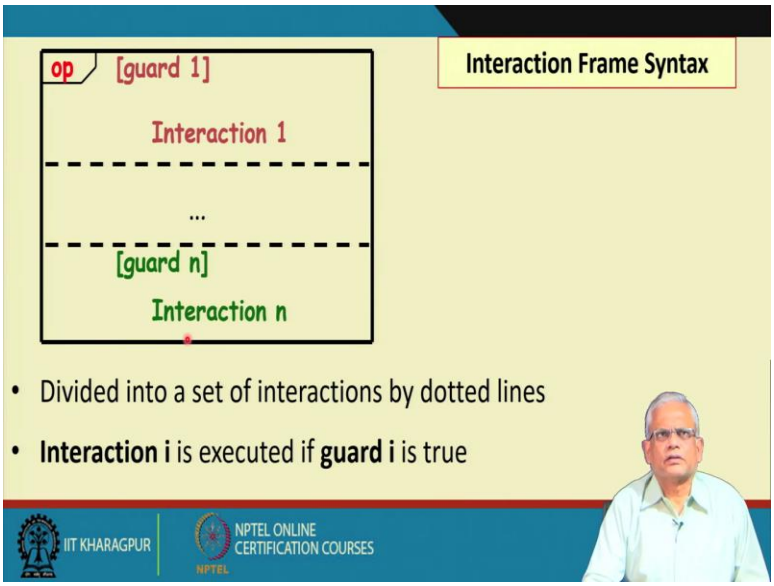
But then it unless there is a return value which is used explicitly, we don't show the return arrows. It just makes the diagram cluttered. We will just delete those return arrows that makes the diagram much more elegant. The returns are implicit, unless there are some value. Return value is explicitly needed when there is some return value.

(Refer Slide Time: 7:46)



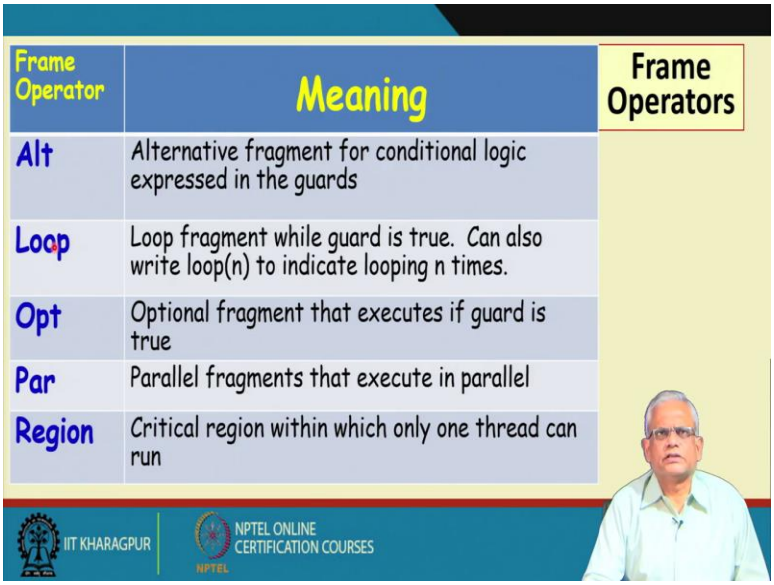
Now, this is another frame in the sequence diagram (in the above slide) and the syntax here is that the big rectangle 'sd sample' is the top-level frame, which is the name of the diagram and this is the operator frame named as 'operator' and one operand here is 'op1' and the operator is written on the left corner. The operand may be a loop operand, and then for all objects it does something.

(Refer Slide Time: 8:27)



We can also have selection like, if then else or a switch. We can write 'opt' for representing 'if' or we will write a specific operand which will indicate that if the [guard 1] is true then do this Interaction 1. If [guard 2] is true, then do this Interaction 2 and if [guard n] is true, then do this interaction. So, that is more like a switch.

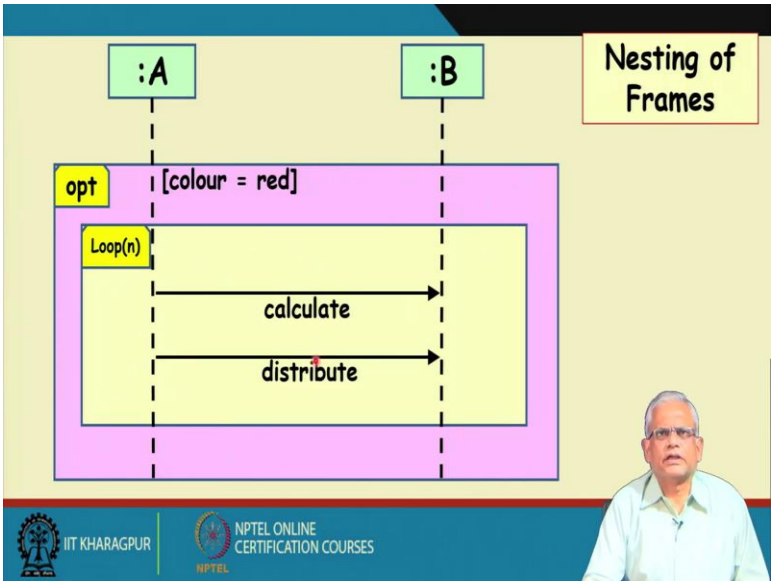
(Refer Slide Time: 9:15)



In the above slide different operators shown. First one is alt, alt is like if then else here there are two alternate fragments depending on the true or false value of the conditional logic either the first fragment or the second fragment in the frame is executed. The loop fragment executes until the guard is remains true you can just write a guard [n > 0] or something or we can also write loop(n) to indicate n time looping.

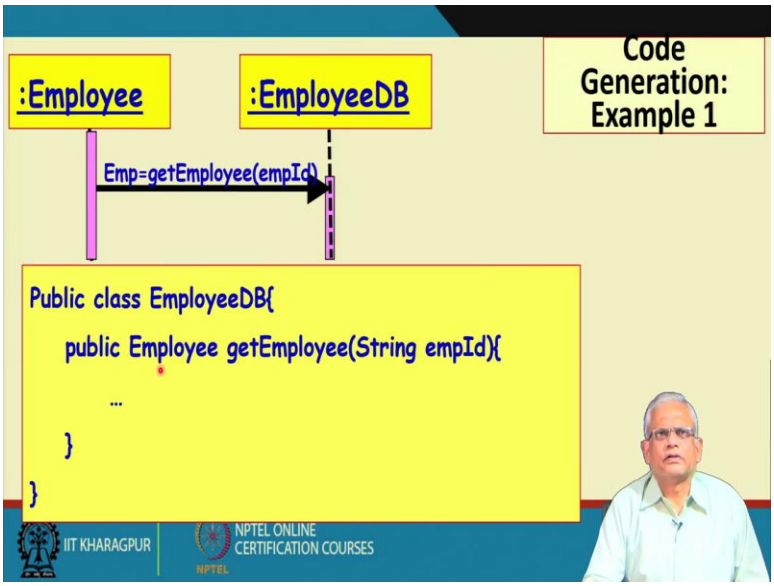
We can use a opt fragments or the optional fragment here it is, like if condition do something. Alt is like if then else and opt is like if the condition is true, then do, we don't have the else part here. The par is the parallel fragments that execute in parallel region. At the critical regions within which one thread can run is represented by Region. Region we don't use but alt, loop, and opt we will use.

(Refer Slide Time: 10:52)



We can nest frames (in the above slide), the first one is opt [colour=red], then this behaviour or this interaction is carried out. The calculate method on the B is called, if colour is red and this occurs for n times that the calculate and distribute is called.

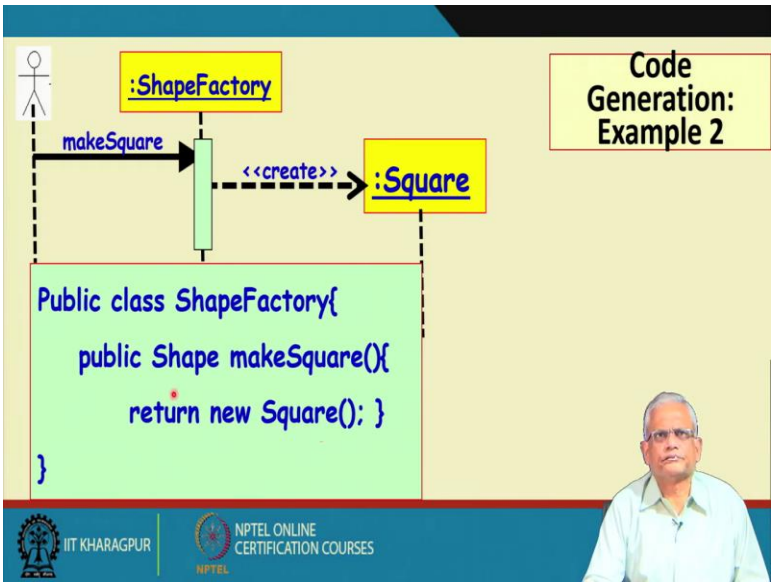
(Refer Slide Time: 11:26)



Now, let's look at some more code generation examples (in the above slide). Let us say we have an employee object and it makes a method call on the employeeDB object and get employee ID and gets the employee details.

How do we write the code? From the above diagram, the code that can be generated is for the employedDB and it must support the get employee. The employeeDB class must support the getEmployee, and it takes the argument employee ID (empId) and returns an employee. This much code is intuitive, can be generated from the sequence diagram.

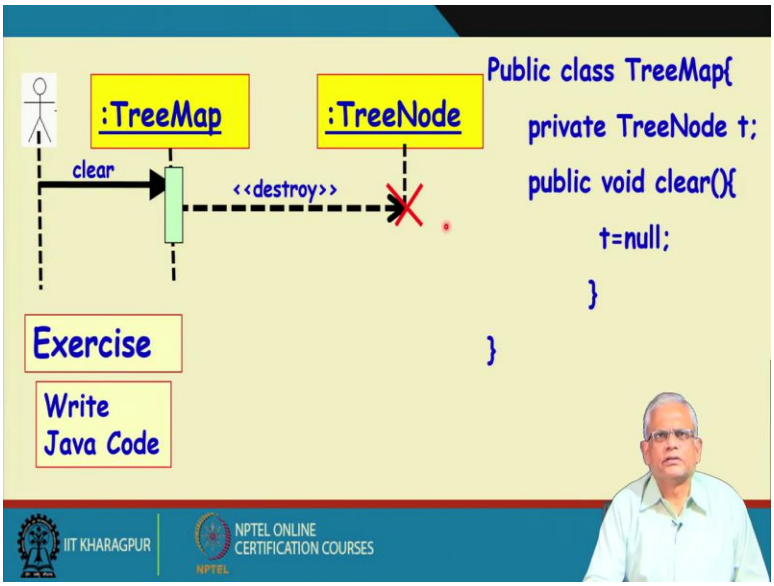
(Refer Slide Time: 12:27)



Now, let's look at another example (in the above slide) that the user calls the makeSquare on the shapeFactory object and there is the methods call occurs, becomes active and then ShapeFactory creates a square object.

Now, based on this diagram, what is the code we can generate? One is that the shape factory class must support the makeSquare method and in the makeSquare method, it must call the new Square. So, here the shape factory it supports the make square and it creates a square object and returns the ID.

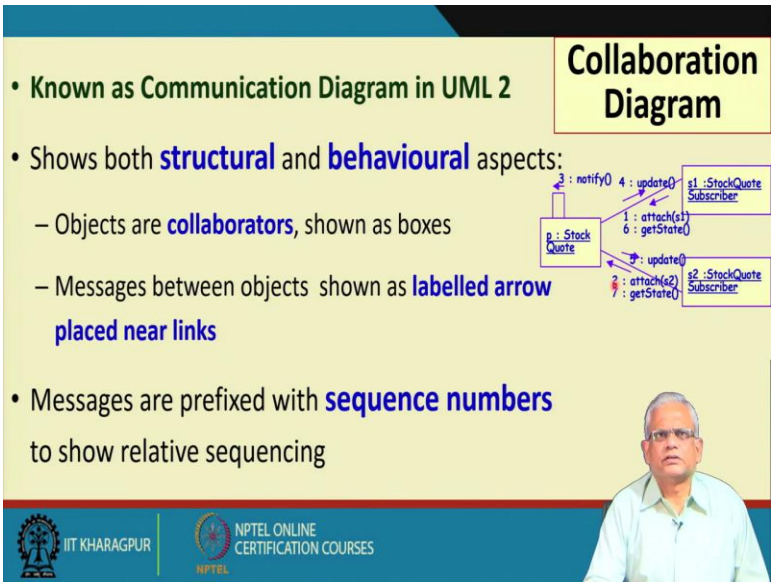
(Refer Slide Time: 13:28)



Now, how do we write the code for this (in the above slide)? Here the user calls the clear method on the tree map object, the tree map object becomes active and it destroys the tree node.

How do we do that in java? How do we destroy a tree node, in the tree map object? We had said that to destroy in Java it goes out of scope or we may assign null to the corresponding reference ID so that the reference ID is permanently lost or is deallocated. So, here are the tree maps we have a TreeNode as private and then as the clear() method is called it sets ‘t = null’ and this causes the tree node to be deallocated or destroyed.

(Refer Slide Time: 14:40)



Now, let's look at the collaboration diagram.

So far, we had looked more or less the details of the sequence diagram that we needed. Let's now look at the collaboration diagram. The collaboration diagram is now known as the communication diagram since the UML 2.0. The sequence diagram is a behavioural diagram where is a collaboration or the communication diagram represents both structural and behavioural aspects.

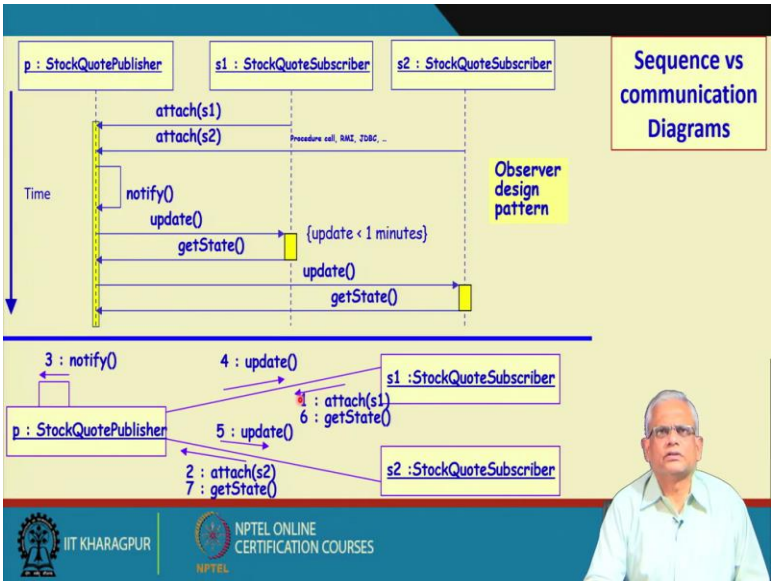
This is a collaboration diagram (in the above slide) and here by looking at the diagram, we can know which object is associated with which another object. And using the ID stored we said that an association is implemented by storing the ID of the corresponding object and then once the ID is stored it calls methods of the corresponding object and in what sequence the methods are called are annotated here using numbers.

The first one is attach the StockQuote calls that attach object and then the StockQuote subscriber calls the attach on the stock quote which is number 2. So, on object s1 first attach called, and on object s2 second attach called to StockQuote. The third one (number 3) is the StockQuote calls notify() which is a self-method, and the fourth (number 4) one is it calls the update() on the StockQuote and also calls the update() on the stock quote on the s2 object and so on.

So, here as you can see that the collaboration diagram or the communication diagram represents not only the behaviour that is in what sequence the messages are called, but also some structural aspect like which classes are associated to whichever class. In our previous example, it becomes visually clear that the StockQuote is associated with the StockQuote subscriber. And here the messages are prefixed with a number that indicates the sequence.

Remember that in the sequence diagram there is an implicit notion of time from top to bottom. But here, these are numbers here.

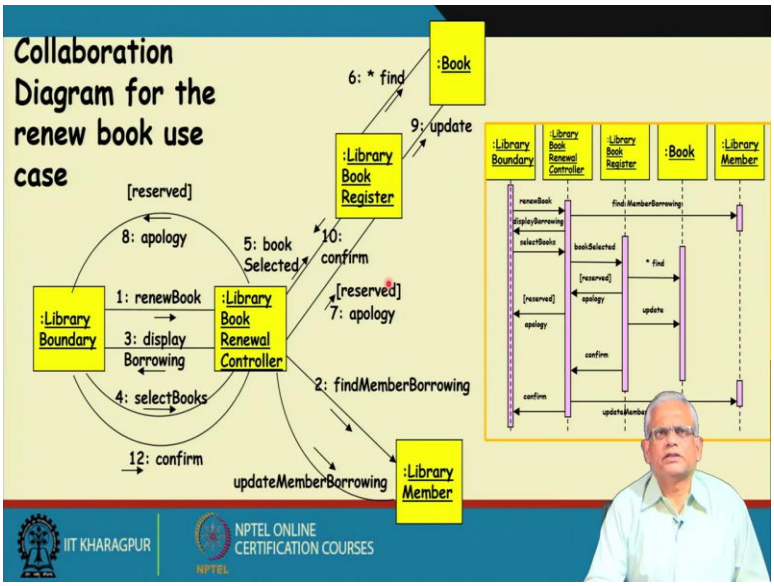
(Refer Slide Time: 17:26)



If this is a sequence diagram (in the above slide), we can generate the corresponding collaboration diagram very simply. Let’s observe how we will do it. And incidentally, this simple example is an observe design pattern. Later in this course, once you start discussing about design patterns, we look at the observer design pattern and there the subscriber's here first attach method on the publisher. And the publisher calls the notify() self-method and then the publisher calls the update() method on the subscriber's and then the subscriber's call the getState() method. To generate the communication diagram from the sequence diagram, we first write all the objects that are appearing at the top in the sequence diagram, we write them and then we read from top to bottom that is time zero. It calls attach on object s1 we represent that in collaboration diagram, that the stock quote subscriber calls the attach and give the ID s2, we draw an arrow from StockQuoteSubscriber to StockQoutePublisher.

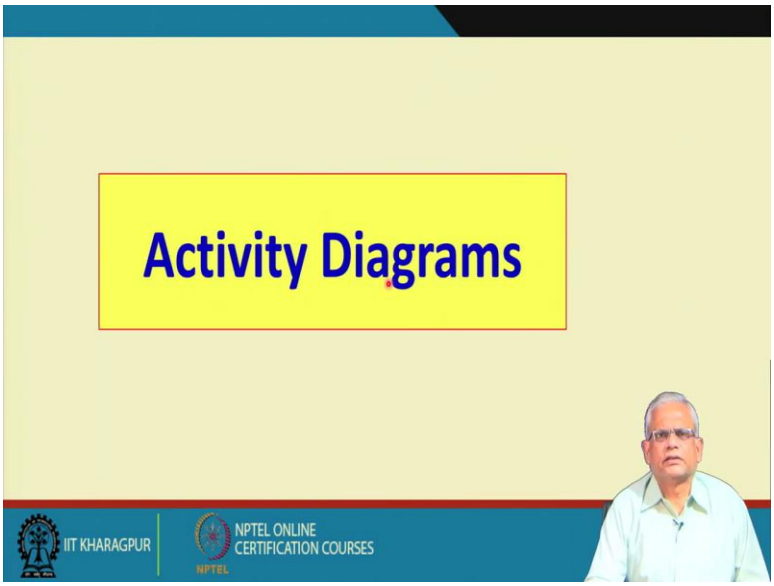
We draw a line between this StockQoutePublisher and StockQuoteSubscriber if there is a method call between these two. This shows that they are associated and write number one attach s1. And then we read the next one attach s2. And we joined this two and then write to attach s2. And then notify() is called on the StockQoutePublisher and numbered as 3 and so on. The time order in which these interactions occur, we write explicit numerical values on this diagram, as you could have seen here that it is really straightforward to develop the communication diagram from the sequence diagram. And one thing that the collaboration diagram explicitly represents which class is associated with which class, which we are not very explicitly represented in the sequence diagram. And also, here we write the numerical values to indicate the order in which the message interchanges occurred. So, generating the communication diagram is no great deal, first write all the objects and then read the diagram from top to bottom, the first message we attach give number one, and then one column attach. And then we draw diagram and draw all the arrows to show the which class is associated with which class. The arrow is here Synchronous arrow. In a case tool, case tool effortlessly generates the communication diagram from a sequence diagram by one click. But if we have to manually do it, then also, it is not very effort intensive, from any sequence diagram we can easily do the communication diagram.

(Refer Slide Time: 21:29)



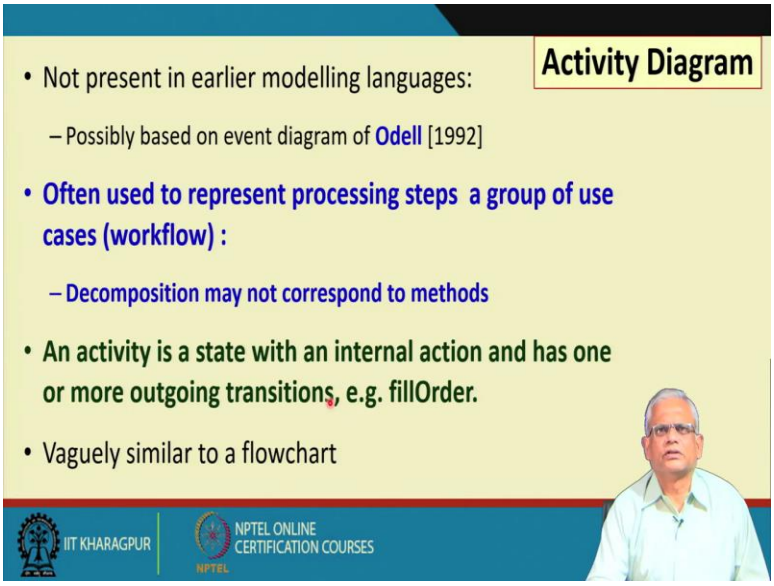
This is another example here (in the above slide). A sequence diagram shown in the above slide and the corresponding communication diagram. We represent all the objects in the collaboration diagram and write the numbers here from the top to the bottom. The renewBook method call is the first one which is between the Library Boundary to the Library Book Renewal Controller. So, Library Boundary to the Library Book Renewal Controller the first message is invoked. The direction of the message invocation is written using a small arrow which connects association. As we can observe, we can not only generate the communication or the collaboration diagram from the sequence diagram, but also given a communication diagram, it becomes easy to generate the sequence diagram as well. Any one of the diagrams has all the information to generate the other diagram. Typically, we develop only one of the diagrams and the other is automatically generated.

(Refer Slide Time: 23:05)



The aspects of the sequence diagram that we will need in our design process, we have already discussed those, even though the sequence diagram in the UML syntax as many more notions. We just did not discuss exhaustively all aspects of the sequence diagram. We restricted to only those things which will be needing in our simple examples. For example, the parallel execution, critical sections, et cetera. we did not discuss.

(Refer Slide Time: 23:49)



Now, let's look at the activity diagram. An activity diagram is a model which was not used earlier. We are saying that before UML there were many other modelling techniques OMT, Rumbaugh, Booch, object-oriented software design and so on, and none of them have something like an activity diagram.

So, this is something new here, not present in other diagrams that we could see, but then possibly the event diagram that was present in the Odell's technique somewhat vaguely resembles this. But the activity diagram is not really like the event diagram of odell, the activity diagram is something new in UML did not exist in the earlier modelling techniques.

The activity diagram typically we use to represent the different steps that occurred in a use case that is a workflow. Activity diagram used to represent complex use cases. We might miss some things, and we first developed the activity diagram for a use case and here we document how the steps occurred, some steps may occur in sequence, some steps may occur in parallel, some may require exchange of some objects between two entities, there may be synchronization parallel activities and so on. All these we can represent and therefore for complex use case diagrams, for complex use cases, it is becoming necessary to draw the activity diagram. The activity diagram is useful not only to generate the sequence diagram, but also, we can generate test cases based on this. The different steps we represent in activity diagram need not correspond to methods, actually. But as you will see that many times, there granularity is similar to methods.

An activity if we look at, an activity is a state with an internal action and has some outgoing transition once it completes. For example, fill order is an activity takes some time and in the fill order process or activity some action goes on internally. After one process or activity complete it goes to another activity.

We can think of this as somewhat similar to a flow chart. In a flow chart also, we had a representation for processes or statements. The boxes used to represent the execution of statements or methods or blocks of statement. And then as the execution ends, it used to go to another box and so on. But then the flow charts are of limited expression capability, the activity diagrams are much more sophisticated, as you will find out.

But then there is a vague similarity that the activities are similar to the boxes represented in a flow chart. Activity is actually a state, at which some internal actions occur and at the end of the activity, transition to another activity or to the end occurs.

With this brief introduction to the activity diagram, we will stop here.

We will continue from this point in the next session.

Thank you.