

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

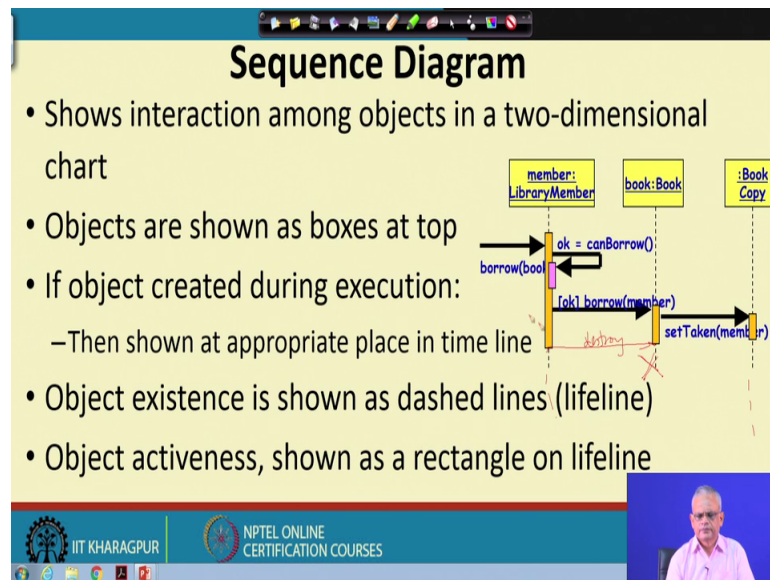
**Lecture - 38**  
**Development of Sequence Diagrams**

Welcome to this lecture. In the last lecture we discussed about how to identify the classes and their relations and also started to discuss the interaction diagrams. The interaction diagrams are a very vital diagram; these are developed for every design. Even for a small system, we want to develop, we need to develop the class diagram and the interaction diagram and the use case diagram.

So, these 3 diagrams are developed for all systems what whether it is trivial or large scale we need this diagrams, but the other diagrams for example, the state chart diagram, activity diagram, etcetera. They are developed for specific cases for example, if the classes have significant number of states we need a state diagram, activity diagram, if a use case is complex or we need to have a overall understanding of a very complex software, we develop activity diagram.

But, interaction diagram is developed irrespective of the problem for all types of problems we need interaction diagram, and we will discuss about the nitty gritty of the interaction diagram and see what purpose they serve in a design. There are two types of interaction diagrams two important types; one is called as a sequence diagram and the other is collaboration diagram. Of course UML 2.0 supports few other types of interaction diagram for example, interaction overview diagram and so on, which we will not discuss in this lecture. Let us look at the sequence diagram and see how we can develop a sequence diagram and what role it plays in a design process?

(Refer Slide Time: 02:25)



The sequence diagram as we discussed in the last lecture, it shows the interaction of the among the objects in a 2 dimensional chart. This is the interaction that occurs when a use case is executed, the objects are shown at the top and if an object is created during a execution these are showed at the appropriate place in the diagram, there is a lifeline for every object.

So, the object exists the lifeline exists and if the object is destroyed we put across on it is lifeline and it does not exist. When a object becomes active it is shown on as a rectangle on it is lifeline. So, this is an example of a sequence diagram just see here that the objects are shown at the top of the diagram, and then we have the lifeline for each object as long as the lifeline exist this dotted line exists the object exists.

If, any time object gets destroyed for example, another object member destroy method on that. And, then we just draw a cross here and stop the lifeline there indicating that the object number exists. So, with this call say destroy, we will put a cross here and the lifeline does not exist for this other lifeline exist.

And, we have this small rectangle on the lifeline indicating that the object has got the control and it is in execution state. And, we write the messages on the arrow here. And, else the messages are sent or method call takes place, then the control is transferred in the other object becomes active.

(Refer Slide Time: 05:05)

### Sequence Diagram Cont...

- Messages are shown as arrows.
- Each message labelled with corresponding message name.
- Each message can be labelled with some control information.
- Two types of control information:
  - condition ([])
  - iteration (\*)

```
sequenceDiagram
    participant member as member: LibraryMember
    participant book as book: Book
    participant bookCopy as :Book Copy
    member->>member: borrow(bool)
    activate member
    member->>member: ok = canBorrow()
    deactivate member
    member->>bookCopy: [ok] borrow(member)
    activate bookCopy
    bookCopy->>bookCopy: setTaken(member)
    deactivate bookCopy
    deactivate member
```

The diagram illustrates a sequence of interactions. It starts with a self-call on the 'member: LibraryMember' lifeline for the 'borrow(bool)' message. This is followed by a self-call on the same lifeline for the 'ok = canBorrow()' message, which is enclosed in a condition box. Then, a message 'borrow(member)' is sent from 'member: LibraryMember' to ':Book Copy', also enclosed in a condition box labeled '[ok]'. Finally, a self-call 'setTaken(member)' is performed on the ':Book Copy' lifeline.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, the messages are shown as arrows it is seen that and it is message is labeled with the message name, and for each message name we can have some control information. For example, depending on some condition the message will be invoked or not. There are two types of control information the condition and iteration. So, message is sent many times or a message will be sent depending on some condition.

So, here we have this condition here is the condition, if is true, is computed here can borrow returns here. So, can borrow is the method that is executed for the library member. And, then is the written that is obtained that is how we represent here. And, then based on the that was computed by the can borrow we check whether indicates true or false and if it is true only then borrow member executes.

(Refer Slide Time: 06:25)

• **iteration marker** \*[for all objects]

• **[condition]**

- message is sent only if the condition is true

• **self-delegation** ↻

- a message that an object sends to itself

• **Loops and conditionals:**

- UML2 uses a new notation called interaction frames to support these

**Gist of Syntax**

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

The gist of the syntax that we discussed for a sequence diagram a simple, we show star on a condition the condition may not be there also, then we say that it is iteration star indicates the iteration condition for all objects etcetera may there may not be there. And, then to represent condition we use the square bracket and the message is sent only the condition is true, when a self delegation that is the object calls a method of it is own loops and conditionals we can use either the star notation or the square bracket, but then UML 2 uses a new setup notations called as interactions frames.

(Refer Slide Time: 07:30)

**Control logic in Interaction Diagrams**

- **Conditional Message**

  - [ variable = value ] message()
  - Message is sent only if clause evaluates to *true*

- **Iteration (Looping)**

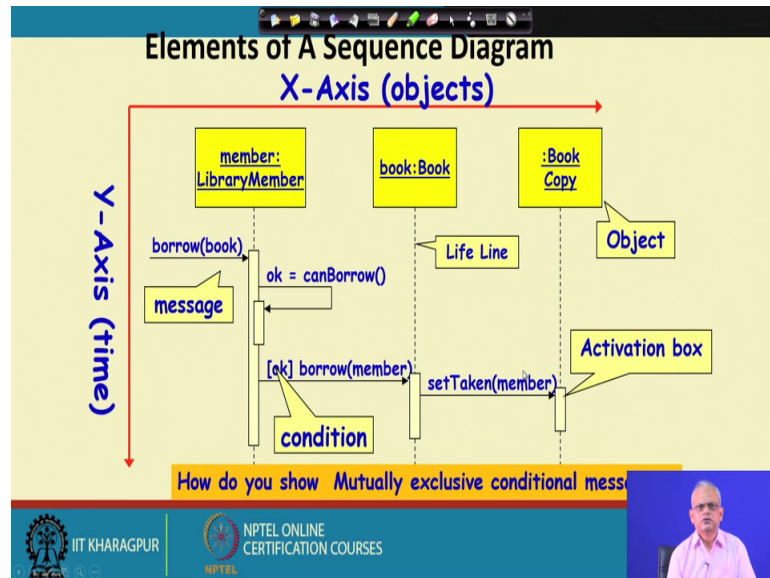
  - \* [ i := 1..N ] message()
  - “\*” is required; [ ... ] clause is optional

  - The message is sent many times to possibly multiple receiver objects.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

We had seen that we can use a conditional message; we can set Boolean value here if variable a some value then the message is sent. Then have the looping here star message the star is the requirement for looping and the condition may not be there.

(Refer Slide Time: 08:01)

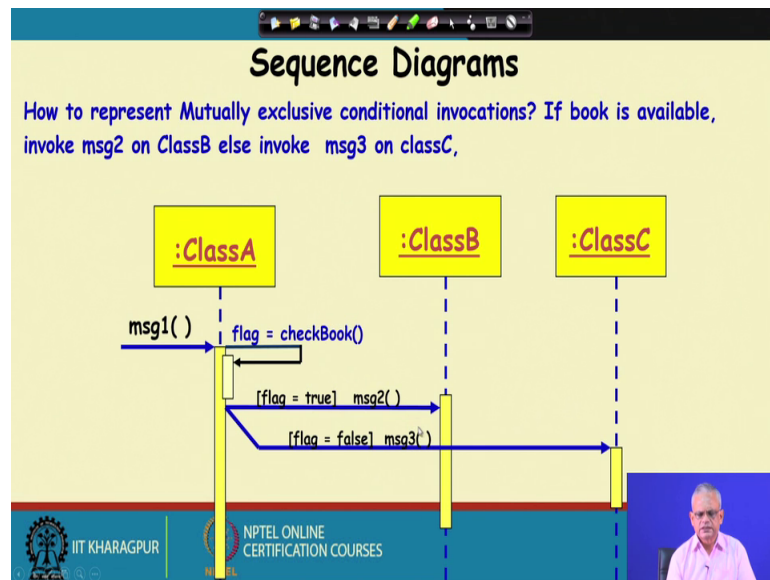


So, let us look at the sequence diagram again this 2 dimensional chart the X-Axis indicates the object and the Y-Axis is the timeline. And, the dotted line here is the lifeline for the objects, the objects have been drawn at the top; that means, a time 0 before the use case starts the objects are already created and they exist.

So, time 0 the objects exist and therefore, they have been drawn at the top of the diagram. And, time during the execution of the use case lapses from top to bottom. First the book the burrow book is invoked on this object and then it invokes a self-method, method of it is own that is can barrow and we read the diagram from top to bottom.

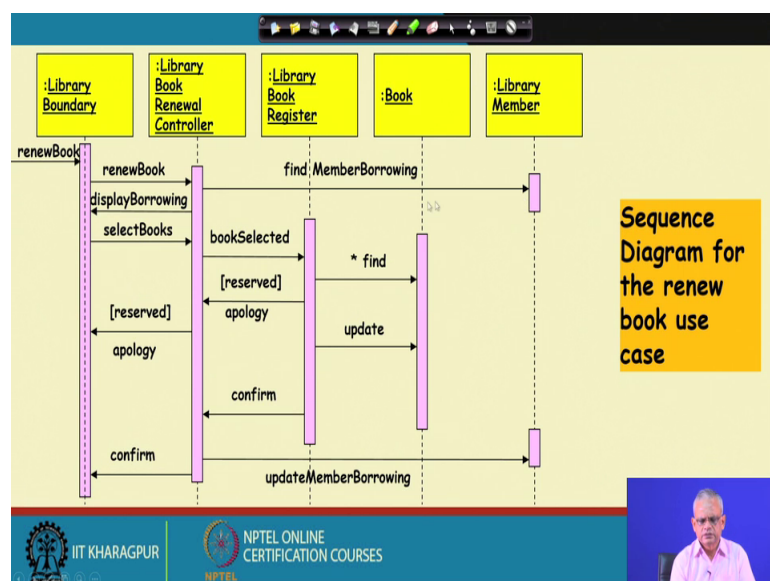
So, these are the objects this is the message, this is the conditional, and this is the activation box, but question is that let us say a message will be sent either to one object or another object not to both objects. Mutually exclusive sometimes a message will be sent to anyone or to the other and not to both how to do we represent that?

(Refer Slide Time: 09:39)



Let us say, if a book is available then we invoke some issue on class B and if not available we invoke record the request on class C, how do we represent this? So, we draw the 3 classes class A, class B, class C and then we have a flag set which is written by execution of a self-method checkbook, and depending on if the flag is true message to sent to class B or if the flag is false message 3 is sent to class C. And therefore, both the classes will not be invoked, they are mutually exclusive one of the classes will be invoked.

(Refer Slide Time: 10:36)



This is another example of a sequence diagram; let us see how to read this diagram. So, this diagram represents the behavior that occurs when the renew book use case is executed. We can also say in alternate words that, this sequence diagram is an implementation of the renew book use case. Here, the objects are shown at the top and just see here call on name of the class under line. And that means; that these are anonymous objects any object of this.

And, then the execution starts by the renew book which is invoked and the library boundary. And, this in turn in books the renew book and a renewal controller and this becomes active and this invokes the find member borrowing on the member. It checks, what are the books they have in borrowed? And, just see here there is no return arrow here the return arrow that is the result that is return by the library member is implicit.

But of course, if you want to show that some variable that is used later in a decision or something we made draw a arrow back arrow here just like, we have see drawn back arrow here we might sometimes draw a back arrow, but then these are normally omitted this invokes and a result that is return is not represented normally. And, a see here there is a control information here that is find the book among all books.

So, that is represented by iteration, is by condition, if it is reserved book is reserved, then generate an apology and finally, it is apology is displayed. So, it is a we can see that the different objects that participate, in the execution of the use case and the messages, there if seen among each other for execution of the use case and the time ordering the messages is represented here.

(Refer Slide Time: 13:31)

**Example: Develop Sequence Diagram**

- A user can use a travel portal to plan a travel
- When the user presses the plan button, a travel agent applet appears in his window
- Once the user enters the source and destination,
  - The travel agent applet computes the route and displays the itinerary.
  - Travel agent widget disappears when user presses close button

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, let us look at few examples. The idea is that given a statement like this that is a details of a use case execution, we need to identify the objects that interact each other and develop the sequence diagram. So, let us see one example here a user can use a travel portal to plan a travel.

So, the travel portal is one object, normally the travel portal is a website and therefore, a website appears on our client machine and we can even consider this as a client object, in the user presses a plan button on the travel portal. So, as the travel portal appears on the client machine, the user presses a plan button and when a presses a plan button a travel agent applet created gets created and it appears in his window, and then on this travel agent the user enters source and destination.

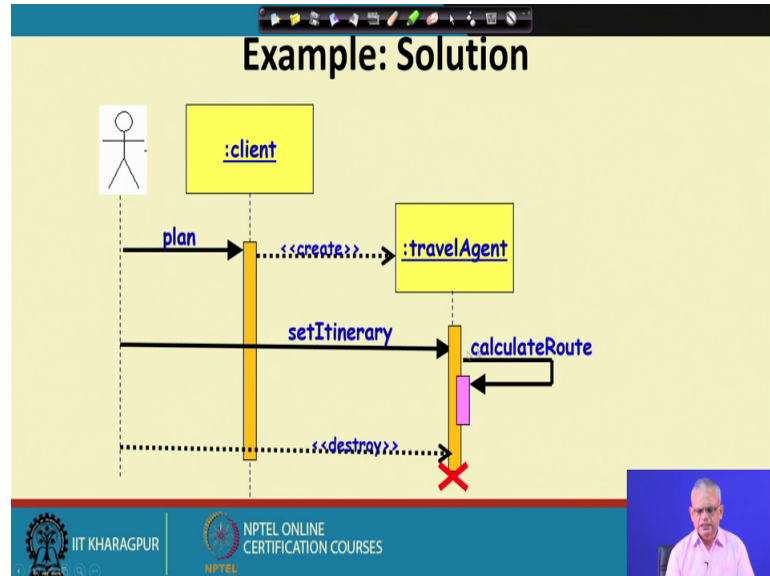
So, the user interacts with the a travel agent and on it enters the source and destination and based on the source and destination the travel agent computes the route and displays the itinerary. And, as soon as the user presses the close button the travel agent visit disappears. So, this a simple sequence diagram, where there are 2 classes here a travel portal and depending on when the user presses the plan button and the travel portal it will create a travel agents.

So, the travel agent object does not exist to start with and only when the plan button is pressed the travel agent button appears. And, then the user enters the source and



destination and the travel agent and the travel agent invokes a self-method to compute the route and then it displays it, and it disappears in the user presses the close button.

(Refer Slide Time: 16:21)




So, this is the user presses the plan button in the client and then it creates see the travel agent object does not exist to start with and only when the plan button is created, it gets created and appears. The creation of an object we represent using this kind of notation a dotted arrow with the stereotype create and the point at which the, create is invoked the object appears in the timeline it does not exist a time 0.


But, only when the create method is called and then the user sets the itinerary on the travel agent and it use it called. So, self-method calculate route, displays the route and that is implicit here. And, then the user invokes the close the, destroy and the object gets destroyed here. So, this is the representation object getting destroyed.

(Refer Slide Time: 17:37)

### Return Values

- Optionally indicated using a dashed arrow:
  - Label indicates the return value. 
  - Don't need when it is obvious what is being returned, e.g. `getTotal()`
- **Model a return value only when you need to refer to it elsewhere:**
  - **Example:** A parameter passed to another message.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | 130

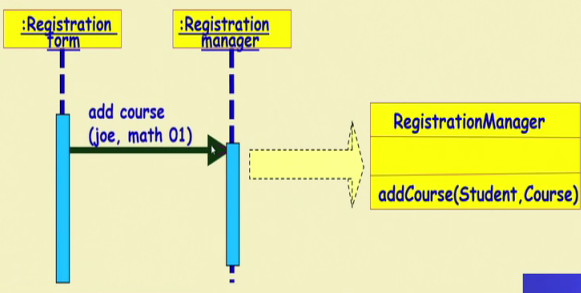


Normally, we do not represent the return values, but sometimes as I said that we need to use the specific value returned in the diagram itself and we need to use this kind of dotted arrow. And, we use this arrow the idea is that we should not complicate the diagram too much and only if we need the return value in the diagram we represent it. Otherwise, it is understood it is implicit we do not represent the return values.


(Refer Slide Time: 18:20)

### Method Population in Classes

- Methods of a class are determined from the interaction diagrams...



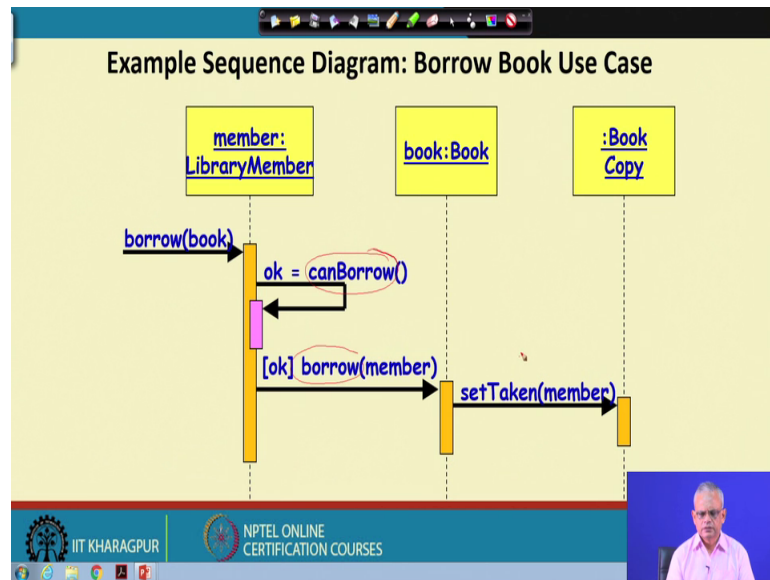
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES



Now, let us see if we develop a sequence diagram how is it useful in our design? First let us start with a very simple use case diagram, let us say there 2 objects registration form

and registration manager and then we have just added a method here between these 2 method call add course with some argument. The, implication of this on our design is that we have the registration manager must have the odd course method and it must take these 2 as it is parameter.

(Refer Slide Time: 19:08)



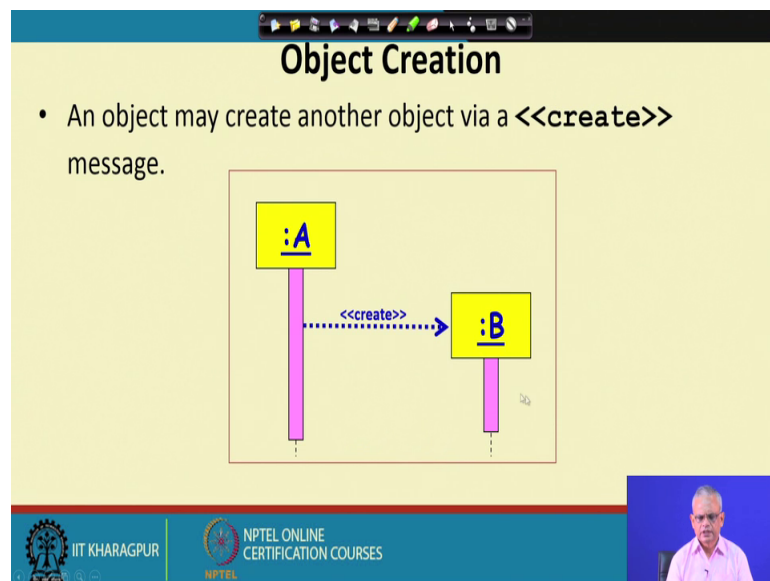
So, it must have the registration manager if you draw this kind of message. So, it must have the registration manager if you draw this kind of message, then the registration counter a sorry the registration manager must have the method add course. Only, if it has the method add course the registration form object can invoke the method and then the argument to this method at the student and the course. So, we can think that the role of the sequence diagram is to populate the methods in the classes. To start with we have just the class name, but as we draw the sequence diagram the methods are added to the classes and most case tools they do that very efficiently.

As, we in the case tool we draw the sequence diagram, we can see that in the class diagram the corresponding methods are been incorporated. At, the end of drawing all class diagram we will know that which classes have what methods? Because, we will draw many sequence diagrams. As, many as there are use cases and after we have drawn the sequence diagram for every use case, we will see that the same class might be participating in different use cases. And therefore, the methods in different use cases the different methods by get executed and they get added here.

So, this is the borrow book use case, now can you identify what are the methods of the different classes that will be populated based on this diagram? What about Borrow book will be a method of the library member class, the name of the method will be borrow and it takes a book type as it is argument. What about can borrow can borrow is also a method of the library member class.

So, from here based on this sequence diagram the library member class will get 2 methods borrow and can borrow, but what about the borrow here, that is the book class will also have an method called as borrow. And, here the argument is a member unlike the burrow method of the library class which as argument of a book, you are the burrow for the book class the argument is a member and then what about the set taken the set taken is a method of the book copy class.

(Refer Slide Time: 22:29)



We had seen the object creation, we need to draw the dotted line and then write the stereotype create on it and then once this is invoked object is created an anonymous object, we write the name of the class call on name of the class underscore is an anonymous object any object of this B class, will be denoted by this and this counts into existence at this point of time did not exist at the beginning of the use case execution.

(Refer Slide Time: 23:05)

### Object Destruction

- An object may destroy another object via a `<<destroy>>` message.
  - An object may also destroy itself.
- **But, how do you destroy an object in Java?**
  - **Avoid modeling object destruction unless memory management is critical.**

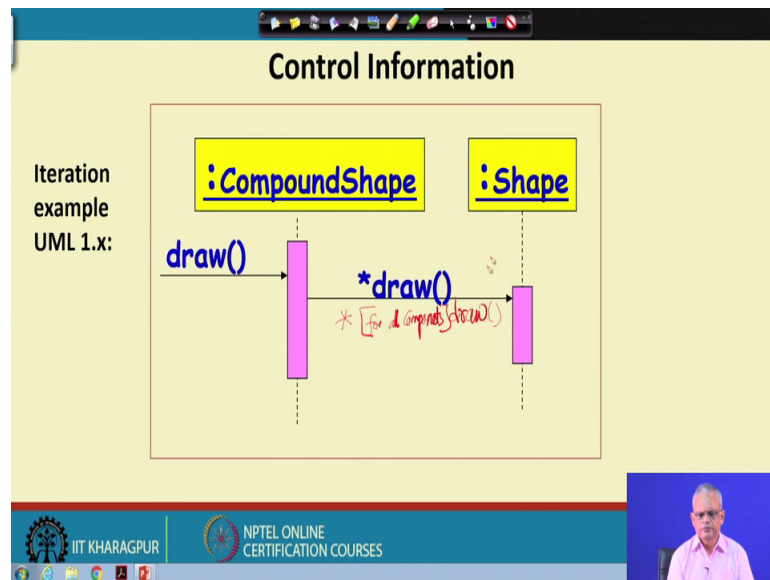
The diagram shows two lifelines, :A and :B. A dashed arrow labeled <<destroy>> points from :A to :B. The lifeline for :B ends with a cross, signifying its destruction.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Object destruction the object B existed, but then when a invokes the destroy method on the B the class of B the object B class, then it is lifeline terminates and objects see just to exist that is these the notation to show that, we draw a cross on it is lifeline and see just to exist. And, it is possible that an object may destroy itself, but then just a question, that how does somebody destroy an object in Java?

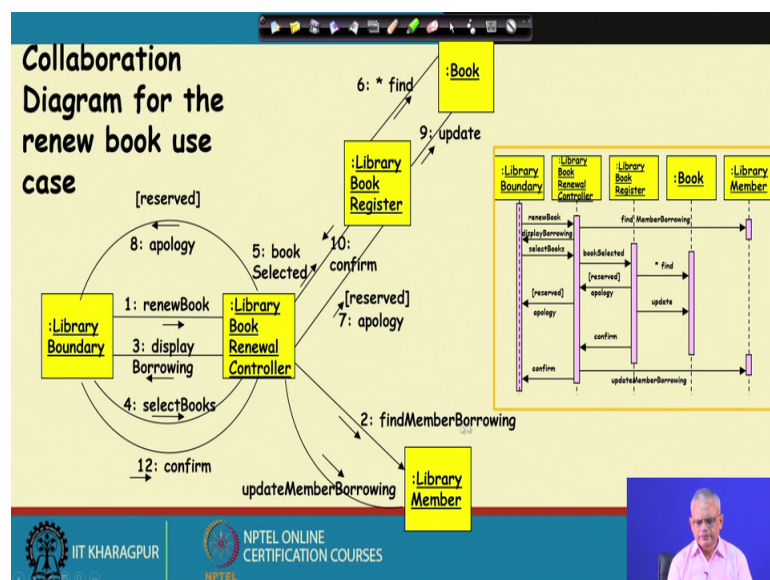
Is there a method called that you can do to destroy an object. In Java normally the objects do not have a method to destroy, they just fall out of scope and then they destroyed. And, typically we do not model to that extent that when objects are destroyed and so on unless memory management is critical.

(Refer Slide Time: 24:19)



Let us see the control information that a star indicates multiple times. So, the draw method when it is invoked and compound shape, it calls the draw method on all its shapes. We can also write here a star for all components to make it clear that draw. So, for all components it draws that is the indication with this star and the compound shape consists of many shapes and the draw that is paint itself on the screen for all components.

(Refer Slide Time: 25:18)



A collaboration diagram is another form of the interaction diagram and this diagram is automatically generated from the sequence diagram. In most of the case tools by press of

a button you get the collaboration diagram and this is just another view of the sequence diagram. Let us see how the collaboration diagram is created, if this is the sequence diagram then here we write all the objects here and then here there is a time ordering of the messages here there is no time ordering of the messages, but we add number in the front of the messages or method call to indicate the order in which it occurs.

For example, the start with the boundary object invokes the renew book and the renewal controller. So, this is the first message and therefore, library boundary library book renewal control, we write one here this is the first message to be interchanged. The next will be the renewal controller it invokes the find member borrowing and the library member.

So, this is the library member and the book renewal controller next invokes the library member borrowing. And therefore, we write 2 here and normally the direction of the invocation is shown by a small arrow here, we just draw a line of cursor a drawn a arrow here it should not be it is just lines and then we just write a small arrow here to indicate the direction of the invocation.

And, then we take the next one display borrowing between these 2 number that 3 library book controller and then library boundary we will make it 3 we draw it and so on, we take one by one and draw this. And, as we can see that it can be automatically drawn based on the sequence diagram the collaboration diagram can be automatically done drawn.

But, then what purpose does it solve? Can you guess; because it supports this diagram it must solve some purpose ok. The purpose it serves is that it clearly shows the class associations. If it invokes the method than their associated and we do not even so, the arrow here by just draw a plane line and remember that the association symbol.

So, as we draw the sequence diagram we get the class associations. Even though we said that associations are identified just by inspecting the statements, problem statement, but often we cannot identify all associations. And they get captured here, when we draw the sequence diagram. We are almost at the end of this lecture; we will stop here and continue from this point in the next lecture.

Thank you.