Object-Oriented System Development Using UML, Java and Patterns

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

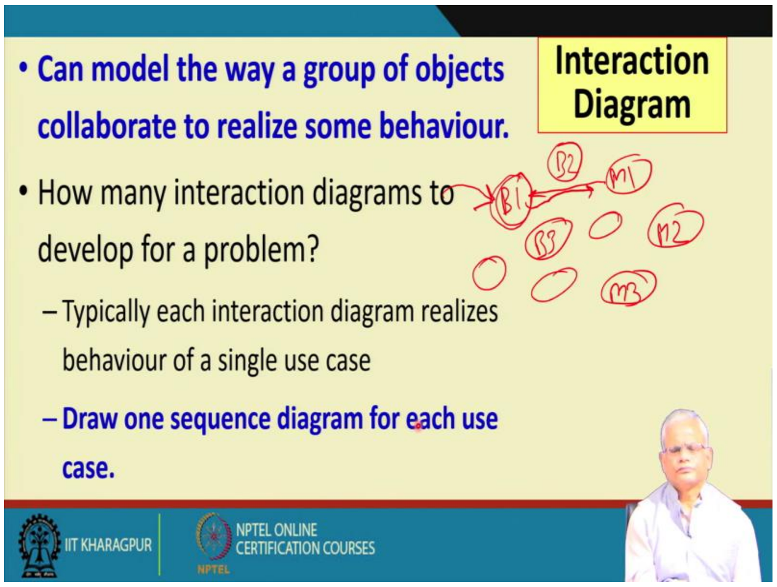
Lecture 25
Interaction Diagrams

Welcome to this lecture.

We will discuss about the UML interaction diagram. These are crucial diagram in the design process. For every problem that we solve, we need to draw an interaction diagram, which captures a crucial behavioral aspect of the problem. And from the interaction diagram, we also generate the code to implement, automatically we generate the code or mechanically, if we are doing manually, manual code generation it is almost mechanical, very easily we generate the code for the system that we are implementing.

If we are using a case tool, then from the interaction diagram, the code gets automatically generated. Now let see the interaction diagram, how to develop it. Given a problem, how do we go about developing an interaction diagram, the syntax, types of interaction diagram, etc. that we will address in this session.

(Refer Slide Time: 01:32)



Interaction Diagram

- Can model the way a group of objects collaborate to realize some behaviour.
- How many interaction diagrams to develop for a problem?
  - Typically each interaction diagram realizes behaviour of a single use case
  - Draw one sequence diagram for each use case.

The interaction diagram, it models the way the objects in a system collaborate to realize some behavior. Every software that we write, we can think of it as a set of objects inside the working software.

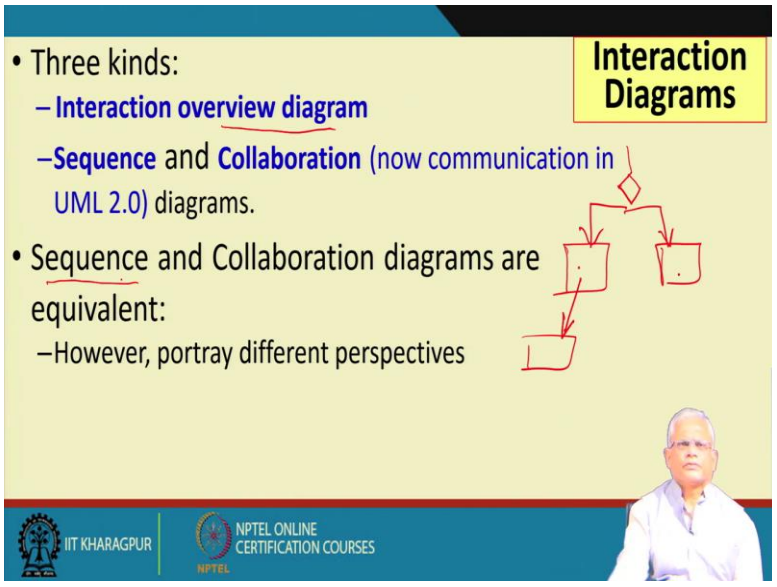
Every program generates large number of objects. Let say in a library software, we have different book objects, book1, book2, book3 etc. and also member1, member2, member3 etc. and so on. Now let’s say we have a request from a member to issue a certain book. Let say, member1 wants to issue book1. Now we check first with the book1, that is whether the book is available, and then we check with the member that whether the member can issue book whether he has not exceeded the quota and whether his membership is valid etc. If that is the case, then again, we call a book1 method to issue out the book in the name of member1 that is member1 is the reference variable for member1 is remembered here in the book1. And also, in member1, we set the book1 as one of the books issued and then print an issue slip. In the issue book example, we saw that there is an interaction first B1 is called, and then from B1, M1 is called, from that B1 again, and so on.

This was very simple case but then we will see that in a typical behavior, many objects interact. And in the interaction diagram, we capture this interaction in the form of a very elegant diagram. But then the question is that, given a problem how many interaction diagrams we should develop?

Here as a hint to the answer, the use case interaction diagram that we discussed here is about the renew book, this is one use case. Renew book use case, for that we developed one interaction diagram. This is not really the interaction diagram but then this the main idea of interaction diagram.

Since, we need to develop one interaction diagram for every use case, and therefore, the number of interaction diagrams required to solve a problem is equal to the number of use cases present in the problem. This is a crucial point, that we need to draw one sequence diagram per each use case, and these can be translated to code easily. And once we have developed the sequence diagram for all the use cases, we have lot of code ready for us.

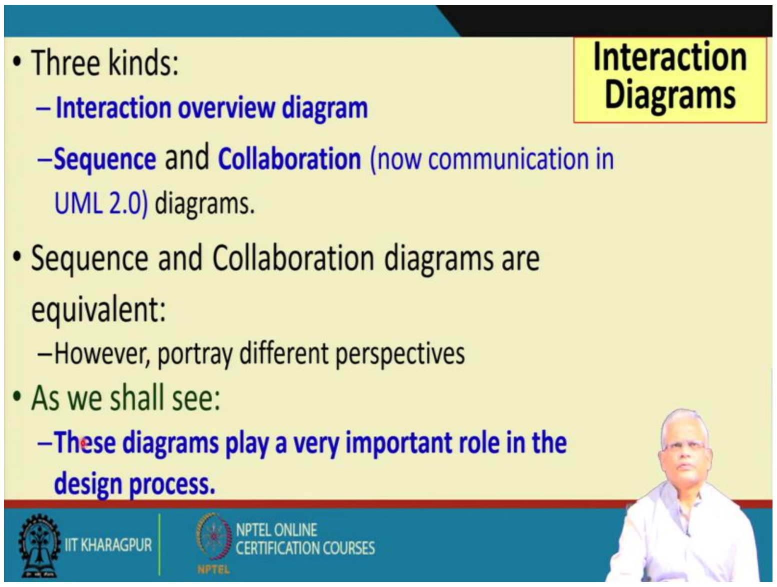
(Refer Slide Time: 06:17)



Interaction Diagrams

- Three kinds:
  - Interaction overview diagram
  - Sequence and Collaboration (now communication in UML 2.0) diagrams.
- Sequence and Collaboration diagrams are equivalent:
  - However, portray different perspectives

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
NPTEL



Interaction Diagrams

- Three kinds:
  - Interaction overview diagram
  - Sequence and Collaboration (now communication in UML 2.0) diagrams.
- Sequence and Collaboration diagrams are equivalent:
  - However, portray different perspectives
- As we shall see:
  - These diagrams play a very important role in the design process.

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
NPTEL

Let’s see the types of interaction diagrams. There are three types of interaction diagram: one is called as the interaction overview diagram, another is called a sequence diagram, and the third is called as a collaboration diagram. But UML 2.0 onwards, the collaboration diagram is called as communication diagram. The interaction overview diagram, as the name says it provides an overview of interaction for complicated behavior. Let me just tell that again, that for some use cases the behavior becomes extremely complicated, we cannot accommodate that using one sequence diagram, we draw multiple sequence diagrams.

For complicated use cases, we might use multiple interaction diagrams. And then using an interaction overview diagram we put together all these interactions, that based on some condition like either one interaction occurs, or another interaction occurs, and so on. And finally, it ends. This is what is modeling the interaction overview diagram, where we model complex use cases where the behavior cannot be accommodated in a simple diagram, we split it into multiple diagrams. And then the interaction overview diagram allows us to piece together or we know which interaction occurs after what and based on which condition, and so on.

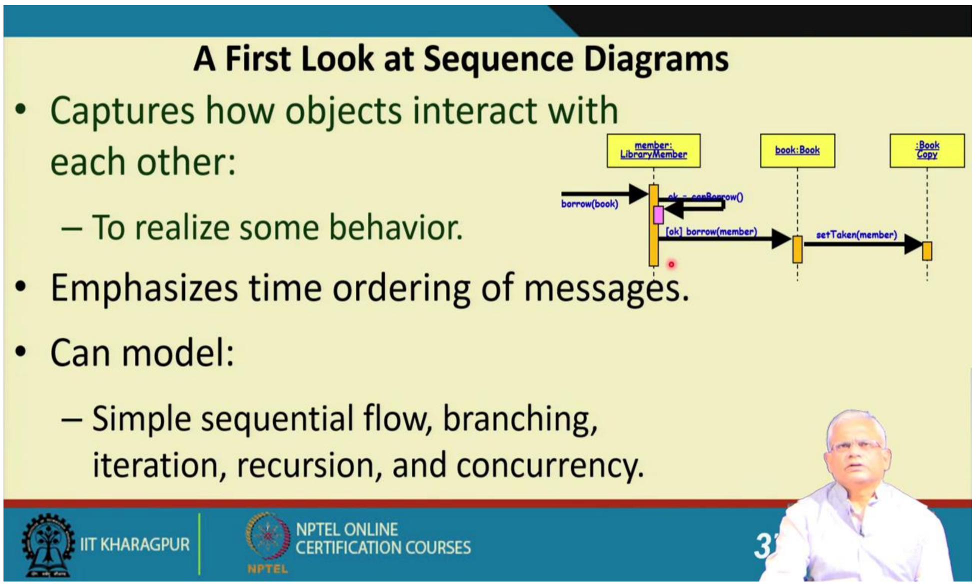
In our discussion here, we will restrict to simple use cases, the behavior is not extremely complex, and therefore, will not really have the opportunity to use the interaction overview diagrams. For us, for each use case, it is enough to draw one sequence or collaboration diagram.

The sequence and collaboration diagrams are actually equivalent. Even though we have two different diagrams named a sequence and communication diagram, but they are actually equivalent. Then the question arises, if they are equivalent, why do we need two diagrams? The answer is that, we will see that these two diagrams even though they are equivalent, like one diagram is generated from the other diagram automatically, but then they portray different perspectives of the system.

Some aspects of the system become clear if we look at the collaboration diagram, typically, we will start by drawing the sequence diagram and will automatically convert a sequence diagram into collaboration diagram. And the collaboration diagram gives an easy understanding of some aspects of the system namely, which class is associated with which class and so on. The collaboration diagram which becomes a little bit cumbersome to identify from the sequence diagram.

Now, in every design process that we will discuss, these diagrams are one of the crucial diagrams. In addition to the class diagram, the interaction diagrams are very essential, they play a very prominent part in the design process and help us to solve a problem. Once you know the sequence diagrams, collaboration diagram, you will use it very frequently.

(Refer Slide Time: 10:42)



A First Look at Sequence Diagrams

- Captures how objects interact with each other:
  - To realize some behavior.
- Emphasizes time ordering of messages.
- Can model:
  - Simple sequential flow, branching, iteration, recursion, and concurrency.

member: LibraryMember
book:Book
:Book Copy
borrow(book)
ok = canBorrow()
[ok] borrow(member)
setTaken(member)
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
NPTEL
31

Now, let’s have a first look at the sequence diagram. As we were saying that it captures how the objects interact with each other to realize some behavior or use case. Given a use case, how do the objects interact with each other, that we will capture using a sequence diagram.

And here we will have the sequence in which the messages occur among the different objects. The diagram would look like this (in the above slide). The top of the diagram we have objects, the book object, the different copies of the book, library member. The objects interact with each other. The way they interact is that they call methods of each other that is encoded in this diagram or modeled in this diagram.

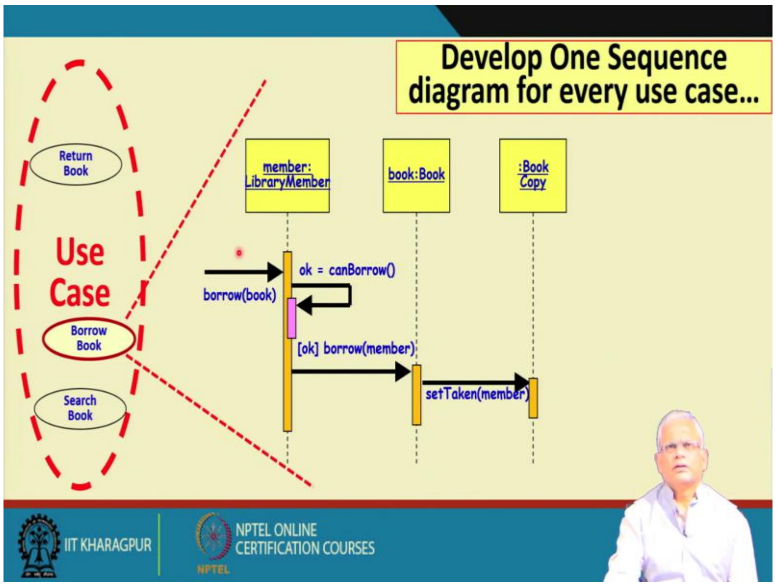
We use a closed arrow like this ($\longrightarrow$) to show the interaction. A method is called under member object, like, borrow(book), and then check whether the member can borrow book whether he has not exceeded the book quota and the membership is still valid or not and based on that, call the method borrow of the book object.

Here we are capturing the interaction, the first interaction occurs when the borrow book method of the member is called (borrow(book)), the second interaction is on the object itself (ok = canBorrow()), here it calls its own method, and then the member object calls the borrow method of the book object(borrow(member)), and so on. And then the book object calls set token (setToken(member)) for the book copy. Then, specific book copy issued out where membership number is written. As you can see here that the method calls occur among different objects in a sequence manner. And that’s why it is called as a sequence diagram.

We write the objects which participate in a particular behavior or use case at the top, and then we model the sequence in which the method calls occur between the different objects. One thing is that the methods calls are in time ordered from the top to the bottom. We have a sequence that occurs in the order in which these are modeled: first book, then borrow book, and then can borrow, and so on.

Not only the sequential flow here, but also, we will see that we can model branching, iteration, recursion, concurrency, and so on. This is a simple glance of the diagram, but it has much more powerful features as we proceed, we will discuss about that.

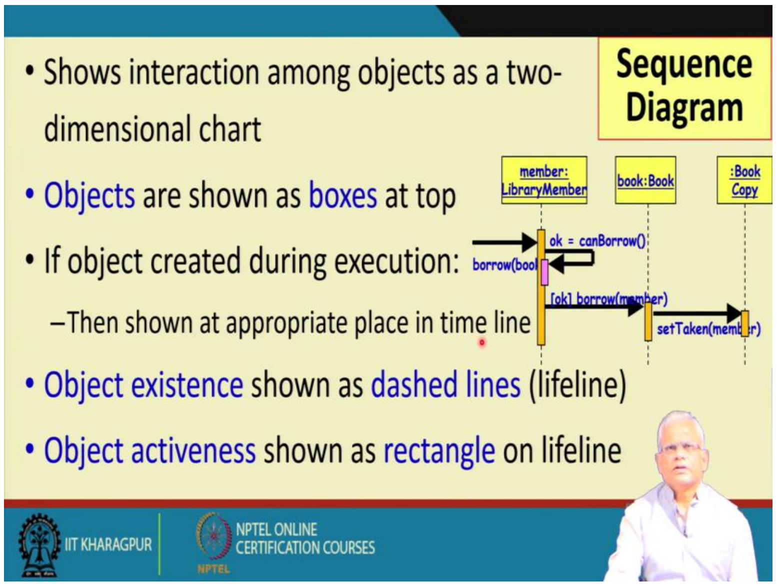
(Refer Slide Time: 14:40)



Develop One Sequence diagram for every use case...

As we have already mentioned, we need to develop one sequence diagram for every use case. If these are our set of use cases return book, borrow book, search book for each one we need to draw one sequence diagram (in the above slide). Let’s consider the borrow book, for the borrow book will develop one sequence diagram. For the return book, we will develop another sequence diagram, search book we will develop another sequence diagram, and so on. And finally, all the sequence diagram will be used to generate a significant part of the code implementing the design.

(Refer Slide Time: 15:28)



Sequence Diagram

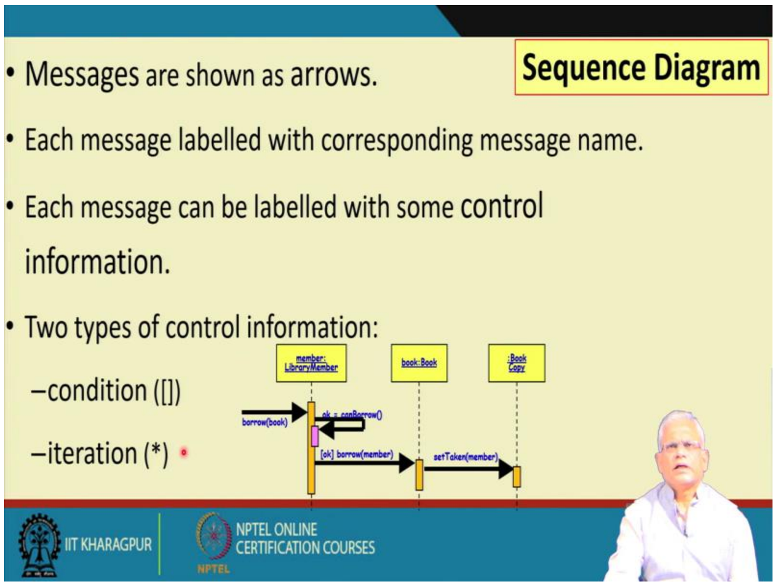
- Shows interaction among objects as a two-dimensional chart
- Objects are shown as boxes at top
- If object created during execution:
  - Then shown at appropriate place in time line
- Object existence shown as dashed lines (lifeline)
- Object activeness shown as rectangle on lifeline

member: LibraryMember
book:Book
:Book Copy
ok = canBorrow()
borrow(book)
[ok] borrow(member)
setTaken(member)
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES

The sequence diagram, we can think of it as a two-dimensional chart. The X dimension are the objects and the Y dimension is the time dimension. On one dimension, we have all the objects that participate in the use case are listed here (X dimension). In another dimension (Y dimension), the time ordering of the different method calls are mentioned. The first dimension are the objects which are shown as boxes. But then sometimes we might have objects that are created after the use case starts.

Let say we are creating a book. In that case, once the constructor for the book is called somewhere, we draw that rectangle at that point of time. The ones that are shown at the top are the time the use case starts execution that means the objects preexist, the start of the use case execution. And sometimes during the execution of the use case, we might have an object getting created and then we show that at the proper place in the timeline. The dotted lines here (- - - -), are called as a lifeline, that is the object continues to exist. If we use a cross on the lifeline sometime that means that the object is destroyed at that point. And, on a method call an object becomes active until the final result is generated. Active method call is shown as a rectangle pipe (▭). So, we have two dimensions here, one is the object dimension, that is the X dimension, the Y dimension is the time dimension.

(Refer Slide Time: 18:10)


Sequence Diagram

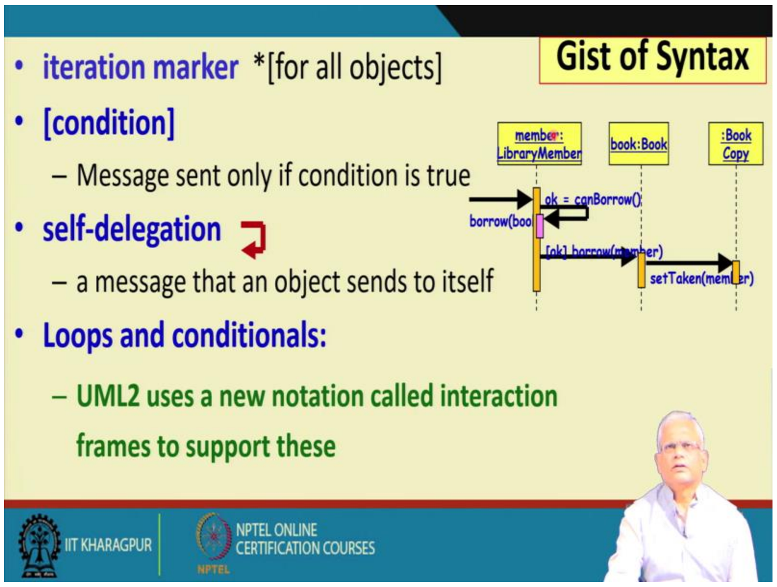
- Messages are shown as arrows.
- Each message labelled with corresponding message name.
- Each message can be labelled with some control information.
- Two types of control information:
  - condition ([])
  - iteration (*)

member: LibraryMember
book:Book
:Book Copy
borrow(book)
ok = canBorrow()
[ok] borrow(member)
setTaken(member)
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES

Now, the method calls are arrows, on the method call we write the name of the method. And also, we can use some control information like a condition under which the method is called, iteration, and so on.

Two types of control information are frequently used. One is condition, we use a bracket here [] and we write the name of the condition inside the bracket. Just like we have written here, [ok], ok is a predicate and then we enclose it in rectangular brackets. That means that this is a condition if it is true then the borrow method of the book object will be called. If okay is not true, then it will not be called. Another one is Iteration shown using a star (* [for all objects]).

(Refer Slide Time: 19:27)


Gist of Syntax

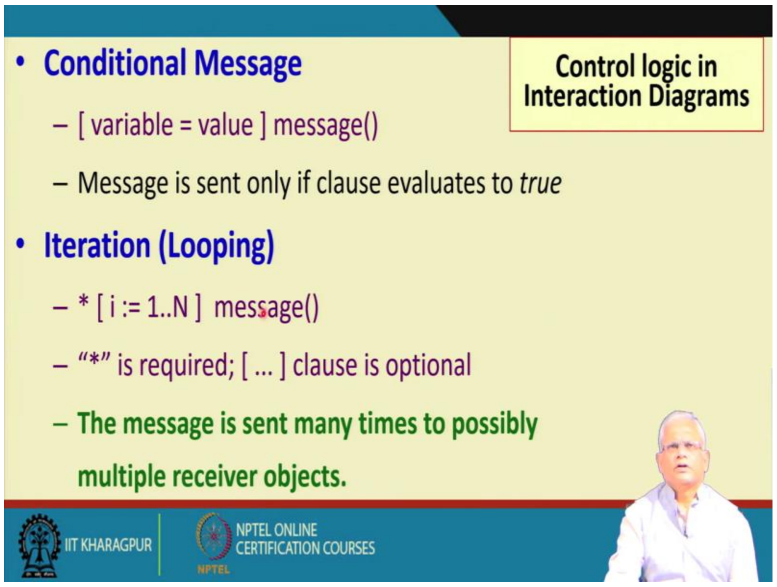
- iteration marker *[for all objects]
- [condition]
  - Message sent only if condition is true
- self-delegation
  - a message that an object sends to itself
- Loops and conditionals:
  - UML2 uses a new notation called interaction frames to support these

member: LibraryMember
book:Book
:Book Copy
borrow(book)
ok = canBorrow()
[ok] borrow(member)
setTaken(member)
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES

The syntax so far, we saw is that iteration is star for all objects (*). Condition right within brackets ([]). Self-delegation that is an object calls its own method, we show a self-delegation. For representing self-delegation, self-loop is used. Some additional features have become available since UML 2, and these are called as the frames.

We will see frames as we proceed with the discussion, initially we will look at these syntaxes like iteration marker, condition, etc. And then little later, we will look at the frames that have been supported since UML2, which help us to do the same thing like looping, decision, and so on. Just see here, on this subject member object there is a self-delegation (ok = canBorrow()) where the member object, calls its own method. It calls the method of the book object at this point, and remember from our early discussions, that in order for an object to call another object must have the id of that object or in other words, the corresponding classes should be associated. There should be association relationship between the member class, like, library member class and the book class, and a link should get established between the corresponding objects. Or in other words, the id of the book object should be present on the member object so that it can call the borrow method of that object.

(Refer Slide Time: 21:49)



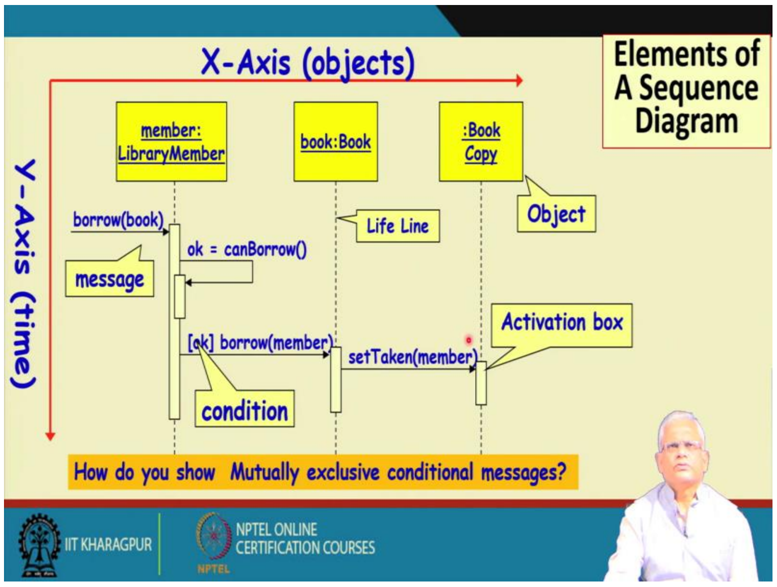
Control logic in Interaction Diagrams

- 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
NPTEL

The conditional, this is an example here (in the above slide), that the variable is having some value we express it like this. Under this condition, let say the flag equals to true, we might write and then the message is called ([variable=value] message()). Iteration, we might write in this that i equal to 1 to n (* [i:=1..N] message) or we might just write star("*"), and here it means that the message is sent to multiple objects.

(Refer Slide Time: 22:31)

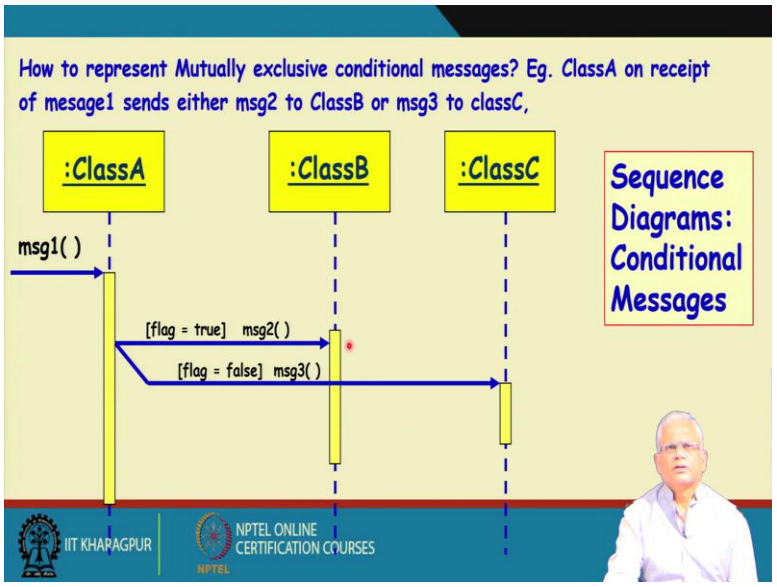


The two dimensions of the sequence diagram; one is the object dimension, all the objects pre existing are shown on the X dimension. To start with, we have all the objects which are already existing and participate in the use case execution. The Y axis is the time direction, the time axis. The first method to be called is the borrow book on the library remember object. And remember the syntax here to represent an object is that colon underline like this one book copy (<u>book:Book</u>) that means any book object. If we write a name here that means that specific object, book 1, book 2, etc. All these are the id of the specific object.

If we write just colon, class name underline that means any object of that class (<u>:Book</u>). Just to let you become familiar with the syntax. At the top, we write the object in the rectangle, and the method call or a message with the arrow. The constraints in brackets are the condition, the lifeline is the dotted arrow, as long as the lifeline exists for an object the object is existing. If we put a cross (X) somewhere that means, the object gets destroyed at that time. And if we draw a square, a rectangle on a lifeline, it means that for that duration of the time, the object is active or a method has been called and it is to produce the result.

But just asking this question, that how do we show, mutually exclusive conditional messages? That is an object either sends a message to another subject, or it sends to another subject but not to both of them (as shown in the below slide). That is mutually exclusive messages to two different objects.

(Refer Slide Time: 25:14)



How to represent Mutually exclusive conditional messages? Eg. ClassA on receipt of mesage1 sends either msg2 to ClassB or msg3 to classC,

Sequence Diagrams: Conditional Messages

Now let’s, see the answer. Slowly we will get used to this kind of constructs and we will use them in our solutions. We use a variable, the variable is having some value, then the message to is called of class B. The class A, calls the message2 (msg2()) where a flag is true, a flag is false, then message3 (msg3()) of class C is called. And once a message is sent to an object or a method is called here, the object becomes active. And class A is also active because it is waiting for results from the method call.

After the completes of class B, it will return the result and similarly, class C also return result after completion. The conditional messages, we use the condition construct here ([flag=true] or [flag=false]) and a variable having some value, we can model that either method of one object or the other object is called. We will see more details of the sequence diagram and we will use it to model some use cases because that’s an important skill in the design process.

In the next session, we will look at further details of the sequence diagram and see how we can use it to construct interaction models of different use cases.

We are almost at the end of this session; we will stop here.

Thank you.