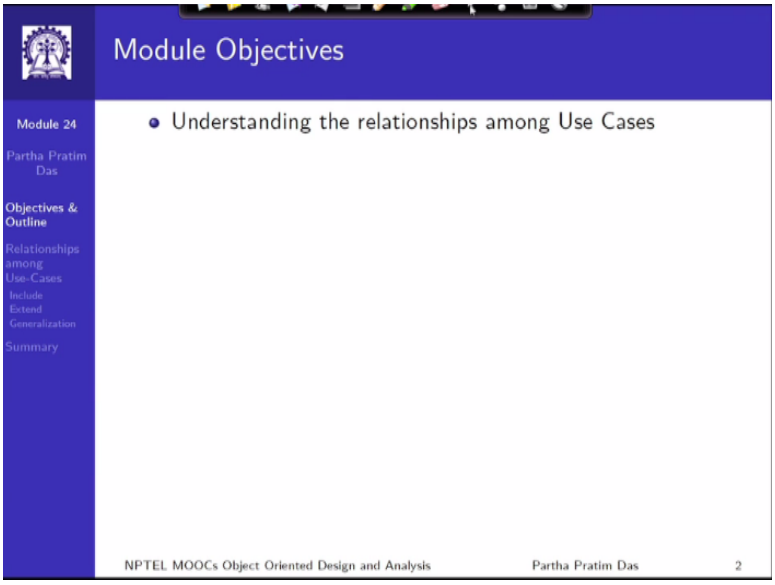
Object-Oriented Analysis and Design
Prof. Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology – Kharagpur

Lecture – 36
Use-Case Diagrams: Part II

Welcome to module 24 of Object Oriented Analysis and Design. From the last module we have started discussing about different UML diagrams; specifically, we have been talking about Use-Case diagrams and we will continue on that.

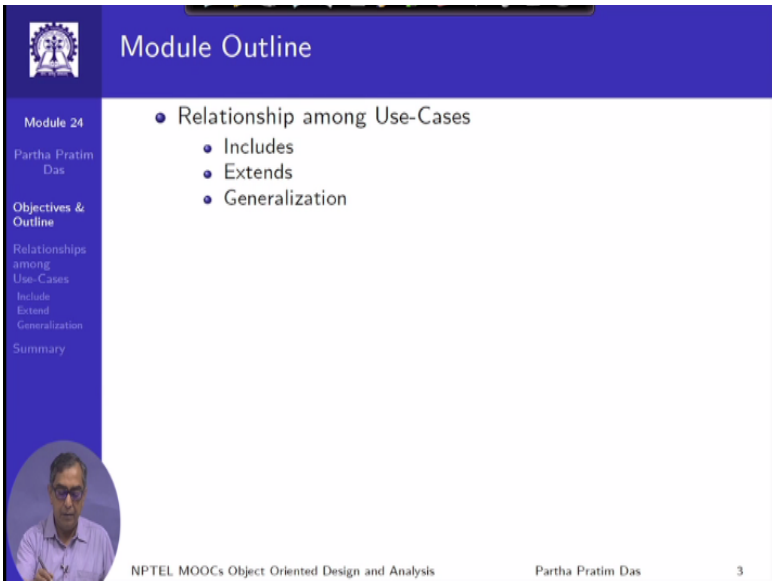
(Refer Slide Time: 00:44)



In the Use-Case diagram we have seen that there are few components that define the diagram the actor, actors in the Use-Case diagram are responsible for starting Use-Case actions or can be responding to some Use-Cases as passive actors. We have seen different classification of actors. The second component are Use-Cases which are well-defined perceived functionalities of the system that we want to put in the place.

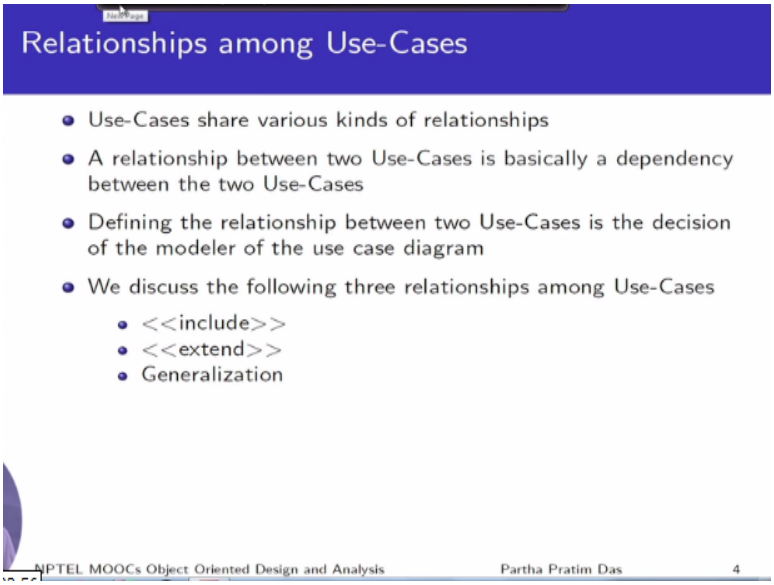
So these are the actions that the actors initiate every Use-Case is described by a name and we have seen that in the specification of the Use-Case we can state its purpose pre-condition, post condition, failure condition, optimal or most expected flow and so on. In this module, we will discuss about relationships among Use-Cases.

(Refer Slide Time: 02:04)



There are typically-- there are three major relationships that exist between Use-Cases; this is given in this outline which will also be available on the left hand side of every slide.

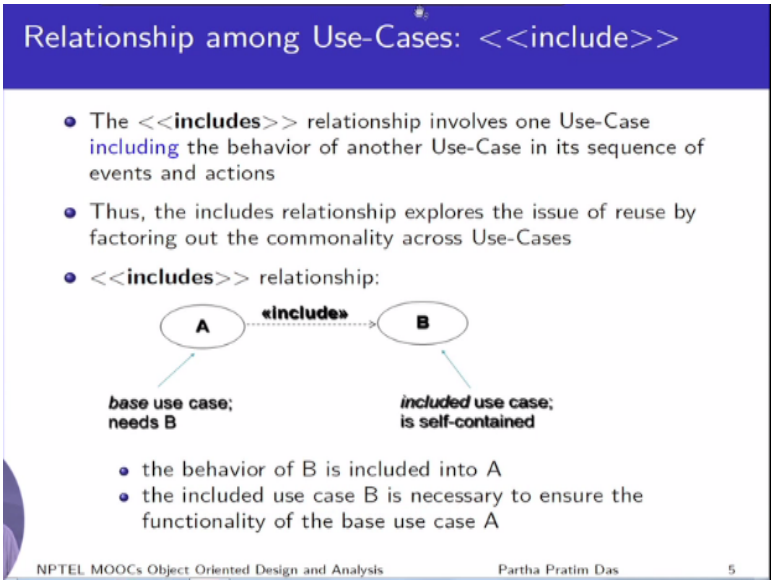
(Refer Slide Time: 02:13)



So the relationship among Use-Cases there could be various kinds. When you talk about relationship it is basically a dependency between two Use-Cases. So a relationship is I have a Use-Case here have another Use-Case here and I am putting some kind of a dependency between these Use-Cases and that dependency is known as relationship. Defining the relationship between the Use-Cases is usually a decision that the person modeling the system would take.

So analyzing the specification looking at different conditions that may exists between these actions, we will identify different relationships. Here we will talk about three relationships the include relationship; the extend relationship; and the generalization relationship.

(Refer Slide Time: 03:18)



So we will start with the <<include>> relationship. The <<include>> relationship involves one Use-Case that involves another Use-Case that includes another Use-Case. So the <<include>> relationship basically explores the issue of reuse. So if you just draw a parallel with the common programming paradigm then <<include>> relationship is kind of one function calling another.

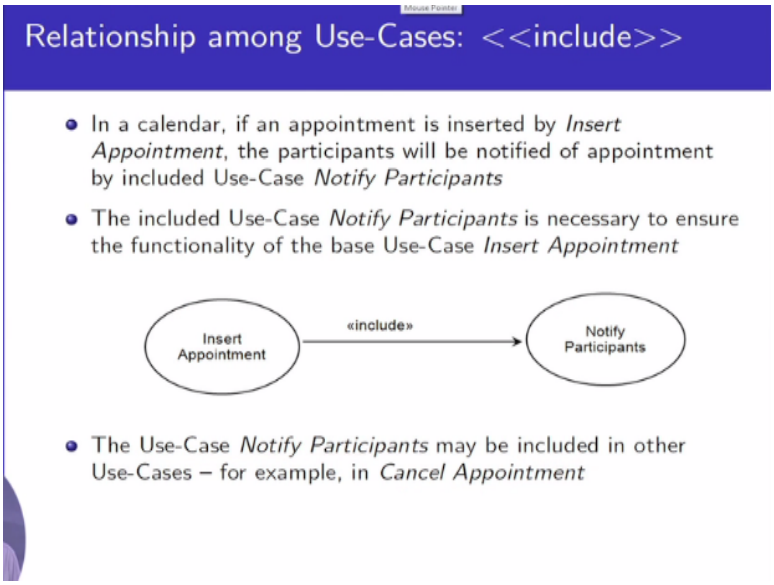
So when we want to perform some action say main function wants to perform some action and that includes printing certain values then main would not be directly printing it rather main will call a constant Printf, to take care of the actual printing process. So if main is one Use-Case, then Printf would be other Use-Case so that is a typical programmatic view but in general we will say whenever we have a Use-Case A and to complete its own functionality.

It might need another Use-Case B then we will write it like this A <<include>> B. So you should note the direction of this relationship A is the base Use-Case which needs B, B is the included Use-Case which we will assume a self-contained. Of course in turn B may include other Use-Cases that is quite possible. So the direction of the arrow would be from A to B and the style of the arrow would be dashed.

And to make it explicit that it is an <<include>> relationship we will use a (()) (05:07) and say that is an <<include>>, so this is the way we write it. So that would mean that the behavior of B is included in the behavior of A because while the Use-Case A is executing at certain point it will invoke B and complete that and the included Use-Case B is necessary to ensure the functionality of the base case.

So without B, I cannot complete the functionality of A and this is the basic notion of <<include>> relationship.

(Refer Slide Time: 05:35)



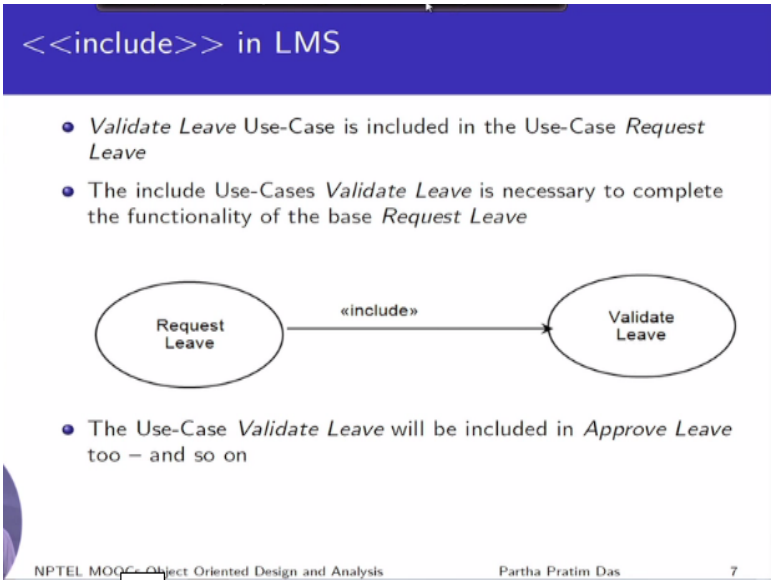
So for example if we have a calendar kind of application were there are different appointments that I can insert, create. Now certainly we will expect that once appointment has been created then there will be notification given to the participation. So one Use-Case-- I am sorry, one Use-Case is the Insert Appointment, the other Use-Case is Notify Participant and there is a <<include>> relationship from the Insert.

So Insert Appointment is the base case and Notify Participant is the included Use-Case in. Now we may note that like in functions which can be called from multiple functions the Notify Participant this Use-Case Notify Participant maybe included in other Use-Cases. For example, if

I later or somewhere, I have another Use-Case Cancel Appointment then certainly when the appointment will canceled the participant need to be told.

So Notify Participant would also be included in the Cancel Appointment Use-Case.

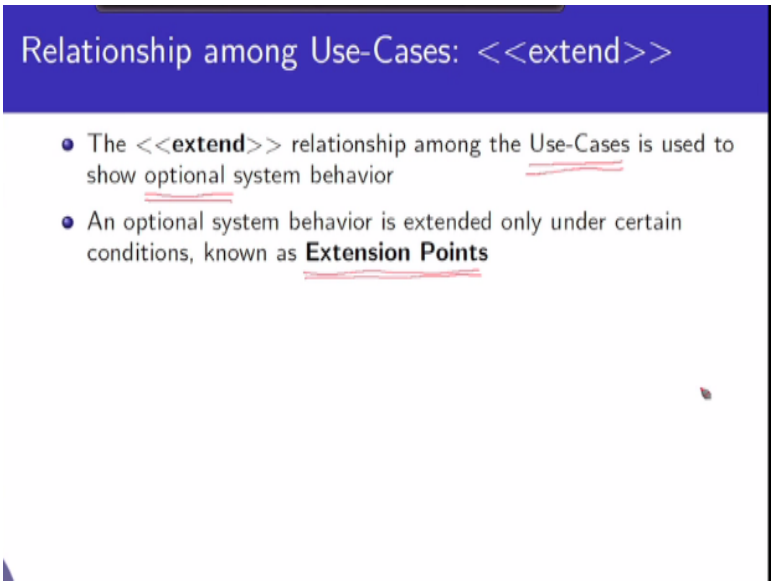
(Refer Slide Time: 06:51)



Talking about ongoing LMS system we have-- for a leave we need to run a validation to check whether a particular leave is valid one. And we say that Use-Case is validate leave which have all the actions performed to validate a leave. When the <<include>> use—this will be included whenever doing a Request Leave. So when an employee requesting for a leave certainly before the request a leave can be created.

We need to valid that the tentative leave record is a valid one, so request leave is a base case and validate leave is an included case that will get included. Certainly validate leave can be included also in other Use-Cases like approve leave. When the lead goes onto approve a leave naturally the -- it will need to be checked that the leave being approved is a valid one and that the conditions that the lead may have imposed on the leave.

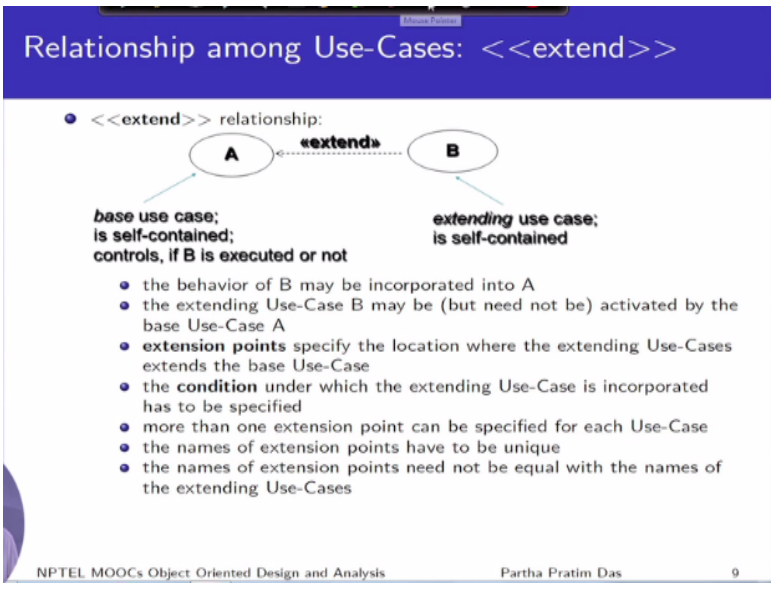
(Refer Slide Time: 07:57)



The second relationship between Use-Case is known as a <<extend>> relationship. <<Extend>> relationship is a kind of an option adding an optional behavior among the Use-Cases. So it can be used to add some optional behavior to what has already been defined for a Use-Case. Now when you want to add this optional behavior; now this optional behavior means that it could actually depend of certain conditions.

So you can also say it is a conditional Use-Case. So the condition is represented in terms of Extension Points. So say if you reach the specific extension point then you invoke the extended Use-Case.

(Refer Slide Time: 08:51)

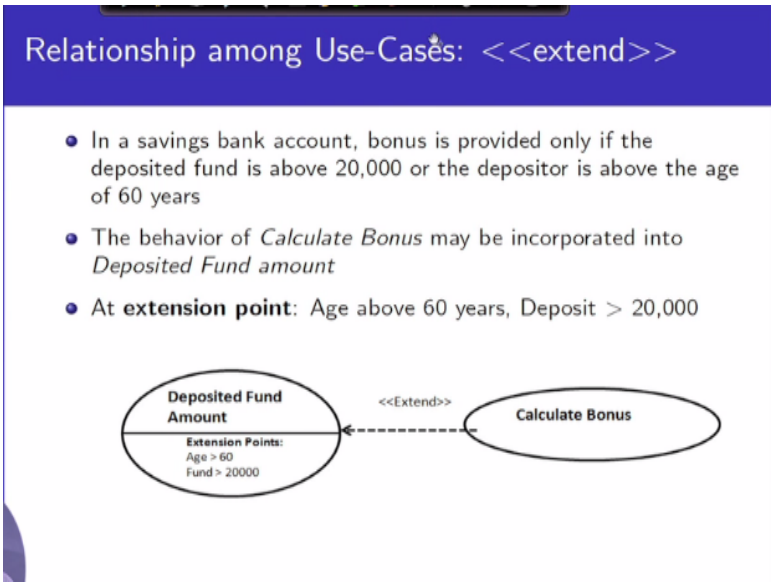


So if we say, we have a base use case A as an here; we have an extending use case B as an here and in this case both A and B are self-contained unlike in <<include>> where B was self-contained but A the base case was partial because it need B always—here that is not the case because this is the case of optional or conditional invocation. But A controls if B should be executed or not, that is a conditional one that is the optional one.

So these are the couple of points we can note the behavior of B may be included maybe incorporated into A provided it is invoked so extending the Use-Case B is optional that is it maybe but it need not always be activated by A. And there are Extension Points that specify the location in Use-Case A where the extending Use-Case will be used. So there is a-- the condition under which this is incorporated the condition is actually given by the Extension Point.

Now some more details will include that more than one extension point can be specified for each Use-Case. There could be multiple different conditions under which the same Use-Cases execute. The names of extension point must be of unique of course otherwise we will not be able to identify them. And certainly since one Use-Case maybe-- may extend multiple extension points it is quite possible that the number and names of extension points and the number of extended Use-Cases may not be one to one, there could be multiplicity of relationship.

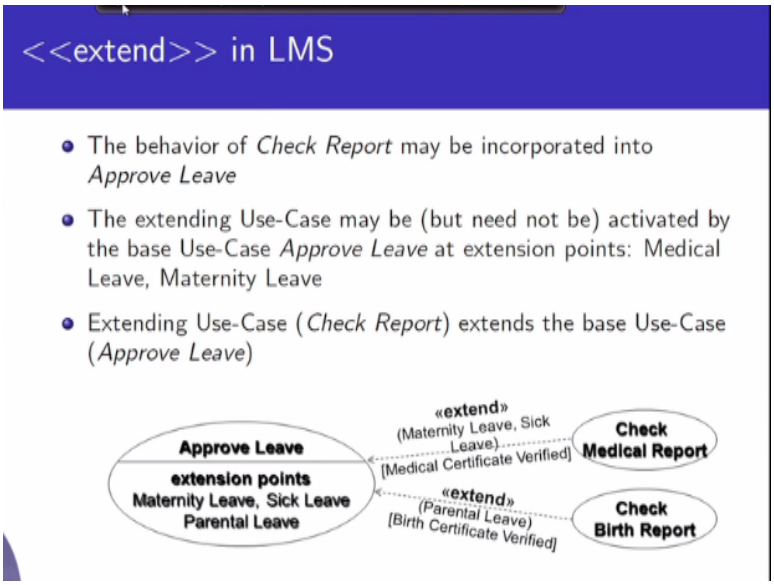
(Refer Slide Time: 10:51)



So as an example say your savings bank account provide bonus if the deposited fund is above 20,000 or the depositor's age is above 60 years either that depositor was a senior citizen or their fund values more than 20,000 then a bonus is provided. And if the Use-Case for providing the bonus is calculated bonus and the Use-Case of actually computing the fund is deposited fund amount then we can have an extension point which incorporates these two conditions.

The senior citizenship condition as an here and the fund level condition as an here and you find those in the diagram. Let me just clean up and show you again. This is the extension point and the deposited fund amount is a base case and this is the extended case that will get invoked at this extension point; that is the basic idea of extended relationship.

(Refer Slide Time: 11:58)



So in LMS, we know that there are certain types of leave; we have seen certain different types of leave which can be approved provided there is a certain condition that will satisfy, for example for a maternity leave certainly the certification is required that the employee in question already has the medical condition of pregnancy or for a sick leave we need the certification that the person was sick and doctor has checked and so on.

Similarly, for a parental leave we will need birth certificate of the baby that has been born to the required-- the associated employee. So let us try to put that behavior of as a Check Report which we may want to incorporate into the Approve Leave which means if the leave has to approve a

particular leave which is of either these are-- you remember the leave types the ML, ML is Maternity, SL is Sick Leave, and PL is a Parental leave, if any of these leave has to be approved then for those leave we will need to check for this.

And certainly for the maternity and sick it is kind of medical report, so I can have-- we can model the extended the Use-Case which is Check Medical Report which has to do all the necessary tasks for actually checking out the validity and authenticity of the medical report and so on. Whereas for Parental leave we need to check some corporation documents, municipality document regarding the birth of the baby having taken place.

So that is another Use-Case check birth report. So we can model this by having an extension point or having actually multiple extension points I should have marked them here in the Approve leave Use-Case. One extension point is Maternity Leave, Sick Leave. So in terms of Approve leave if the condition is that you have a situation of Maternity or Sickness then the corresponding check medical report is Use-Case has to be invoked.

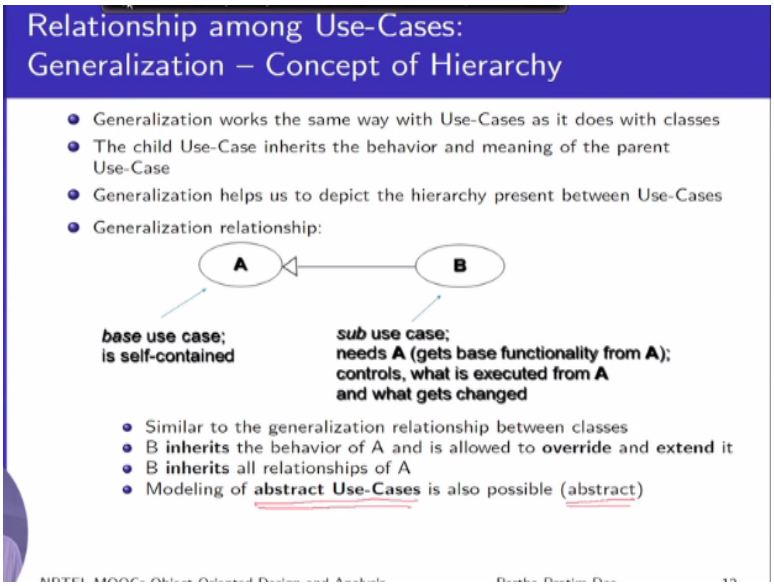
So that is designated by this arrow so here also note that the direction of the arrow is from the extend Use-Case to the base use case. And on that again we put the (()) (14:48) <<extend>> to mark that it is an <<extend>> relationship. On top we put the condition of extension which is extension point name, so the name of the extension point is Maternity Leave, Sick Leave so that name has to be match here Maternity Leave, Sick Leave.

And then below that we try to mark what is the action that this extension will take which in this case verification of medical certificate. So for example for the other extension point this are the other extension point where we have Parental leave so the extend relationship will show what is a Parental leave condition on which this checkbox report to be done and here we specify is to what is action that this will take.

So extension often is a very convenient way to relate different conditional Use-Cases to one or more base Use-Case, it is possible that some extended Use-Case may be associated with multiple different extension points in different base Use-Cases as well. For example, checking medical

report here is happening in terms of approval it may need to be also invoked in case of say revoking a leave, you may really need to check on the report to actually decide whether the revocation can at all be permitted in the system or not.

(Refer Slide Time: 16:31)



So this was the second kind of relationship. The third kind of relationship that may exist between Use-Cases is the Generalization relationship. The concept of generalization and you must have experienced by now that this basic concept of generalization specialization or hierarchy is repeatedly occurring in the scenarios. So it works in the same way as does with classes. So there is a base case A; there is a sub use case. Here is use the terminology sub use case B.

So base case A is self-contained, that it has a within that it needs for the invocation. But the sub case B needs A because what it does as in class associated generalization specialization or inheritance the sub use case inherits-- this inherits all the functionalities of the base case A. And after inheritance-- so this is similar so it inherits the sub use case inherits the behavior of A, the base case but it is allowed to override and extend it.

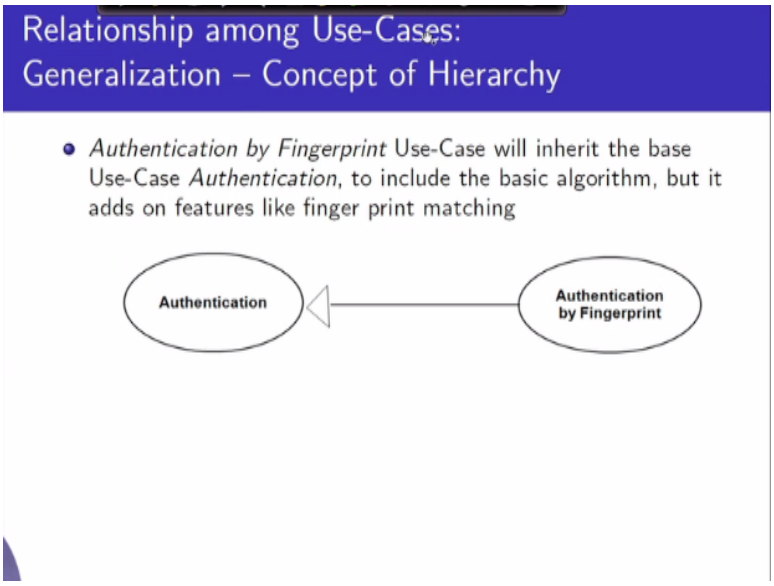
Override means that once Use-Case has been inherited it has different behavior it has different possible operations that the Use-Case may invoke. The sub case B make use these operations as it is from A or it may want to redefine the behavior of the operation. If it redefines the behavior of the operation this basically means redefine then we say it has overridden the particular

operation from the base use case. Or it may want to extend it which means that it will first execute the operation as in the base use case and then it will add something more.

So the possibility is as it is that is just inherit, override that is use the same name of the operation but have a different functionality or extend that is first use what you have inherited and then do something more, these are all the possibilities. So in overall the B inherits all the relations of A, the whole functionality of A. And in this context it is possible that we can have some Use-Case which is abstract.

So if a Use-Case is abstract will normally write within the Use-Case that this is an abstract Use-Case. What is the meaning of the abstract Use-Case is that this is a scenario of action which will never take place in the real-world but this is being put in place more for the modeling purpose to unify a number of different possible Use-Cases that can happen. And like in abstract classes this is also a regular functionality of Use-Case.

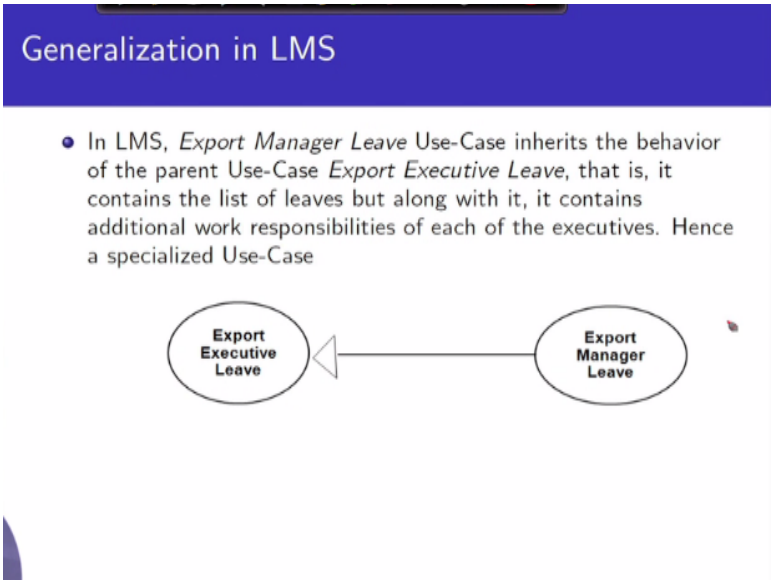
(Refer Slide Time: 20:16)



So as an example let us say I have an authentication system. So I have Use-Cases for authentication which does variety of authentication like password base authentication maybe SMS, OTP based authentication and so on. And I want to define another Use-Case authentication by fingerprint which specializes from the authentication use case. So what is does, it inherits all the different algorithms and operations that the authentication Use-Case has.

But, in addition, it will include some fingerprint matching strategies in the authentication in this Use-Case and that is how the authentication Use-Case will get a specialized in the authentication by fingerprint.

(Refer Slide Time: 21:09)



Talking about LMS example if we think of a Use-Case to export executive leave and another Use-Case as exporting manger leave then we will have again a generalization, specialization relation, so this being the base. If I want to export the manager leave certainly I will do everything that is required for exporting the leave of an executive which is the base case. But in addition, I will need to possibly do some more export of the data in terms of the different employees who report to this manager and their availability during the leave of the manager.

So this will typically have then when I extend I inherit all the functionalities of the export executive leave Use-Case and then execute them and do something more on top of that. In LMS we could also think of abstract Use-Case for example we can say that we have different Use-Cases like ‘Request’ is a Use-Case then ‘Approve’ is a Use-Case then ‘Avail’ is the Use-Case and so on.

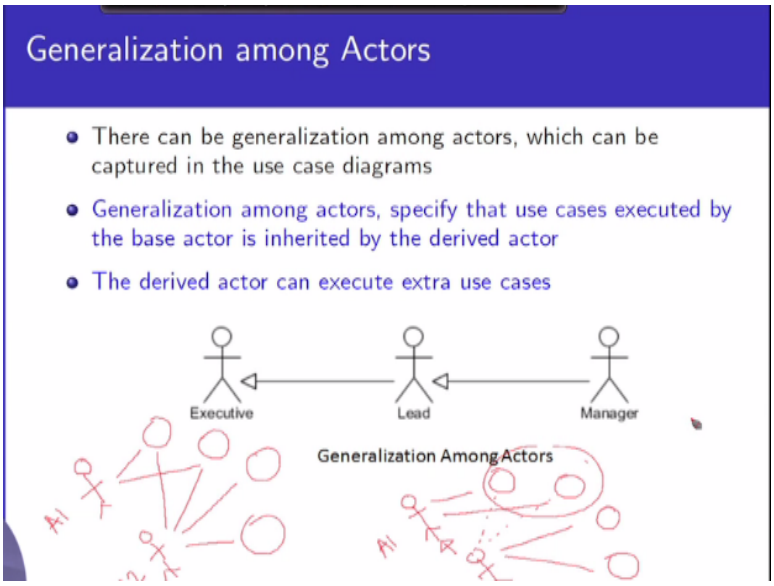
Again say that Use-Case ‘Act on Leave’ which is an abstract. And we can say these are these basically are all specialization of this particular ‘Act on Leave’ Use-Case. So what is the ‘Act on

Leave' Use-Case specifically, it is some action on the leave so this will have the basic functionality of checking out on the leave record maintaining in the leave store and so on which is something which all of these leave Use-Cases will need for its own invocation for its own implementation.

So instead of that all those leave store management related stuff being implemented as a part of all of these specific Use-Case we can club them together and conceive of an abstract Use-Case 'Acting on Leave. Now is this abstract? Because by itself just the functionality of the leave being maintained on the leave database, leave records that being accounted and so on just by itself is not a stated functionality or a perceive functionality if we look into the leave management system.

So- but this will come out through the discussion and analysis of the LMS specification that such abstraction could actually help the system get organized in a better manner.

(Refer Slide Time: 24:12)



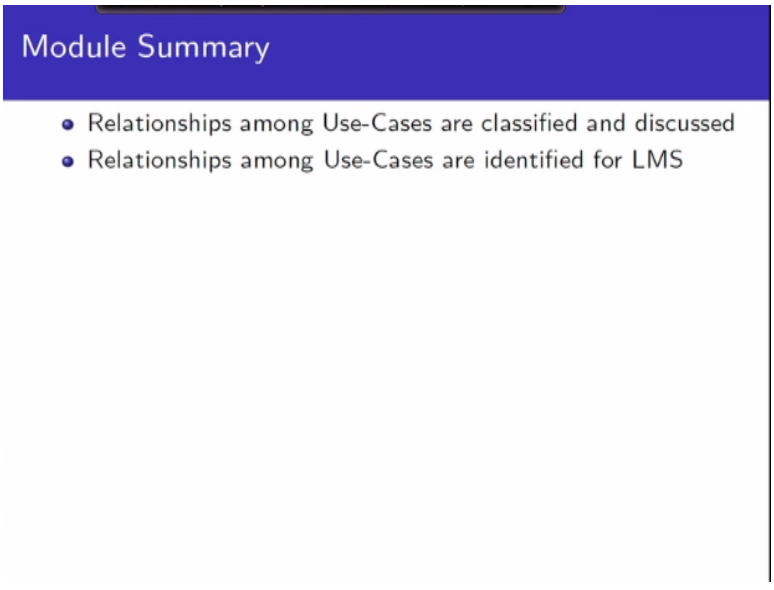
Besides the relationship amongst the Use-Case there could be some more relationship like there could be a generalization relationship between actors. That is we can think that actors like an executive the example is a best way to look at executive is an actor lead is an actor and we have already in terms of identifying classes we observe this that lead as a employee can do anything that executive as an employee can do but lead can do something more.

So again have a base so this is the base actor and this is the sub-actor or the specialized actor who inherits all the functions of what the executive actor and do but lead actor can do something more and in turn that could be generalized-- specialized fact in terms of the manager actor. So what happens is often this is very useful as we will see that if we have two actors one is here and one is here and suppose we have four different Use-Cases, and this actor can use this two and this actor can use say this four instead.

So basically it means that say the actor A1 and actor A2, A2 can do anything that A1 can do then it maybe more explicit to model this separately that I say this is A1 this is A2 and there is a specialization of A2 from A1 and of the different Use-Cases I will associate these two with A1 and only these two A2.

I actually do not draw this association because since A2 is a A1 it is implicit that all these Use-Cases will also be associated with A2 that often simplifies the (()) (26:28) and simplifies a model much better and give a clarity in terms of the Use-Case diagram.

(Refer Slide Time: 26:36)



So to summarize this module here we have for continued to our discussion on the Use-Case diagram and in addition to the actors and Use-Cases we have now seen that there could be relationships between Use-Cases and in specific we have talked about <<include>>, <<extend>>

and Generalization relationship between Use-Cases. And we have also says examples so of the LMS system.

We will continue this discussion in the next module as well were we will try to explore and consolidate the Use-Case diagram for the LMS system.