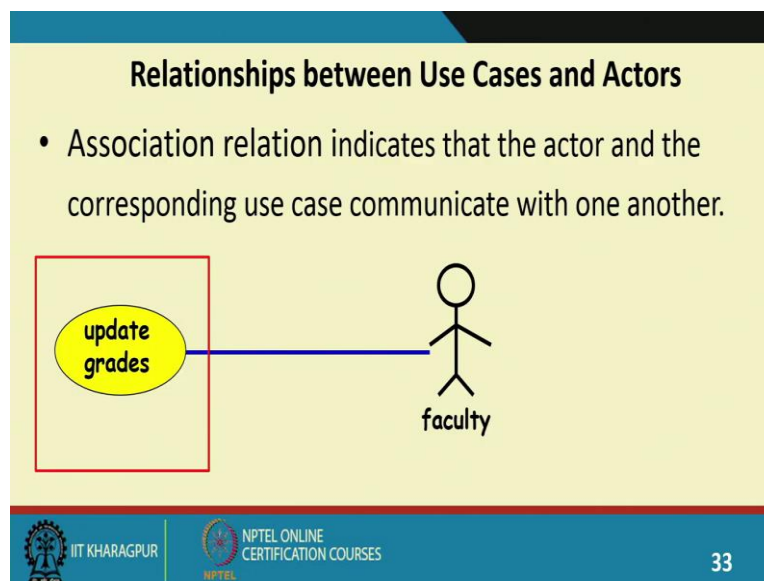


Object-Oriented System Development using UML, Java and Patterns
Professor Rajib Mall
Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur
Factoring Use Case
Lecture 04

Welcome to this lecture.

In the last lecture, we had started discussing the Use Case Model. We had said that the Use Case Model is the core model. Use Case Model is developed first among all models in a development process. Also, we have seen that Use Case Model is a very simple model, easy to draw, easy to develop, and easily understood by the user also with very limited knowledge of software and computer. The Use Case Model expresses the user's requirements, and based on this model, the other models are developed by refining this model. In the last lecture, we were just discussing how the use case is represented, the communicates relation between the users, and the use case.

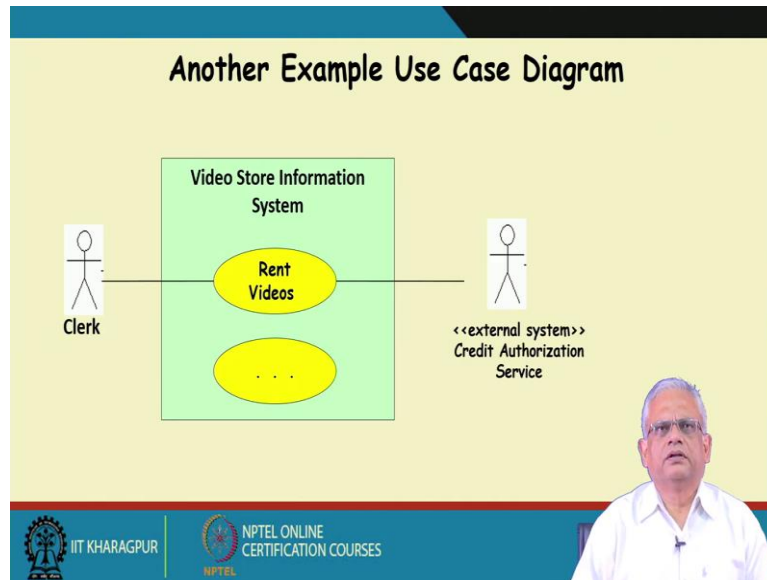
(Refer Slide Time: 01:16)



We discussed that the line drawn between the use case and the actor is called the communicates relation: it is an association. Association relation between the use case and the actor indicates that they communicate with each other. But it does not indicate that data is input or data is displayed. It's just indicated that the user or the actor invokes the use case. But of course, it may also happen that the user inputs certain data at certain times and the system produces some results visible to the user at different times. However, that is not mandatory to represent in Use Case Model. The user may just invoke, and there may not be

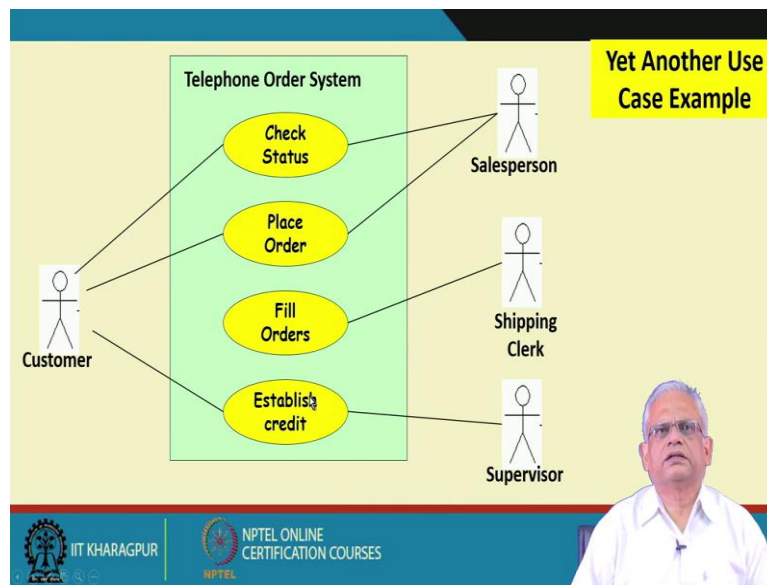
any data display. But typically, the user not only invokes but also inputs data, and also some display is given to the user.

(Refer Slide Time: 02:41)



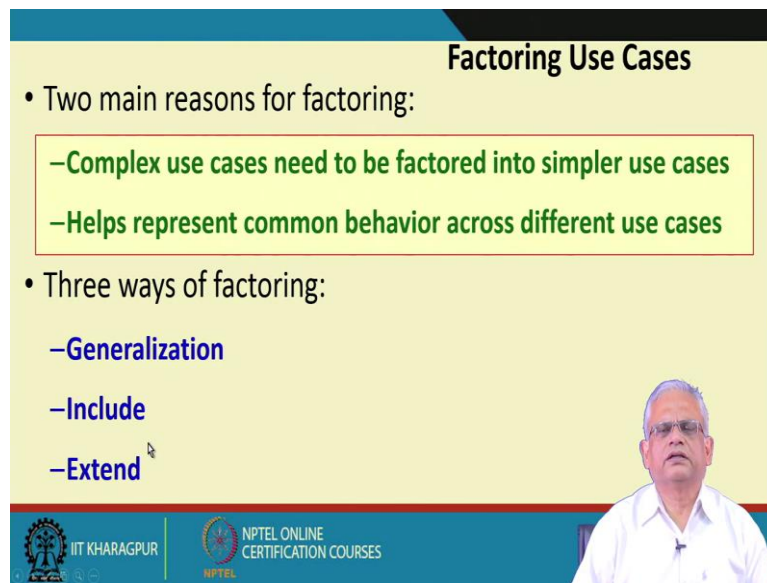
This is another example of a Use Case Diagram (in the above slide). The name of the software is 'Video Store Information System'. Here, one of the functionality or use case is Rent Videos. The clerk is authorized to rent videos. So possibly, a customer chooses a video and gives it to a clerk sitting there. The clerk issues the video through the system and then asks the customer to make payment for the video and takes the card from the user and tries to make payment. There is also an external software or external system called as the 'credit authorization service'. As the card is given it tries to authenticate the card using the credit authorization service. There are other use cases but which are not indicated here.

(Refer Slide Time: 04:04)



Now we will see a slightly larger example. This is a ‘Telephone Order System’ (in the above slide). This is e-commerce software where the orders are placed over the telephone. The customer telephones to the system, and there is a message played. The user can choose which option, and based on that he can check status of an order. Here, salesperson also can check status of an order. The customer may be using a telephone interface and the salesperson may be actually sitting in front of a terminal with keyboard and checking the status. The customer can place order on the telephone order system. The salesperson also can place an order and the shipping clerk checks out what are the orders that have been placed and fulfils the orders and enters the details in ‘Fill Orders,’ and during placing order, the customer may establish credit, and here supervisor may check the credit establishment.

(Refer Slide Time: 05:40)



The slide is titled "Factoring Use Cases" and is presented by a speaker whose video feed is visible in the bottom right corner. The slide content is as follows:

- Two main reasons for factoring:
 - Complex use cases need to be factored into simpler use cases
 - Helps represent common behavior across different use cases
- Three ways of factoring:
 - Generalization
 - Include
 - Extend

At the bottom of the slide, there are logos for IIT KHARAGPUR and NPTEL ONLINE CERTIFICATION COURSES.

One thing is that, the diagrams we just now saw, in that high-level functionalities represented straightaway. But sometimes these high-level functionalities need to split into smaller use cases. More clearly can be said that sometimes the high-level requirements that are represented on the Use Case Diagram need to be split into smaller use cases. There are two main reasons why we want to split or factor into smaller use cases. One is that if the use case is very complex, then it becomes difficult to implement efficiently. As we go to the design process, we will see that a complex use case cannot even fit into a page when we convert into other diagrams. More clearly, the other diagrams or the other models we develop based on a complex use case, those diagrams become extremely complex and cannot even fit into one screen or page, becomes difficult to develop. Therefore, we split them into simpler use cases so that the later diagrams become simpler.

Another reason is that if we can split a use case into simpler use cases, some of the simpler use cases may be part of other use cases. So, reuse becomes possible. If we split a use case, the small functionalities they are used across other use cases. So, it becomes possible to have the common behavior factored out in the Use Case Diagram itself so that the code is not repeated or the design is not repeated. Expert use case modelers, they know the common behavior across use cases and they factor this out in the Use Case Diagram itself so that the later model becomes clean, efficient.

Otherwise, if we don't factor out use cases here, big chunks of functionality are common among different use cases. In the later design process, it is very likely that there may be

duplication of design, and the model will become complicated. So, the code becomes large and more effort is required.

But The question is how do we factor? Given that a use case is complex what do we do to factor it?

Actually, there are three ways we can factor. One is called as generalization. The second is include and third is extend. Let see how these three techniques, generalization, include and extend can be used for factoring a complex use case.

(Refer Slide Time: 09:24)

Generalization

- The child use case inherits the behavior of the parent use case.
 - The child may add to or override some of the behavior of its parent.

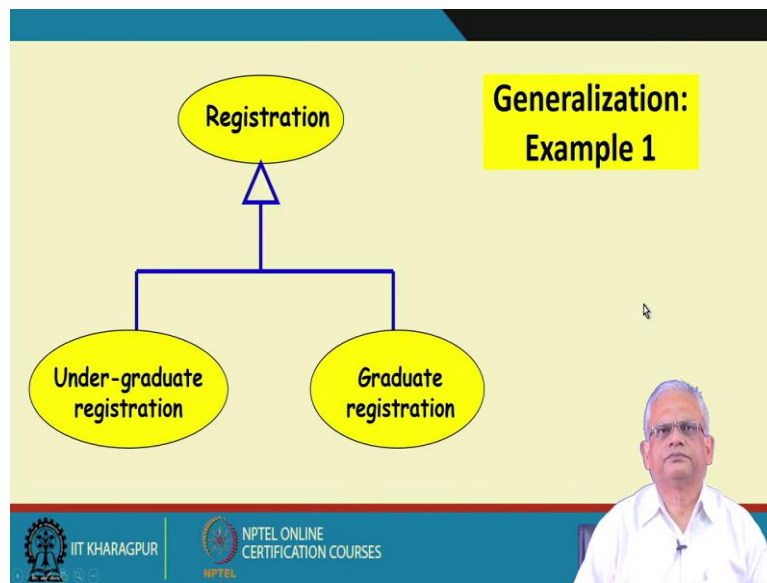
The diagram shows a yellow oval labeled 'parent' at the top and a yellow oval labeled 'child' at the bottom. A blue arrow points from the 'child' oval up to the 'parent' oval, indicating inheritance.

Logos for IIT KHARAGPUR and NPTEL ONLINE CERTIFICATION COURSES are visible at the bottom of the slide.

Let's look at the generalization first. Here we represent them using an arrow (as shown in the above slide). One use case has been split into two use cases, the parent, and the child. The parent is a base use case or simple use case, and the child is the more complicated use case. But some part of the child is already present in parent because child inherited parent here.

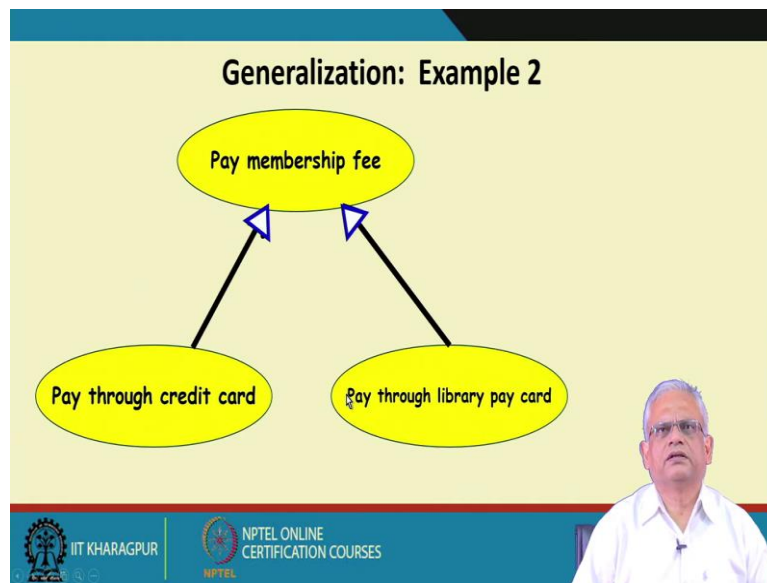
The child use case inherits the behavior of the parent use case, and the extra behavior that is needed are represented in the child use case. So, we have effectively split a complex use case into two parts. One is the parent part, the basic functionality, and the extra functionality is added in the child use case. And the parent behavior is inherited by the child behavior, so it may possible some of the functionality in the parent overridden by the child use case.

(Refer Slide Time: 10:49)



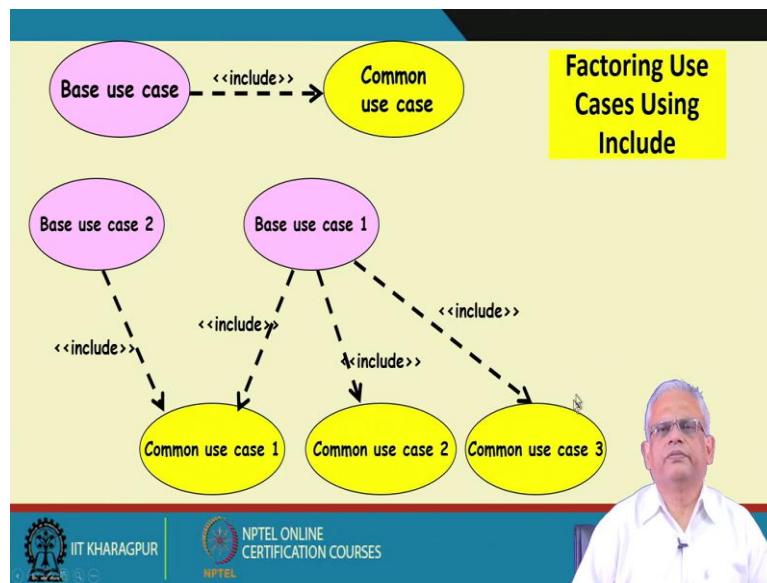
Just to give an example of a generalization, let say we have a use case for student registration before a semester starts. Now there are different categories of students. And the use case becomes very complex if all categories of students use the same use case. During the registration process, some of the functionalities are common. For example, enter the roll number, verify the address, identify the subjects to enter, to credit, and so on. But then, in let say the UG registration, they just need to indicate the hostel in which they will reside, whereas in a graduate registration, they may stay outside but they indicate a bank account in which their scholarship will be credited. The graduate students get scholarship. The undergraduate students don't get scholarships. The undergraduate students stay in the hostel and the graduate students don't stay in the hostel. This is the only difference. Otherwise, the registration process is almost similar. So, the similar things are done here in the parent use case, and the extra things are done here under the undergraduate registration and the graduate registration (in the above slide).

(Refer Slide Time: 12:41)



Let's take another example. Let say in a library, the membership fee paid by a member has some procedure. For example, need to give the library member number, choose the duration whether you want to have the membership for six months or one year or two years, and so on. There are two options exist for paying fees. One is that, can pay through credit card, and another option is pay through the library pay card. In pay through library card, the library card is internally validated and the procedure is slightly different: the amount is debited from library card balance. In pay through credit card option, specific code has to be sent to an external system, authenticated, and then the amount has to be debited and deposited in the library. So, a different procedure for pay through credit card and a different procedure in pay through library pay card. But both of these inherit the pay membership fee. So, the basic procedures like entering the library membership number, selecting the duration for which the membership is to be renewed, etc. are the same for both the paying methods.

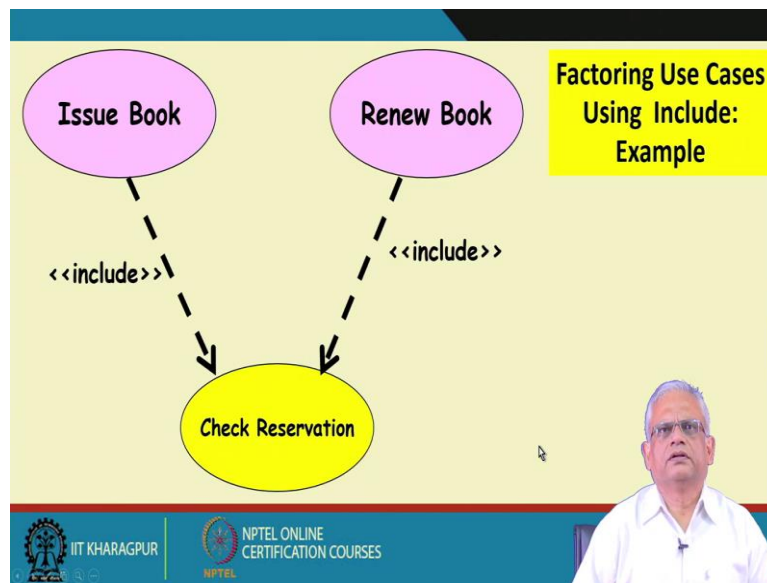
(Refer Slide Time: 14:30)



Now let's look at the include technique for factoring use cases. Here we use a dotted arrow (as shown in the above slide). For the extend, the inherits or the generalization we were using a solid arrow with a closed arrow head. But, for include we use a dotted arrow with open arrow head. The notation is important because these are standard notations. In the above slide, Base use case 1 include common use cases. So, the behavior of common use cases have factored out from the base use case and this behavior may get included in many use cases achieving reusability.

We can also see from the above slide, where there are two base use cases (Base use case 1 and Base use case 2) and each of these include the common use case 1. So, the functionality for common use case 1 is elaborated and that serves towards both the base use case. In the later design process this will be elaborated into different models and that serves towards both of these. Otherwise if we didn't have this, we will develop the model corresponding common use case 1 for both the part. So, it helps if we can identify the common functionality using the include relation. The Base use case 1, includes these three (Common use case 1, Common use case 2, and Common use case 3) use cases and the Base use case 2 includes only Common use case 1. Later, may be these common use cases are included by other use cases or maybe they are not included at all in other use cases.

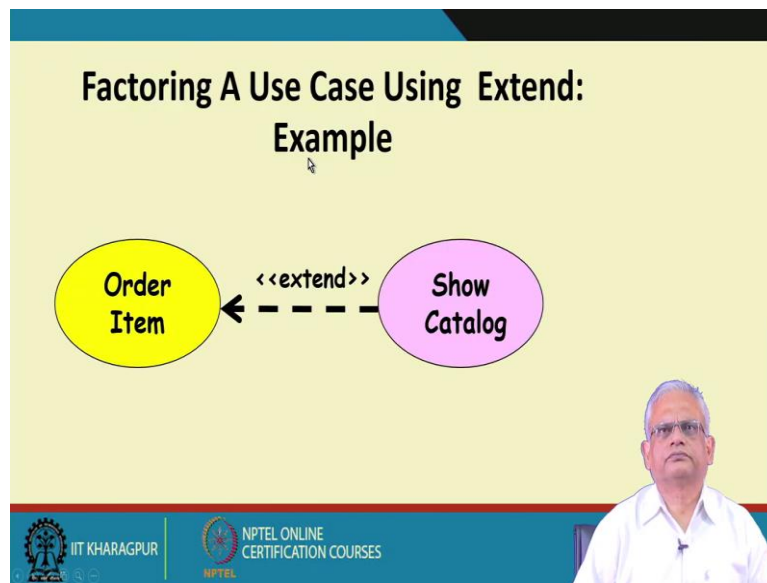
(Refer Slide Time: 17:09)



Now let's look at an example of the include relation (in the above slide). Let say during Issue Book, once a member tries to issue a book out, the book and the library membership code are scanned and then it is checked whether the book can be issued by checking whether any other user has reserved it. So, the check reservation functionality is invoked to check if any other user has reserved the book. We say that the issue book compulsorily includes the check reservation use case. Similarly, when a book is to be renewed the Renew Book use case is invoked and it compulsorily includes the check reservation during the renew book process.

If any other user has reserved the book then the Renew Book will fail. It will say that the book has been reserved, cannot renew the book. So effectively this diagram we have factored the Issue Book and Renew Book into simpler use cases and also it serves in identifying the common use cases that are part of both of these and this is developed only once. Otherwise there will be duplication while developing the Issue Book and Renew Book.

(Refer Slide Time: 19:07)



Now let's look at the third way of factoring using the extend (in the above slide). The notation is almost similar to the include relation. It is a dotted arrow with open arrow head but then the stereotype is here extend. And also, another thing to notice is that 'Order Item' is the base use case and 'Show Catalog' is the use case which is to be included part of the Order Item but optionally. In the include relationship it is compulsorily included and there the arrow head on 'Show catalog' side and also it was written include instead of extend. In case of extend, during the Order Item, the user may not need a catalog. He just knows the item code and he is very familiar with the product, the warranty, et cetera and just goes ahead and orders the item without even looking at the catalog. But some users, during the ordering process, they might need the catalog. So, there may be a button on the user interface for Order Item. If you click on that the catalog will be shown. In the catalog the details of the item would be annotated. For example, what is the item code, what is the warranty, price, et cetera, et cetera. So, by using the extend relationship we split a use case into simpler use case but then the factored use case is optionally included. Sometimes it may get included in the Order Item, and for some user they may not need this use case.

(Refer Slide Time: 21:47)

Extension Point

- The base use case may include/extend other use cases:
 - At certain points during execution, called extension points.
- Note the direction of the arrow

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Another thing is that we might also indicate the extension point during the include and extend. That is at what point of execution actually this is gets included or extended. We indicate that with this line (in the above slide perform sale done after check out) and after that we write the point of extension that is the Gift wrap product may get invoked after the checkout. So once the sale is performed and the checkout is done, at that point the Gift wrap product option will be shown and it's possible to select the Gift wrap product option and the product will get gift wrapped. So, the point at which the gift wrap option will be given, so that can be indicated here as after checkout. So that's called as the extension point.

(Refer Slide Time: 23:03)

Use Case Relationships

```
graph TD
    Actor[Sales Person] --- PlaceOrder[Place Order]
    PlaceOrder -.->|<<include>>| SupplyCustomerData[Supply Customer Data]
    PlaceOrder -.->|<<include>>| OrderProduct[Order Product]
    OrderProduct -.->|<<include>>| ArrangePayment[Arrange Payment]
    ArrangePayment -.->|<<include>>| CashPayment[Cash Payment]
    ArrangePayment -.->|<<include>>| CreditPayment[Credit Payment]
    RequestCatalog[Request Catalog] -.->|<<extend>>| PlaceOrder
```

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

This is a use case Place Order (in the above slide) which is factored into many use cases. And see in the slide, there are three include: the Place Order invokes or includes the Supply Customer Data use case (gets the customer data), includes the Order Product use case (here the order procedure is carried out) and also includes Arrange Payment use case. Arrange payment is factored into Cash Payment and Credit Payment. And also, during the Place Order, optionally the Request Catalog functionality may be invoked. Here, decomposition of the Place Order into smaller use cases incorporates all the three decomposition techniques, that is include, extend and also the generalization.

(Refer Slide Time: 24:23)



The slide, titled "Example 1: Video Store Information System", lists the following business functions supported by the system:

- Video Store Information System supports the following business functions:
 - Entering the information about all videos that the store owns
 - This database is searchable by staff and all customers
 - Store information about a customer's borrowed videos
 - Access by staff and customer. It involves video database searching.
 - Staff can record video rentals and returns by customers. It involves video database searching.
 - Staff can maintain customer and video information.
 - Store Manager can generate various reports.

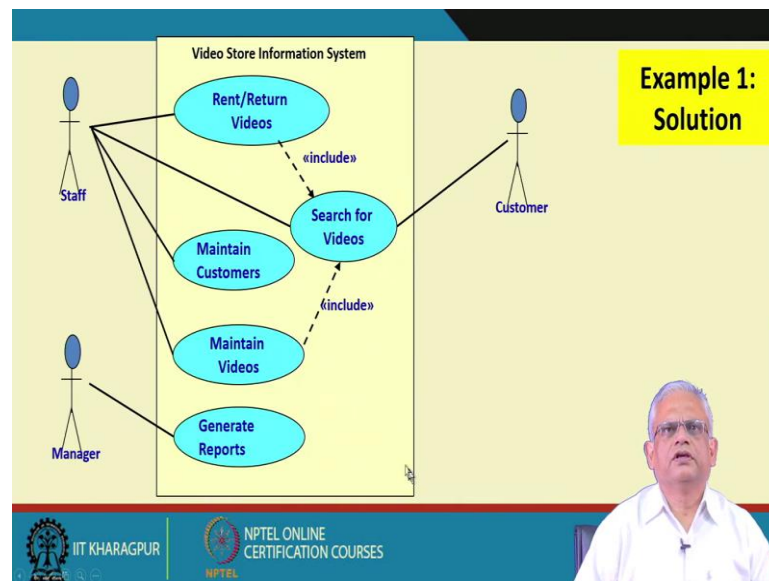
The slide also features the logos of IIT KHARAGPUR and NPTEL ONLINE CERTIFICATION COURSES, and a small portrait of a man in a white shirt.

Now let's try to do one example. We will try to develop a use case for the video store information system (in the above slide). We have an informal text description and based on that we will identify what are the use cases. Remember that use case is a functionality to be performed by the system and we will identify the user who will invoke this functionality and then we will document each of this functionality as a use case and also connect them to the appropriate users. The first functionality is about entering the information about the video store. All the videos that the store owns need to be entered one by one into a database and the database once created will be searchable by staff and customers to find whether a video exists.

So, there are two use cases actually here. One is that enter all the videos and then the searching. And once a customer borrows a video that information is entered. Again, this borrowed information can be accessed by staff and customer and it involves video database searching. The staff can record the video rentals and the returns made by the customer and

this also involves video database searching. The staff can maintain customer and video information, that is they can delete some customers, delete some videos and so on. So, if we look at it, these are the functionalities (rent videos, maintain customers, maintain videos, generate reports etc.) and there are some functionalities which is included as part of many use cases, one is video database searching.

(Refer Slide Time: 27:01)



If we represent all of that in the form of a Use Case Diagram, we will see that rent and return videos that's done by the staff. The customer can search for videos and rent video and maintain video. The staff can maintain customers and the manager generates reports. So, we have captured all the functionalities to be supported as was described in the text and represented that in the form of a Use Case Model.

We will stop here, and we will continue in the next lecture with more examples and also, we will see that this diagram is definitely a large part of the Use Case Model. But then we also need a systematic text description of the diagram. This diagram does not convey some information which we will note down in a text description. This is the basic model but it will be accompanied by a text description.

We will stop here and continue in the next lecture.

Thank you.