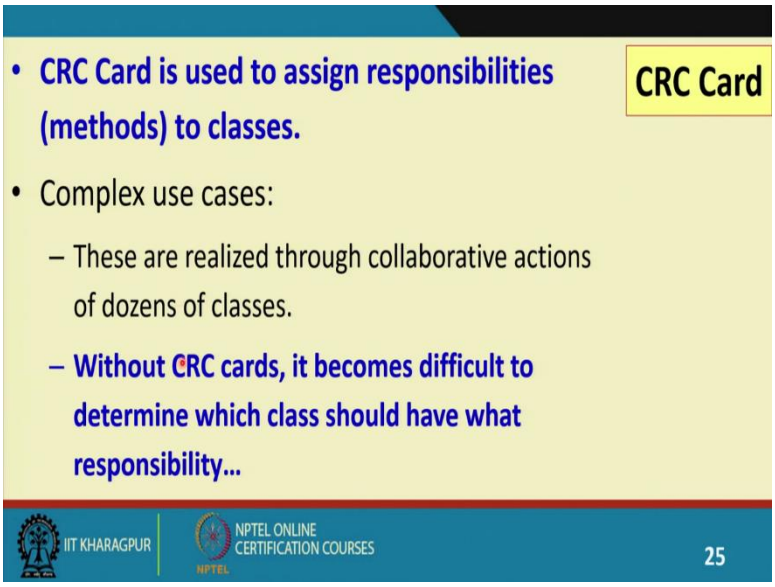
Object- Oriented System Development Using UML, JAVA and Patterns
Professor Rajib Mall
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur
Lecture 33
CRC Cards

Welcome to this session! In the last session we had taken a small problem and we worked through all the designs steps, starting from use case development, the domain model and then the sequence diagram and from the sequence diagram, the class diagram could be developed where the different methods are populated in the class diagram.

But for large problems where the use case involves interaction among dozens of objects. The simple method we discussed earlier by drawing sequence diagram from reading the problem may turn out to be very difficult and error prone. And for that the CRC card technique has been developed to handle such use cases where large number of objects interact.

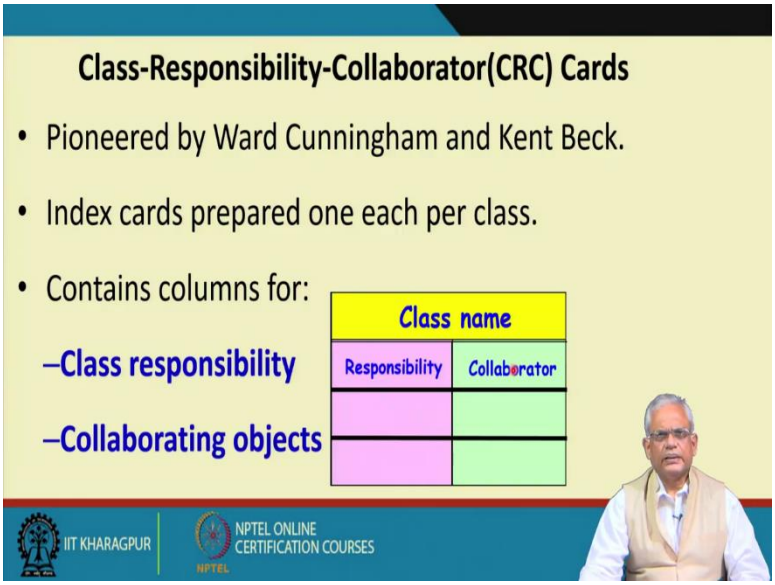
In this session, let us start discussing about the CRC card. The CRC card stands for Class Responsibility Collaboration. Remember that the responsibility in our design terminology is basically the methods. So, using the CRC cards we will assign responsibilities or the methods to the classes which in other words is basically the sequence diagram. Once the methods are assigned by the CRC card we can have the sequence diagram easily developed.

(Refer Slide Time: 02:13)



Now, for complex use cases, dozens of object may interact and without a CRC card it becomes very difficult and error prone to draw the sequence diagram.

(Refer Slide Time: 02:30)



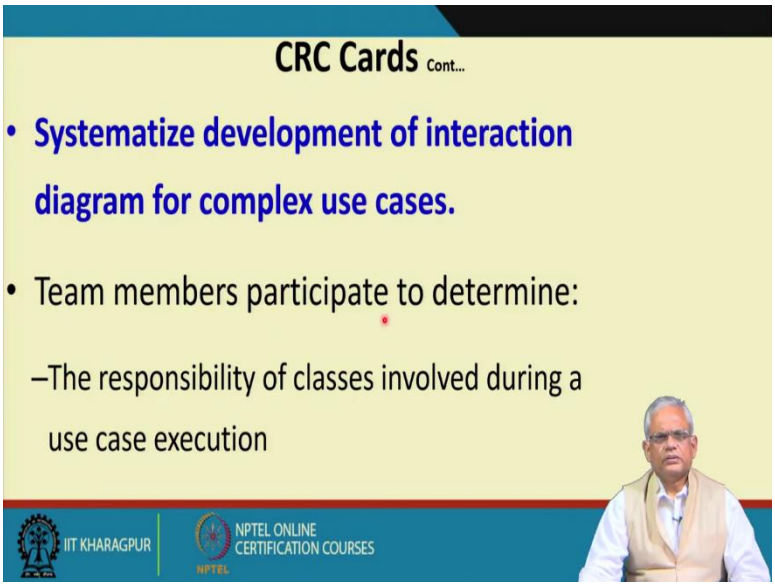
But what exactly is a CRC card? A CRC card is just like a playing card. We take small white papers and on that we draw this kind of table as shown in the figure and on the top of the table for each use case we are trying to develop the interaction diagram, we identify what are the objects that are going to interact. Of course we might miss some of them, we will identify those missed objects as we start the activity with the CRC card.

Or we might have some extra classes, which will eliminate during the CRC card process, for each use case we identify the classes and then from the domain model these are identified. We look at the domain model what are the classes present, we prepare one class one card for each class present in the domain model.

We write the class name and then these columns initially are blank and this is the method which was developed by Cunningham and Kent Beck. There are two columns here, the first column is the responsibility. The responsibility we have been saying is the same as the method supported by a class. The method supported is the responsibility of the class.

And to complete the method, the responsibility, the object may need to take help of another object. For example, it may get some details from another object and so on. And those objects we will write down here and they are the collaborators. For a given responsibility there may be one or more collaborators we will write them down here. We will just go throw the simple example to see how we will use this card.

(Refer Slide Time: 04:53)



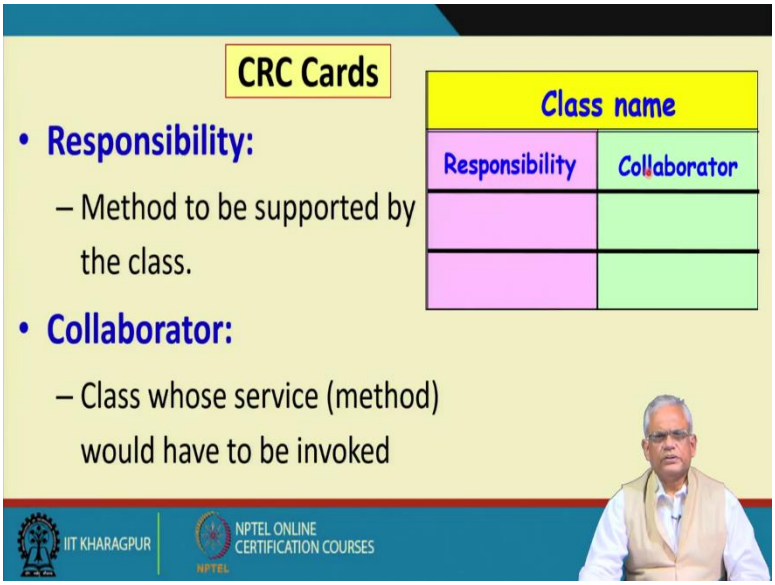
The CRC card basically systematizes the sequence diagram drawing process. The sequence diagram that we drew was more institutive, we had a good understanding of the use case execution and then we just developed the sequence diagram based on our understanding.

But here the CRC cards once they are made for each class in the domain model, these are distributed to a set of team members, maybe 3, 4 team members participate depending on the complexity. If there are 20 classes in the design model, may be 4-5 team members participate in the CRC card process. If there are 30 domain objects then may be 6-7 developers participate, each of them gets about 4-5 cards.

Typically, a developer is not given more than 5-6 cards otherwise again becomes very complicated for that developer. And once the developers have been distributed 4-5 cards each, then the use case is read out. We have this use case model where the use cases are there in the pictorial form and also there is a text description.

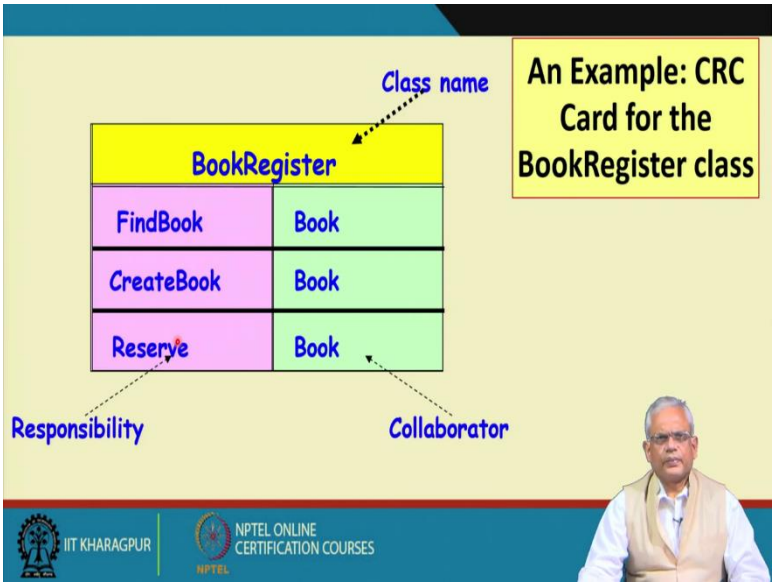
The text description identifies various scenarios starting with the main line scenarios the other scenarios and the interaction that occurs between the user and the computer, as these are read out then it is determined by the developers holding the card. That what exactly needs to be done by a class. And the appropriate class, the person who is holding the appropriate class who will take off the step in the execution of the scenario, he just writes down the specific responsibility of that class.

(Refer Slide Time: 07:45)



And that is the method to be supported. He also identifies any collaborator that may be required these are the class from which some service will be required to accomplish the given responsibility.

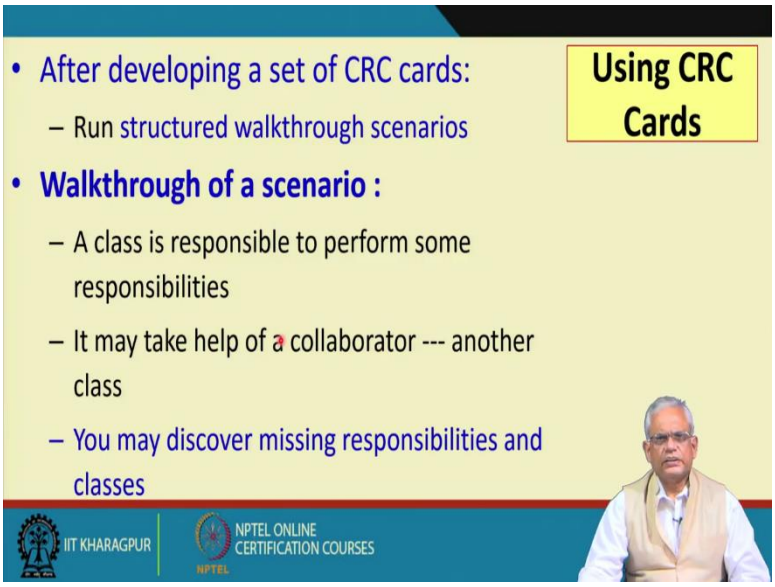
(Refer Slide Time: 08:07)



Just to give an example of a CRC card process, this is a library example for the register, this is the book register class and as the different use cases are read out. Then the member holding the book register class will assig different responsibility to the book register class. For example, find book is the responsibility of the book register and typically as the different responsibilities are read out that is in this step of use case scenario, certain things need to be done.

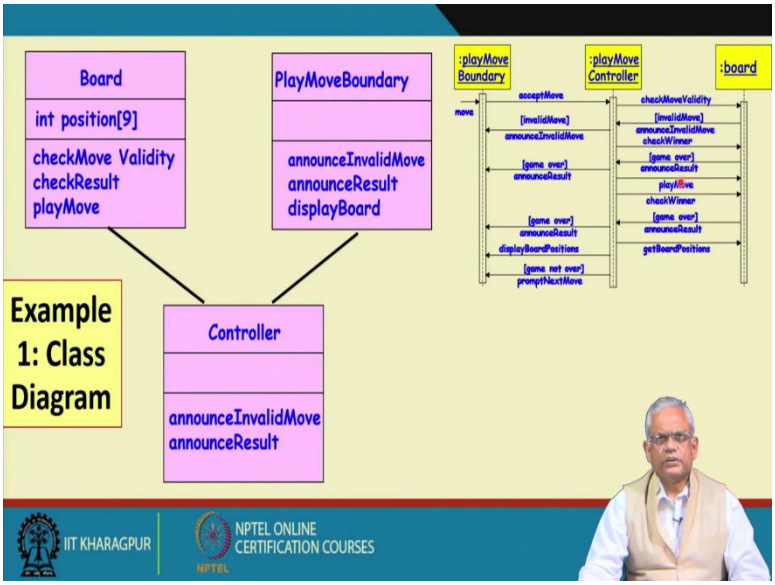
The members may debate among themselves which class is most suitable for handling that responsibility. And the class which is identified to be the most suitable class for performing that responsibility is written here as shown in the figure. For example, find book is the responsibility of the book register, create book is also a responsibility of book register and reserve book is also the responsibility of the book register. And at the end of all the use cases as they are read out, all the responsibilities would have been assigned to all the classes and then based on that, the sequence diagram can be drawn.

(Refer Slide Time: 09:49)



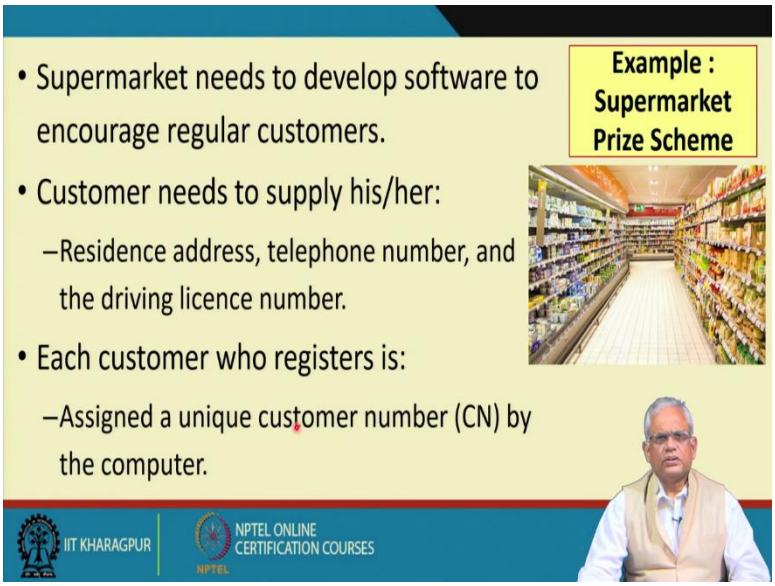
In the CRC card terminology, the team members participate in structured walkthrough of scenarios, that means once the cards have been distributed, the scenarios are read out and they walkthrough the scenario to identify which class needs to do what. The walkthrough of a scenario consists of identifying the responsibility of a class and also identifying the collaborator for that responsibility. The process might find that some responsibilities have been missed in the use case description as well. So, this is the crucial activity.

(Refer Slide Time: 10:48)



Now, once we have the sequence diagram drawn, the population of the methods on the class diagram is rather straight forward. It is done automatically by a case tool. For example, the play move boundary here it has the announce result, announce invalid move and so on. And the controller has the responsibility of accept move and announced invalid move and announced result. The board has check move validity, check result and play move. This is the play move here.

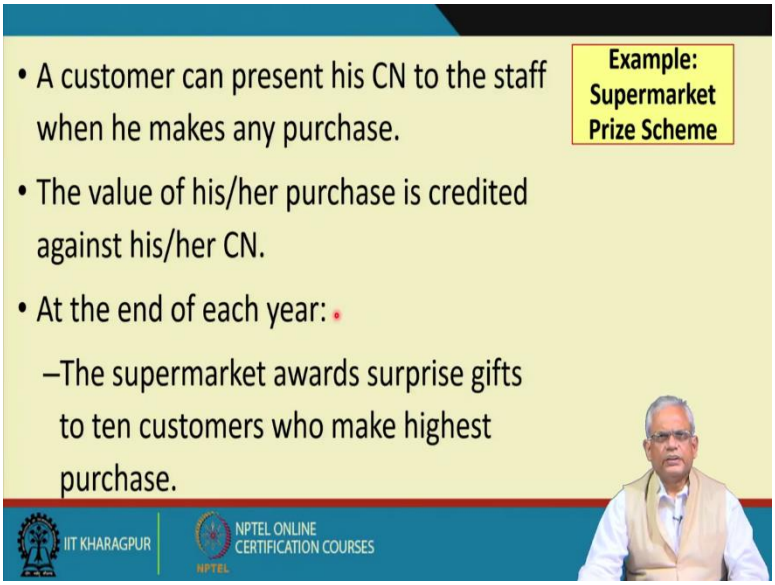
(Refer Slide Time: 11:55)



Now, let us look at another problem because finally we need to solve many problems to get expertise on the basic design process. This problem you may please try yourself before looking at the solution. So, here a super market like this as given in the figure, it needs to

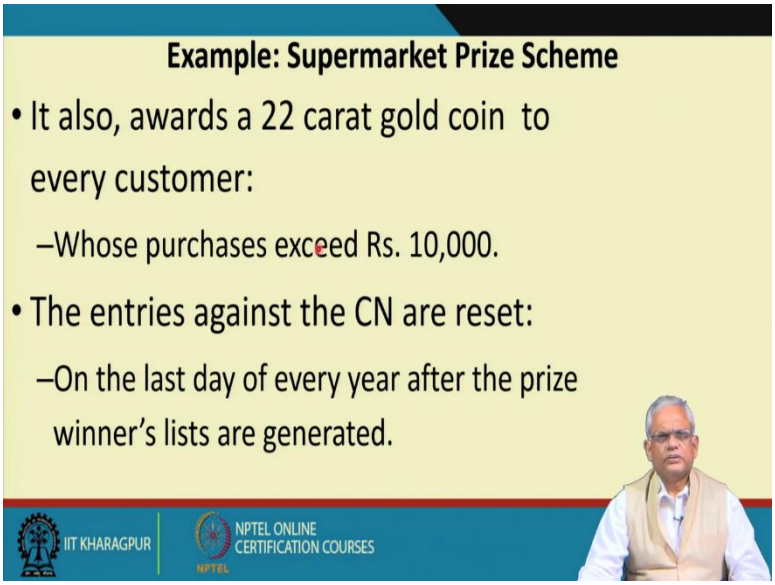
develop a software to encourage regular customers. In this software, the customers will have to first register and they register by supplying their residence address, telephone number and driving license number or Adhaar card number. Each customer who registers is assigned a unique customer number by the computer and the customers can quote their customer number for any interaction with the software.

(Refer Slide Time: 12:55)



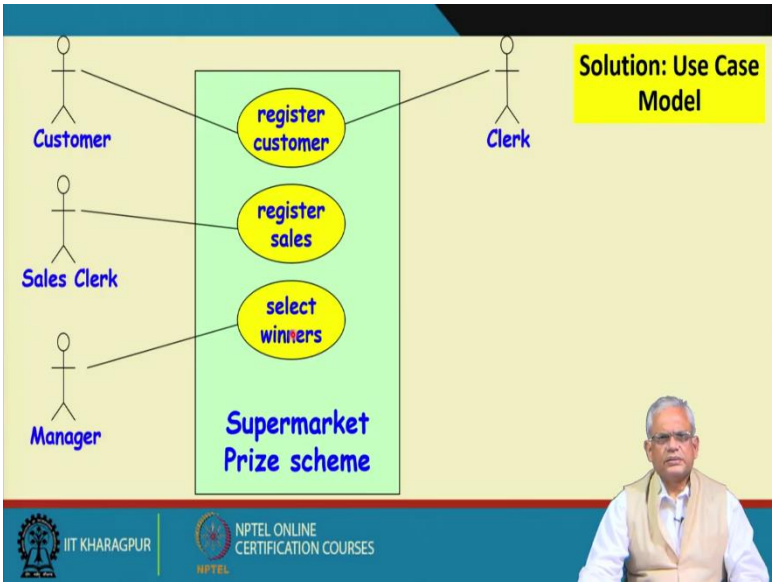
And during the purchase a customer can tell his customer number and the amount purchased is registered in his account. At the end of the year the super market awards gifts those who have made maximum purchases. Just looks through, the software needs to look through all the purchase records and find out to which customer made the maximum purchase.

(Refer Slide Time: 13:29)



And also all the customers who has made certain amount of purchase like 10,000 or something they are awarded a gold coin. And at the end of the year all the entries are reset and in the new year, new entries for purchase start. Now, the first step in designing this is to draw the use case diagram and if we go through the problem, we find that there are only few functionalities of the software which the users invoke. For example, register customer, register purchase and award price and if we think of reset is another function which is evoked by a calendar, we can represent them here.

(Refer Slide Time: 14:27)



So, register customer is done by the customer or the clerk, register sales the sales clerk to whom the customer tells the CIN- the Customer Identification Number and the register sales

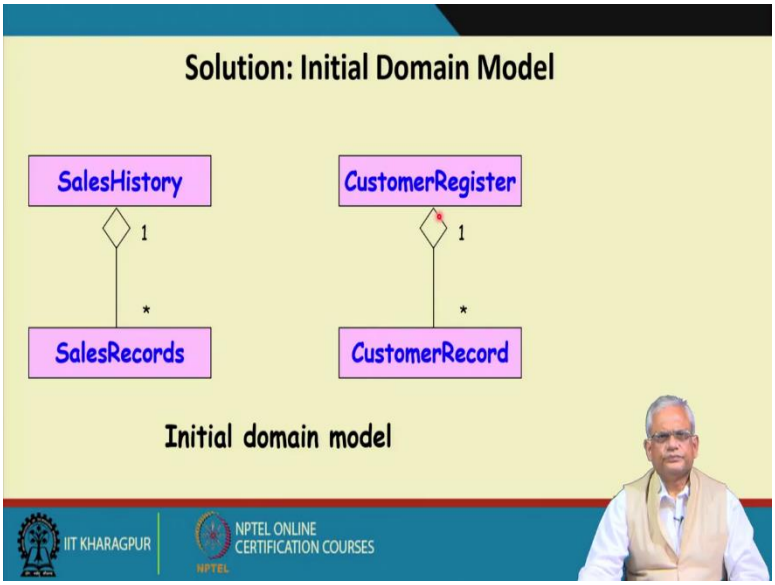
the sales that occurs to the customer is registered in the software. At the end of the year the manager selects the winners for prizes and we can also have another use case here reset. The data is reset by a calendar event, you can draw a calendar actor and then have a reset use case.

Now, having drawn the use case diagram the next step would be to identify the domain model. In the domain model for this example we will have 4 boundary classes, since there are 4 boundary classes, we will name them as customer boundary, clerk boundary, sales boundary and manager select winner boundary.

There will be 3 controller classes because there are 3 use cases, we can name them as register customer controller, register sales controller and select winner controller. Now, the main thing comes to us is to identify the entity classes. But our hint is that the entity classes do store information, do some activities on this, based on that we read through the problem statement, we will find that the customer is the entity class.

Because the customer details need to be stored and we can query about the customer what is address, phone number and so on. And each time a sales occur, the sales data need to be stored for later prize award and therefore the sales records are entity objects, which we need to aggregate in the form of a sale register aggregates the sale records. And select winner does not need any new entity classes.

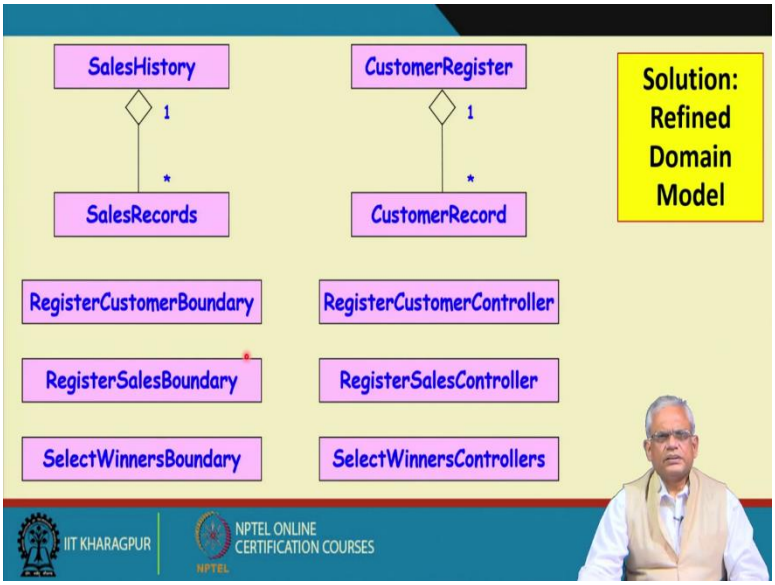
(Refer Slide Time: 17:08)



So, we have these 2 entity classes, the sales records are aggregated by the sales register or the sales history. And the customer record is aggregated by the customer register. Now, the next

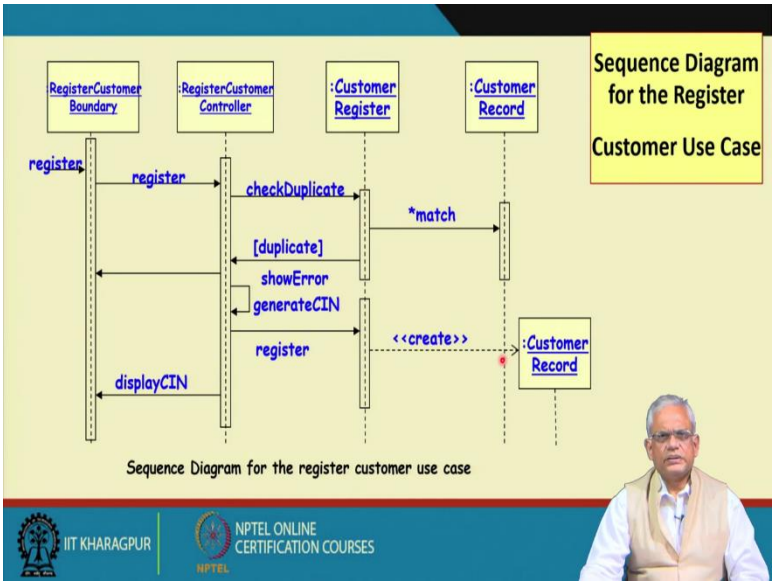
step is to draw the sequence diagram for each of the use cases. We have 3 use cases and this is not a very difficult problem, we do not need a CRC card here, only few objects participate and we can draw.

(Refer Slide Time: 17:40)



This is the complete domain model where we have the entity classes, 3 controllers and 4 boundary classes. We have written only 3 boundary classes, there is one more boundary class. Now, the next step is to identify and draw the sequence diagram, since there are 3 use cases, we will need 3 sequence diagram.

(Refer Slide Time: 18:14)



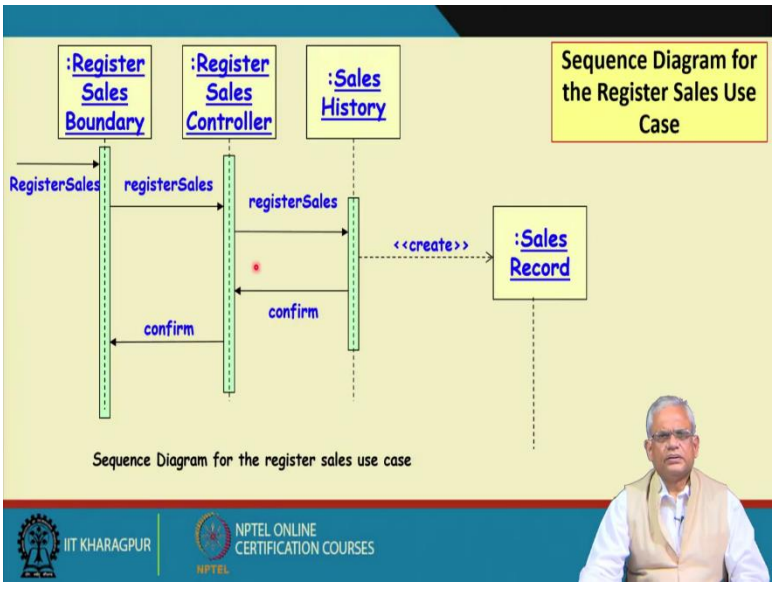
The first sequence diagram is for register customer. In the register customer 4 objects participate, one is the register customer boundary, the register customer controller and then

the customer register and the customer record. Once the register request comes is reported to the controller, the boundary does not do much, it does not have much processing responsibility, it just gets the data and reports to the controller.

The controller has the business logic and it knows what to do when a request comes. The first thing it needs to do is to check duplicate. And that it requests the customer register to identify if there are duplicate. And for that, it needs to match the given details with all the customers record and this is the iteration here and all the records of the customer. And if there is a duplicate then an error is shown here and if there are no errors then the Customer Identification Number is generated and then this is needs to be stored.

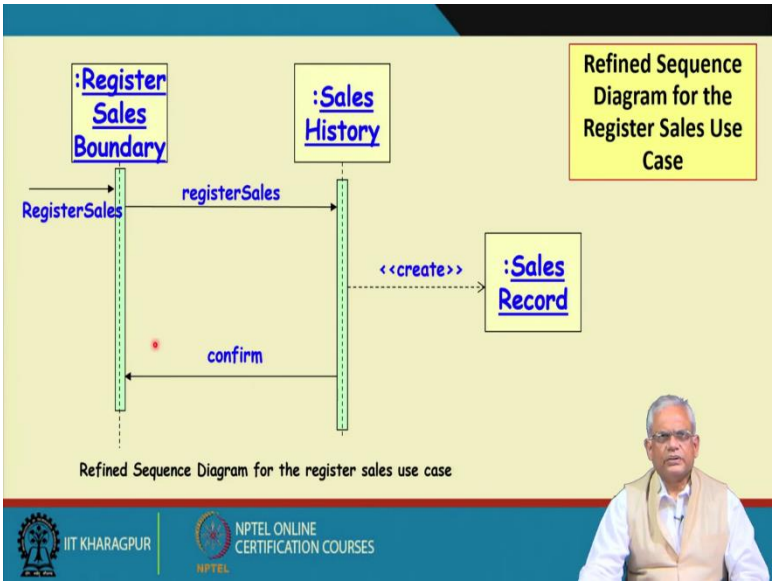
For that the controller request the controller register to register the customer number. And in response the customer register creates the customer record and finally the CIN is displayed. And as this diagram is drawn all these methods will be populated. For example, controller will have the register show error and generate CIN. These will be the methods for the controller and for the customer register check duplicate and register are the two methods that we will be populated for the customer register and for the customer record match and create are the two methods.

(Refer Slide Time: 20:48)



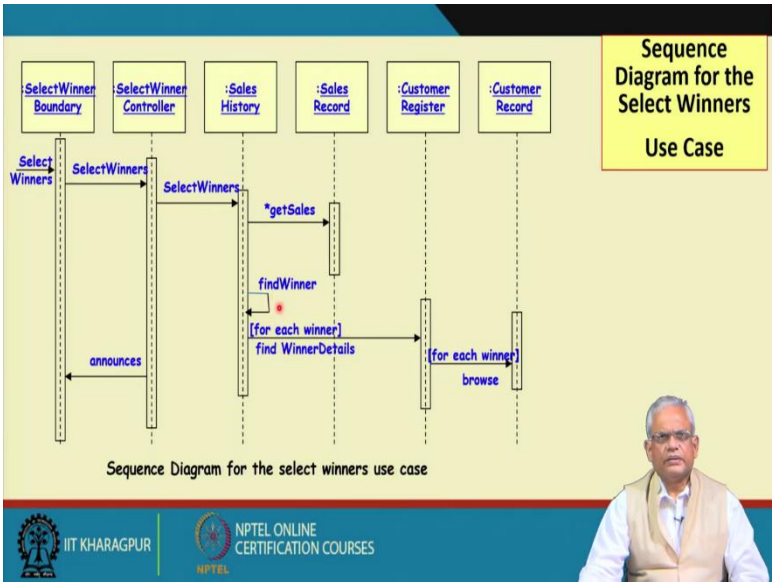
Now, we can draw the sequence diagram for the other use cases. I will not go through the details here, please try to convince yourself that from the problem description this is the use case, you can first draw yourself and then check your answer with this diagram. Of course, it is possible that for the same use case we can have slightly different sequence diagram, but more or less the structure will be similar.

(Refer Slide Time: 21:21)



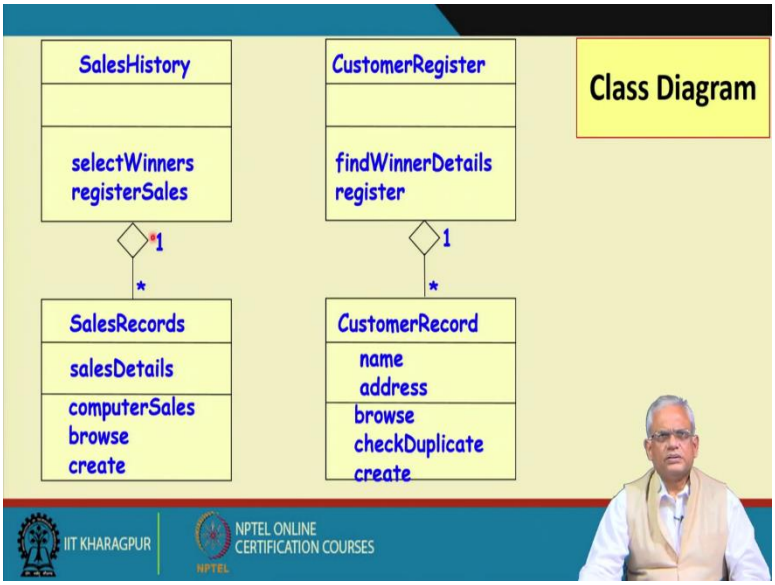
And this is the third used case is about the sale. Once the sale comes, the sale needs to be registered and the sales record is created and it is confirm the boundary.

(Refer Slide Time: 21:35)



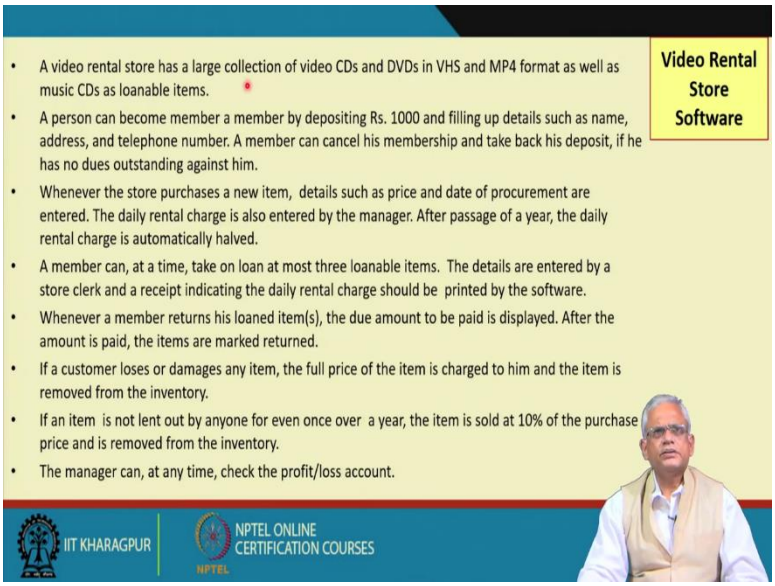
This is the select winner and this more number of objects participate. And if we know that the steps that occur in the sales select winner we can easily draw this, please try to draw this and match with this solution.

(Refer Slide Time: 22:00)



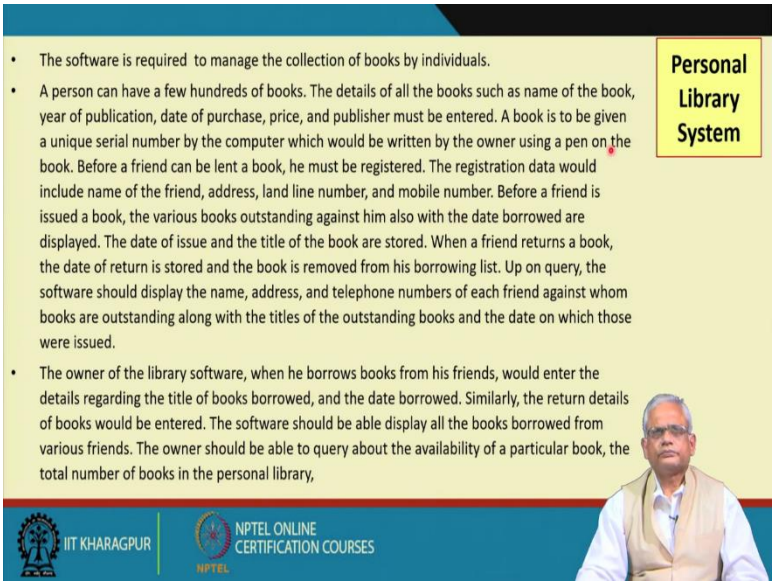
And based on that we can populate the methods on the class diagram.

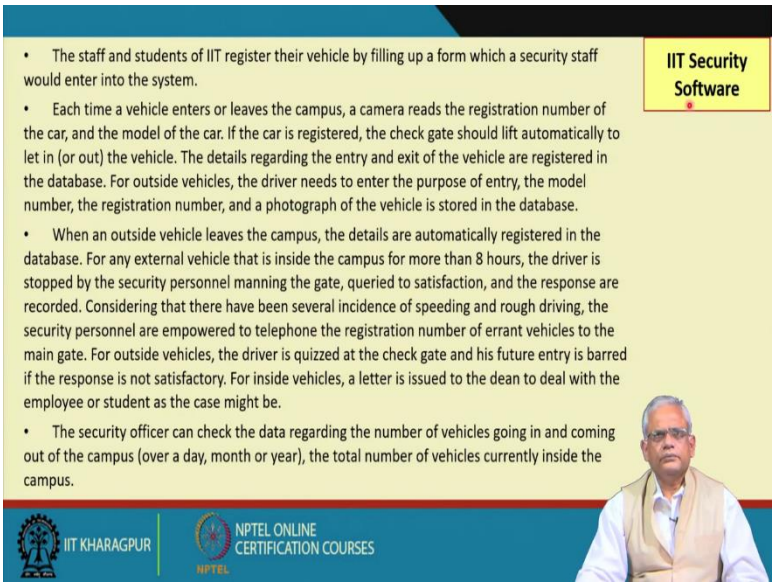
(Refer Slide Time: 22:07)



These are some practice problems, please try to solve this.

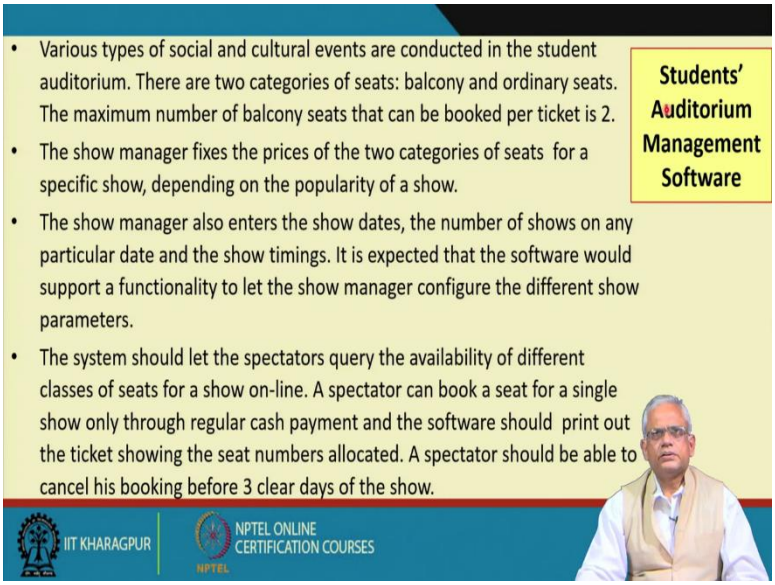
(Refer Slide Time: 22:16)





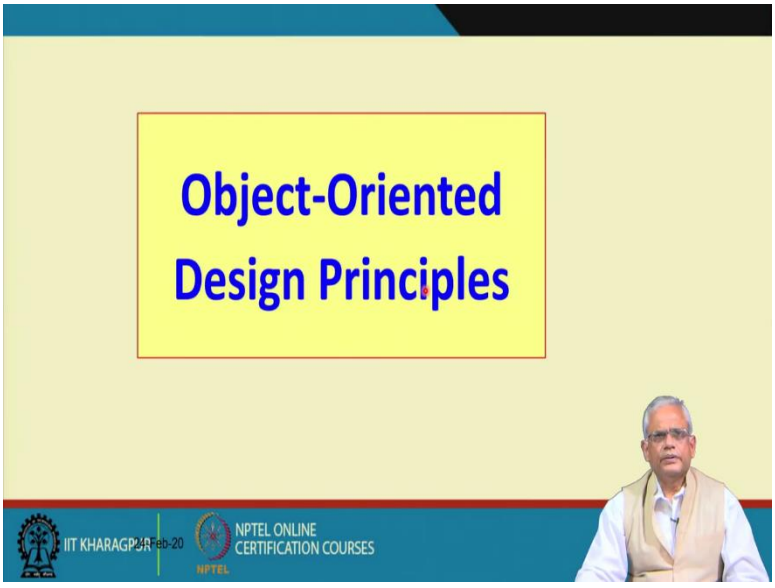
We have displayed few practice problems and this is the personal library system and this is the security software for an institute.

(Refer Slide Time: 22:30)



And finally, this is a student auditorium management software, please go through the problem statement, try to develop the use case model, develop the domain model and based on that, draw one sequence diagram for each use case and finally have the class diagram.

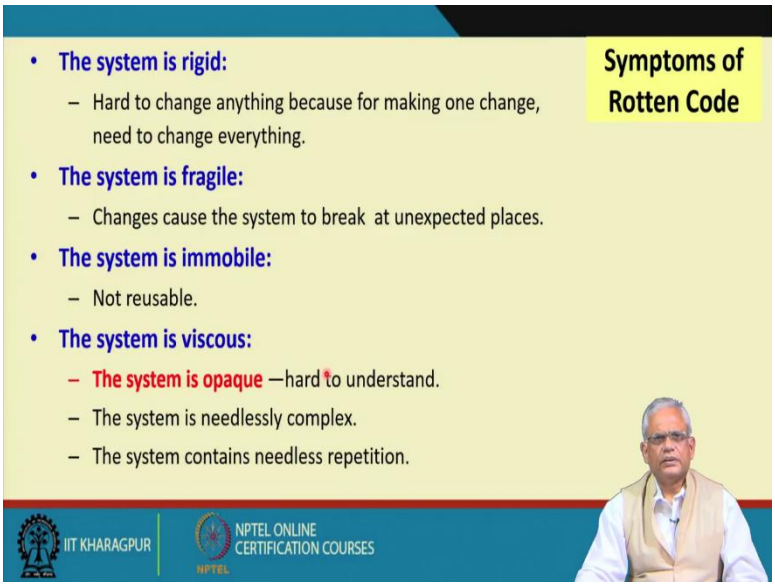
(Refer Slide Time: 23:03)



So far, we have discussed the basic design or the core design method using which we can come up with reasonable design to small problems. Now, we will see how to come up with a better design once we have the ability to come up with the basic design, from now onwards, from this session, we will try to concentrate on how to arrive as a better design using certain design principles and then we will see that these principles are embedded in the patterns.

We will also discuss about the design patterns, the object-oriented design principles we will take only few sessions, may be 2, 3 sessions and from there we will start, from that point onwards we will start discussion about the design patterns and those are the good solutions. We will try to reuse those in our design process.

(Refer Slide Time: 24:20)



We will first tell about what are the bad code or the bad design and then we will discuss about the principles that try to overcome this bad design. And once we are aware of the principles, we will try not to commit those problems. This part of the discussion is based on the work of Robert Martin and his book on object oriented design.

A bad code, the system is rigid and becomes very difficult to change if you change one thing you want to just change one feature but you find that the features are so interlinked that all features need to change. The system is fragile you changed one feature and other features have stopped working. To get some methods or objects from here and reuse another in development is very difficult because these are bound to many variables and specific situations that you cannot really take them out and reuse.

The system is hard to understand, the system is extremely complex and there are lot of repetitions which are hard to get rid of. These are example of a bad code and what are the principles we must have to avoid having such bad code? That is our discussion over the next few sessions, we are almost at the end of this session, we will continue in the next session, thank you.