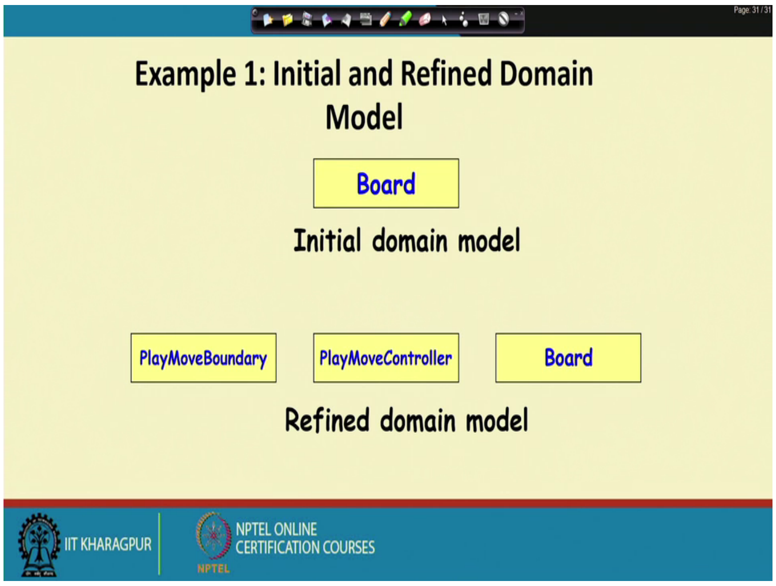
Software Engineering
Prof. Rajib Mall
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture – 42
Examples of object-oriented design

Welcome to this lecture. In the last lecture we were doing some practice initially we discussed about how to do the domain analysis? And, then we took some examples and we try to develop the domain model. We took the example of a Tic-Tac-Toe computer game. And, we identify the entity classes there and that was our initial domain model. And, then we had a refined domain model, where we added the boundary and the controller classes.

(Refer Slide Time: 00:52)

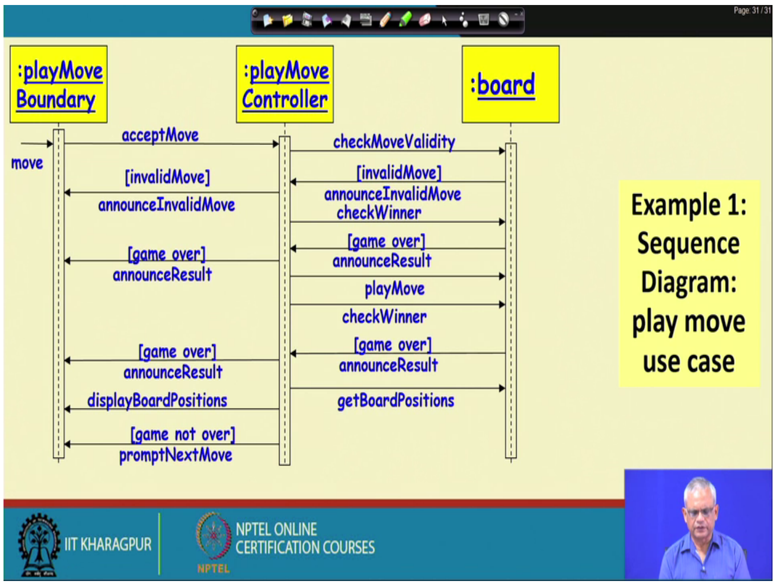


Example 1: Initial and Refined Domain Model

Now, let us proceed from there. So, this in the last class we had identified the board class from a noun analysis we did not really get down to do noun analysis, but mentally we rejected all those nouns which cannot be entity classes, but as we get more experienced we do not even have to do that you just read through the problem. And, we can identify what are the entity classes. And, once we identify the entity classes that, forms our initial domain model. The boundary and the controller classes are easily identified almost mechanically from the use case diagram and, then we add them here and that forms our refined domain model.

And, from here the next step would be to identify the sequence diagram. We know that we have to develop one sequence diagram for every use case. And, for this problem of Tic-Tac-Toe you have only one use case that is play move and therefore, we need to develop only one sequence diagram. In the sequence diagram, when the user starts a play how do this classes interact? And finally, produce the result we display that graphically in the sequence diagram.

(Refer Slide Time: 02:26)



And, for the Tic-Tac-Toe problem to develop the sequence diagram, we need to understand the business logic well read the problem many time and understand what really happens? And, then we can draw the sequence diagram easily let us see here, as the user makes a move that is detected in the play move boundary. So, these are the 3 classes which interact to realise the play move. Once the move in initiated by the user that is conveyed to the controller.

The controller is the one which actually correlates all other objects, the boundary is only for getting the data and reporting it to the controller or taking data from the controller and displaying to the user. So, the accept move as the move is a made, that is conveyed to the controller. The controller knows what to do knows that whether need to check the validity, whether the user has marked on a square which is already occupied or was it a ambiguous move.

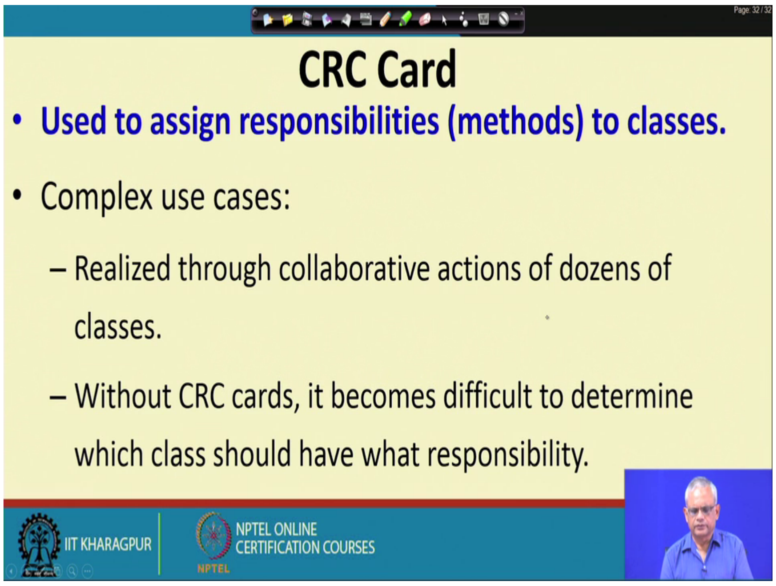
If, it is given graphically, and it checks the move validity, but how will it check? It needs to check through, what are the current marked squares on the board? And, that data is there with the board. So, it request the board to check the validity. If the board conveys invalid move, then it displays invalid move and prompts to give an you move and the use case would end here.

But, if it is a valid move then it is registered on the board and the controller knows that once the move is valid needs to check if there is a winner. And therefore, it invokes the check winner in the board. If, there is a result either the game is over or there is a winner that is game is drawn or the game is there is a winner it will be announced.

But, the controller again knows that if there is no winner or game is not drawn the computer has to play the next move. And, it request the board to make the next move because board class knows all the squares which are marked and which are available. And this is the best class that can play the move, request the play move and then once it has made the move it again checks for winner, if there is a result announces result. Otherwise, it gets the board position and displays the board position and if the game is not over prompts for the next move.

So, as you can see that the controller actually implements the business logic it coordinates the actions of other objects and then realizes the use case.

(Refer Slide Time: 06:29)



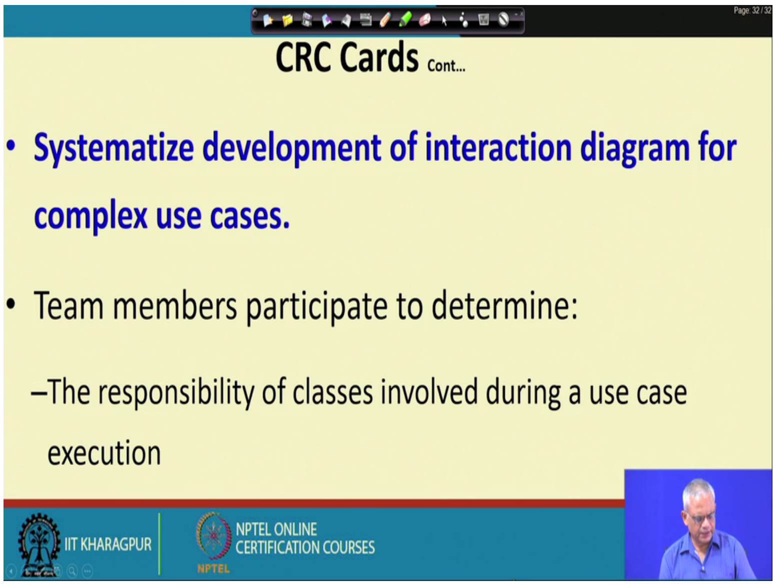
CRC Card

- **Used to assign responsibilities (methods) to classes.**
- Complex use cases:
  - Realized through collaborative actions of dozens of classes.
  - Without CRC cards, it becomes difficult to determine which class should have what responsibility.

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES

But, see that the sequence diagram can become complex.

(Refer Slide Time: 06:40)



CRC Cards Cont...

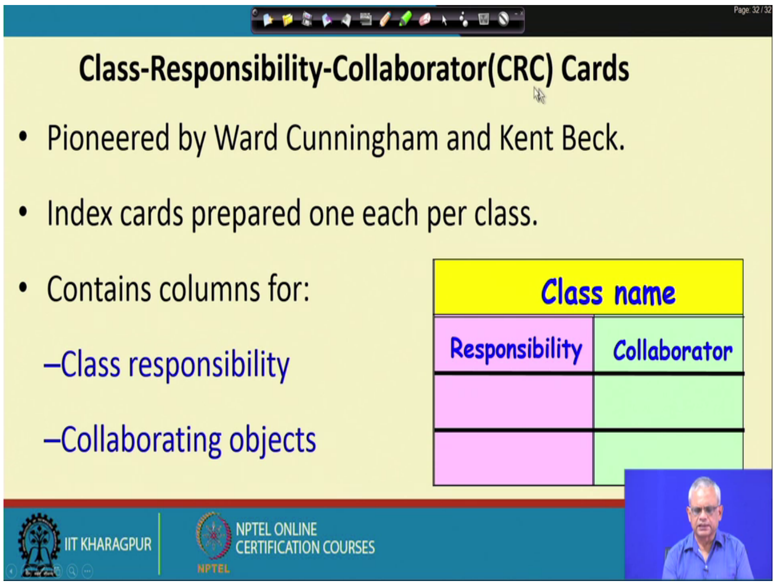
- **Systematize development of interaction diagram for complex use cases.**
- Team members participate to determine:
  - –The responsibility of classes involved during a use case execution

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

In our previous example we had only 3 classes, but then the message exchangers there are many exchangers, but imagine if we have the dozen classes. And, there are large numbers of message exchangers among them, it becomes very difficult for a person to keep this in mind and draw the sequence diagram. And, for that the CRC Card has been developed the CRC Card stands for class responsibility collaborator card.

It is used to assign responsibilities to classes, that is responsibilities are actually the method in the classes, that is which class should have which method. The CRC card simplifies the development of sequence diagram, because here by using the CRC card we can easily identify, which class would have, which responsibility.

(Refer Slide Time: 07:56)



Class-Responsibility-Collaborator(CRC) Cards

- Pioneered by Ward Cunningham and Kent Beck.
- Index cards prepared one each per class.
- Contains columns for:
  - –Class responsibility
  - –Collaborating objects

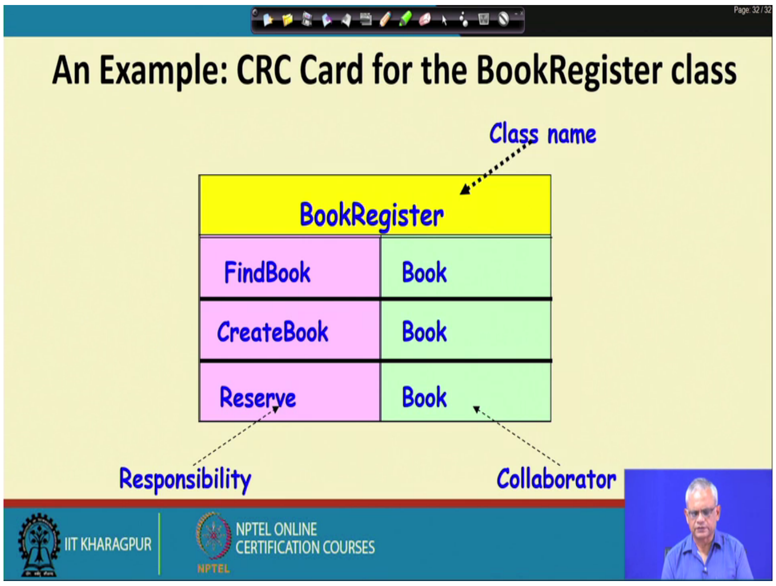
Class name	
Responsibility	Collaborator

And, then we can develop the sequence diagram. CRC card stands for class responsibility collaborator, the pioneered developed by Cunningham and Kent Beck. And, these cards are like playing cards small cards may be 4 inch by 6 inch.

And, here at the top of the card the class name is written and then there is a small table here. So, the classes are already identified during domain analysis. The entity classes boundary classes and controller classes. For every class identified during domain analysis a small card like this is made the name of the class is written here and, then on these rows we write the responsibility and collaborator.

The class responsibility is that, what exactly it should do during a use case execution? And then it can take help of which class, which what responsibility it will perform and for performing this responsibility, it will take help from use class. So, that is a collaborator. So, the class responsibility collaborator diagram, if we develop for every use case then we will have the responsibility here field, but we do it for one use case by another and then we draw the corresponding sequence diagram.

(Refer Slide Time: 09:59)



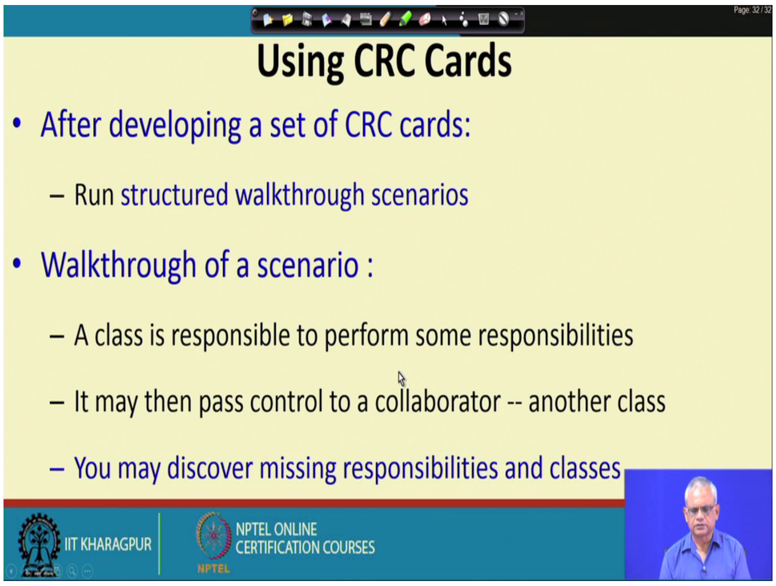
An Example: CRC Card for the BookRegister class

If, we think of it the CRC cards helps us systematize the development of the sequence diagram. For complex problems, for simple problems, just by reading it we can do it for example, for the Tic-Tac-Toe game we could do it without developing a CRC card, but for more complex problems we need to develop the CRC card. The responsibility here are the methods that will be supported and collaborator is the class whose services will be invoked by this class.

So, this is an example of a CRC card name of the class and then this is the method it needs to support find book. And to be able to find book it needs to invoke the book class method. It needs to support a create book and for creating a book needs to invoke the book class method.

Similarly, for reserve book it supports a reserve method and then as part of the reserve method, it invokes a method of the book class.

(Refer Slide Time: 11:09)

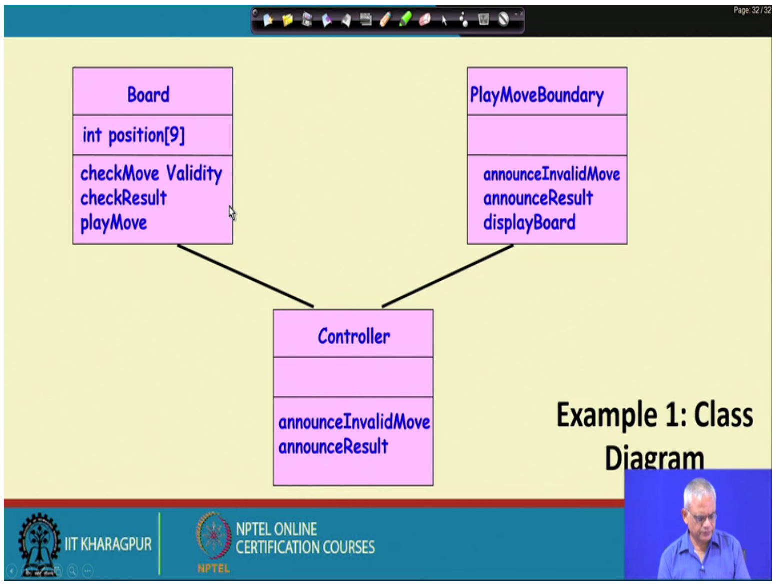


Using CRC Cards

- After developing a set of CRC cards:
  - Run structured walkthrough scenarios
- Walkthrough of a scenario :
  - A class is responsible to perform some responsibilities
  - It may then pass control to a collaborator -- another class
  - You may discover missing responsibilities and classes

But, how does we go about developing the CRC card? Once having the CRC card identify a use case and then determine roughly, how many classes would be required to collaborate for this use case? And then we get some team members may be 3 or 4 each team member is distributed some of the card 2 3 card. And, then the use case text description is read out. And, at the use case text description is read out the team members identify, if their class need to do anything.

(Refer Slide Time: 12:04)

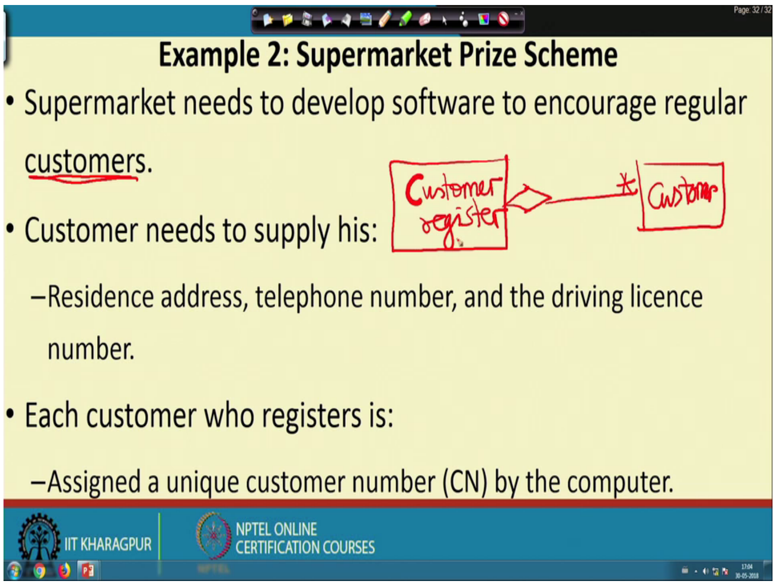


This is called as a structured walkthrough at the use case scenario is read out. Let say book to be issued to a member, then initially the member identifies the book. And, then the member's lists of issued books are checked. And, once it is read out that the members list of issued books have to be checked then the team member holding the member CRC card CRC card for the member class. We will say that for checking if the books, how many books have been issued to the member?

The member class can check it and then he writes the responsibility as check, how many books issued? Check number of books issued and so on as the text description is read out each of the team member identifies, if the class they are holding the CRC card for the class they are holding, whether it needs to do take some action and that will appear as the responsibility .

Now, once the sequence diagram is done the class diagram is automatically developed, if we were using a case 2. Otherwise, we need to check the sequence diagram identify all the messages that are send to this class. And, put them here these are the responsibility of this class. For every class, if we go through the sequence diagram we will see that these are the messages that are sent to this class. And, we just list them out here as the methods to be supported and based on the method to be supported, we write the attributes and that forms our class diagram.

(Refer Slide Time: 14:31)



Example 2: Supermarket Prize Scheme

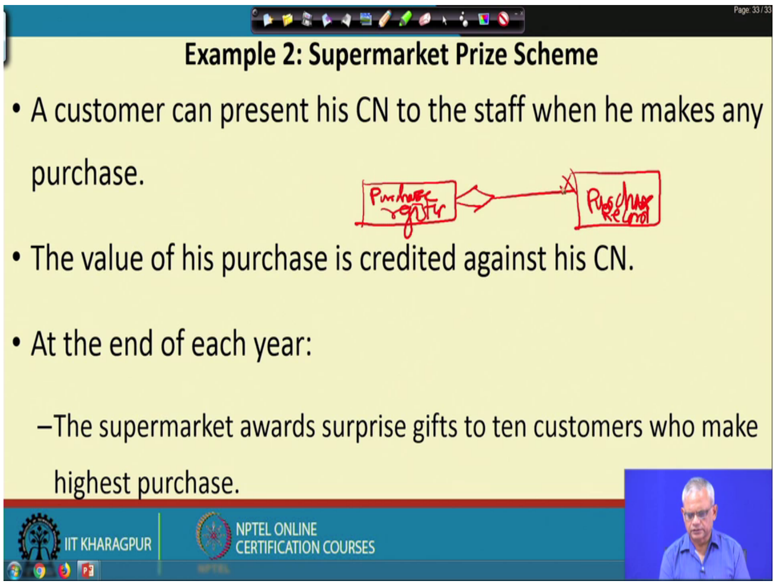
- Supermarket needs to develop software to encourage regular customers.
- Customer needs to supply his:
  - Residence address, telephone number, and the driving licence number.
- Each customer who registers is:
  - Assigned a unique customer number (CN) by the computer.

Now, let us take the next problem, which is the supermarket prize scheme. We had also developed the use case diagram for this problem, but just to refresh your memory I will just quickly read through that description of the problem. And, since the use case diagram was already developed, we need to develop the domain model. And, for that first we need to identify the entity class. And, once we identify the entity classes we can look at the use case diagram and easily identify the boundary and the controller classes.

Now, please keep in mind that you will have to identify the entity classes and with that perspective while I read the problem, identify what can be the entity class? Supermarket needs to develop a software to encourage regular customers. So, if you are looking for entity class, the customer is a noun and there are many customers and the supermarket needs to remember the residence address, telephone number, and driving licence of the customer. And from this description we can identify that the customer is a entity class.

And, we need to aggregate the customer using a customer register. The customer register class will aggregate the customer class. And, there will be many customer records that will be created and the customer record here itself you can find that the attributes are residence address telephone number and driving licence. You can write the attributes if we can and each customer is also assigned customer number. So, customer number also becomes attributes. From this part of the description we can identify customer with entity class.

(Refer Slide Time: 17:31)



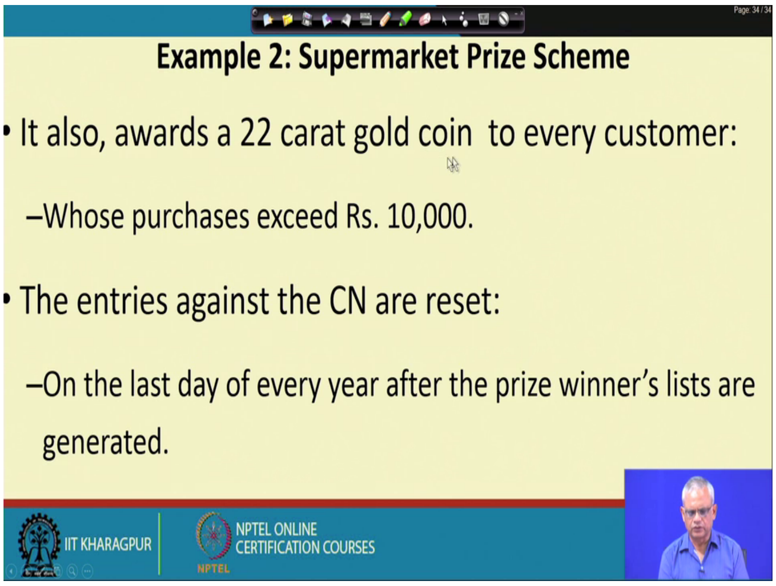
Example 2: Supermarket Prize Scheme

- A customer can present his CN to the staff when he makes any purchase.
- The value of his purchase is credited against his CN.
- At the end of each year:
  - The supermarket awards surprise gifts to ten customers who make highest purchase.

A customer can present the CN to the staff, when he makes any purchase. The value of his purchase is credited against his CN. So, all his purchase needs to be remembered. And, these will form the purchase record. And, at the end of the year the supermarket awards surprise gift to 10 customers, who make highest purchase? So, all the purchase has to be remembered, because at the end we need to look at the computer needs look at the purchase records and identify the customer who has made the highest purchase.

So, this and this description we will write purchase register aggregates the purchase records and so, 2 entity classes we have so far identified.

(Refer Slide Time: 18:50)

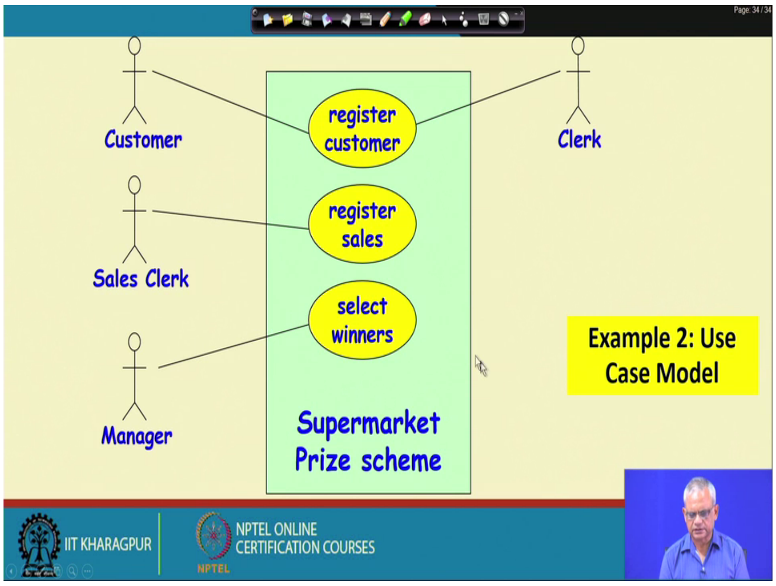


Example 2: Supermarket Prize Scheme

- It also, awards a 22 carat gold coin to every customer:
  - –Whose purchases exceed Rs. 10,000.
- The entries against the CN are reset:
  - –On the last day of every year after the prize winner's lists are generated.

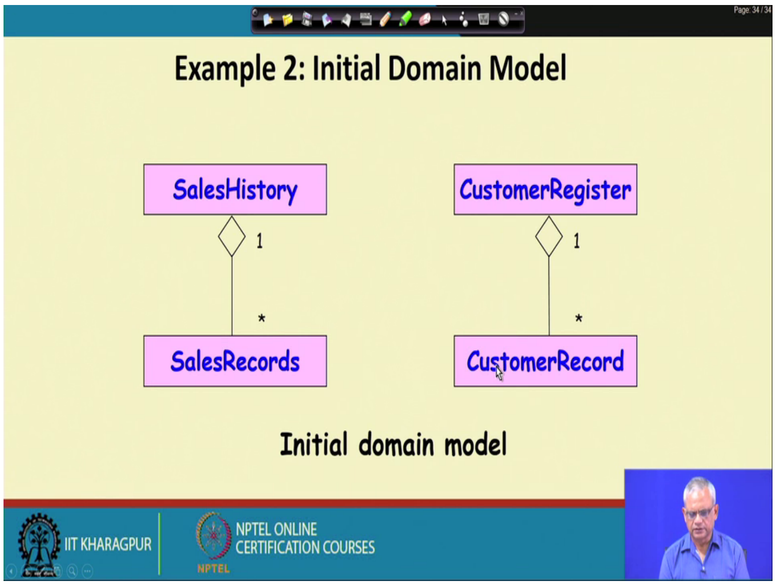
It also awards a 22 carat gold coin to every customer; whose purchase exceeds rupees 10,000. So, no entity class is here the entries against the CN are reset on the last day of every year after prize winners list are generated so, here also no entity classes. So, based on the noun analysis we had identified 2 entity classes.

(Refer Slide Time: 19:11)



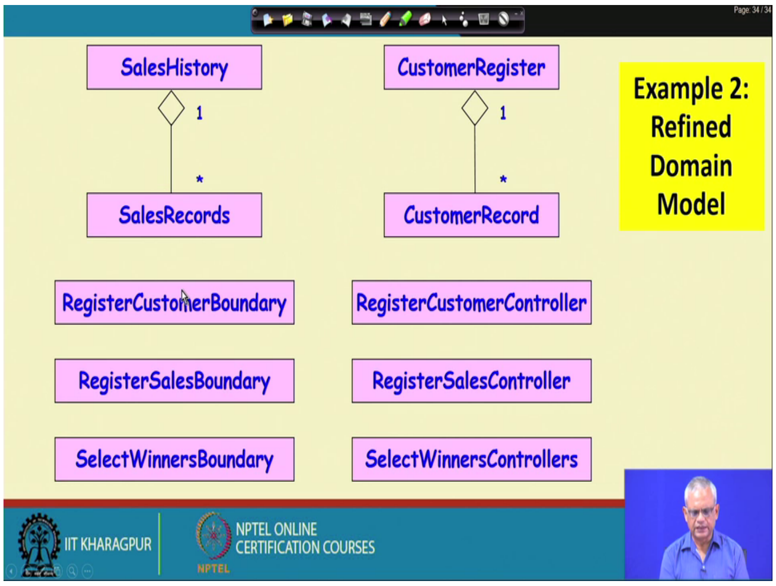
And, we can then look at the use case model. And identify the boundary and controller classes, 3 controller classes and 4 boundary classes.

(Refer Slide Time: 19:34)



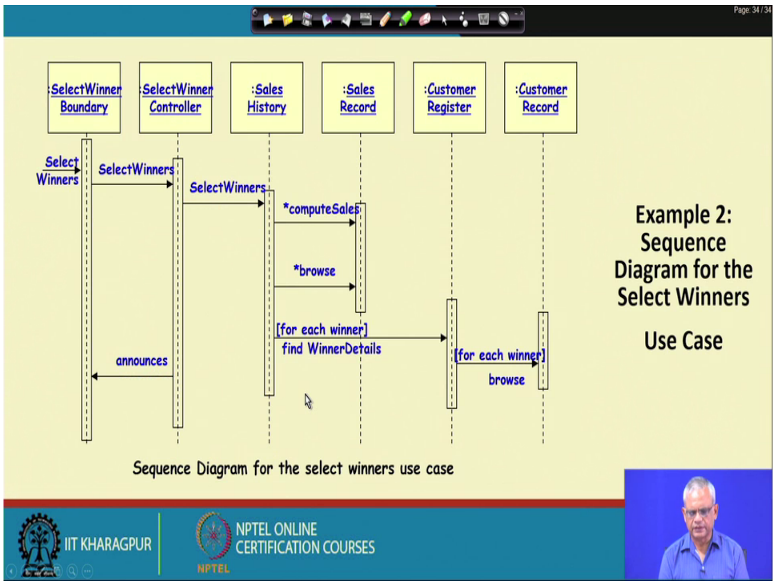
Now, this is our domain model, the customer register and customer record. We have rename the purchase record, because these are actually the sales by the company and the purchase is the from the customer perspective and these are internal data. So, we keep it as sales record and sales history. So, these are the 2 important entity classes.

(Refer Slide Time: 20:08)



And, then we can identify and add the controller classes and the boundary classes 3 controllers and 4 boundary classes.

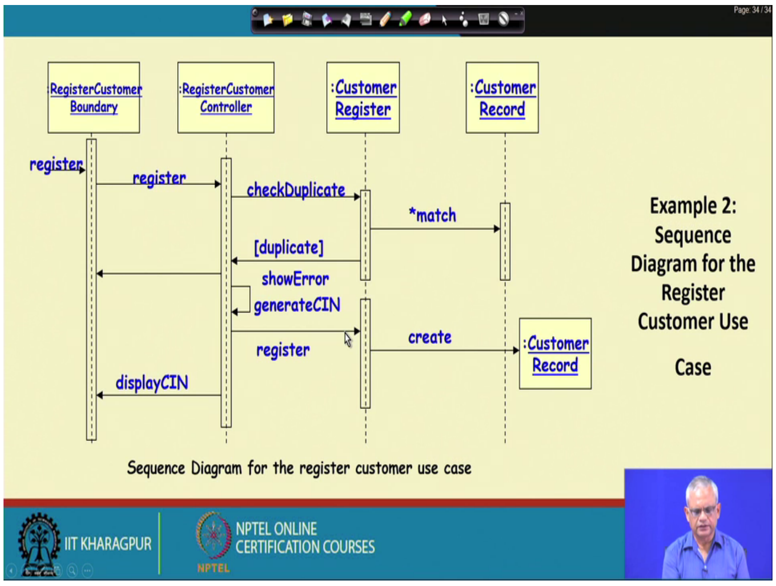
(Refer Slide Time: 20:18)



And, then next we develop the sequence diagram. First for the select winner this is possibly the simplest sequence diagram, we need to look at the sales history.

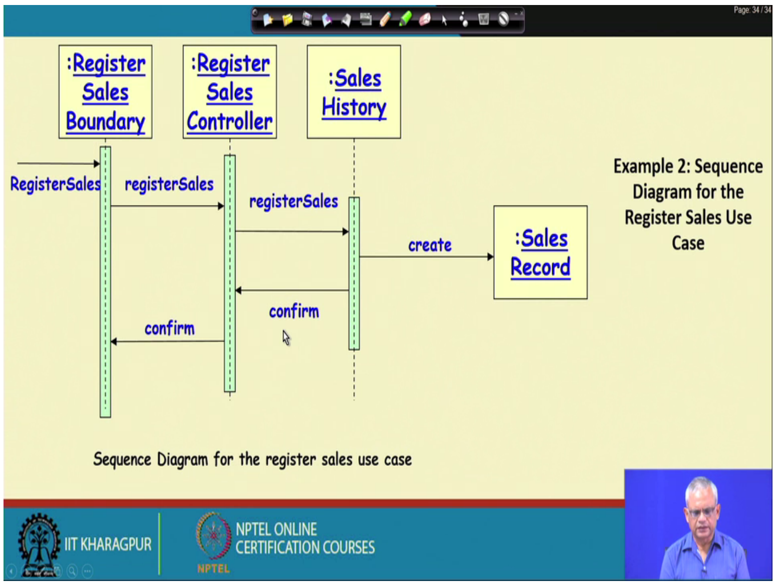
First the controller gets charge, the controller has the business logic, and it knows that the select winner can be obtained from the sales history. The sales history in terms computes the sales address sorry the total sales and then reports to the controller. The controller finds the top winner details ask it to finds the top winner details. And, then it for each winner detail identifies the address from the customer register. And, then the controller ask the boundary to announce the winners similarly, register customer.

(Refer Slide Time: 21:20)


RegisterCustomer Boundary
:RegisterCustomer Controller
:Customer Register
:Customer Record
register
register
checkDuplicate
*match
[duplicate]
showError
generateCIN
register
create
:Customer Record
displayCIN
Example 2: Sequence Diagram for the Register Customer Use Case
Sequence Diagram for the register customer use case
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES

Once the register customer boundary we have the register request coming given to the controller, the controller first checks if it is already registered. For that it request the customer register to match, if there is a duplicate, it requests, it declines to register shows error. Otherwise, it generates a CN and request the customer registers to create a customer record and the customer record is created here and then displays the CIN.

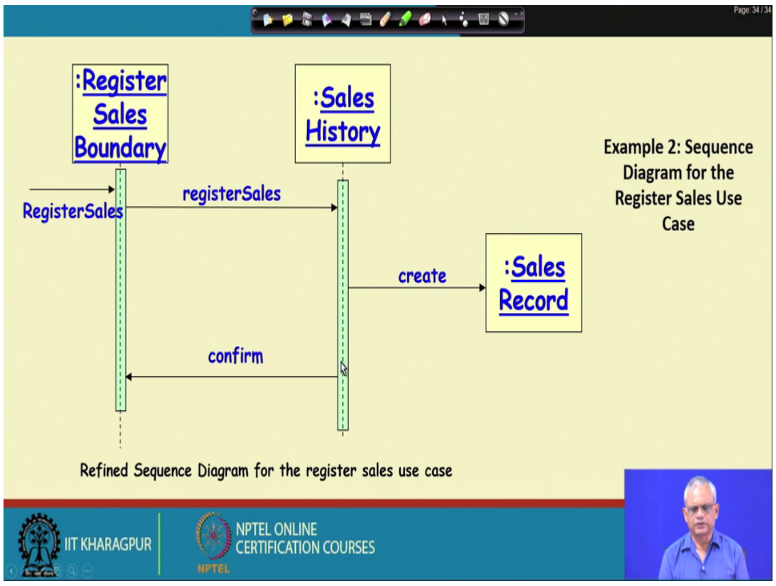
(Refer Slide Time: 22:03)


:Register Sales Boundary
:Register Sales Controller
:Sales History
RegisterSales
registerSales
registerSales
create
:Sales Record
confirm
confirm
Example 2: Sequence Diagram for the Register Sales Use Case
Sequence Diagram for the register sales use case
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES

Similarly, register sales is reported the controller, the controller request the sales history to create a sales record create and it is confirmed.

So, we have so, far looked at domain analysis. And, development of the sequence diagram. For simple examples, we will not spend more time on the object oriented design part we will.

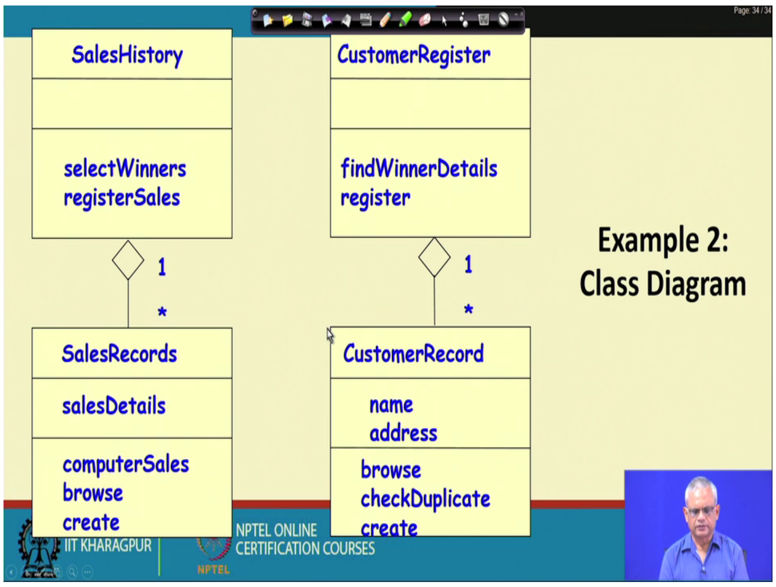
(Refer Slide Time: 22:40)


:Register Sales Boundary
:Sales History
RegisterSales
registerSales
create
:Sales Record
confirm
Example 2: Sequence Diagram for the Register Sales Use Case
Refined Sequence Diagram for the register sales use case
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES

Now, look at the testing part. Ok. This is also for the previous diagram, we have a small refinement here. The controller class actually has very little role here. It just gets the message and transmits it to sales history and then it just gets confirmation here and transmit.

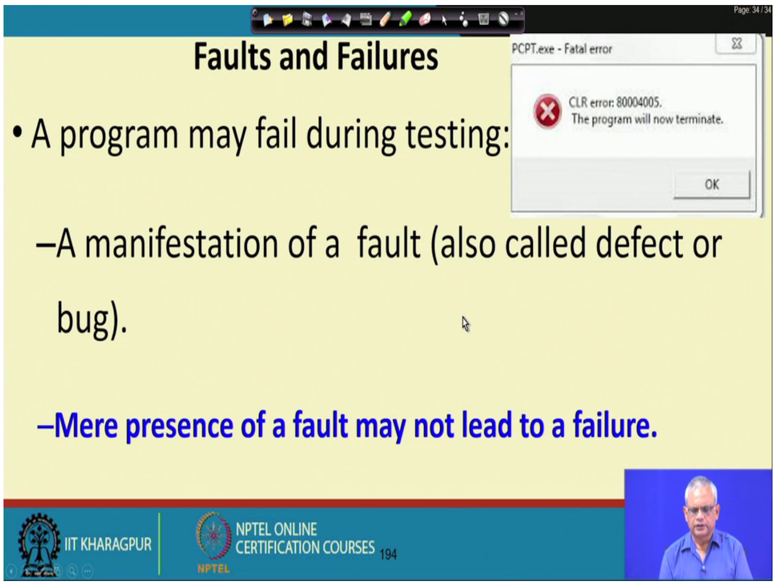
So, the business logic here is very simple. And therefore, we can eliminate the controller class and we can redraw the small business logic and be part of the sales history. So, it is a very trivial business logic. And, we have eliminated the controller class. Similarly, when the controller class has too much of business logic, we might split the controller into 2.

(Refer Slide Time: 23:31)



And, based on that we develop our full class diagram, where we write the methods based on and looking at the sequence diagram a case 2 will automatically do it. And, then we write any attribute that we identify.

(Refer Slide Time: 23:58)



Faults and Failures

- A program may fail during testing:
  - A manifestation of a fault (also called defect or bug).
  - Mere presence of a fault may not lead to a failure.

PCPT.exe - Fatal error

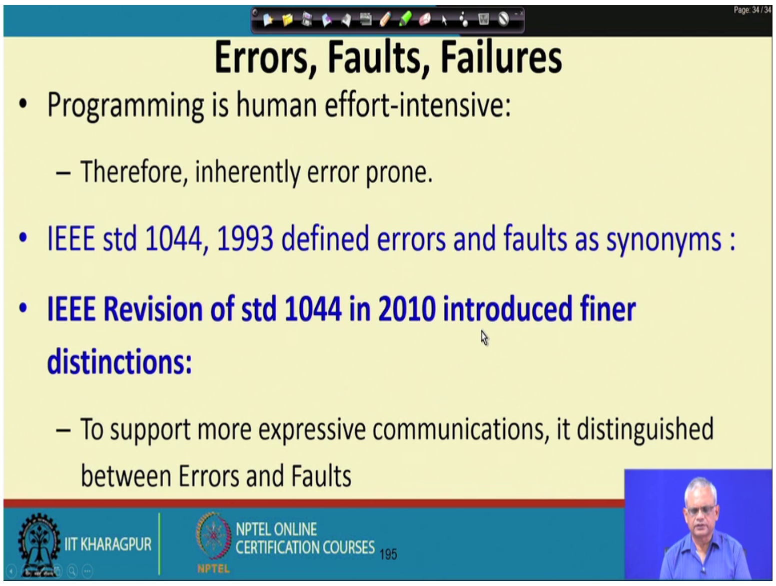
CLR error: 80004005.
The program will now terminate.

OK
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
194

Now, let us look at software testing. Once we write the program, while testing it may fail. We may see a failure a symptom of the failure it says that fatal error program will now terminate, but then we observe that the program has failed, but we do not see the bug, this is the symptom of the failure by testing, we identify the failure. If there is a failure, but we do not really see the bug or the error which had cause the failure, for that we need to do a debugging to identify the bug.

So, it is important to see here that when we run test cases, we see the failure; we do not see the bug. A failure is a manifestation of a bug or a fault, there was a bug in the code and while running that manifested itself in the form of a failure. We do not see the bug, but we see the manifestation of it, but then we must also remember that even if there may be a bug it may not lead to a failure. For example, the data that we give does not lead to the error to express it itself.

(Refer Slide Time: 25:53)



Errors, Faults, Failures

- Programming is human effort-intensive:
  - Therefore, inherently error prone.
- IEEE std 1044, 1993 defined errors and faults as synonyms :
- **IEEE Revision of std 1044 in 2010 introduced finer distinctions:**
  - To support more expressive communications, it distinguished between Errors and Faults

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
195

Let us now try to distinguish between error fault and failure. Programming is effort intensive in main manually write the code and therefore, inherently error prone. In 1993 the I EEE standard defined errors and faults as synonyms.

But, then a revision of that in 2010 introduced the distinction. That errors and faults there is a difference actually errors are the mistakes committed by the programmer and faults are what happens due to that mistake? So, there is a bug or a defect. So, error and mistake and synonyms and faults bugs and defects are synonyms, we are almost at the end of the time here. We will stop at this point and continue on this topic in the next lecture.

Thank you.