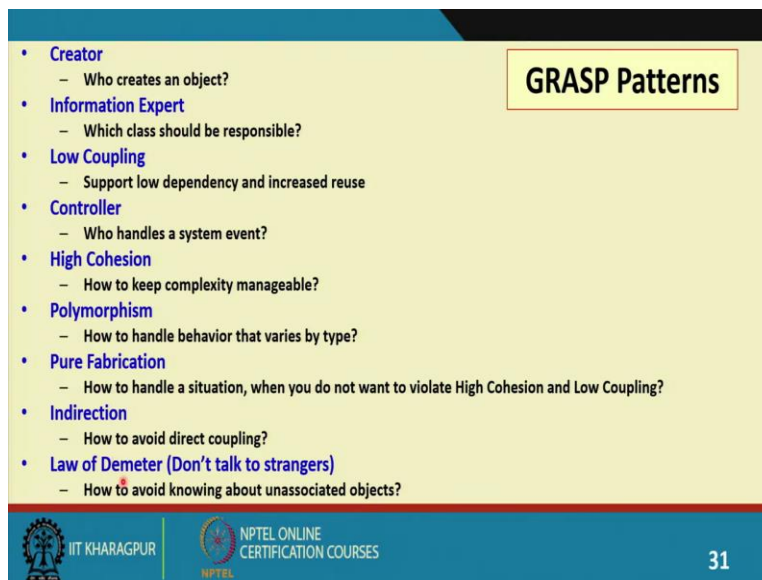


**Object-Oriented System Development using UML, JAVA and Patterns**  
**Professor Rajib Mall**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology Kharagpur**  
**Lecture 39**  
**Expert and Creator Patterns**

Welcome to this session! Over the last couple of sessions, we were discussing design patterns. The design patterns are essentially reusable solutions. If we know the patterns in our design, we can reuse this, not a plug and play kind of reuse. But then we can adapt a pattern solution to our design. There are many patterns. But we will start with the simplest pattern and we said that the GRASP patterns are the simplest.

These patterns deal with assigning methods to classes. And many of these patterns are intuitive, very simple. Some patterns are almost common sense, but then these patterns address very common problems and therefore, if we learn these patterns, we will be able to use these patterns in almost every design solution.

(Refer Slide Time: 1:41)



The slide, titled "GRASP Patterns", lists nine design patterns with their key questions:

- **Creator**
  - Who creates an object?
- **Information Expert**
  - Which class should be responsible?
- **Low Coupling**
  - Support low dependency and increased reuse
- **Controller**
  - Who handles a system event?
- **High Cohesion**
  - How to keep complexity manageable?
- **Polymorphism**
  - How to handle behavior that varies by type?
- **Pure Fabrication**
  - How to handle a situation, when you do not want to violate High Cohesion and Low Coupling?
- **Indirection**
  - How to avoid direct coupling?
- **Law of Demeter (Don't talk to strangers)**
  - How to avoid knowing about unassociated objects?

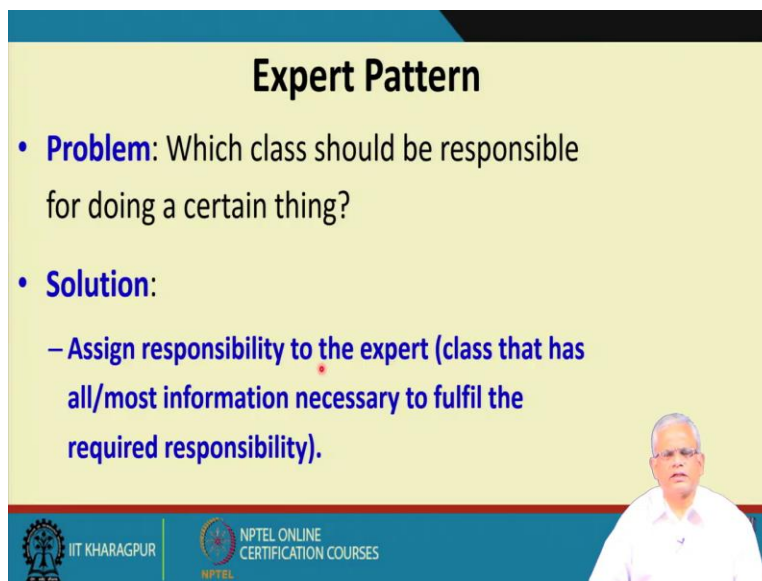
The slide footer includes the logos of IIT Kharagpur and NPTEL Online Certification Courses, along with the page number 31.

There are 9 of these patterns, we will start discussing with the creator pattern. The creator pattern have several classes and object is to be created, then which object would have the responsibility to create the object? The information expert pattern or simply the expert pattern addresses when we need to assign a responsibility or a method to add a class to which class we should assign.

And then we have the low coupling pattern, which is how to enhance low coupling, low dependency among classes. Similarly, high cohesion, how do we have high cohesion among the classes and keep the complexity manageable. The polymorphism pattern, how do we handle the behavior when there is an inheritance hierarchy and we do not want to check the type and then call the method, we should use polymorphism.

Pure fabrication, this is about when a class has unrelated responsibilities in a design, we inspect and find unreasonable set of responsibilities on cohesive class. And then we use pure fabrication to delegate some of the responsibilities which is contributing to making it a bad design. In the fabrication step we create a new class and delegate responsibility to the new class. Indirection, this is to avoid direct coupling between classes, unrelated classes, we will see this pattern and then the Law of Demeter to restrict the method invocation to only related classes.

(Refer Slide Time: 3:54)



**Expert Pattern**

- **Problem:** Which class should be responsible for doing a certain thing?
- **Solution:**
  - Assign responsibility to the expert (class that has all/most information necessary to fulfil the required responsibility).

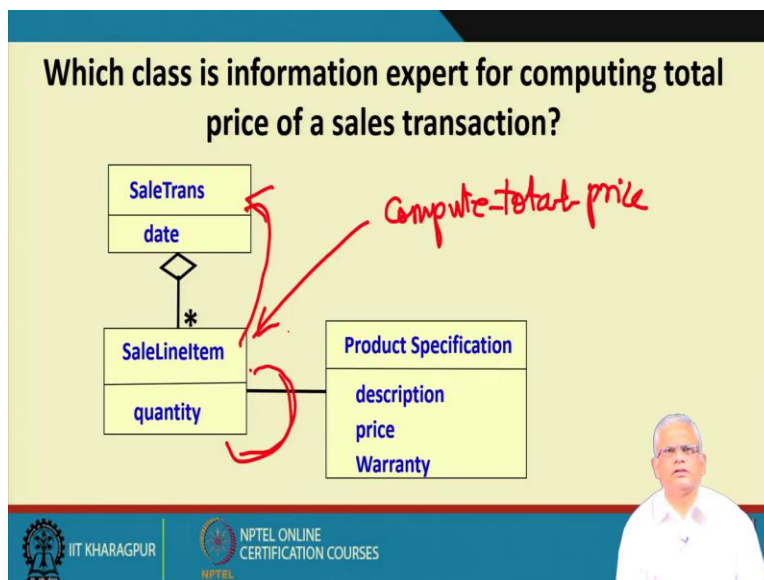
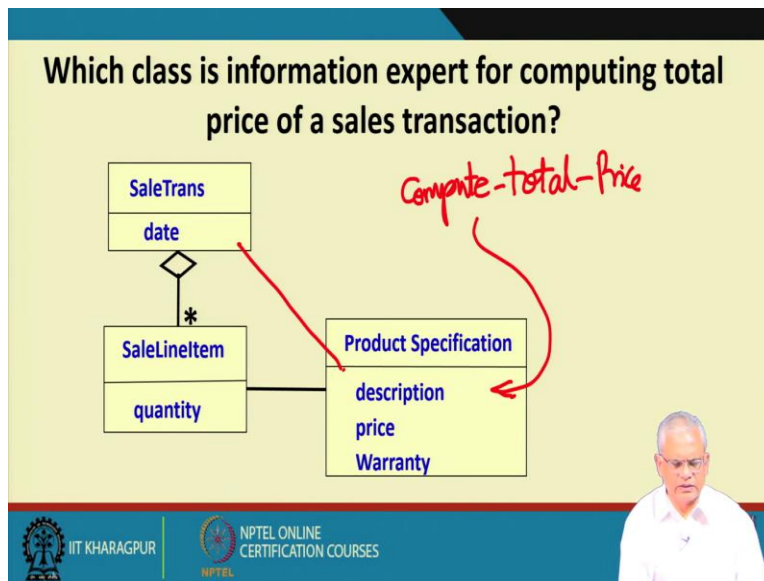
The slide features a yellow background with a blue header and footer. The footer contains the logos for IIT Kharagpur and NPTEL Online Certification Courses, along with a small portrait of a man in a white shirt.

Now, let us start with the expert pattern. One of the simplest pattern but widely used. The problem that pattern addresses is that which class should be responsible for doing certain thing, i.e., we have something to be done maybe while doing a huge case analysis, we find that during use case execution, certain things need to be done. But which classes exactly will be able to do these things, we cannot just arbitrarily assign the responsibilities, different classes, and this pattern addresses to which class, the responsibility or the method be assigned. The main idea of

this pattern or the main solution offered by this pattern is that assign responsibility to the information expert to do certain things you need, certain information to be able to do that.

This pattern says that, check the different classes and find the class which has the maximum set of information needed to carry out this responsibility. A class might have all the information necessary to carry out the required responsibility, then we definitely assign to that class. But even if it has most of the information, but only a few information is there with other classes. Still, we assign this to the one with the highest set of information. We will see the implication of this pattern, it reduces coupling between classes, make the classes cohesive and so on.

(Refer Slide Time: 6:09)



Now, let us try to explain this with the help of an example. Let us say we have this class diagram aright from a domain analysis as shown in the figure, let us say we have a sales transaction class, which aggregates a set of sale line item. These are the specific orders in the sale transaction, sale transaction may consist of several items being purchased and each sale line item addresses one of the product. And the sale line item is associated to the product specification.

Because given the quantity, the sale line item does not contain the price, the warranty information, description of the product and so on are there for a product specification is a separate class here. And let us say a use case requires us to calculate the total price of a sales transaction. Now, we need a method called as compute total price. We find that we need to have this method compute total price for the sale transaction.

Now, should we assign the compute total price to the product specification or to the sale line item or to the sale connection. Let us just say that we assign this responsibility compute total price to product specification. Then the method will work by first finding out that for this product how many quantities have been ordered.

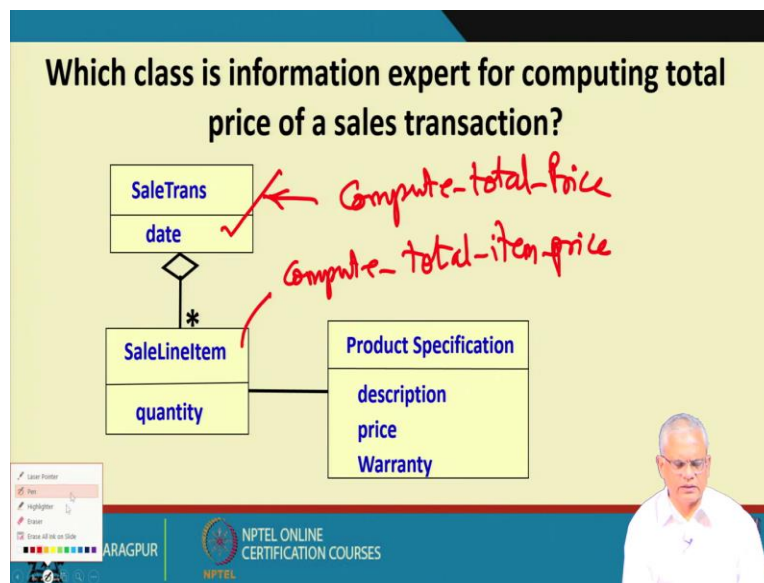
To use the association here and find out how many orders and then it will compute for that specific line item. But also it is needs to find out all other items and therefore, it would have to communicate to the sale transaction. Find out what exactly are the other line items and then the sale transaction, from sale transaction it again has to communicate with the corresponding sale line item objects and then compute the total price and then return.

And there are too many new association relations are necessary links basically, a sale line item has a link to one product specification. But in this situation compute total price for this object, one product specification object, we will not only have the link to the corresponding sale line item, but also we need a link to the sale transaction object and also all other sale line item objects, definitely a bad solution. But what if we assign the responsibility to sale line item compute total price.

Now, for that specific line item, it would have the association with the specific product and get the price for that multiply with quantity and get the price, the total cost for that line item, but what about the other line items? For that, it would need an association with sales transaction linked to the sale transaction object and then request what are the other line items.

And then it would need a link to the other sale line items, that is a self-association to the other objects have the sale line item and then request what is the item purchased and also it will need link to the other product specification objects also extremely complicated. Too much of extra coupling between different objects is introduced which is not a good solution.

(Refer Slide Time: 11:46)

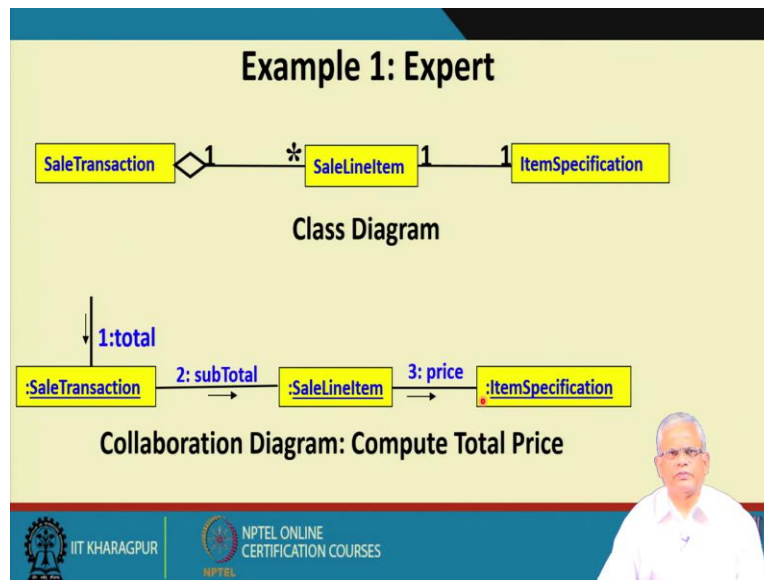


But what about we assign the compute total price to the sale transaction. Now the sale transaction has all the information regarding what are the sale line item, that is the implication of this aggregation relation that the sale transaction object aggregates all the sale line item for that sale transaction. Now, it can request the sale line item to compute price for sale line item, it will assign the responsibility or invoke the responsibility compute total item price. And to compute the total item price, the sale line item object has the quantity but does not have the price information. So it has already an association with the product specification. We will get the price information, compute the total item price and then return that to the sale transaction and the sale transaction can aggregate all the compute total item price returned by various sales line item objects.

And there is no extra coupling introduced, no extra link between the objects required just for completing the compute total price because sale transaction is the object having almost all the necessary information. And therefore, the sale transaction is the right class to assign the responsibility compute total price. So far, we have been doing this in our designs unconsciously,

informally, but this pattern, the expert pattern systematizes the process of assigning responsibility to a class.

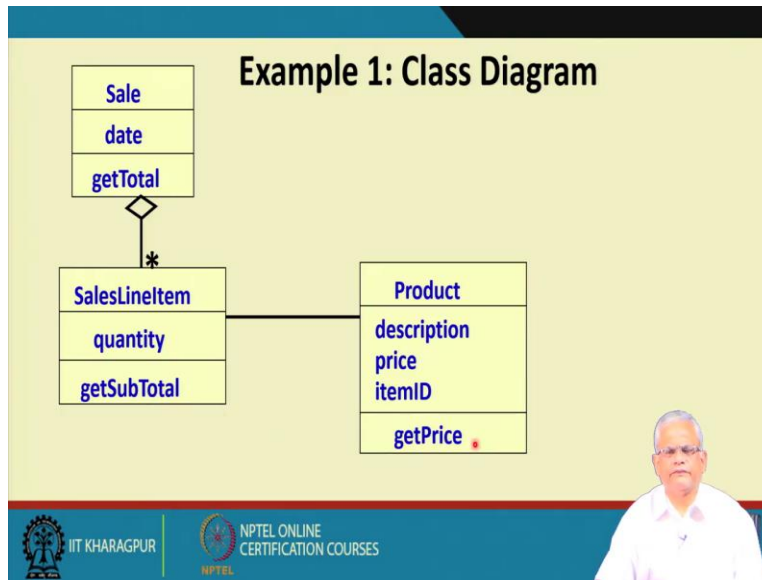
(Refer Slide Time: 14:12)



And once we assign the responsibility, we can draw the corresponding class diagram and there will have the sale transaction consists of many sale item and each sale item is specified by an item specification. And once we have the responsibilities assigned, we can draw the collaboration or sequence diagram.

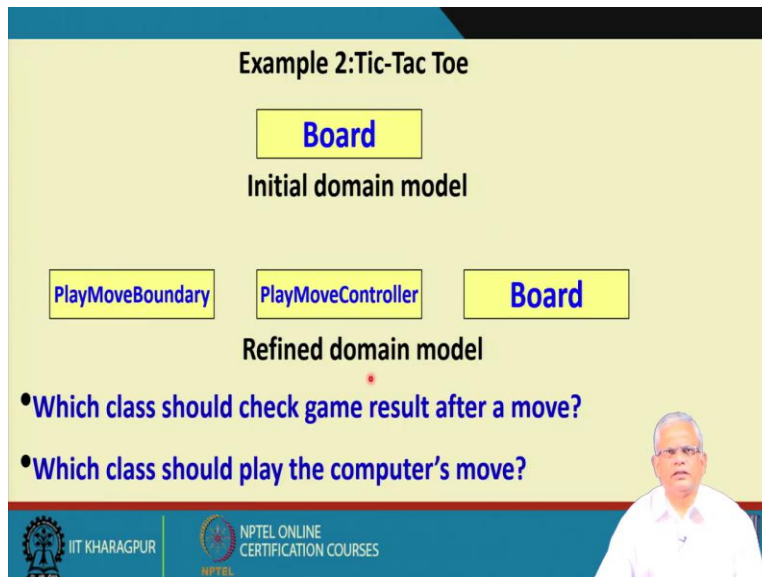
The total is the responsibility of the sales transaction and that in turn, for every sale line item that it aggregates will invoke the method subtotal. And then each sale line item object, it will invoke the corresponding get price method of the item specification. And this is the solution recommended by the expert pattern, very intuitive but very useful.

(Refer Slide Time: 15:10)



And once we do the assignment of the responsibilities, we will have this class diagram with the responsibilities assigned for based on this used case, the get total is the responsibility of the sale transaction, the sale line item has the responsibility- get subtotal and the product has the responsibility- get price.

(Refer Slide Time: 15:37)



We have also used this pattern while we designed the problem Tic-Tac Toe. There unconsciously we had used the expert pattern, but now we are systematizing which class should have some responsibility. Let us revisit the Tic-Tac Toe problem, remember that initially we had board as

the initial domain model and then we refined it, we added the boundary and controller classes based on the use case diagram for the Tic-Tac Toe. And then we were going through each use case and trying to assign the responsibility to different objects here.

Now, let us look at some specific responsibility, for example, which class should check the result of the game after a move, because after every move as per the use case description, we need to check whether the game is done or the game is won by the player or the computer. Now, is it the play move boundary? No, the play move boundary cannot have the check game result because it does not have the information about what is the current game position.

To check whether the game is done, we need to find which player that is the computer or the human player have marked which square. What about the play move controller, the controller also does not have the necessary information? If we have this solution that it will compute the check game result, then it would need to get all the information from the board.

And that will lead to high coupling because coupling is not only a method invocation, but how much information flows between two objects, it has to get all the information from the board because board has the information about which player is occupying which square. And what about the board? Yes, what is the right class, it has all the information about which player has marked which square and based on that we can have a simple algorithm to implement the method that will check whether the game is done.

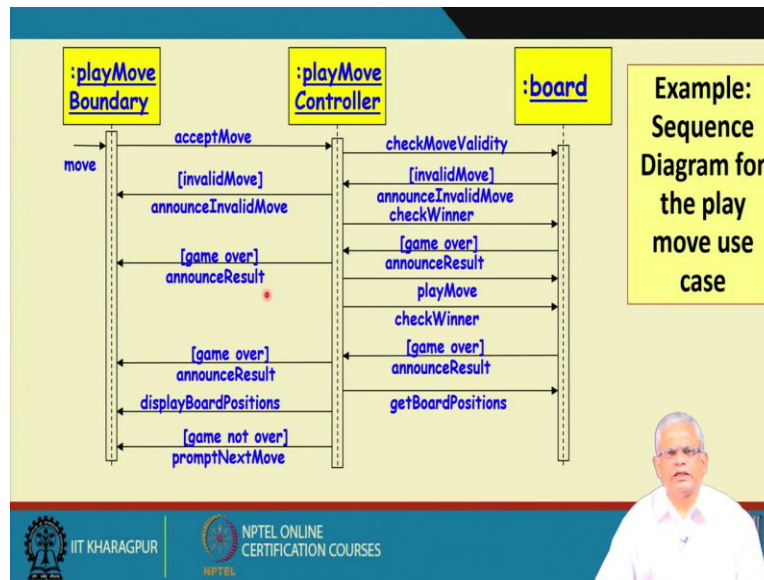
Now, another responsibility is that once the human has made a move, and the check game result shows that neither has won and the game is also not done. And then a computer move has to be done. And this responsibility of making the computer move, should it be there with the play move boundary class? No. It does not have the information because to make a next move we need to check which are the empty squares available and what are the squares occupied by each side.

What about play move controller? No, that is also not a good solution, because it does not have the necessary information about the board positions. And if at all we assign the make next move to the play move controller then it has to get all the information about the board from the board class and then have the algorithm play next move and it will unnecessarily increase the coupling severely.



Now what about the board. Can it have the play computer move? Yes, the board has all the necessary information, which player has marked which square and therefore, it has all the information necessary to have the method play computer move completed and then have the return, the result returned to the controller and then the controller can display it through the play move boundary.

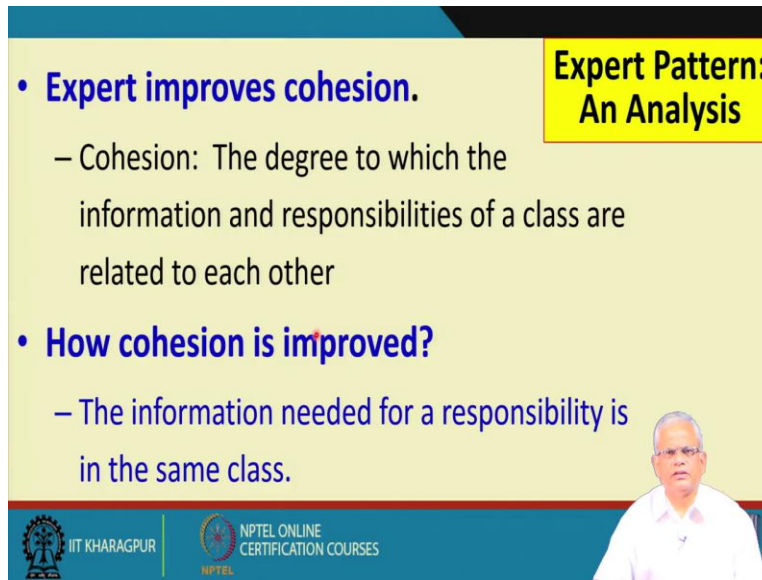
(Refer Slide Time: 20:37)



Now, based on this simple intuition of the expert pattern, we can assign the two responsibilities, that is to check result and to play the next move. To the board class check winner is the responsibility of the board class, we had drawn it earlier. Intuitively, but now, the expert pattern formalizes that, many novices do the problem, do the mistake actually of assigning method to an class which does not have the necessary information.

And similarly, the play computer move is, the play computer move is also a responsibility of the board class, we had drawn this solution earlier, but that was intuitively but a novice can make the mistake of having the play move assigned to the controller or to the boundary and that will be a bad solution.

(Refer Slide Time: 21:54)



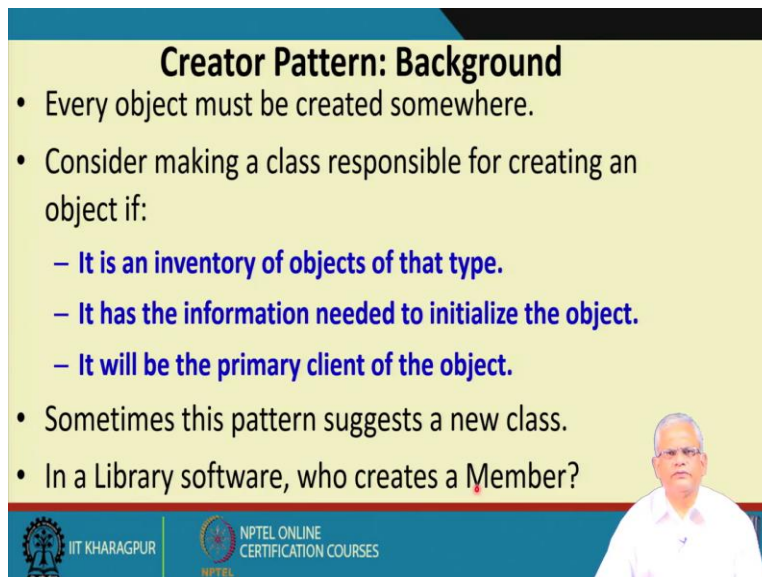
**Expert Pattern: An Analysis**

- **Expert improves cohesion.**
  - Cohesion: The degree to which the information and responsibilities of a class are related to each other
- **How cohesion is improved?**
  - The information needed for a responsibility is in the same class.

IT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

The expert pattern is extremely useful pattern usable in almost every design improves cohesion of the classes. Please remember that cohesion is the degree to which information and responsibilities of a class are related to each other, a class does not do something it should not do. And if it needs to do that, it needs to get the necessary information and increase the coupling and reduce the quality of the designer. The cohesion is improved because the information needed for responsibilities in the same class and does not have to depend on other classes.

(Refer Slide Time: 22:47)



**Creator Pattern: Background**

- Every object must be created somewhere.
- Consider making a class responsible for creating an object if:
  - It is an inventory of objects of that type.
  - It has the information needed to initialize the object.
  - It will be the primary client of the object.
- Sometimes this pattern suggests a new class.
- In a Library software, who creates a Member?

IT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, let us look at the next pattern, which is the creator. The problem that the creator pattern addresses is that every object must be created somewhere. In every program, large number of objects are created but each object creation is the responsibility of which class. Now if we try to solve this problem, we might sometimes assign the object creation very wrongly and that may degrade the design solution.

But this pattern, the creator pattern recommends that we need to assign the responsibility of object creation to a class which aggregates objects of that type. For example, let us say the sale transaction aggregates the sale line item, then creating the sale line item is the responsibility of the sale transaction class. If aggregation is not involved, we had remarked that aggregation is involved for almost every entity object.


But there will be some objects for which there is no aggregator. In that case, the responsibility to create that object will be assigned to the class which need to provide the initialization information for that object. If initialization information is also not required, then the creation of the object responsibility will primarily be with the client of the object, the one that invokes services of this object maximum. And sometimes, of course, this pattern suggests a new class, which will assign the responsibility.



Now, let us try to use this pattern, because if we, any pattern if we use few times, we will have the insight to use the pattern solution in our next solution. In the library software, which pattern which class will create the member object. Now, let us try to apply these rules, the member object is aggregated by the member register. And therefore, the responsibility of creating the member object should be with the member register object.

(Refer Slide Time: 25:50)

### Creator Pattern

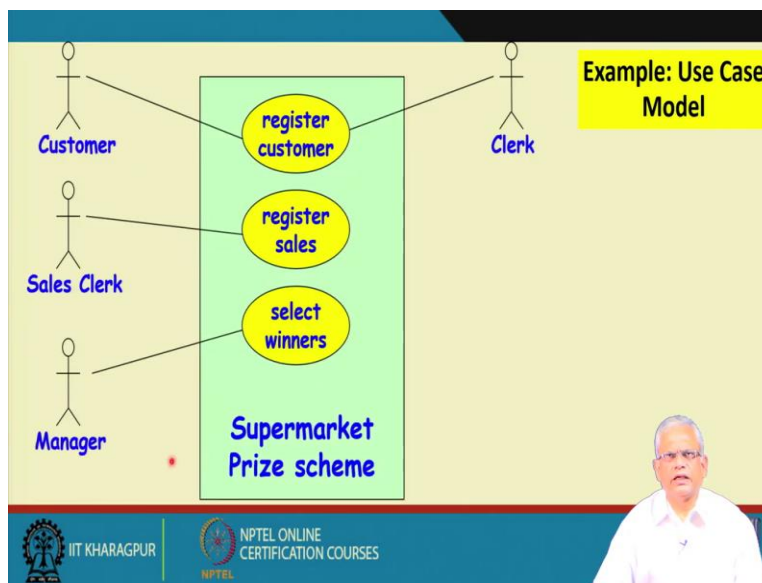
- **Problem:** Which class should be responsible for creating a new instance of some class?
- **Solution:** Assign a class C1 the responsibility to create class C2 object if
  - C1 aggregates objects of type C2
  - C1 contains object of type C2
  - C1 closely uses C2
  - C1 has the initializing data for C2



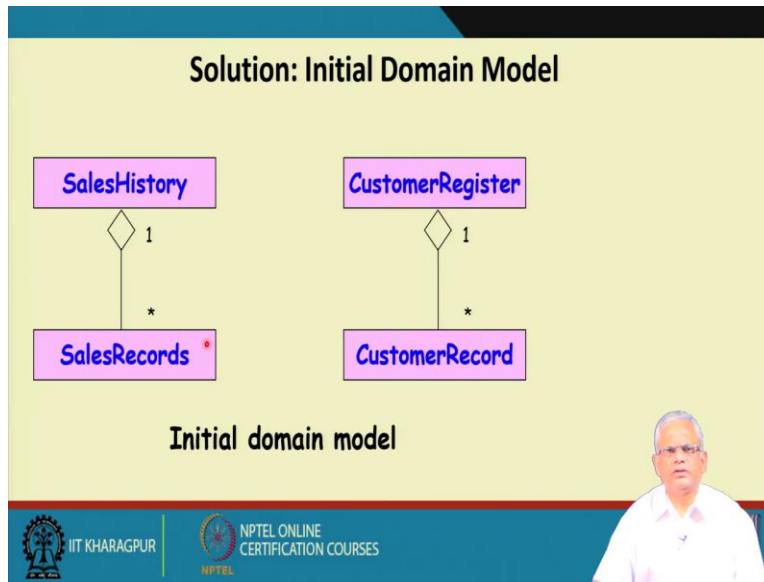
Now, we can document the pattern in the typical documentation of a pattern. The problem addressed is that which class should be responsible for creating a new instance of some class. We assign a class C1 the responsibility of creating an instance of class C2, if C1 aggregates objects of type C2, C1 contains an object of type C2 this is a composition relation. The first one is aggregation relation it closely you just C2 or it has data required for initializing object of C2.

(Refer Slide Time: 26:47)



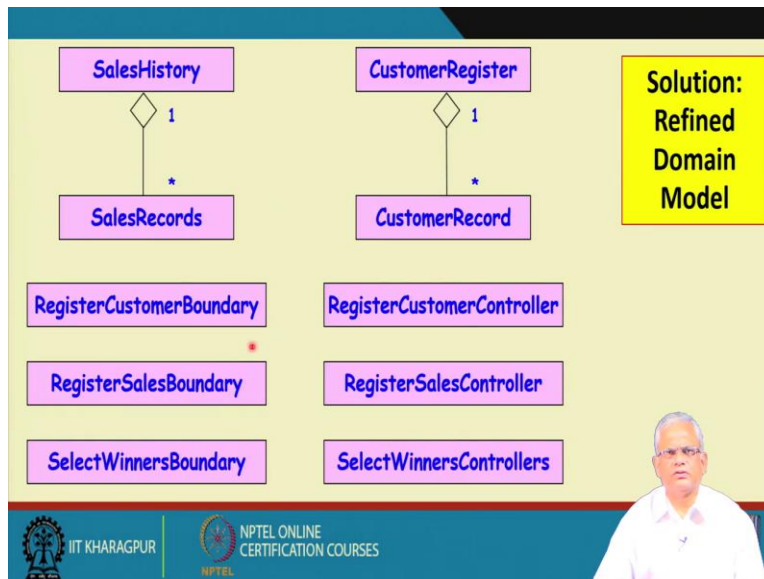
We had used this pattern while we try to solve the problem for supermarket prize scheme.

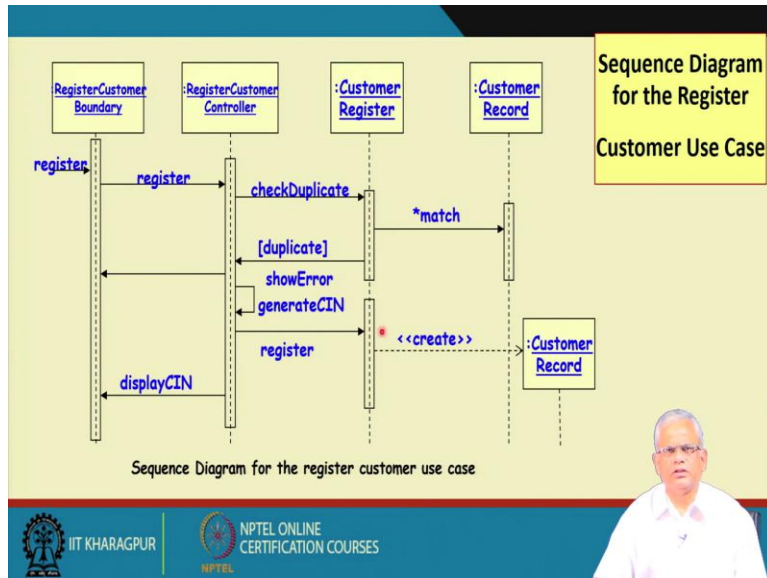
(Refer Slide Time: 26:56)



The creator pattern, remember that we had several sales record objects created and several customer record objects created. And in each case, we had used the corresponding aggregator is creating an object. That was based on our intuition, but this pattern formalizes that.

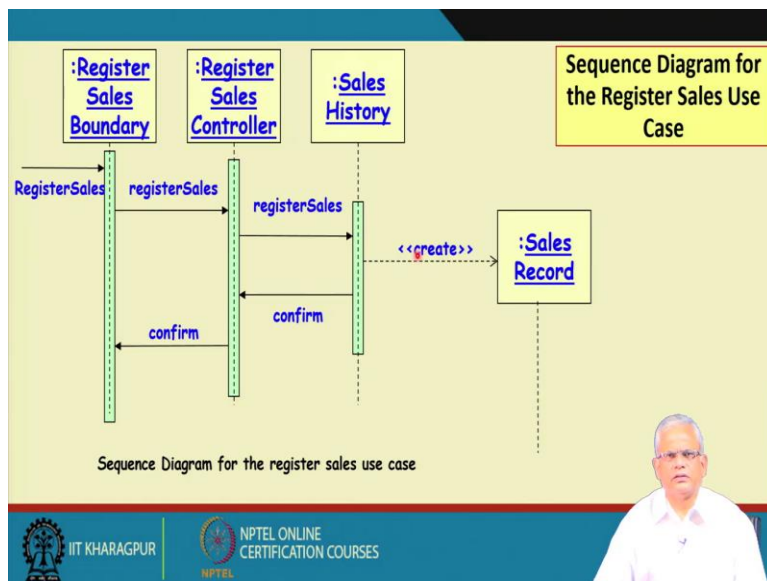
(Refer Slide Time: 27:17)





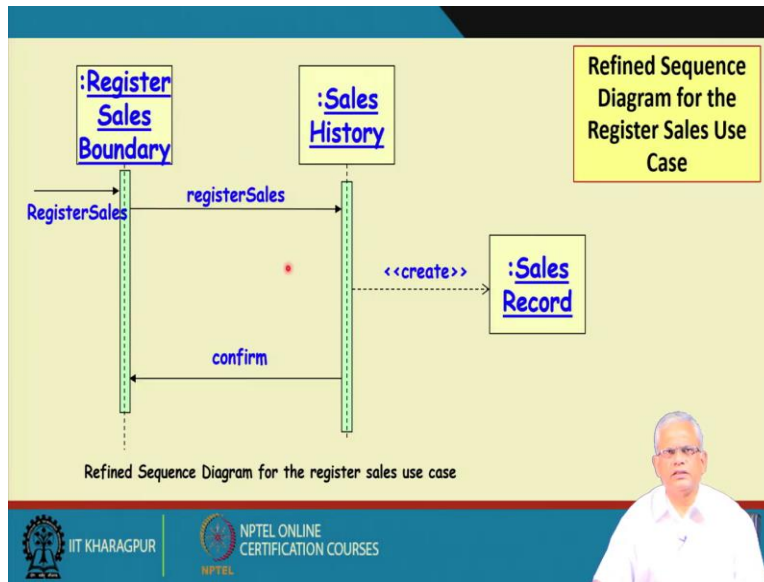
We had refined our solution and then look at the solution that we had developed, the customer record creation is assigned to the customer register, the customer register calls the create and the customer record, that is the responsibility of the customer register to create the customer record.

(Refer Slide Time: 27:44)



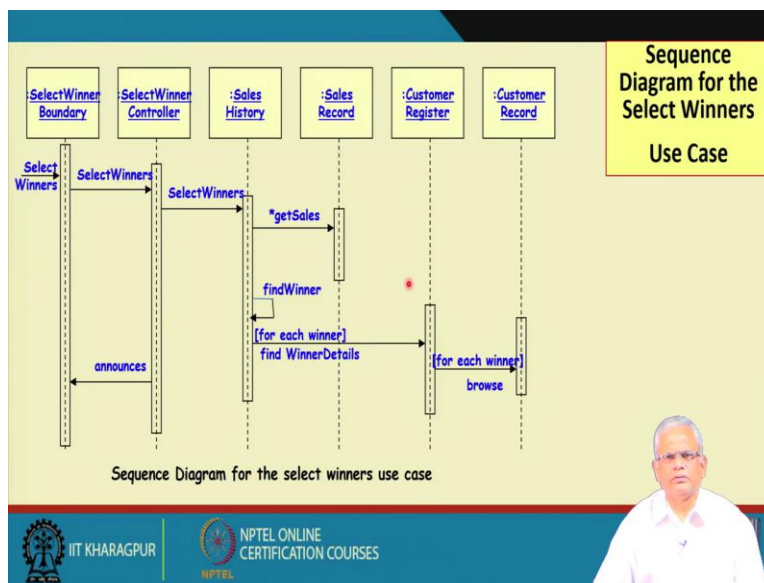
Similarly, the sales history had the responsibility to create the sales record.

(Refer Slide Time: 27:55)



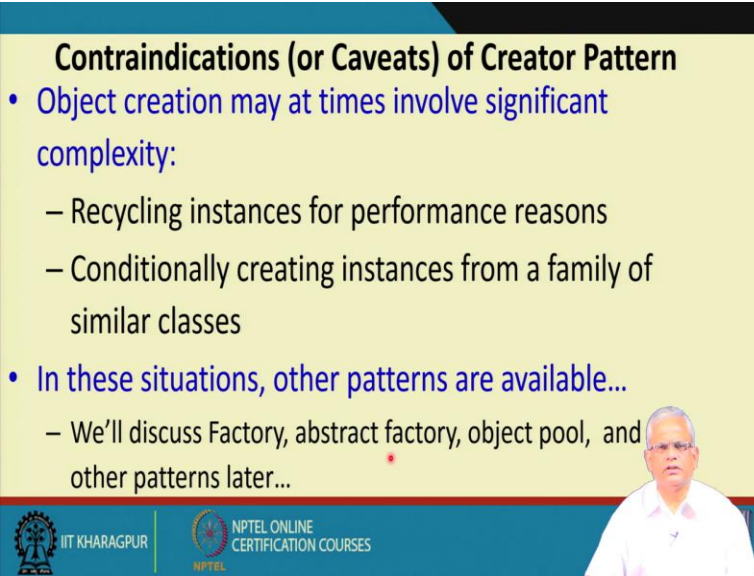
Of course, we had this, later we had simplified it into removed the controller class, and then we had the sales history creating the sales record.

(Refer Slide Time: 28:07)



And we had also used, we had finally got the solution for the select winner, we did not have any object creation and this.

(Refer Slide Time: 28:19)



**Contraindications (or Caveats) of Creator Pattern**

- Object creation may at times involve significant complexity:
  - Recycling instances for performance reasons
  - Conditionally creating instances from a family of similar classes
- In these situations, other patterns are available...
  - We'll discuss Factory, abstract factory, object pool, and other patterns later...

The slide features a small video inset of a man in a white shirt and glasses in the bottom right corner. The footer contains the logos for IIT KHARAGPUR and NPTEL ONLINE CERTIFICATION COURSES.

But then, using this pattern, we need to be slightly careful, in some situations the pattern is not suitable. For example, for efficiency we need to recycle instances of objects, then the creator pattern is not suitable. We will look at pattern like object pool later, which will be more suitable when reuse of objects is required, that is, we create an object but we just do not delete it, next creation we reuse it.

Conditionally creating a family of similar, objects of similar classes that we will use the abstract factory pattern. So, these are some situations reuse and creating a family of objects from a class hierarchy. We use other patterns that become more appropriate but the creator pattern is one of the most used pattern, but very intuitive. We are almost at the end of this session, we stop here and continue in the next session. Thank you.