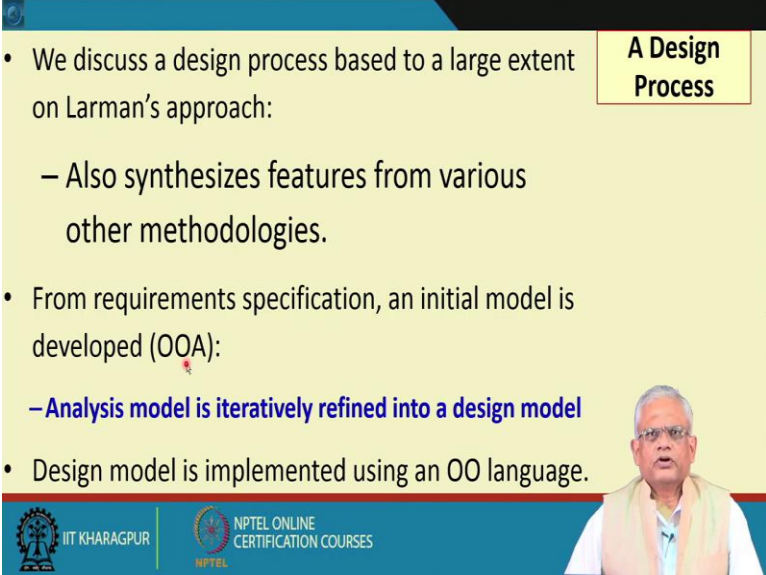


Object-Oriented System Development Using UML, Java and Patterns
Professor Rajib Mall
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur
Lecture 30
OOAD-I

Welcome to this session.

In the last session, we had just started to discuss about object-oriented analysis and design. We have so far in this course completed the basic notations of the UML, the different diagrams and the notations used. And now we will see how to use those notations to perform object-oriented analysis and design for a given problem. We just started the topic object-oriented analysis and design in the last session.

(Refer Slide Time: 0:53)



A Design Process

- We discuss a design process based to a large extent on Larman's approach:
 - Also synthesizes features from various other methodologies.
- From requirements specification, an initial model is developed (OOA):
 - **Analysis model is iteratively refined into a design model**
- Design model is implemented using an OO language.

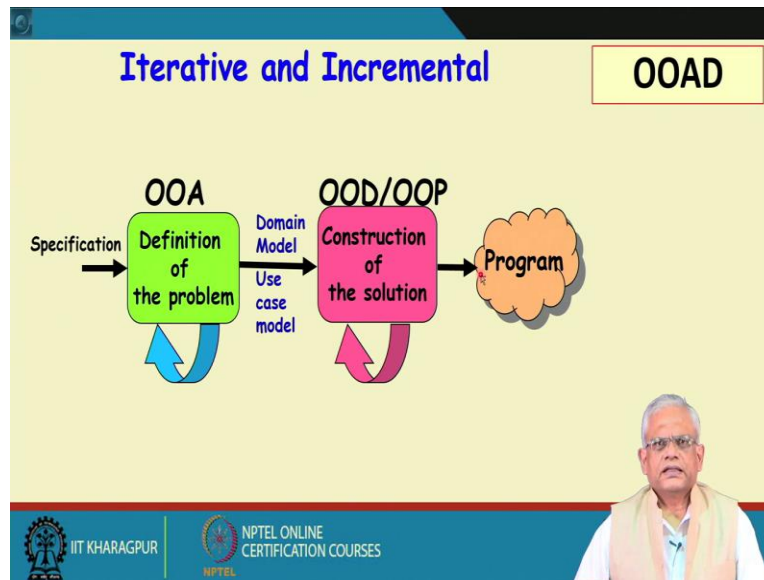
IT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

The approach that we are discussing is largely based on the Larman's approach, but then it has features from other approaches as well. And even though there are many approaches that are prevalent being used in the industry but more or less, they share almost similar steps. There may be small variations, but more or less the same steps.

In all these object-oriented analysis and design processes, we start with the requirements specification. And then we performed the object-oriented analysis. At the end of the object-oriented analysis process, we get the initial domain model and the use case diagram. And later,

we refine the domain model to get the class diagram and the class diagram is such that we can easily generate code from it maybe in Java, C++ or any other languages. So, we should be able to easily generate code from the object-oriented design model that we arrive at the end of our design process.

(Refer Slide Time: 2:35)




Pictorially we can represent as the object-oriented analysis process (in the above slide), which is iterative process where we elaborate the initial definition of the problem given to us by the customer. At the end of the analysis process, we get the domain model and the use case model. And based on these two models we will carry out the object-oriented design process which is again iterative process. At the end of it we will get the class diagram and then using any object-oriented programming language we can easily translate this to code.

(Refer Slide Time: 3:24)

OOA versus OOD?

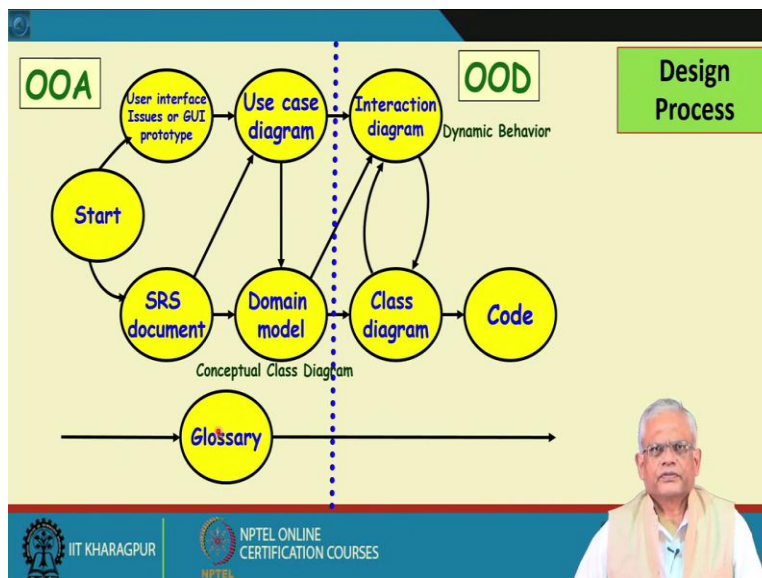
- **Analysis:**
 - An elaboration of requirements.
 - Independent of any specific implementation
- **Design:**
 - A refinement of the analysis model.
 - Takes implementation constraints into account



IT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

We are just discussing the difference between analysis and design steps. In analysis we just elaborated the problem and represent that in a UML diagram. On the other hand, in the design process, we refined the analysis model into a model which is easily translatable into code.

(Refer Slide Time: 4:00)

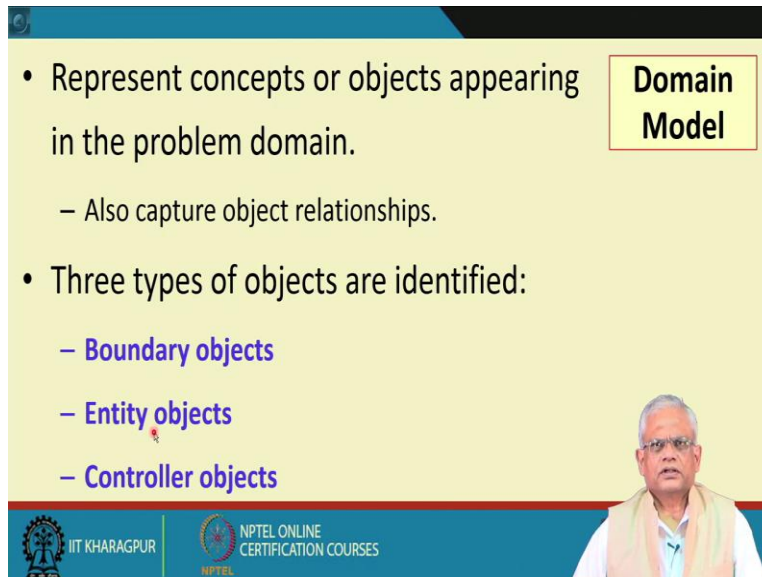


And we were mentioning that in the procedural design techniques, the analysis and design are very distinct phases, using very different notations, diagrams, but here in the object-oriented analysis and design, the diagrams are almost similar.

It seamlessly changes from analysis to design steps but then if we graphically capture our analysis and design process, we can say that the left side of this are the analysis and the right side is the design (in the above slide) and on the left side, we start with the initial user requirements and also a GUI prototype. Earlier while discussing the use case diagram we had seen that a user interface prototype is very helpful to develop the use case diagram, it helps us to effortlessly create packages of use cases and also take the SRS document, initial user specification and the prototype of the user interface, and based on that, we get the use case diagram. Once we have the use case diagram, we use the use case diagram and the SRS document to get the domain model. The domain model is a very important artefact. It is the conceptual class diagram. We will have some practice problems so we will see the steps of the domain modelling.

Given a problem, how to get the domain model we will identify the steps and also apply those to some example problems using which we can get the domain model and once we have the domain model, we can create the interaction diagram with the help of the use case diagram and the domain model. Then once we have the interaction diagram, we use the domain model and the interaction diagram to iteratively define the class diagram, and the class diagram can be easily translated to code. On each of these processes that we carry out, we introduce new terminologies class names, attributing names, method names and so on and that forms our glossary. All through the design process, the glossary builds up.

(Refer Slide Time: 7:22)



The slide is titled "Domain Model" in a yellow box. It contains the following text:

- Represent concepts or objects appearing in the problem domain.
 - Also capture object relationships.
- Three types of objects are identified:
 - **Boundary objects**
 - **Entity objects**
 - **Controller objects**

At the bottom right of the slide is a video feed of a man with glasses and a white shirt. The bottom of the slide features logos for IIT KHARAGPUR and NPTEL ONLINE CERTIFICATION COURSES.

Now, let's look at the domain modelling we know so far based on our discussion on use case modelling. We know, given a problem how to arrive at the use case model. After we have got the use case model, the next big step is to arrive at the domain model. The domain model is a representation of the concepts or objects that appear in the problem description. And also, in addition to identifying the objects or classes, we also identify the relations among classes. During the identification of the classes, we look for three types of classes: The boundary classes or the boundary objects, the entity classes or the entity objects and the controller classes or the controller objects. These are three main types of classes in any problem we can identify and we will see that the roles of these classes are different. As we proceed this session, we will see the exact purpose of the boundary classes, the entity classes and the control classes and how to identify them from a given problem description.

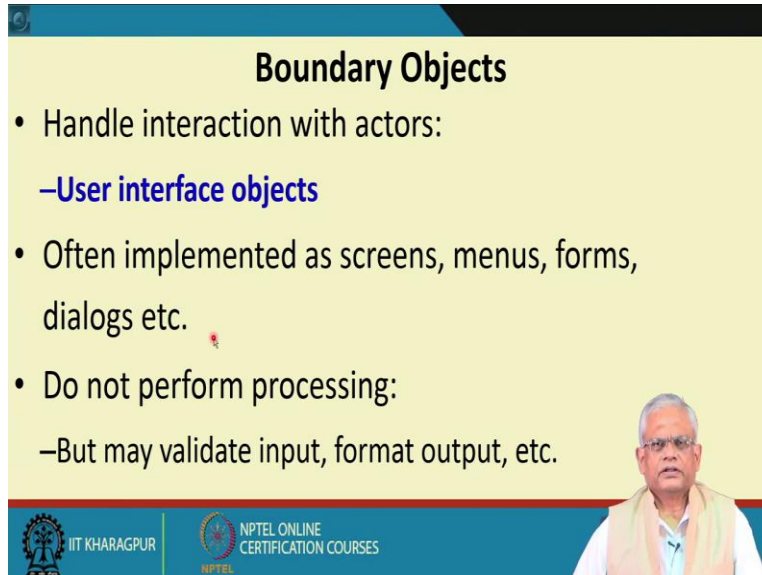
(Refer Slide Time: 8:59)

The slide, titled "Class Stereotypes", explains that three different stereotypes are used to represent classes: `<<boundary>>`, `<<control>>`, and `<<entity>>`. It illustrates these with three examples: a Boundary class named "Student Interface" represented by a circle with a vertical line on the left; a Control class named "BookIssue manager" represented by a circle with a curved arrow on the top-left; and an Entity class named "Book" represented by a circle with a horizontal line on the bottom.

These are three classes (in the above slide) which are almost always identified from a problem description. Identify the three classes from a problem description is the first step in domain model construction. And since these are identified for almost every problem so there are case tools present. Even someone can manually identify these three classes. Even while manually developing it helps to have some stereotypes because these are all classes after all. The boundary, the control and the entity classes have different roles. Their role is not similar.

Many of the case tools use this notation for a boundary class (as shown in the above slide). For example, a student interface which is a Boundary class will be represented using this symbol (in the above slide) in many case tools. The control classes (in the above slide) are represented in this symbol. An example of Control class is a book issue manager. An entity is represented using this symbol (in the above slide). An example of entity class is Book. These are symbols that can be easily drawn in a case tool, but in our manual problem-solving which we do in the next few slides, it is a bit cumbersome to draw this symbol and therefore will not use it actually. But just for your information, these are the stereotypes you might find in case tools.

(Refer Slide Time: 10:51)



Boundary Objects

- Handle interaction with actors:
 - User interface objects
- Often implemented as screens, menus, forms, dialogs etc.
- Do not perform processing:
 - But may validate input, format output, etc.

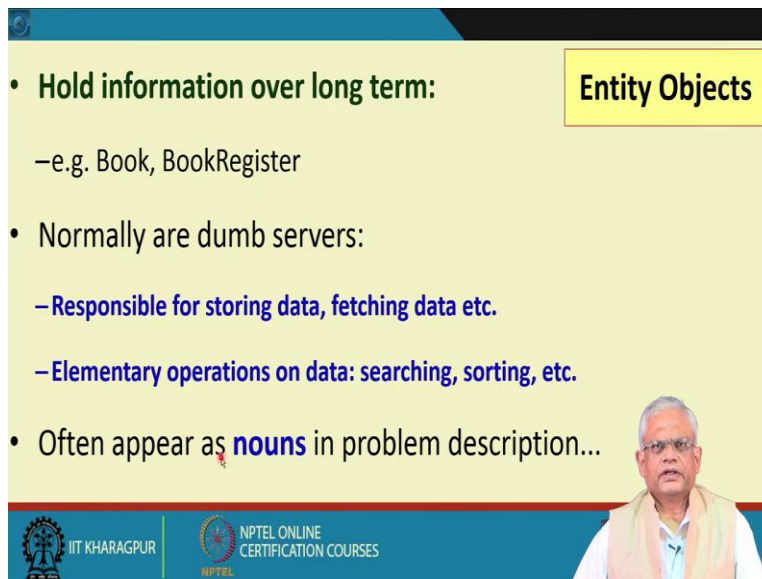
The slide features a video feed of a speaker in the bottom right corner. The footer contains the logos for IIT KHARAGPUR and NPTEL ONLINE CERTIFICATION COURSES.

Now, let's look at the first type of classes to be identified from a problem, these are called as boundary objects. The role of boundary object is to handle user interactions with the software. We can say that these are the user interface objects, you might wonder whether it would have been a good idea to call this as interface objects or user interface objects. But then in Java and other programming languages, the term interface is used for a different purpose. The interface classes, which have a different purpose, they are not really user interface classes. And therefore, UML avoids using the interface objects, calls these as boundary objects, but their role is to handle interactions with the actors. Examples of boundary objects are screens, menus, forms, dialog boxes et cetera with which user interacts and gives input, gets output from the system.

It is important to note that the boundary objects do not do any processing, their main purpose is to gather user input and give it to the other objects or to gather information from other objects and properly display it for the user. Boundary objects are mainly concerned with input and output, formatting, maybe sometimes validating the input, but nothing beyond that they do not do any processing steps. But then the first-time programmers who do not really use this kind of design processes they end up having some processing activity also associated with the user interface but which is not a good idea it makes the design difficult to maintain and also, the design becomes complicated.

Here in our approach we will have the boundary classes which do purely input and output operations: formatting the output properly displaying it, getting the user input from the keyboard, from mouse, et cetera and reporting this to the other objects in proper formatting. To present the data in proper format it may do some pre-processing that some characters are accepted the integer ranges between 1 to 10 and other integers are not acceptable. Only that kind of processing is done but nothing beyond that.

(Refer Slide Time: 14:16)



The slide is titled "Entity Objects" in a yellow box. It contains the following text:

- **Hold information over long term:**
 - e.g. Book, BookRegister
- Normally are dumb servers:
 - Responsible for storing data, fetching data etc.
 - Elementary operations on data: searching, sorting, etc.
- Often appear as **nouns** in problem description...

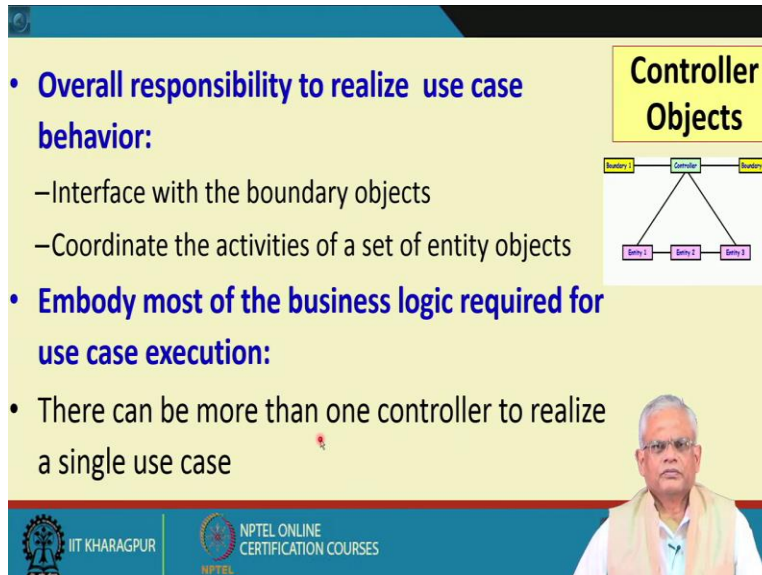
At the bottom of the slide, there is a video feed of a man speaking, and logos for IIT KHARAGPUR and NPTEL ONLINE CERTIFICATION COURSES.

The entity objects, these are the ones which store information for a long term. For example, we might have book objects, the book objects once created the name of the book, the author, the subject index, the ISBN number, et cetera are stored with book object. Once the book is created remains relatively long time maybe a couple of years, several years.

Similarly, a book registered is an entity object stores what are the authors, book number, and so on. And once created remains for long time. Unlike a user interface, where each time an interaction occurs the user interface object is created and after the end of the interaction, the interface is destroyed. But here the entity objects are long term once the entity object is created, remains for a reasonably long time. And the main purpose of the entity object is to store data. It also supports some methods for fetching data and also maybe some operations like search for data, sort the data items, et cetera. The entity objects typically appear as nouns in the problem description. A little while later, we will do the noun analysis to identify the entity objects.

Typically, the entity objects are identified based on a noun analysis of the problem description. We identify the nouns in the problem description, and some of those nouns will see under what situation we accept them to become the entity objects. Not all nouns are entity objects.

(Refer Slide Time: 16:24)



The slide features a yellow background with a blue header and footer. On the right side, there is a diagram titled "Controller Objects" showing a central box labeled "Controller" connected to two boxes labeled "Boundary 1" and "Boundary 2" above it, and three boxes labeled "Entity 1", "Entity 2", and "Entity 3" below it. A small inset image of a man in a white shirt and glasses is visible in the bottom right corner of the slide area.

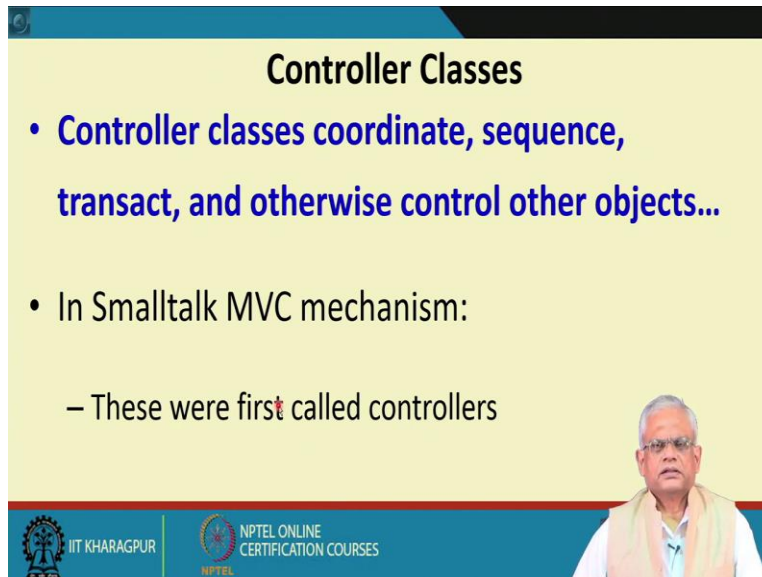
- **Overall responsibility to realize use case behavior:**
 - Interface with the boundary objects
 - Coordinate the activities of a set of entity objects
- **Embodiment of most of the business logic required for use case execution:**
 - There can be more than one controller to realize a single use case

The third category of the objects are the controller objects. The controller objects often cannot be identified from the problem description. We can say that these are synthesised objects. They do not really correspond to a physical object, but this plays very important role. We know that in a use case execution, several objects interact. We had seen in the interaction diagram that in one use case execution, several objects might participate. But then how do they interact with each other? What I mean to say is that there must be some object which directs the subjects so that they interact in some order. That is the role of the controller object, we will see as we proceed that these are a vital class objects. These are the objects who hold the business logic that means once an issue book or a request for the renew book is given to the software the controller objects take up and they know what exactly to do. They request first the corresponding member object to check whether the member had exceeded quota. And then they invoke a method on the member object to check if the membership is valid and then they check whether the book can be renewed means it has not been reserved by some other members. And then they register the book in the name of the member by calling an appropriate method on the member, and also on the book copy the member id is stored.

All these are handled by the controller class. In a complex use case, a controller might direct dozens of other objects to do their role at appropriate time. The controller object embodies the business logic that means for every use case we will have a controller who knows that once a request comes for that use case execution, it knows which objects to request in what order and what activities need to do that is, which methods to invoke on those objects. That we call as the business logic. For every use case the business logic is embedded in the corresponding controller object.

This implies that every use case should have one controller object and typically we write all our programs that way and our design is that way that for every use case we will have one controller object which has a business logic for that use case and directs all other objects that participate in that use case to do their appropriate roles. But sometimes the use cases are complex, maybe several dozens of objects need to be interrupted, in that situation controller becomes extremely complicated. In that situation, we may split the controller into multiple controllers for one use case, but normally we have one controller per use case.

(Refer Slide Time: 20:54)



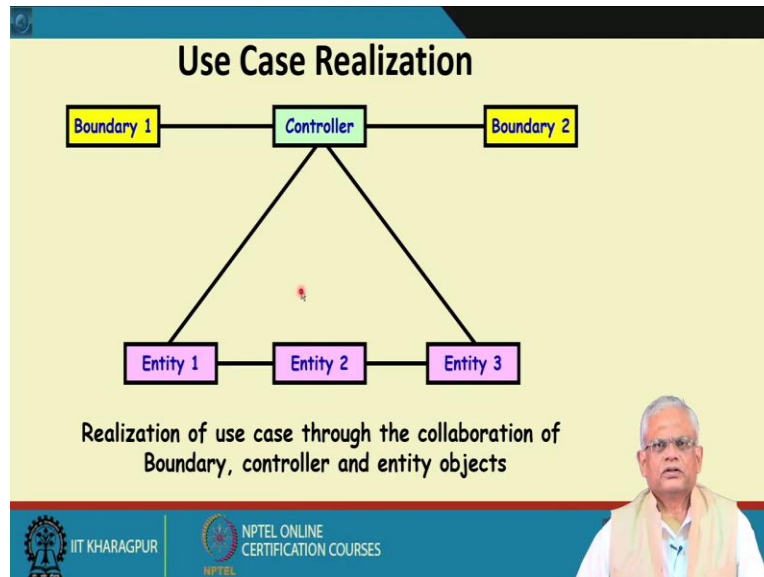
Controller Classes

- **Controller classes coordinate, sequence, transact, and otherwise control other objects...**
- In Smalltalk MVC mechanism:
 - These were first called controllers

The slide features a yellow background with a blue header and footer. The title 'Controller Classes' is centered at the top. Below it, two bullet points are listed. The first bullet point is in bold blue text. The second bullet point is in black text and includes a sub-bullet point. In the bottom right corner, there is a small video feed of a man with glasses and a white shirt. The footer contains the logos for IIT KHARAGPUR and NPTEL ONLINE CERTIFICATION COURSES.

The controller classes are synthesized objects they do not appear in the problem domain normally. We just synthesize them or we artificially invent these classes and these are very important classes. They coordinate, the interaction among other objects, they sequence these interactions or they controlled other objects. In the small talk, MVC mechanism model view controller (will see the model view controller pattern, as we start discussing) were first called as controllers and later these classes have been called as control objects or controller objects.

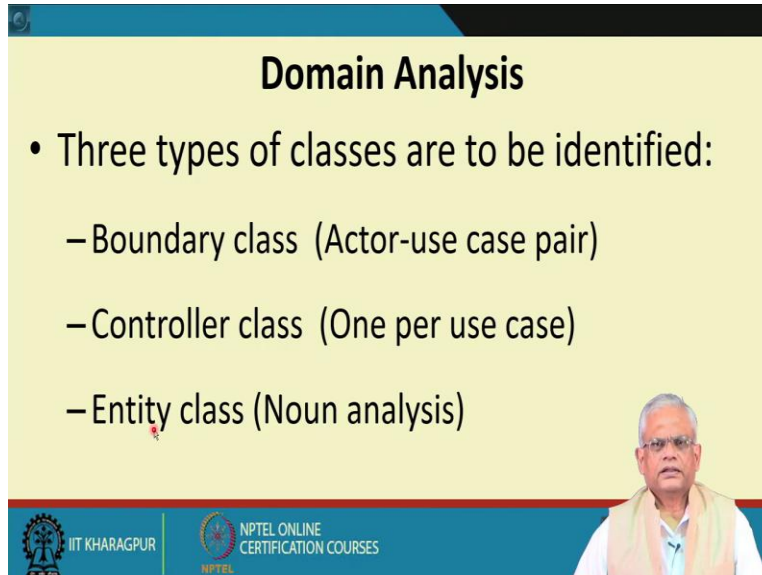
(Refer Slide Time: 21:55)



This diagram (in the above slide) just illustrates the role of a controller object. In a typical use case, the boundary object gets the data from the user that is the input from the user, and once it gets the input, it passes on to the controller. The controller knows what exactly needs to be done. For execution of the use case which class to contact. It contacts some class that may take help of other classes, and it may return the result to the controller and in the controller, it may display it on the boundary.



This is a simple situation, but then a controller might interact at different points with different objects. To explain the role of the controller we use this simple example but a controller might need to interact with several classes at different points of time.

(Refer Slide Time: 23:03)



Domain Analysis

- Three types of classes are to be identified:
 - Boundary class (Actor-use case pair)
 - Controller class (One per use case)
 - Entity class (Noun analysis)

Now, let's proceed with our domain analysis based on the use case diagram and the SRS document. We will perform the domain analysis and we will identify three types of classes: one is the boundary class, the second is a controller class and the third is the entity class. The first two classes are almost mechanically identified.

The rule for boundary class is we will have one boundary class for every actor use case pair. Very fairly straightforward rule, we will take some examples and explain that just by looking at the use case diagram we can identify the boundary classes. Similar is the controller classes just by looking at the use case diagram, we can identify one controller class per use case.

Possibly the most challenging part is the entity classes, we need to practice several problems before we become proficient in identifying the entity classes. To identify the entity classes, we have to really do the noun analysis, identify what are the nouns, and then eliminate some of those nouns and so on until we get the entity classes. But as we become more experienced, we will see that just by reading the problem description we will be able to identify the entity classes.

So, in a summary the boundary and the controller classes are the easiest to identify for a given use case diagram. The entity classes need some effort, we read the problem description and then mentally we do a known analysis, eliminate some of the nouns and other nouns that remain become the entity classes.

(Refer Slide Time: 25:21)

Identification of Boundary Objects from Use Case Diagram

- Need one boundary object :
 - For every actor-use case pair

IT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Identification of Boundary Objects from Use Case Diagram

- Need one boundary object :
 - For every actor-use case pair

IT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Let see how the boundary classes can be identified from a use case diagram. Let's look at this very simple use case diagram (in the above slide). There is only one-use case in the tic-tac-toe game which is Play Move and one actor the Player. And the rule here is that we need one boundary object for every actor use case pair. And here we have only one-use case and one actor, so there is one boundary. So, there is one boundary and this boundary helps the player to interact with the use case Play move. And we need only one boundary class for this problem and will name this boundary class as the 'player-play move boundary'.

So, we just identify the pairs, the actor and the use case and we write the name of the boundary. We might take a slightly bigger problem. Let say we take this problem (in the above slide Supermarket Prize scheme) where there are three use cases and four actors. So, we have four use case actor pair. And if we identify these, we will see that there is a boundary here (pointed by red circle in the above slide). So, one boundary for customer and register customer, another boundary sales clerk and register sales, another boundary is manager and select winners and one more is clerk and register customer. These boundaries are basically some user interface objects which take care of the interaction of the customer with a register customer use case and the clerk with the register customer, the sales clerk with the registered sales, and so on.

So, there are four boundary classes that can be identified from this diagram very mechanically, and we will be named this appropriately as a 'customer register customer boundary', 'sales clerk registered sales boundary', 'manager select winner boundary', and 'clerk register customer boundary'.

We are almost at the end of this session. In the next session will try to identify how to determine the entity classes and the controller classes. The controller classes are rather straightforward, just like the boundary classes and by inspecting the use case diagram we can identify the boundary classes. And for the entity, classes will do a formal noun analysis, but as we become an experience to solve a dozen problem, we will see that we do not need a formal noun analysis, just a mental noun analysis and by just reading through the problem description, we can identify the entity classes.

We are almost at the end of this session,

We will stop here and we will continue in the next session.

Thank you.