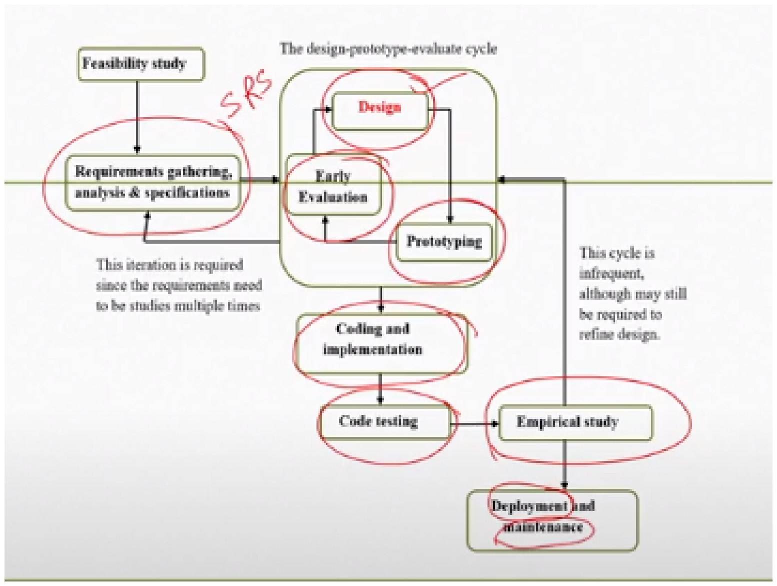
Design & Implementation of Human – Computer Interfaces
Prof. Dr. Samit Bhattacharya
Department of Computer Science & Engineering
Indian Institute of Technology, Guwahati

Module No # 05
Lecture No # 25
UML

Hello and welcome to the NPTEL MOOC'S course on design and implementation of human computer interfaces lecture number 22. Where we are going to introduce a particular language called unified modeling language, or UML which is suitable for object oriented design. If you recollect we are currently discussing the interactive system development life cycle consisting of several stages; the life cycle is designed to help develop interactive systems in a very systematic manner.

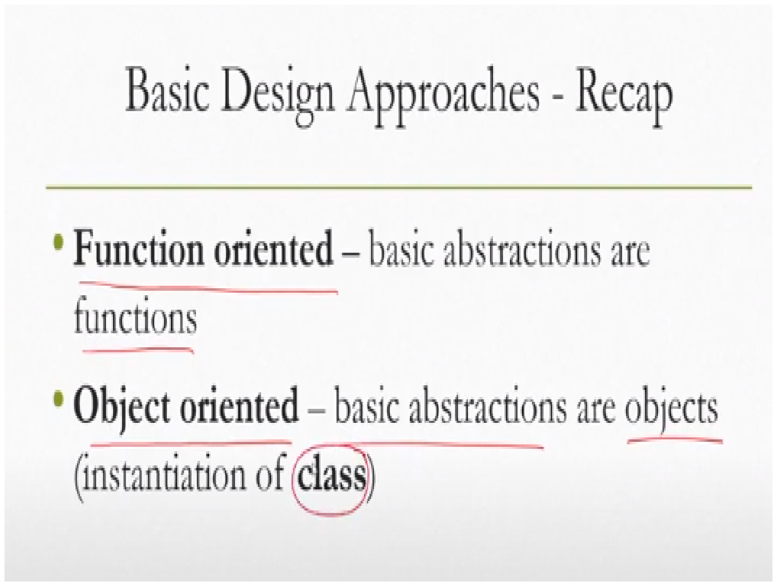
(Refer Slide Time: 01:12)


The design-prototype-evaluate cycle
Feasibility study
SRS
Requirements gathering, analysis & specifications
Design
Early Evaluation
Prototyping
This iteration is required since the requirements need to be studies multiple times
This cycle is infrequent, although may still be required to refine design.
Coding and implementation
Code testing
Empirical study
Deployment and maintenance

In the development life cycle we have already covered in details the requirement gathering, analysis and specification stage at the end of this stage we get an SRS document or software requirement specification document. Then we have talked about this design prototype and evaluation cycle consisting of these 3 sub-stages. Now this cycle is primarily meant for interface design once we arrive at a stable interface design taking care of usability concerns we go for code design.

So there this prototype and evaluation is not very important we concentrate on the design part, currently we are discussing the design of code or system. Once that design is done; we go for coding and implementation of the design followed by testing of the code. Then the whole system is tested for usability with end users, and finally we go for deployment and maintenance stages.

(Refer Slide Time: 02:31)



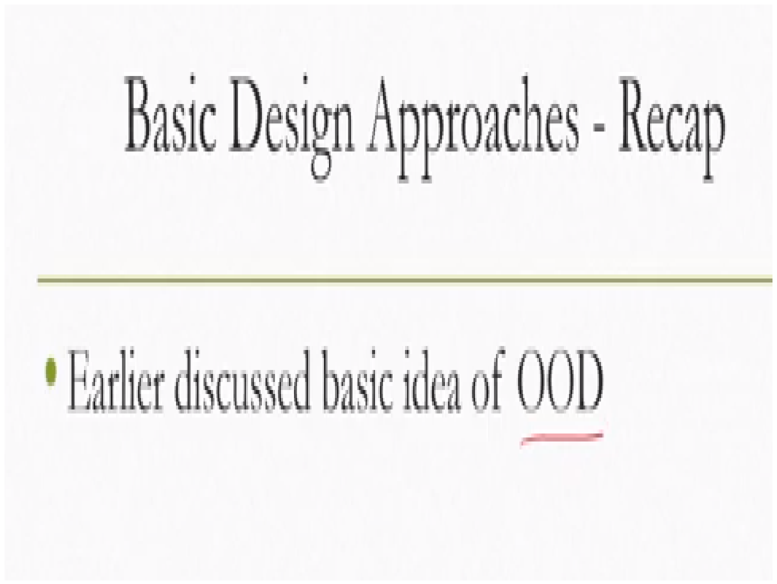
Basic Design Approaches - Recap

- **Function oriented** – basic abstractions are functions
- **Object oriented** – basic abstractions are objects (instantiation of **class**)

So the design stage we started our discussion in the previous few lectures particularly related to code design. Now if you may recollect we have mentioned 2 broad design approaches one is function oriented design, other one is object oriented design. So earlier we talked about function oriented design in details which relies on the idea of functions, so in function oriented design the basic abstraction that are used for designing a system are functions.

In contrast in object oriented design basic abstractions are objects which are instantiation of generic concepts called classes or class. So in object oriented design basic abstraction units that are used are objects now earlier in the previous lecture we introduced the concept of objects and classes in terms of some examples.

(Refer Slide Time: 03:34)

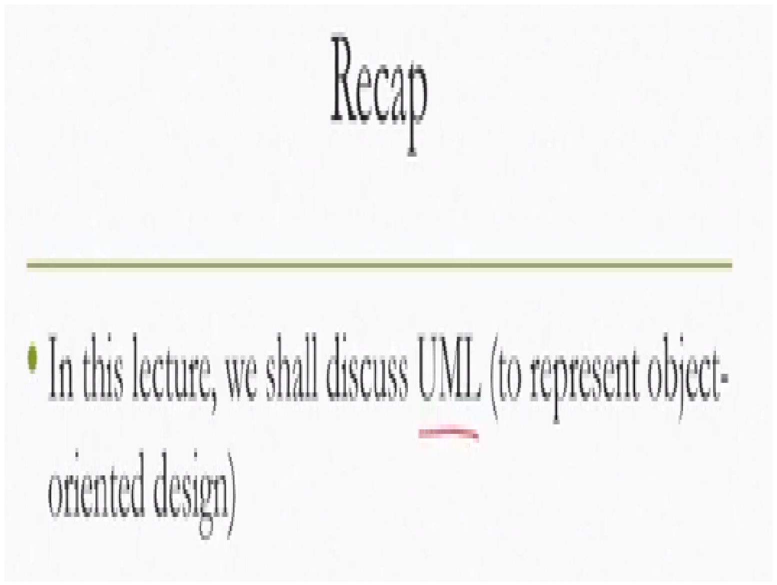


Basic Design Approaches - Recap

- Earlier discussed basic idea of OOD

So the basic idea of object oriented, design have already been covered in the previous lecture.

(Refer Slide Time: 03:39)



Recap

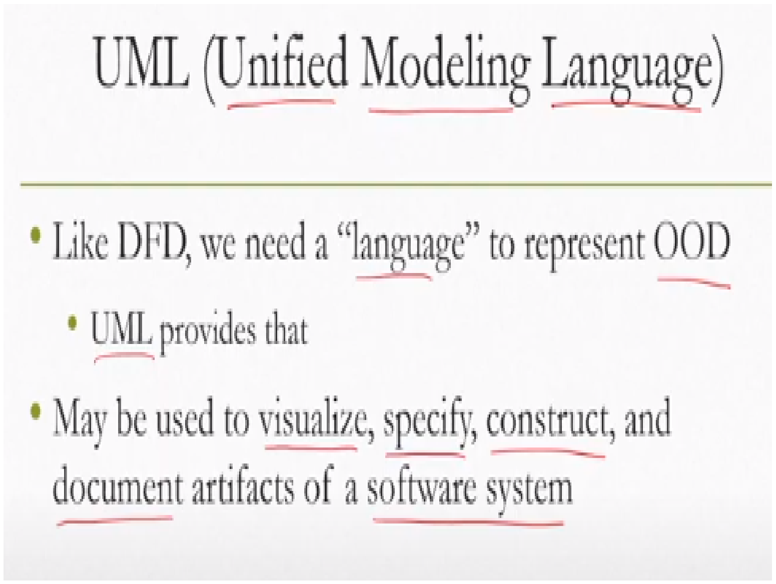
- In this lecture, we shall discuss UML (to represent object-oriented design)

In this lecture what we are going to do we are going to learn about UML or unified modeling language which is used to represent object oriented design. Remember that earlier what we mentioned is for any design activity there are primarily 2 concerns; where to start the design process, and how to represent the design. In case of code design also these concerns are there where to start as we have already seen the starting point is the SRS or software requirement specification document.

And how to represent in the case of function oriented design we have seen that representation language is DFD or data flow diagram, and also we have learned about in brief ER diagram or entity relationship diagram together they can be used to represent function oriented design of a system. For object oriented design again the starting point can be the SRS document and to represent object oriented design we need some language.

So in this lecture we are going to talk about one of such languages which is basically a graphical language called UML or unified modeling language. So let us see what; is UML what are the notations and symbols used in UML and how we can use those notations and symbols to come up with a system design.

(Refer Slide Time: 05:11)



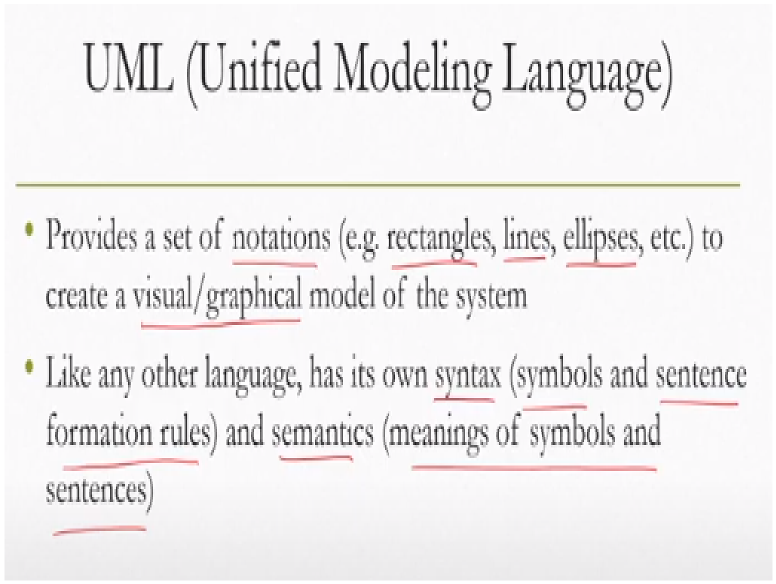
UML (Unified Modeling Language)

- Like DFD, we need a "language" to represent OOD
  - UML provides that
- May be used to visualize, specify, construct, and document artifacts of a software system

As I already mentioned UML stands for unified modeling language so this is an acronym for unified modeling language like DFD. We need a language to represent object oriented design which can be achieved with UML, but remember that UML is one of many such languages but it is the most popular one; so what we can do with UML.

Once we come up with a design or once we conceptualize a design we can use UML to visualize the design specify the design construct its components and linkages and document various artifacts that are part of the overall system. So using UML we can do a plethora of activities including visualization documentation and specification.

(Refer Slide Time: 06:25)



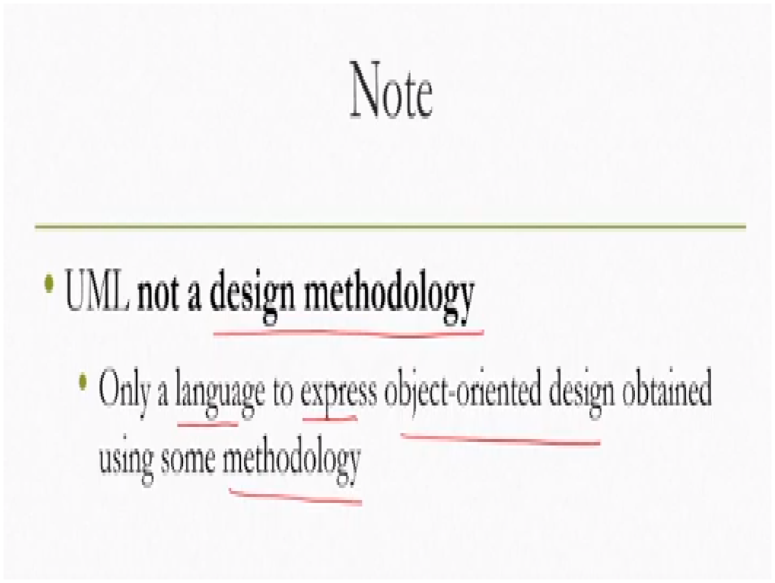
UML (Unified Modeling Language)

- Provides a set of notations (e.g. rectangles, lines, ellipses, etc.) to create a visual/graphical model of the system
- Like any other language, has its own syntax (symbols and sentence formation rules) and semantics (meanings of symbols and sentences)

In a nutshell what is UML? UML is a language that allows us to represent object oriented design, how it allows us to do that it provides a set of notations. For example rectangles, lines, ellipses and so on to create a visual or graphical, model of the system. So this is a graphical language like DFD it allows us to create a graphical representation of the system using notations like rectangles, ellipse, lines, and so on.

Like any other language it has its own syntax that is the symbols and sentence formation rules and semantics that is the meanings of the symbols and the sentences. Of course this syntax and semantics are different than the natural languages like English or Hindi or Bengali because this is a different sort of language. But conceptually it also has syntax like any natural language and semantics like any natural language.

(Refer Slide Time: 07:37)



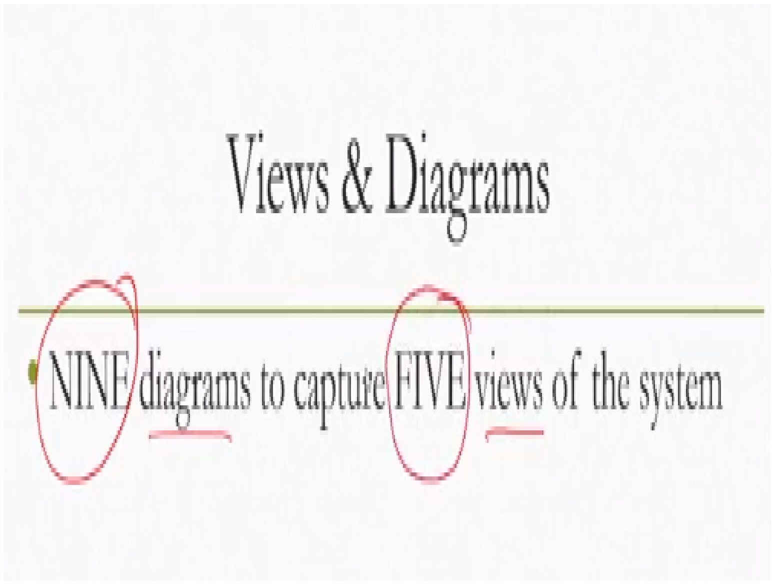
Note

- UML not a design methodology
  - Only a language to express object-oriented design obtained using some methodology

One thing we should always keep in mind is that UML is not a design methodology, so we should not have this impression that UML helps us to design. UML does not help us to design it does not give us any guidelines or any hints for a good design; rather it allows us to express our design. So that distinction should be always kept in the mind. It is nothing but just a language to express object oriented design obtained using some other methodology.

So with UML we can only express a design with UML we cannot get any help for the design that is the crucial thing we should always keep in mind.

(Refer Slide Time: 08:39)

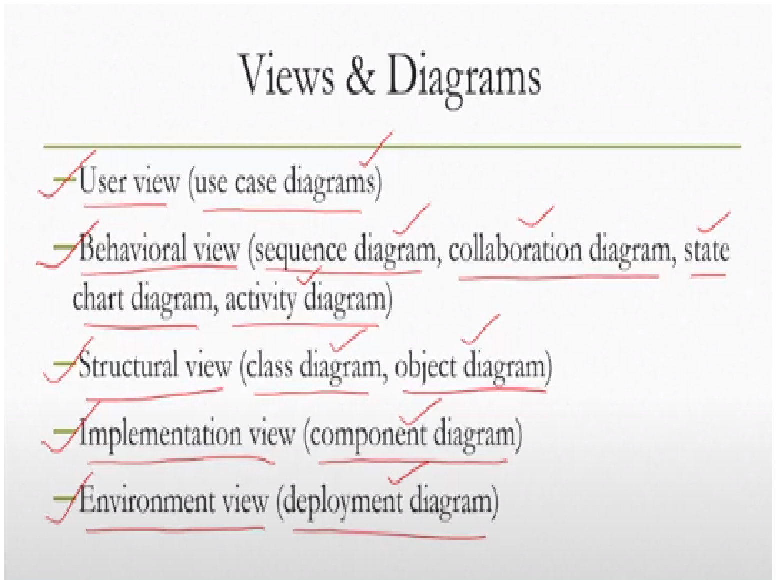


Views & Diagrams

- NINE diagrams to capture FIVE views of the system

So with UML we can do several things broadly it allows us to create diagrams to capture different views of the system. So essentially we look at the system from different points of views and UML allows us to create those views using diagrams it supports 9 such diagrams to capture 5 different views of the system.

(Refer Slide Time: 09:16)



Views & Diagrams

- User view (use case diagrams)
- Behavioral view (sequence diagram, collaboration diagram, state chart diagram, activity diagram)
- Structural view (class diagram, object diagram)
- Implementation view (component diagram)
- Environment view (deployment diagram)

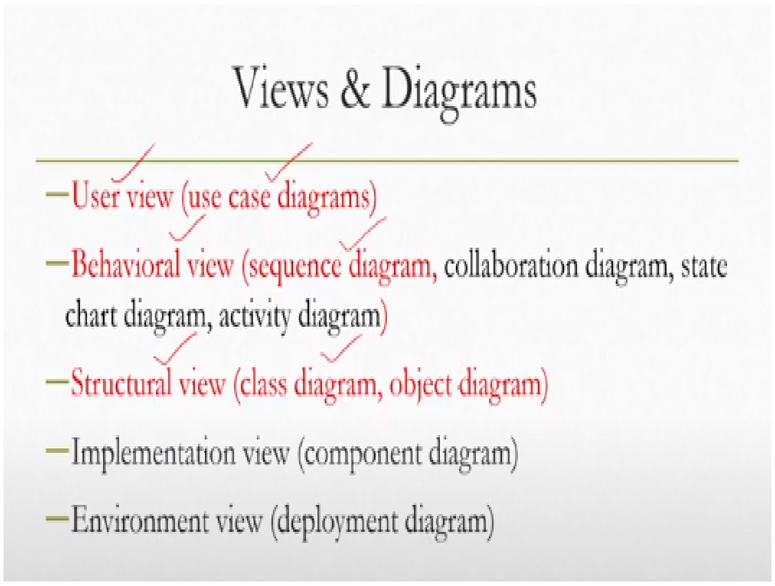
So what are those views and diagrams first is user view, how a user perceives the system. So the system can be viewed from the point of view of the user and UML provides that view in terms of use case diagrams. Then we can have a behavioral view of the system that means how the system components behave during execution of the system. To get the behavioral view different diagrams are there and UML supports different such diagrams; such as sequence diagram, collaboration diagram, state chart diagram, activity diagram.

Any of these diagrams can be used to represent the behavioral view of the system; that means how the system behaves during runtime. Then we can have a structural view how the system is structured around its units for that 2 diagrams; are there supported by UML one is class diagram, one is object diagram. Then we have implementation view and for representing this view we can make use of the component diagram.

Finally we have the environment view and this view can be expressed in terms of the deployment diagram. So these are the 5 views that are supported by UML user view, behavioral view, structural view, implementation view, and environment view. And in order to represent these views 9 diagrams are there which can be used. Use case diagram can be used for user view; any of the 4 diagrams sequence diagram, collaboration diagram, state chart diagram, and activity diagram can be used for representing behavioral.

View any of the 2 diagrams class diagram or object diagram can be used to represent structural; view component diagram can be used to represent implementation view. And finally the deployment diagram can be used to represent environment view. Among all these views and diagrams in this lecture we are going to restrict our discussion to few of the views and diagrams, which are most important.

(Refer Slide Time: 12:01)

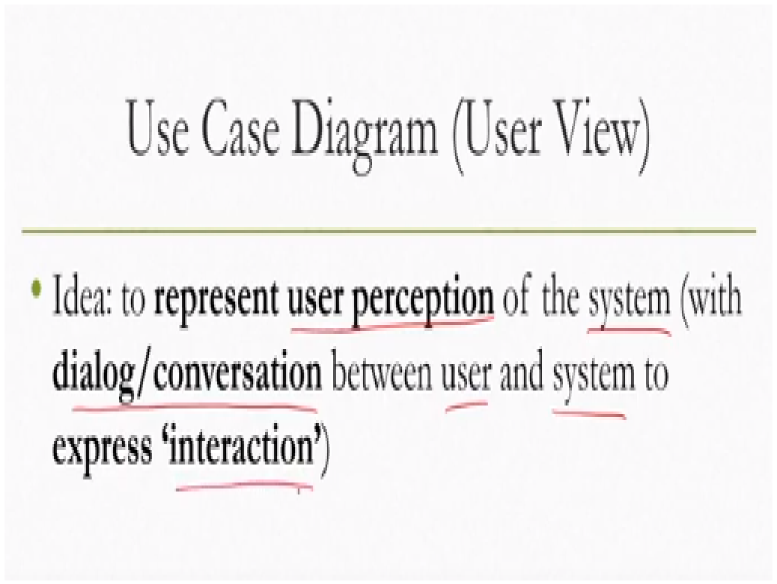


Views & Diagrams

- User view (use case diagrams)
- Behavioral view (sequence diagram, collaboration diagram, state chart diagram, activity diagram)
- Structural view (class diagram, object diagram)
- Implementation view (component diagram)
- Environment view (deployment diagram)

Namely we will learn about what is an, user view and how to use that, how to represent that view using the use case diagram. Similarly we will learn about behavioral view and how to represent that view using sequence diagram. So among all the 4 different diagrams that can be used to represent behavioral view we will learn about sequence diagram. And we will also have a look at the structural view of the system in terms of class diagram, object diagram we are not going to discuss in this lecture. So let us start with the first view that is user view what it means and how it can be represented using the use case diagram.

(Refer Slide Time: 12:55)

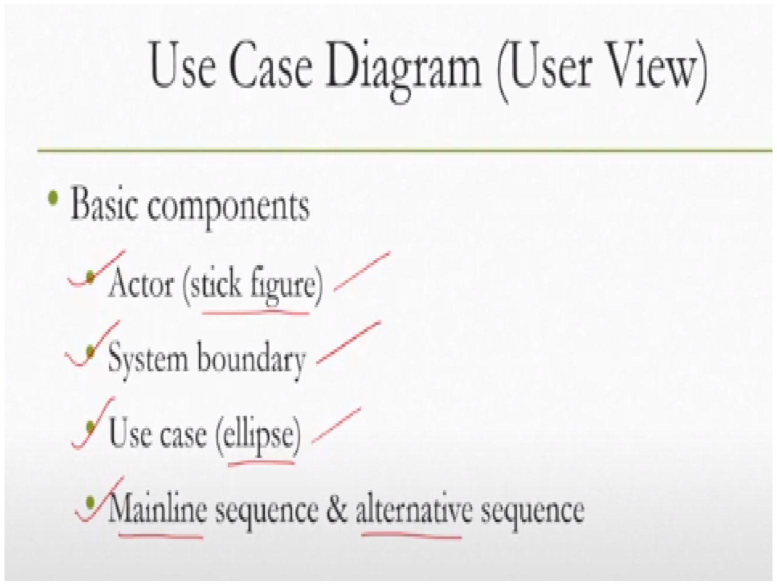

Use Case Diagram (User View)

- Idea: to **represent user perception** of the system (with **dialog/conversation** between user and system to **express 'interaction'**)

When we are talking of user view the idea is that we want to represent user perception of the system; that is how the user perceives the system. Note that this is different from the developer perception of the system, where developer is mostly concerned about the internal working of the system whereas users are mostly concerned about how input can be provided and outputs are generated.

Now to represent users, perception in UML we can make use of dialogue or conversation between the user and the system to express interaction between these 2 entities, namely the user and the system.

(Refer Slide Time: 13:42)

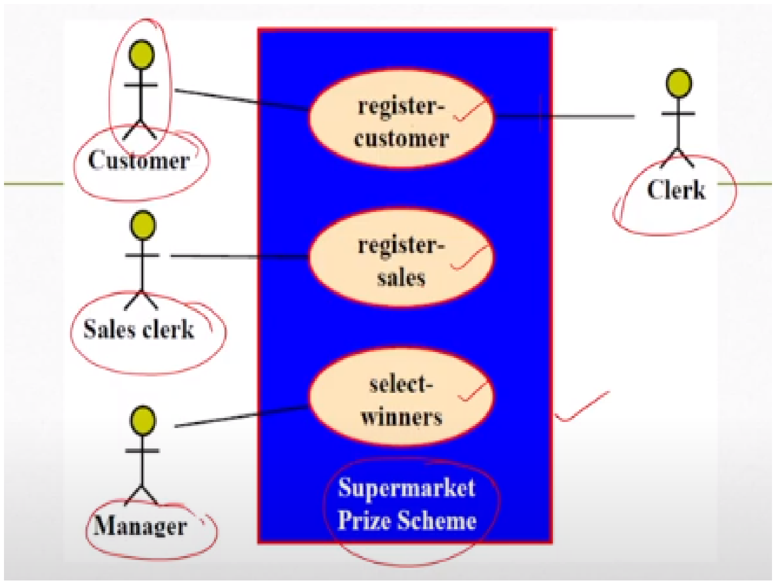

Use Case Diagram (User View)

- Basic components
  - ✓ Actor (stick figure)
  - ✓ System boundary
  - ✓ Use case (ellipse)
  - ✓ Mainline sequence & alternative sequence

So use case diagram is nothing but representation of the dialogue between a user and the system. Now in order to capture that in order to capture the interaction we need some so what are the basic components for use case diagram there is this actor component represented with a stick figure, what is that we will see so shortly in an example. Then we have system boundary, so one is actor one is system boundary then use case represented as ellipse, and finally the main line and alternative sequences.

So there are primarily these 4 components which makes up the use case diagram actor represented with a stick figure, system boundary, use case represented with ellipse, and the concepts of mainline sequence and alternative sequence. Let us try to understand what these components mean and how they can be used to represent use cases.

(Refer Slide Time: 15:02)



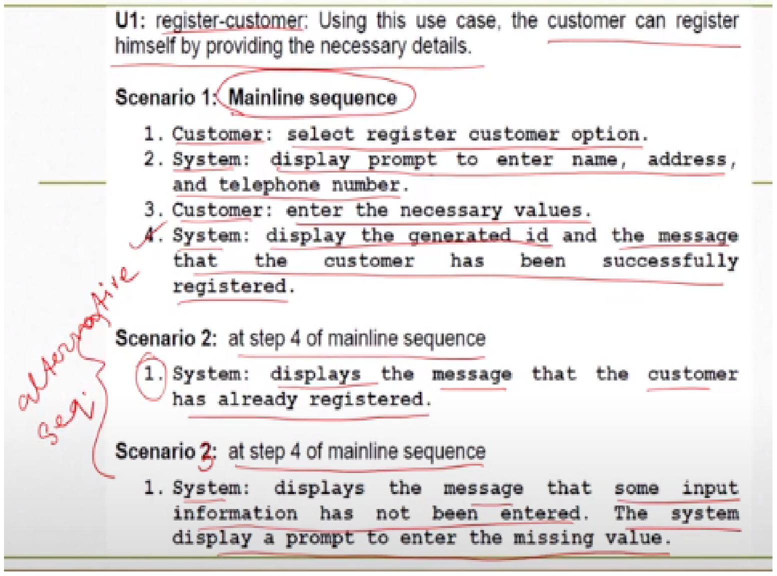
Let us try to understand that in terms of one example, suppose this is one system which allows a store to manage price awarding scheme. Let us call it supermarkets price scheme that means, if some customers come to the market they can participate in some lottery and win some prizes. This is the system now from the point of view of the users, so who are the users in this case there can be broadly 4 types of users which are very apparent.

One is the customer, one is the sales clerk, one is the manager, one can be a regular clerk. Of course you can think of more or less number of stakeholders or users but let us stick to these 4 types of users for the system. So for customer there can be one use case; that is register customer for sales clerk, one use case can be register sales for manager, it can be select winners. Register customer can be use case for the clerk type of user also.

Now here you can see that these users are represented with stick figures like this one, the use cases are represented with ellipse like register customer, register sale select winners. The whole system is represented with this rectangle with boundaries shown in red lines, so this is the system boundary. And there are these lines from the user to the, use cases indicating which use case is meant for which user.

If there is only one user and one use case of course and that will be very apparent, and we can simplify the notation; otherwise we have to make use of this type of notation. So as you can see there are 4 users and 3 use cases we can think of, now that is the user's point of view. Now to represent the actual interaction between the user and the system we need a dialog that dialog is called mainline sequence. And if there is some deviation in the dialog which needs to be taken care of then we call it alternative sequence.

(Refer Slide Time: 17:44)



U1: register-customer: Using this use case, the customer can register himself by providing the necessary details.

Scenario 1: **Mainline sequence**

```
1. Customer: select register customer option.
2. System: display prompt to enter name, address,
   and telephone number.
3. Customer: enter the necessary values.
4. System: display the generated id and the message
   that the customer has been successfully
   registered.
```

Scenario 2: at step 4 of mainline sequence

```
1. System: displays the message that the customer
   has already registered.
```

Scenario 2: at step 4 of mainline sequence

```
1. System: displays the message that some input
   information has not been entered. The system
   display a prompt to enter the missing value.
```

alternative seq.

For example consider the use case register customer, for the customer user; so using this use case the customer can register him or herself by providing the necessary details. Now from the customer's point of view what should be the interaction with the system; what should be the interaction, and if there is some issue then what should be the interaction. So ideal interaction is called main line sequence, assuming there is no problem in the ideal interaction.

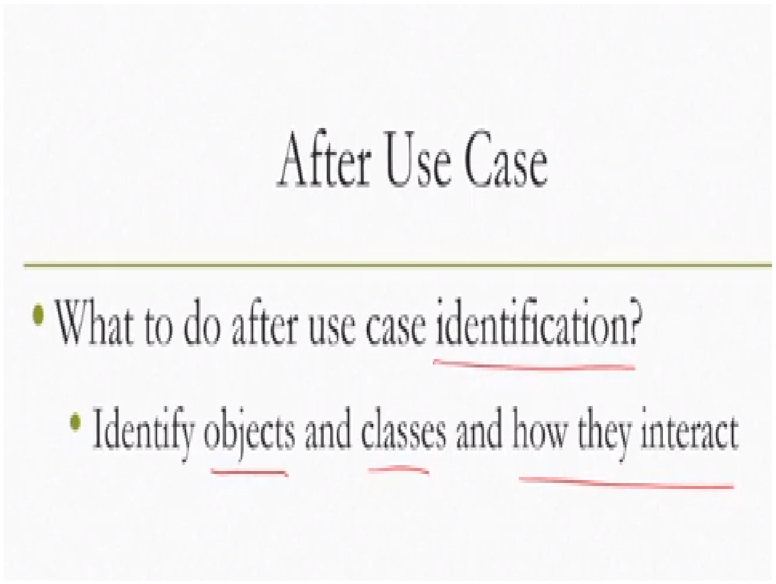
Customer selects the register customer option from the interface then the system responds by displaying a prompt to enter the details like name, address, and telephone number. Then customer enters all the necessary details as prompted by the system finally the system displays the generated id customer id and the message that the customer has been successfully registered. So ideally this should be the interaction and this should be the view from the point of view, of the user.

However if there is some issue with this sequence then we have to go for alternative sequence. Now in step 4 some deviations may happen, for example at step 4 of the main line sequence the system may display the message that the customer has already registered no need to re-register. So in mainline sequence ideally it should generate id and display the message that registration is successful.

So the message that the customer is already registered, so in that case mainline sequence will not be followed in totality 1, 2, 3, 4 up to this point it will be followed then this dialogue will be followed. There may be further deviations as well like at step 4 of the main line sequence system may also display the message that some input information has not been entered, so the system displays a prompt to enter the missing values.

So in ideal case mainline sequence should be followed if the case is not ideal, then either of scenario 2 or scenario 3 it should be scenario 3 so either of scenario 2 or scenario 3 should be followed; now these are called alternative sequences. So this is how we can represent the user view using the use case diagram, so the diagram consists of the stick figures, the lines, the ellipses, and the system boundaries which is accompanied by this main line and alternative sequences. I hope the idea of the user view and the use case diagram is clear with this example. Let us now move to the next view that is the behavioral view of the system.

(Refer Slide Time: 21:02)

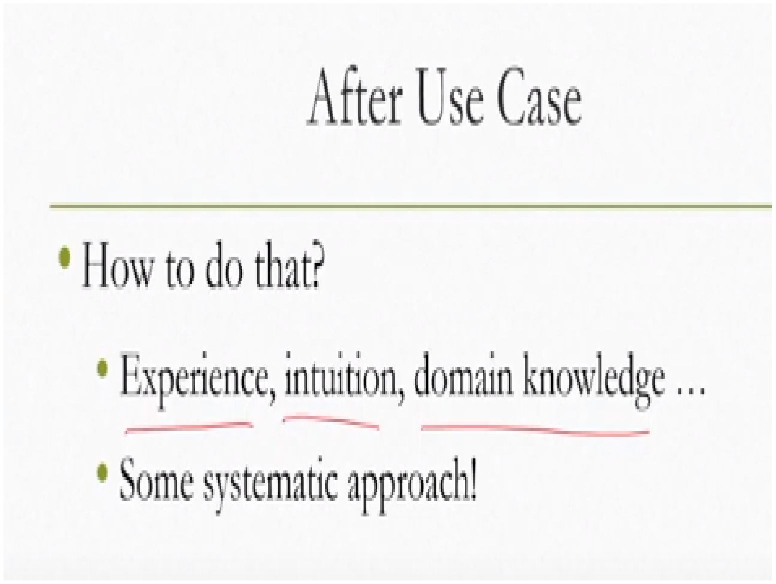


After Use Case

- What to do after use case identification?
  - Identify objects and classes and how they interact

So once we are able to identify the use cases and represent the use cases using the use case diagrams, what is next; what to do after the use case identification. Remember we are using an object oriented design approach, so our next objective should be to identify objects and classes and how they interact. So our next objective is identify, the objects and the classes that represent those objects.

(Refer Slide Time: 21:36)

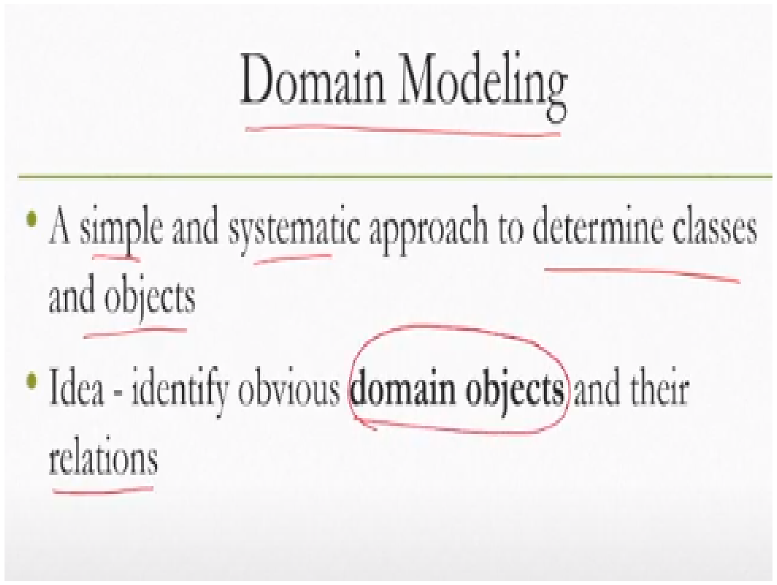


After Use Case

- How to do that?
  - Experience, intuition, domain knowledge …
  - Some systematic approach!

Identification of objects from the use case diagram is definitely not very easy task so, how we can do that? If we have experience of doing such things, or we can rely on our intuition, or we have some domain knowledge. Then definitely those are helpful but if we do not have any of these or our experience is not sufficient, or domain knowledge is not sufficient then we may face problems.

(Refer Slide Time: 22:15)

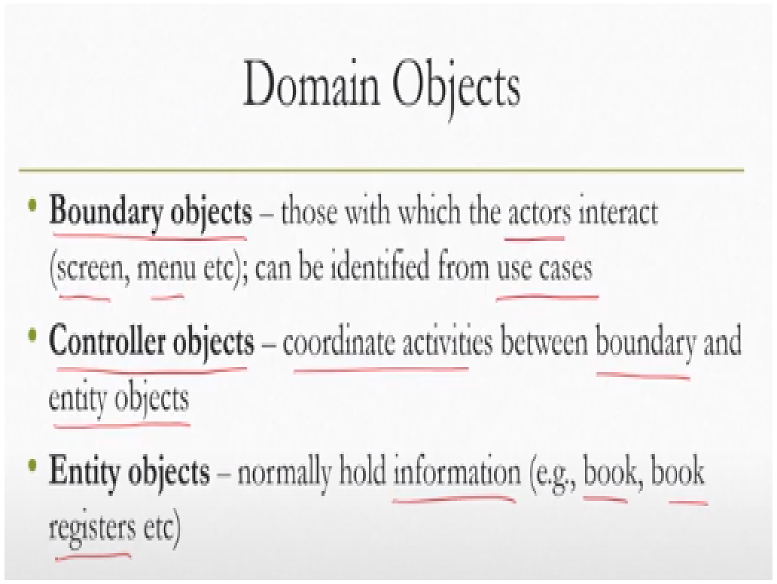


Domain Modeling

- A simple and systematic approach to determine classes and objects
- Idea - identify obvious **domain objects** and their relations

Then can there be some systematic approach? There can be one very simple and systematic approach called domain modeling approach. Using which we can identify some simple objects, and then from there we can go for class definition for those objects. So domain modeling is nothing but a simple and systematic approach to determine classes and objects, here the idea is to identify obvious domain objects and their relations.

(Refer Slide Time: 22:45)



Domain Objects

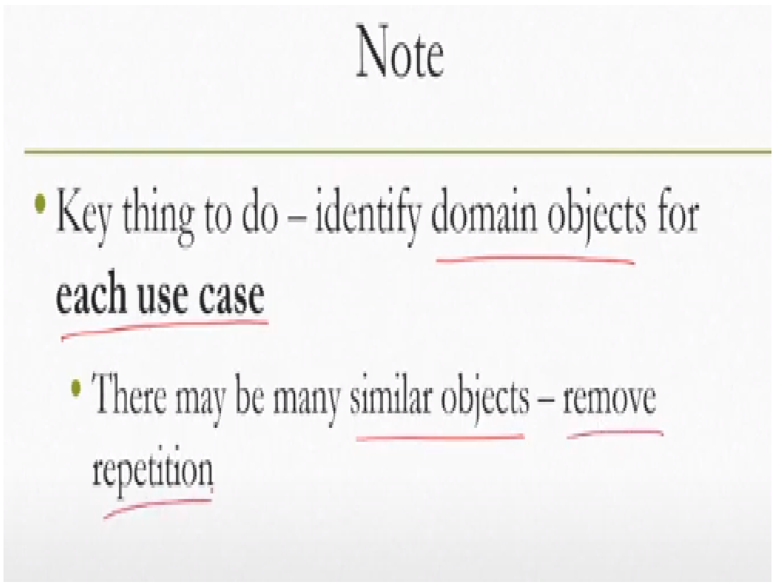
- **Boundary objects** – those with which the actors interact (screen, menu etc); can be identified from use cases
- **Controller objects** – coordinate activities between boundary and entity objects
- **Entity objects** – normally hold information (e.g., book, book registers etc)

Now what are domain objects; there can be broadly 3 groups of domain objects one is the boundary objects, these are the objects with which the actors interact for example screen, menu, etc now these objects can be identified from use cases. So once we are ready with our use cases we can immediately identify the boundary objects. The next category of objects; are the controller objects; these are objects that coordinate activities between the boundary objects and another class of objects that is called the entity objects.

So what are these entity objects, these are objects that normally hold some information so in our system if we are using some data storage or information storage we can label them as entity objects. For example a book, book registers and so on; so earlier we are using the term data storage now we will be using the term entity objects to represent those objects.

So this is the idea of domain objects broadly 3 categories boundary, controller, and entity objects; boundary objects are easy to identify from use cases, also entity objects are not very difficult to identify. And once we are able to identify these 2 we can come up with intermediate objects that connect these 2 types of objects which we call controller objects.

(Refer Slide Time: 24:16)

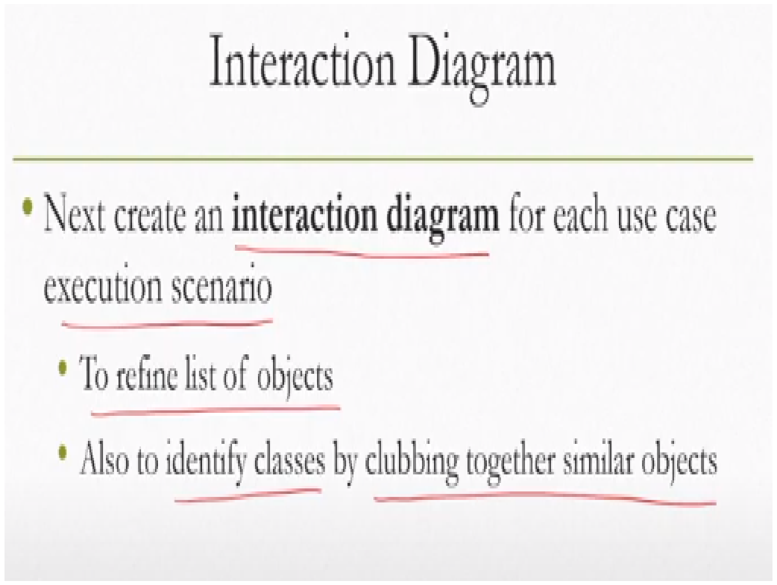


Note

- Key thing to do – identify domain objects for **each use case**
  - There may be many similar objects – remove repetition

So the key thing to do is to identify domain objects for each use case, once we identify all the use cases for each use cases we have to act separately and identify these 3 types of domain objects. Then we need to combine them together to come up with a combined set of objects for the system. It may so happen that in this combination some objects are repeated it is quite likely, because there may be similar objects in different use cases. So we have to be careful and remove those repetitions from the combined list of objects.

(Refer Slide Time: 25:09)



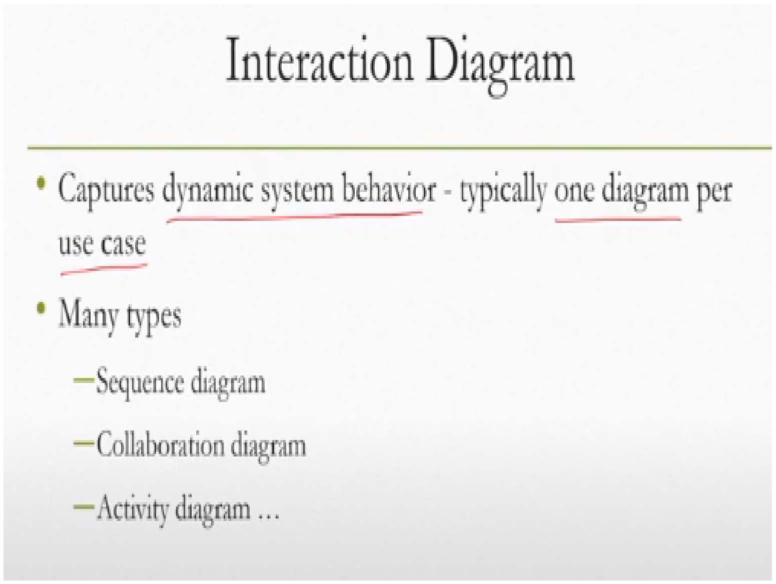
Interaction Diagram

- Next create an **interaction diagram** for each use case execution scenario
  - To refine list of objects
  - Also to identify classes by clubbing together similar objects

Once we are able to create the list of objects our next task is to create an interaction diagram for each use case execution scenario. Why we need this diagram to represent the behavioral view of the system that is how the system behaves during execution, in other words how the objects interact with each other during execution of the system. Now that behavioral view we can represent with an interaction diagram,

Remember that we mentioned several such diagrams sequence diagram, collaboration diagram activity diagram, state chart diagram, any of these diagrams we can use to represent the interaction between objects. Now this diagram can be useful to refine the list of objects and also to identify classes by clubbing together similar objects. So these are the primary 2 purposes for creating an interaction diagram between the objects which represent the interaction between them during execution of the system.

(Refer Slide Time: 26:12)

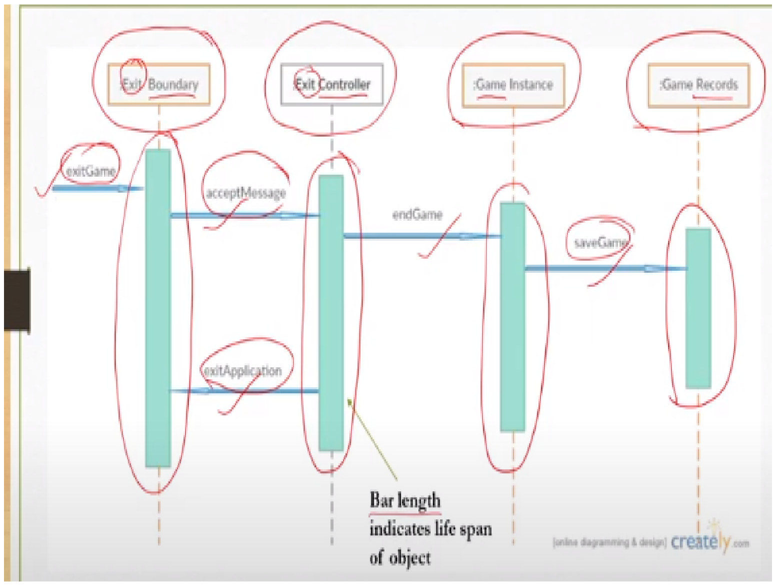


Interaction Diagram

- Captures dynamic system behavior - typically one diagram per use case
- Many types
  - Sequence diagram
  - Collaboration diagram
  - Activity diagram …

So interaction diagrams captures dynamic system behavior again it is advisable to go for one diagram per use case, later on we can merge this diagram together to avoid repetitions. Already I mentioned that there are many such types of diagrams sequence diagram, collaboration diagram, activity diagram, state chart diagram; out of these we are going to briefly learn about sequence diagram.

(Refer Slide Time: 26:55)



With one example like we did for use case diagram so this image shows a sequence diagram for a system, an imaginary system, a game system a video game. Now as you can see on top within rectangles we have marked certain things, these are essentially the objects that we have identified from the use cases. Exit menu option is a boundary object, exit controller object, then game instance which is another object, and game records that is an entity object.

So suppose we have identified these 4 objects using domain modeling approach, for each object we have created one vertical bar, like shown here. Now these bars are having some significance what they signify as you can see there are 4 bars each representing one of the objects. And the bars are of unequal length now this bar length indicates the life span of the object during execution of the system.

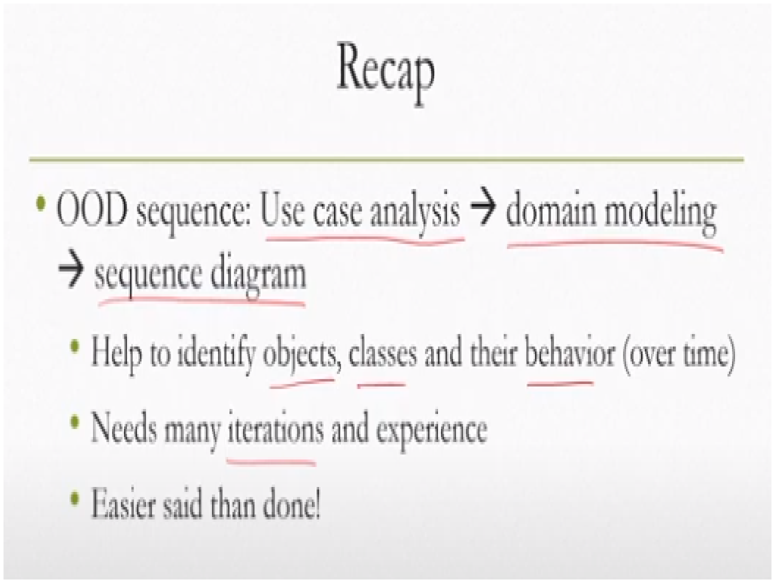
In this image, in this diagram as you can see the exit boundary object has the maximum length that means it remains active for the maximum duration of execution. Exit controller object has somewhat lesser length than the exit boundary length that means it dies or becomes inactive before the exit boundary object. So boundary object may remain active but the controller object remains inactive after performing its assigned tasks.

The game instance object is even more, short which means it dies quote, unquote dies before the boundary and controller objects. And finally the records object is the shortest which dies before all the other objects the arrows as shown here indicates the messages that goes or passed between the different objects. So each arrow is labeled with some text which indicates the message name.

For example exit game is a message sent to the, exit boundary object accept message and exit application are 2 messages passed between boundary and controller objects save game is a message passed between game instance and game records objects and so on. So we have basically these notations namely rectangles on top indicating the name of the object vertical bars indicating the life span of the objects; and message passing between them using arrows with labels, the direction of the arrows indicate the direction of the message.

So this essentially indicates the behavior during execution that is how long an object remains active; what messages are passed between objects and so on. So this is the behavior that we capture diagram which is sequence diagram, same behavior we can capture with other interaction diagrams that is collaboration diagram, activity diagram, state chart diagram etcetera.

(Refer Slide Time: 30:49)



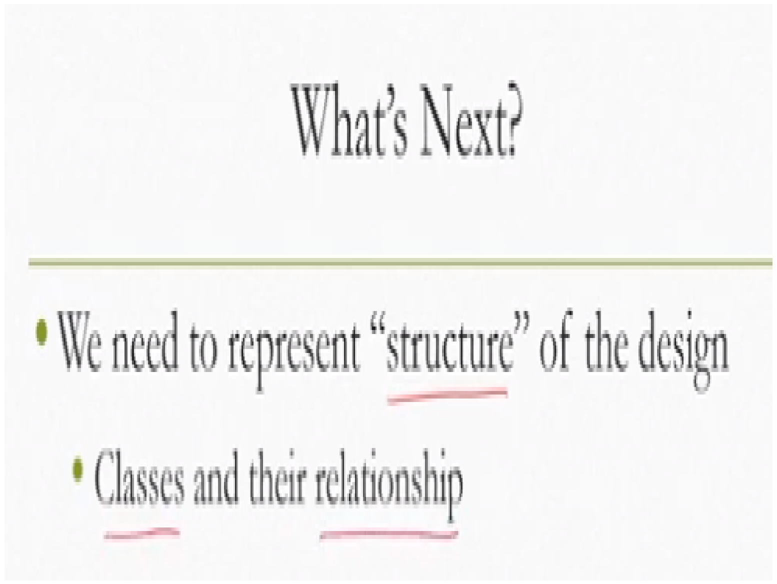
Recap

- OOD sequence: Use case analysis → domain modeling → sequence diagram
  - Help to identify objects, classes and their behavior (over time)
  - Needs many iterations and experience
  - Easier said than done!

So what we have learnt so far, so the sequence to be followed to create a design and represent using UML when we are using an object oriented design approaches. First we go for use case analysis and then get the user view with the use case diagram. From there we use domain modeling, to go for identification of objects and those objects are then used to create an interaction diagram which can be any of sequence collaboration activity or state chat diagram to get the behavior of the system during execution.

So if we follow this sequence then we will be able to identify objects, classes, and their behavior over a period of time essentially during execution of the system. Of course this is easier said than done it requires many iterations and good amount of experience to find out the objects and create the behavior, it is not a very easy thing to do. Once we manage to get the behavioral view what next?

(Refer Slide Time: 32:06)



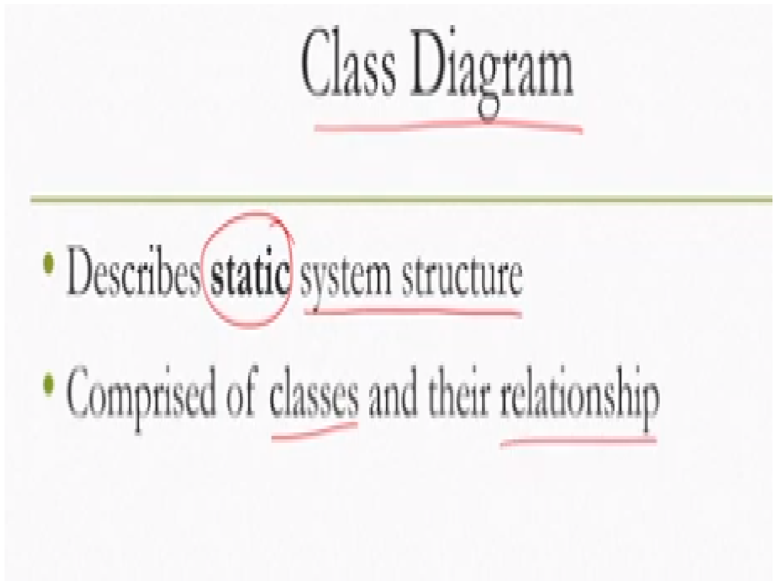
What's Next?

- We need to represent "structure" of the design
  - Classes and their relationship

Next we need to go for representing the structure of the design that means classes and their relationships. So earlier we talked about objects and how they interact with each other during execution that is behavior. Now behind the objects there are classes and the classes are related to each other with some relations we need to capture that, so use case analysis and interaction diagrams help us to identify objection from their classes.

Then we need to represent these classes and their relationships in some form or we need to represent the structure of the system, so that is our next step, let us see how we can do that. So next we will go for creating a structural view of the system.

(Refer Slide Time: 33:02)

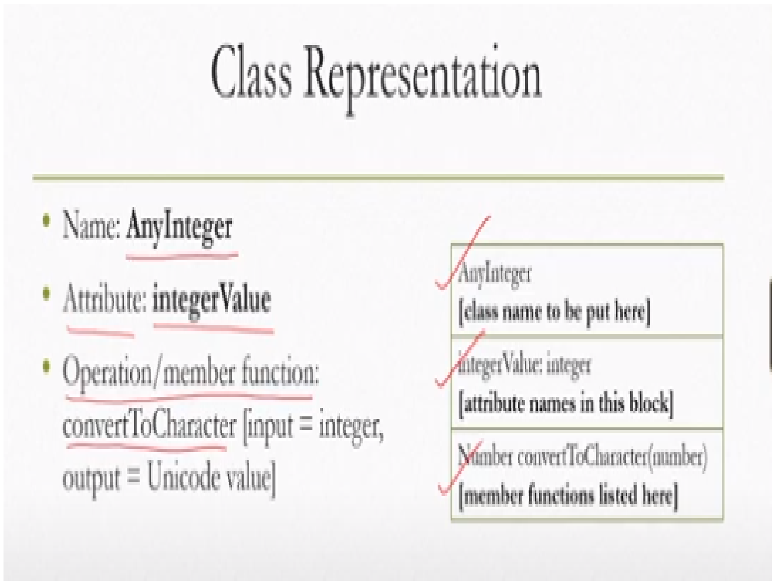


Class Diagram

- Describes **static** system structure
- Comprised of classes and their relationship

Now the structural view we will create using one type of diagram called class diagram. A class diagram describes static system structure, unlike interaction diagram which represents a temporal interaction between objects during execution of the system. Now class diagrams, consists of classes, and their relationships let us try to understand these concepts, with respect to some examples.

(Refer Slide Time: 33:46)



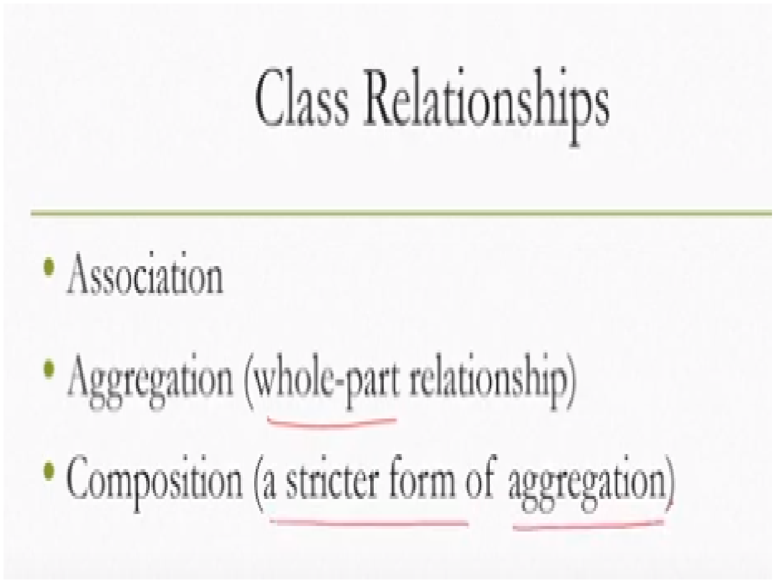
Class Representation

- Name: **AnyInteger**
- Attribute: **integerValue**
- Operation/member function: convertToCharacter [input = integer, output = Unicode value]

AnyInteger **[class name to be put here]**
integerValue: integer **[attribute names in this block]**
Number convertToCharacter(number) **[member functions listed here]**

First of all recollect from our previous lecture the idea of class we said that a class has a name with some convention followed attribute which contains the data values. And some operations or member functions again with some naming conventions and input and output specified. Now this class we can represent it in this rectangle with 3 cells name, attributes, and member function names, so that is how we can represent classes. Once we are able to represent classes we can represent the relationships between the classes.

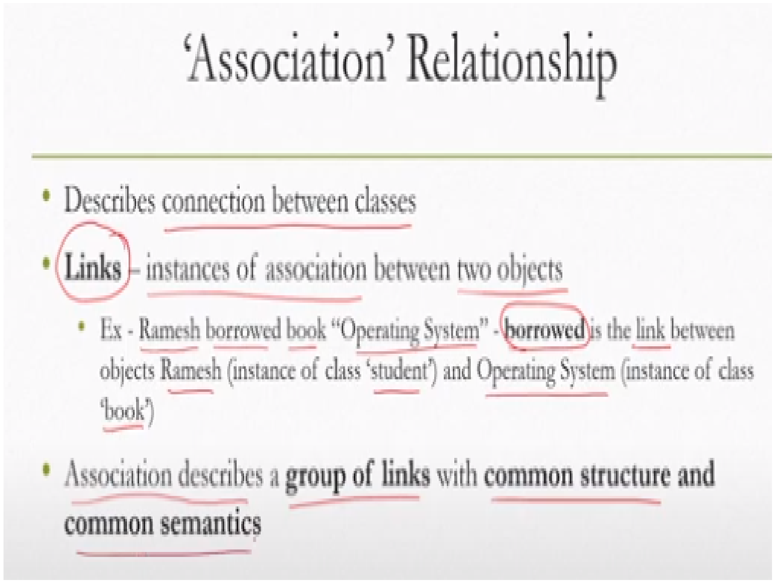
(Refer Slide Time: 34:34)


Class Relationships

- Association
- Aggregation (whole-part relationship)
- Composition (a stricter form of aggregation)

What are the relationships that can exist between classes several relationships can be there association, aggregation and composition. So classes can be related with association relationship they can be related with aggregation relationship or the whole part relationship. And they can be related with the composition relationship that is a stricter form of the aggregation relationship.

(Refer Slide Time: 34:54)

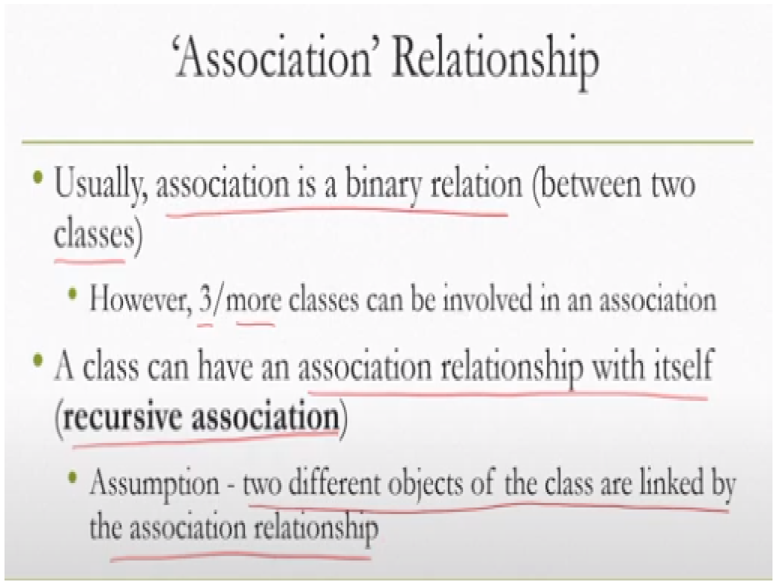

'Association' Relationship

- Describes connection between classes
- **Links** – instances of association between two objects
  - Ex - Ramesh borrowed book "Operating System" - **borrowed** is the link between objects Ramesh (instance of class 'student') and Operating System (instance of class 'book')
- Association describes a **group of links** with **common structure and common semantics**

Let us learn about the association relationship, now association essentially describes connection between classes we can use the concept of links to represent instances of association between 2 objects. For example suppose a person named Ramesh borrowed a book named operating system, so here borrowed can be considered to be the link between objects Ramesh. Which can be an instance of the; class student and operating system which can be the instance of the class book.

So we have 2 classes, student and book, and 2 instances Ramesh and operating system; these 2 instances can be connected with the association relationship or a link called borrowed. So association essentially describes a group of links with common structure and common semantics that is how this relationship is defined, a group of links with common structure and common semantics.

(Refer Slide Time: 36:12)



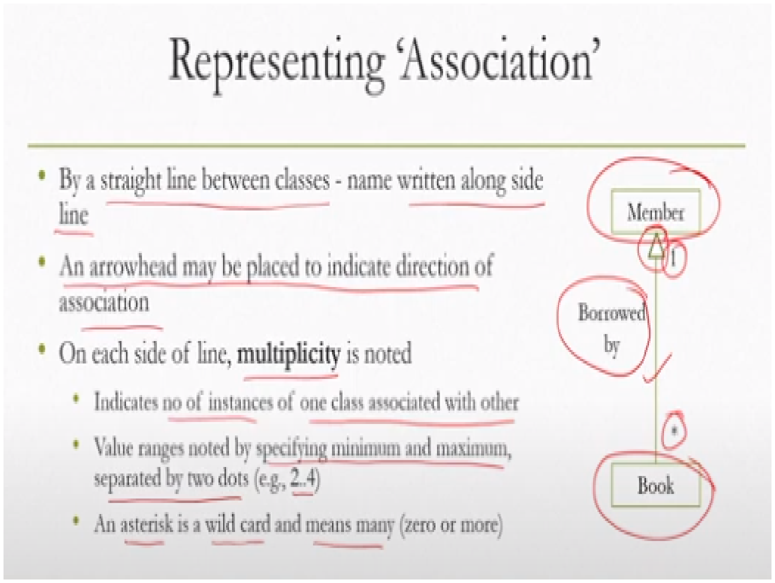
'Association' Relationship

- Usually, association is a binary relation (between two classes)
  - However, 3/more classes can be involved in an association
- A class can have an association relationship with itself (**recursive association**)
  - Assumption - two different objects of the class are linked by the association relationship

Usually association is a binary relation, that means between 2 classes however it is possible to have 3 or more classes to have this relationship, but that is not very frequent mostly it is a binary relationship. A class can have an association relationship with itself that is also possible which is called recursive association, where the assumption is 2 different objects of the class are linked by the association relationship.

So this is a concept related to the association relation how we can represent the relationship; remember that in UML we need some notation for graphical representation of everything.

(Refer Slide Time: 37:08)



Representing 'Association'

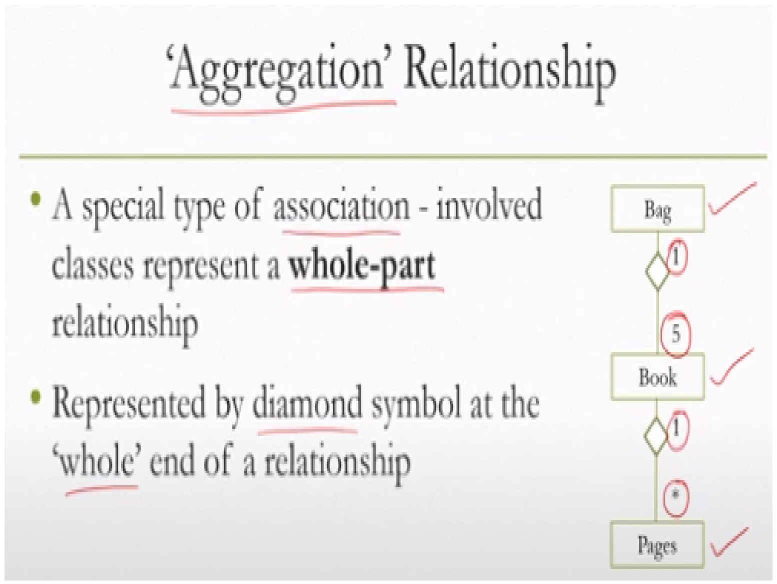
- By a straight line between classes - name written along side line
- An arrowhead may be placed to indicate direction of association
- On each side of line, **multiplicity** is noted
  - Indicates no of instances of one class associated with other
  - Value ranges noted by specifying minimum and maximum, separated by two dots (e.g, 2..4)
  - An asterisk is a wild card and means many (zero or more)

So for association we can use such a notation. We can use a straight line between classes where the name of the relation is written alongside the line. So it is a labeled line like; shown here this is a book class, and this is a member class and this straight line indicates the association relation borrowed by an arrowhead may be placed to indicate direction of association like shown here, it is advisable to show the direction.

Now on each side of the line a concept called multiplicity is mentioned, this indicates, like here we have mentioned 1 and star. Now this indicates number of instances of one class associated with other value ranges noted by specifying minimum and maximum separated by 2 dots; like 2 dot, dot 4 so we can also specify value ranges rather, than a single number. And if we put an asterisk it is a wildcard notation and means many.

For example when this asterisk is put here it indicates that a member can borrow many books, that is, the idea of this multiplicity notation placed alongside the line that is about association.

(Refer Slide Time: 38:42)



'Aggregation' Relationship

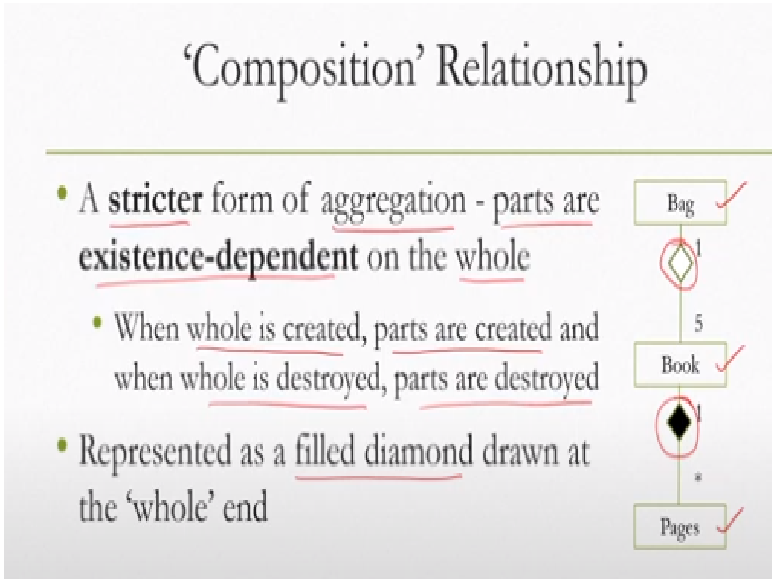
- A special type of association - involved classes represent a **whole-part** relationship
- Represented by diamond symbol at the 'whole' end of a relationship

Bag
1
5
Book
1
*
Pages

Next comes aggregation relationship this is a special type of association where the involved classes represent a whole part relationship. To represent such a relationship a separate symbol is used a diamond symbol at the whole end of a relationship. For example consider this example, we have a bag class, a book class and pages class; now between pages and book there is a whole part relationship.

So pages are part of book it means that one book can contain many pages, similarly between books and back there is a whole part relationship which indicates that one bag can contain 5 books. The notations and symbols are mostly same with association except the use of the diamond symbol instead of an arrow.

(Refer Slide Time: 39:43)



'Composition' Relationship

- A **stricter** form of aggregation - parts are **existence-dependent** on the whole
  - When whole is created, parts are created and when whole is destroyed, parts are destroyed
- Represented as a filled diamond drawn at the 'whole' end

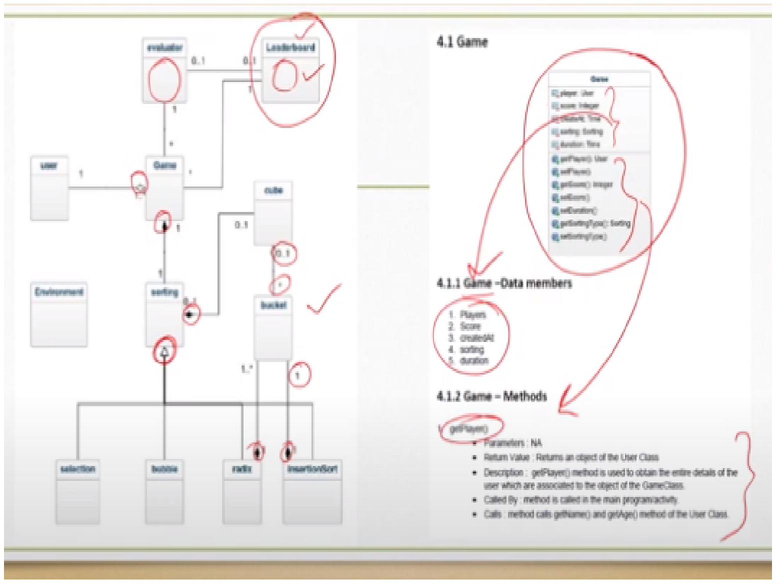
Bag
1
5
Book
1
*
Pages

Finally comes the composition relationship; this is a stricter form of aggregation relationship note that aggregation is a stricter form of association and composition is a stricter form of aggregation. Here parts are existence dependent on the whole that means when the whole is created parts are created and when the whole is destroyed parts are also destroyed they no longer exist. For example we can define the relationship between the 3 class's bag, book and pages in a slightly different way, between pages and books we can define a composition relationship.

That means when a book is created the pages are created and when a book is destroyed all the pages are destroyed. However between book and bag we need not define composition relationship, that means when a bag is created books need not be created and when a bag is destroyed books inside the bag need not be destroyed it is up to us how we define the relationship between classes.

So between book and bag we can keep the aggregation relation and between pages and book we can have the composition relations. Now here as you can see there is slight change in notation, so for aggregation we use diamond now for composition, to indicate composition relationship we use the field diamond at the whole end. That means here between pages and books book is the whole pages are part of it. So at the book end of the relation we place this diamond field diamond to indicate that the relationship between them is a composition relation.

(Refer Slide Time: 41:36)



4.1 Game

4.1.1 Game –Data members

1. Players
2. Score
3. createdAt
4. sorting
5. duration

4.1.2 Game – Methods

1. getPlayer()
   - Parameters : NA
   - Return Value : Returns an object of the User Class
   - Description : getPlayer() method is used to obtain the entire details of the user which are associated to the object of the GameClass.
   - Called By : method is called in the main program/activity.
   - Calls : method calls getName() and getAge() method of the User Class.

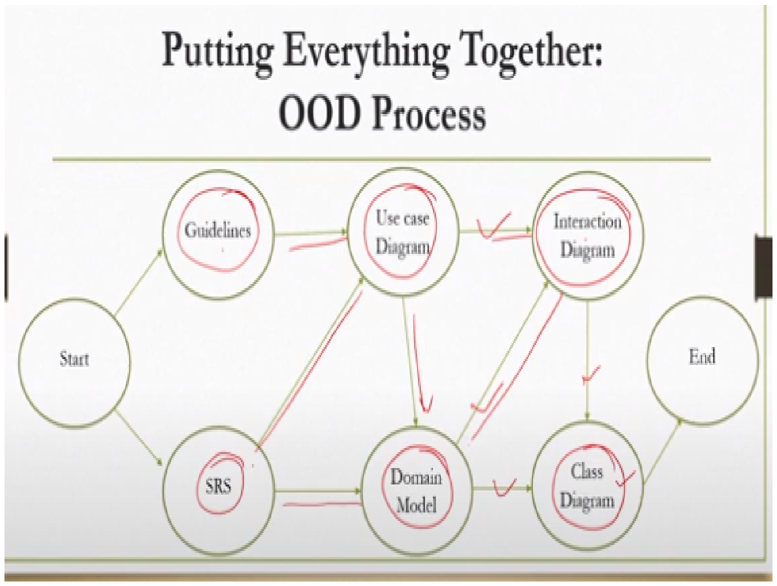
This is one example class diagram showing the structure of the system where we have used the rectangles to represent the classes class name and in this part we will keep the ideally we should keep the attribute and member function details as shown in the earlier example. Then we have these notations, various notations used multiplicity values range aesthetics even single numbers as shown.

So the relationship between classes with this aggregation composition and association notations are represented this is an again a hypothetical and imaginary system model represented with the class diagram. If there is a positive space in the diagram that means we are unable to keep the attribute and member function details within the rectangle then it can be mentioned separately as shown here.

Like for this game class these are the attributes and these are the methods with input, output details and the permission details, that can be also another way to represent the classes. And then in the diagram simply show the relationship between them both are okay. Here is shown that class details with members functions and they are explained here separately rather than doing the same thing here itself because there is less space.

So you can create this diagram and you can separately create these details to represent the classes. So that is in summary what we can do with class diagram, we can represent the classes and the relationships between them in terms of the 3 types of relationships association, aggregation and composition. So let us summarize what we have learnt so far.

(Refer Slide Time: 44:01)



Putting Everything Together: OOD Process

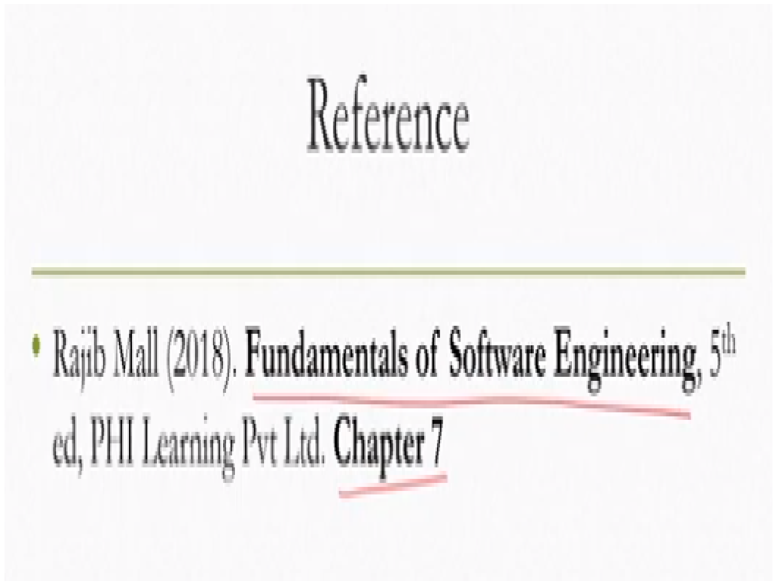
If we want to follow an object oriented design approach and come up with an object oriented design for a system we can use UML notations to represent the design. However to create the design we can follow certain methodology like we can follow guidelines or create SRS using requirement analysis, to come up with use case diagram. From SRS we can create domain model also or from the use case diagram we can create domain model.

Now again from use case diagram or the domain model we can go for interaction diagram in fact both we require to, create the interaction diagram use case diagram and the domain model. And again domain model and interaction diagram both can be used to create class diagram and the class diagram gives us the design of the system.

So we can make use of guidelines as well as SRS to go for use case diagram, use the use case diagram as well as SRS to go for domain model; use the domain model as well as use case diagram to go for interaction diagram. Use the interaction diagram as well as domain model to go for class diagram and that gives us the design of the system. So we can follow this approach to come up with a object oriented design of the system and use the UML notations to represent that design.

In UML we learned about 3 views user view, behavioral view, and structural view and 3 diagrams to represent those views. Namely the use case diagram to represent user view, sequence diagram to represent a behavioral view, and class diagram to represent structural view. All 3 together give us the overall system design. Remember that if we are designing or trying to express a design of a system we need all these views not a single view.

(Refer Slide Time: 46:18)



Reference

- Rajib Mall (2018). **Fundamentals of Software Engineering**, 5th ed, PHI Learning Pvt Ltd. **Chapter 7**

Whatever I have covered today can be found in these book fundamentals of software engineering you may go through chapter 7. I hope you could enjoy the topic that has been covered today in this lecture later on we will try to understand these concepts in terms of one case study looking forward to meet you soon in the next lecture thank you and goodbye.