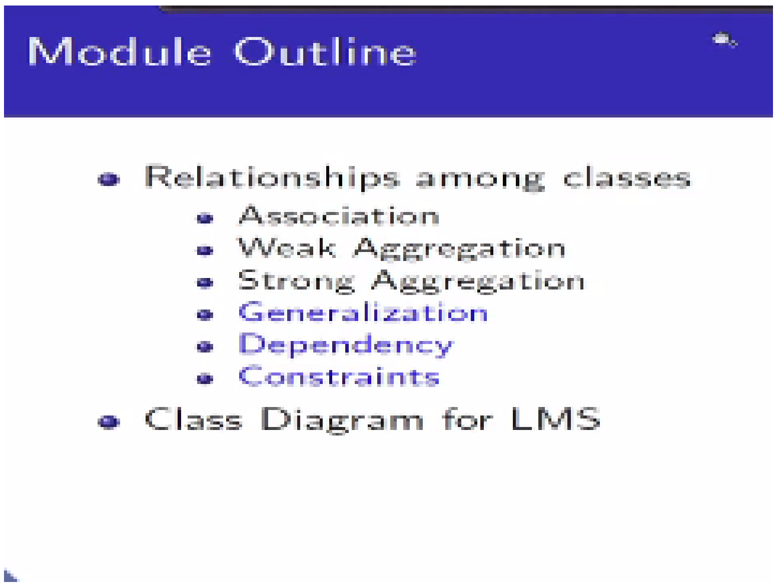
Object-Oriented Analysis and Design

Prof. Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology- Kharagpur

Lecture - 40
Class Diagrams – Part 3 (Generalisation, Dependency, Constraints)

(Refer Slide Time: 00:36)



Module Outline

- Relationships among classes
  - Association
  - Weak Aggregation
  - Strong Aggregation
  - Generalization
  - Dependency
  - Constraints
- Class Diagram for LMS

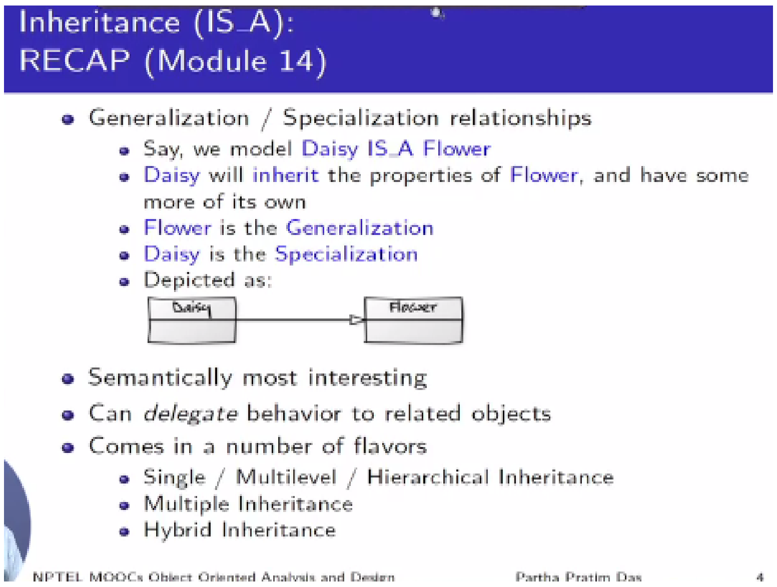
Welcome to module 28 of object oriented analysis and design. This module is the 3rd and concluding part of our discussions on class diagrams as UML diagrams. So in the earlier 2 modules, we have first studied the basics of class diagram, how do you represent class? And how do you represent properties of a class? The operations of a class, how to control the visibility of different members, the property that attributes.

How to specify different details behind this properties and operations like if a property is read only? What if a property is derived? and so on. Then we started discussing about this was in the last module, discussing about the different relationships that may exist between 2 or more classes and we observe that there are variety of relationships, spanning association, which basically relates to 2 or more classes in a certain way.

In any general form, a relationship can be viewed, first is an association; which has 2 ends that can be qualified, there is a reading direction, there is a navigability in the association which basically specifies us to how does one class relates to another. Associations mostly are binary but they could be ternary or N-ary in general. Then we take a look at different forms of aggregations.

We had earlier seen, 2 major aggregation structures possible among classes. One is the composition or component structure, where like car has 4 wheels, so wheels are composite the car, so this is has a relationship, which is in class diagram module has strong aggregation and we also saw a weaker version of the aggregation where you have certain class has 1 or more objects of some other class related to it, associated to it, to complete the functionality.

Library has multiple users, who use the library but they really do not, they are not really contained within the class in the sense of composition, so we designate them as weak aggregation and we have seen for an association, weak aggregation as well as strong aggregation, what are the notational details? The multiplicities, the MPN field dimers and all that, so in view of that, in continuation of that we will discuss 3 other kinds of relationships.
(Refer Slide Time: 03:39)



Inheritance (IS_A): RECAP (Module 14)

- Generalization / Specialization relationships
  - Say, we model Daisy IS_A Flower
  - Daisy will inherit the properties of Flower, and have some more of its own
  - Flower is the Generalization
  - Daisy is the Specialization
  - Depicted as:
- Semantically most interesting
- Can *delegate* behavior to related objects
- Comes in a number of flavors
  - Single / Multilevel / Hierarchical Inheritance
  - Multiple Inheritance
  - Hybrid Inheritance

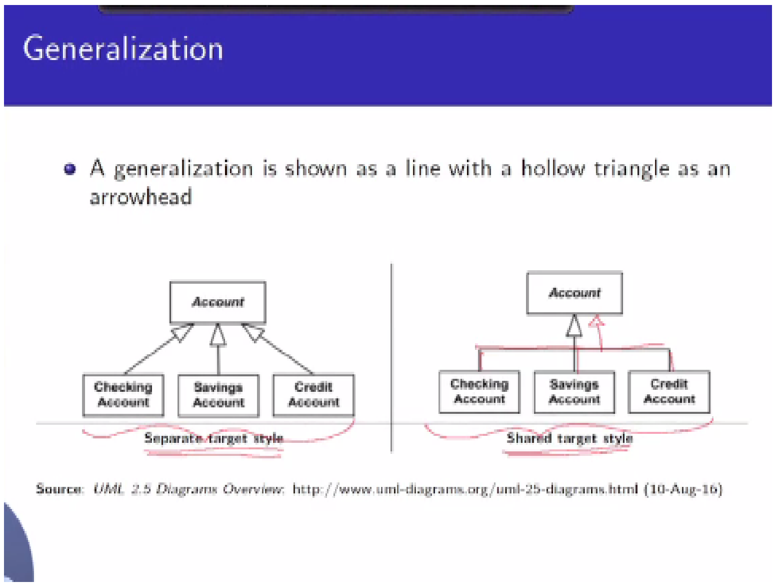
NPTEL MOOCs Object Oriented Analysis and Design Partha Pratim Das 4

The generalisation, the dependency and the constraints, these are further relationships that may exist between 2 or more classes. Besides that, in this module, we will also quickly take a look into our running example of LMS and try to improve on the class diagram of the LMS that we have been constructing so far. So first we start discussing about generalisation, I would like to remind you regarding what we had talked of in terms of generalisation several times earlier.

So it is a generalisation, specialisation relationship between any 2 abstractions that are possible. So when I say daisy is a flower, I have a daisy class or a daisy concept, I have a flower class, flower concept and by inheritance or generalisation specialisation, I mean that if daisy s a flower, then daisy has all the properties that the flower has but daisy may have some more additional properties, we say flower is a generalisation and daisy is a specialisation.

We had seen without actually formal introducing class diagram; we had seen the depiction in terms of a diagrammatic form. So I would ask that before you go through further of this module, please refer to the earlier modules where inheritance and these relationships have been discussed, so that you can understand this in stronger detail particularly you should look into various flavours of inheritance that are possible.

(Refer Slide Time: 05:00)



Generalization

- A generalization is shown as a line with a hollow triangle as an arrowhead

Source: *UML 2.5 Diagrams Overview*: http://www.uml-diagrams.org/uml-25-diagrams.html (10-Aug-16)

All of those flavours can be represented in terms of the class diagram. So this is the basic representation of generalisation, if I have should formally say it is; this is the generalised class and these are the specialised classes and we draw an arrow from the specialised class to the generalised class and the arrow head is an empty triangle which represents that it is indeed is a relationship.

So we read it as checking account, is an account like this, savings account is an account, credit account is an account and with this, what in semantically it means that account has some properties and some behaviour, some operations, for example, account level properties, in terms of a holder’s name, in terms of the account number, the account balance and so on, so being specialisations all of these specific types of accounts like checking account, savings account or credit account.

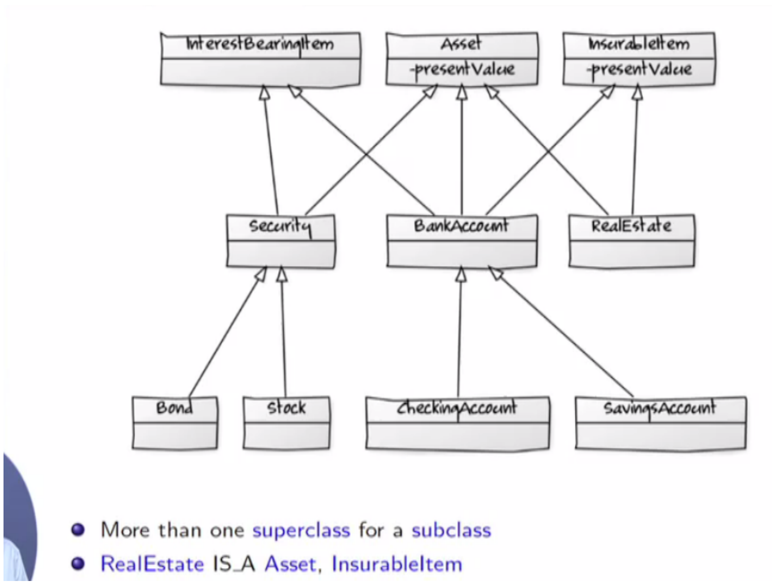
In Indian banking we also have term deposit account, we have current account and all those, they will certainly have all these properties that is the holder, the number, account number, balance, but in addition if I talk about specifically a savings account, then it will have certain further properties may be savings account has a restriction that I must have a balance of say 1000 rupees at the minimum in the savings account which may not be applicable in terms of the savings, checking account, it may not have that property.

Savings account may have that I can transact at most say 1 lakh rupees within a single day, but checking account may have unlimited options for transactions and so on. So these are different specialisations which used the properties that the generalised class has an adds further properties at their own level. So this clearly makes the depiction of the reality of real world.

So that whenever possible we would like to work with the generalised concept and this will certainly enhance the understandability as well as the reusability of the design. Now in terms of the diagram representation, you can say that we have 3 specialised classes of one generalised class and we can draw 3 separate arrows, if we do that then we say it is a, the drawing is in terms of a separate target style.

We could draw a common bar line and let all of this specialised classes hang from there and this bar in turn actually points to the generalised class, this is called a shared target style. Now in class, UML class diagram, both these styles are available depending on the context, depending on what is easier to join represent, you could choose any one of them from time to time.

(Refer Slide Time: 08:14)

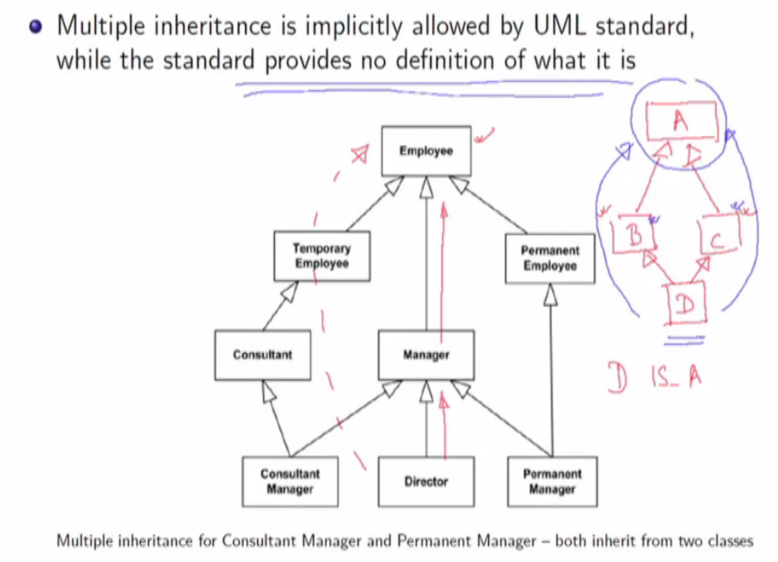


- More than one superclass for a subclass
- RealEstate IS_A Asset, InsurableItem

Naturally in terms of inheritance, you could have multiple inheritance, this is again from a slide from module 14, where we have drawn that a one bank account, this is the specialised class, this is specialised form interest bearing item, asset as well as insurable item. So it says that the interest bearing item has the property that I can earn interest on the deposit that I have.

Bank account has that property, bank account allows us to do that, so we have a specialisation on this. Similarly, the asset is having certain fund value which you can pledge to somebody or which you can use to take loan and so on. So bank account can be used for that purpose with the balance that I have. So bank account is also an asset, so here we see that the same class is a specialisation of more than one classes.

(Refer Slide Time: 09:44)



- Multiple inheritance is implicitly allowed by UML standard, while the standard provides no definition of what it is

Multiple inheritance for Consultant Manager and Permanent Manager – both inherit from two classes

Whenever that situation happens, then we say that we have a situation for multiple inheritance as this example shows that you would have multiple inheritance from 2 or more classes, here bank account has 2 generalisation security, I am sorry; bank account has 3 generalisations, whereas security has just 2 generations and so on. So certainly, we can represent multiple inheritance exactly using the same notation as a single inheritance.

So you can see here that director is a manager, manager is an employee and we have consultant manager and permanent manager. The consultant manager is a manager on one site, and is consultant on the other. So it gets bore this property, so being a manager it will get the rights of the manager, being a consultant it will get the flexibility that a consultant gets possibly it is not full time, possibly it gets paid according to how much his work and so on.

On the other hand, there could be permanent managers, who are managers again get the same manager rights, but is the permanent employee, so the permanent manager also inherits the permanent employees benefits like retirement benefits, like leave, like fixed salary and so on. So multiple inheritance is a very important features which is often used in modelling different real life scenarios and place a critical role in terms of the class diagrams.

Now one point that I would like to highlight is the multiple inheritance as such is implicitly allowed by the UML standard, in the sense that since you can draw from one specialised class, you can draw multiple relationship arrows to other classes, so the multiple inheritance gets represented in a class diagram, but as such the standard has not try to provide any specific definition that if I have multiple inheritance and this is what the inheritance would mean.

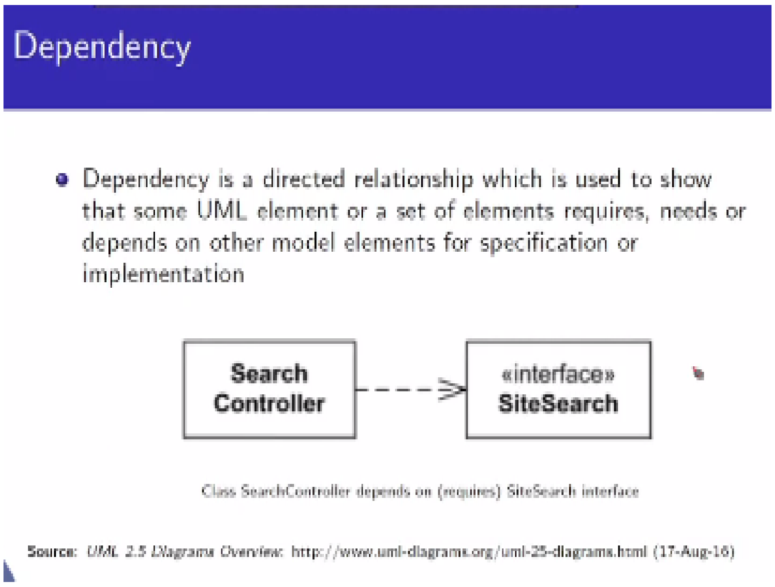
Particularly there are certain critical situations like if I have a class like this and I have 2 classes specialised from that, this is fine, but then I have one class which multiply inherits from these 2 classes, A, B, C and D, so D multiply inherits from B and C both. But B in turn is the specialisation of A, C in turn is the specialisation of A. The question is certainly if I use the transitivity of the inheritance, that is transitivity means, director is the manager, one generalisation.

Manager is the employee, the other generalisation, so there is an implicit transitive inheritance between the director and the employee that is the director has inherited all the properties of an employee through the manager, so if I use that logic here, then D actually inherits the properties of A through B as well as the D inherits properties of A through C. Now does it mean that some of the properties will be inherited twice?

Some of the operations may have been redefined in B, may have been redefined in C, then what will happen to those operations? How will D interpret those operation, because it put inherit the same operation from B and a C, because they existed in A, the original route class, but it may be inherited in 2 different forms, so what should D do in terms of defining its own operations, own properties?

So there are issues in terms of how the multiple inheritance should be interpreted and you should keep in mind that while it is implicitly allowed, standard has not provided any specific definition further, so you will find that a multiple inheritance model as created in UML class diagram, when you try to take it to the OP an object oriented programming language like say C++, then specific OP language will had specific interpretation, specific semantics for multiple inheritance.

(Refer Slide Time: 14:12)



Dependency

- Dependency is a directed relationship which is used to show that some UML element or a set of elements requires, needs or depends on other model elements for specification or implementation

Search Controller
«interface» SiteSearch

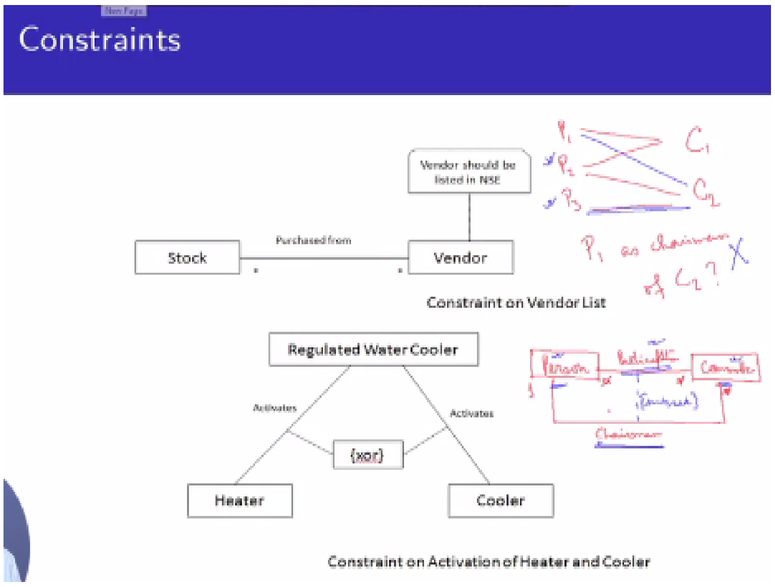
Class SearchController depends on (requires) SiteSearch interface

Source: *UML 2.5 Diagrams Overview:* http://www.uml-diagrams.org/uml-25-diagrams.html (17-Aug-16)

So you will further have to understand what is the semantics for multiple inheritance for a specific language and then use that in the implementation model, so some cleared is required to actually model with multiple inheritance and then take it forward in terms of building up certain actual implementations. Another form of relationship that is possible is known as dependency.

Dependency is somewhat represents the little bit of a different concept. It just says that between 2 classes, this one class depends on the other, one class is affected by another, one class may be using another. So unlike association which tells you that specific messages at intended to be sent through association or when we talk about aggregation we are basically talking about strong or weak compositional aspects of the model.

When we are talking about generalisation, we have certainly talking about a very strong structural similarity between the classes. In dependency, we more or less just try to highlight that how does one class may be effecting another? Or may be dependent on other and that is the typically drawn with dotted arrow at the arrow head at the end and depicts that this class on the left requires the other class or depends on the other class.
(Refer Slide Time: 15:30)



Yet another form of relationship could be expressed in terms of constraints, so constraints for example, let say there are 2 forms that we have shown here, so we are saying there is a class stock and there is a class vendor. So basically it is a scenario of procurement, so stock is the inventory and vendor is the provider from whom we can buy and build up the stock, build up the inventory.

So you have an association purchased from the direction, the reading direction is not shown, it is clear from the way it is written, so the reading direction is like this, the multiplicities are anything, so any item can be purchased from any vendor, you may not purchase from anybody also. Given this, we need to also specify that well we will not purchase from actually vendor availability in the country.

We want to purchase only from the vendors who were listed in the national stock exchange, so we represent that by with the dotted line and this is kind these are typically called notes, I am sorry. Let me write it again. These are typically called notes with this corner cut, so this kind of a note attached here which say that well in this relationship the vendor may actually contained several vendors who are not listed in this.

They may be related to the other classes in the system and for that the reason may be there, but for this particular relationship, the vendor would need to list in NAC, so you are putting some kind of a restriction, some kind of a constraint on that. Similarly, let us say we have a regulated water cooler, so it is a regulator water cooler means it is a water cooler which tries to set a certain some temperature and maintain that.

So internally it will have a heater and a cooler, if the temperature has to go up, the heater has to be put on, if the temperature has to go down, the cooler has to put on. So represent that we have a association showing activates. So this activates relation is applicable for the heater as well as for the cooler, but the reality of the situation would be that you never conceive that you have a state of the regulator water cooler, where both the heater as well as the cooler is working.

You can conceive of this in terms of particularly car air conditioners, or room air conditioners usually do not have the heater path, unless you are in an appreciation air conditioning system, but in a car air conditioning, you will always find that there is a dial like this which can be rotated and it show that there is a blue part usually on this end and there is a red part usually on this end.

The blue part designates that your cooling, the red part designates that your heating. Obviously we will not do both of them together, so how do you represent that in the class diagram, so you, one profusion is you have an XOR relationship, exclusive OR, which means that the heater can, heater activates regulator water cooler is valid, cooler activates the regulate water cooler is valid, but both cannot be valid at the same time.

This exclusive R, XOR is exclusive OR, usually OR means inclusive, but it could be either this or this or both, both exclusive OR means that both cannot have. So these are different forms of constraints. For example, I could have another constraint say let us say I have a person and let us I say have a committee, so basically forming a committee from a group of persons.

Naturally the association could be something like participation, okay, say I can have ***, that is any person number of person can participate in any number of committees. Then in every committee, there is a chairman, so I can model that again I have a committee, I have a chairman, so my relationship is chairman. Naturally I would expect that every committee has one chairman and a person could be chairman of any number of committees including the person may not be a chairman at all.

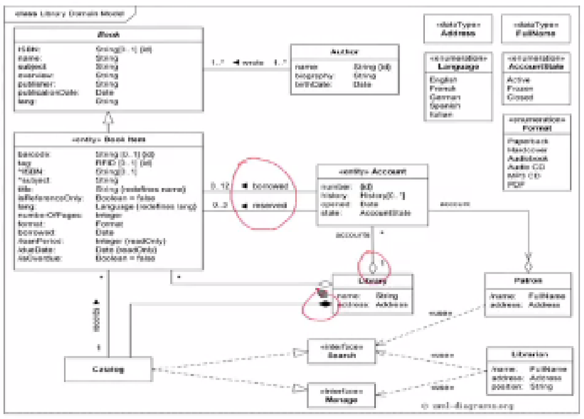
Now if I just modulate like this, then it will be quite possible that if I have a committee C, and I have persons P1, P2, P3, I have another committee say 2 committee C1, C2, let us say that, the participation is like this. They participate here and they say participate here, now can I have P1 as chairman of C2, can I have that is, if I say my blue is the chairman link, can this association be possible? P1 is a chairman of C2?

An other to be absent, right, this person does not participate in the committee but how can he cheered the committee, so this infeasibility have to be represented we again use constraint for doing that. For example, the way this can be denoted is you can join these 2 associations with the dotted link and write subset, so this is the subset constraint, which say that the chairman, the chairperson association has to be subset of the participation association.

Which means that if there is a relation between a person, a person and a committee, as a chairman then that combination, that doublet of the person and the committee must be a member of the participation, so this will be ruled out, so if C2 has to be cheered, it have to be either P2 or P3, so if I say that this is what I have P3 is a chairman of C2, then that it will be valid, but P1 being the chairman of C2 is not valid.

(Refer Slide Time: 22:31)

Library Domain Model



Source: *UML 2.5 Diagrams Overview*: http://www.uml-diagrams.org/uml-25-diagrams.html (17-Aug-16)

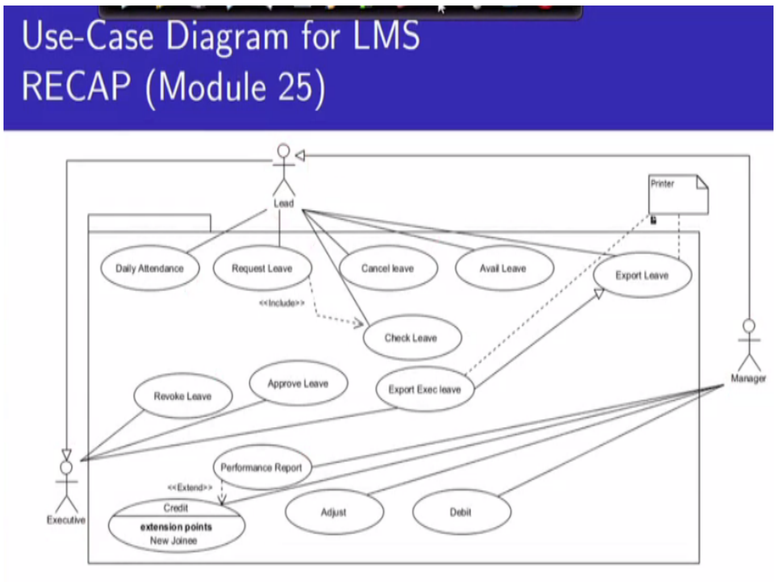
NPTEL MOOCs Object Oriented Analysis and Design Partha Pratim Das

So in this way, different forms of constraints will have to be considered, so that we can model that reality in am more precise manner. So this is just you know take in a look into the Library domain module, we earlier looked at this in terms of the let me go back to that in terms of the properties, now you can see the various kinds of other relationship for example, we conceive a generalisation relationship here.

A book item is a book we can see different this is say is a librarian users, librarian manages the library, librarian searches the library, the patron, here means the user, searches the library. So these are like search manages these are like interface classes, which basically give you different operations that you can do as a part of that class, so how does the way patron relate to the such interface is through this users dependency.

So there is used dependency between so the patron need to use the such interface to be able to perform the search. So this is how all this can be done, this we have already seen, these are the different associations, this is the weak aggregations, strong aggregation, I am sorry, yes; weak aggregation, this is the strong aggregation and so on, so these are all different types of relations that we can have.

(Refer Slide Time: 24:20)

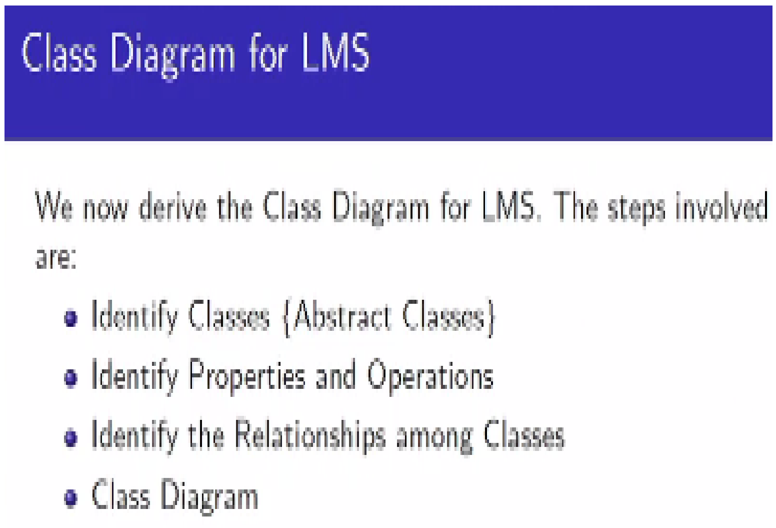


Use-Case Diagram for LMS
RECAP (Module 25)

You can use this particular on the same example, here it is annotated, so you can clearly see the different kind of usage dependence is interface realisation which is an either kind of dependency and so on. Now so that is all about the relationships among class diagrams so we can quickly go back to our leave management system and try to advance the class diagram, the UML diagram for the leave management system, by little bit more.

So I just pick up the use case diagram here done in module 25, so we know that these are the different actors we have already identifies, so naturally they will transform themselves into classes in the design in the class diagram design and we have several different types of use cases of how we use and what is not explicit in the use case diagram is certainly all of these, all majority of this use cases need some other abstraction which is of leave.

(Refer Slide Time: 25:27)



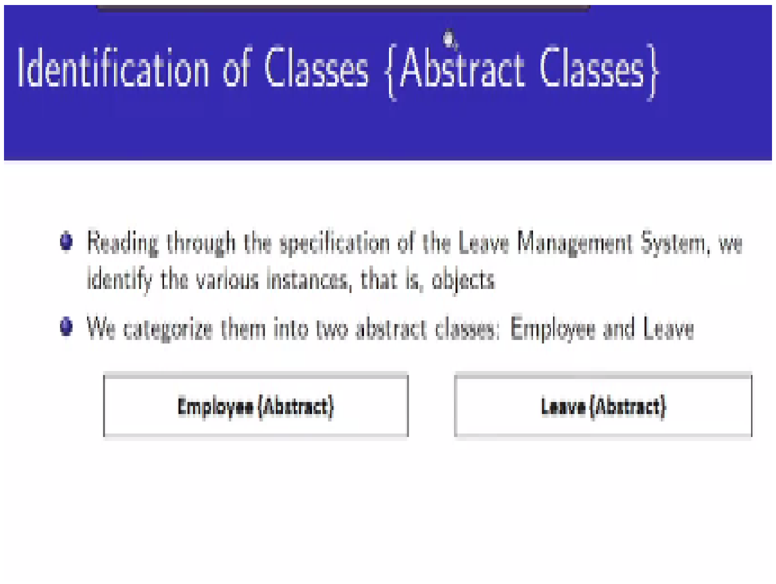
Class Diagram for LMS

We now derive the Class Diagram for LMS. The steps involved are:

- Identify Classes {Abstract Classes}
- Identify Properties and Operations
- Identify the Relationships among Classes
- Class Diagram

Because when you apply for leave, when you request for leave, or when we approved leave we need that abstraction. So now we will see how we got those in terms of a consolidated class diagram. So in terms of the class diagram, the tasks are 2, identify classes, often you start with identifying abstract class, identify the properties and operations, identify the relationships and finally combined them all together, in terms of the class diagram.

(Refer Slide Time: 25:55)



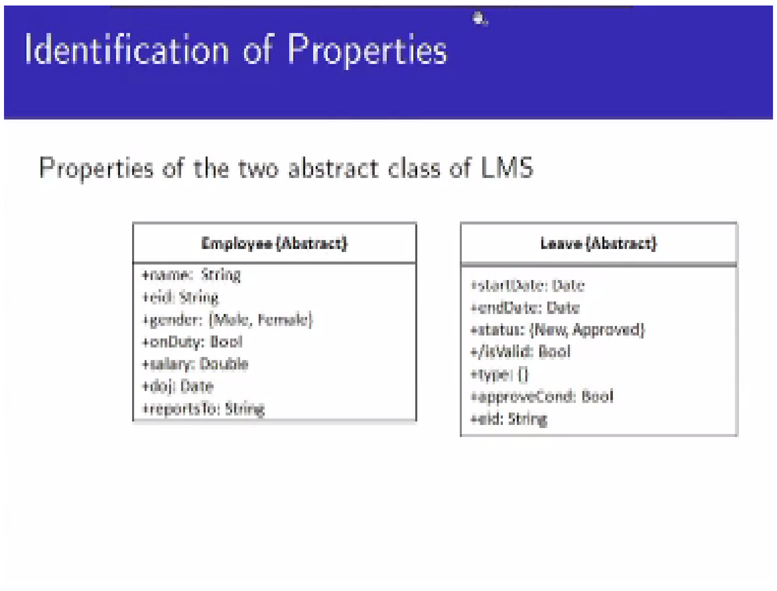
Identification of Classes {Abstract Classes}

- Reading through the specification of the Leave Management System, we identify the various instances, that is, objects
- We categorize them into two abstract classes: Employee and Leave

Employee {Abstract}
Leave {Abstract}

So these are task, which we have been performing all through the past several modules and so we can just quickly glance through the result of that, certainly in the notation of class diagram or main abstract classes, there would be any more, but main abstract classes are employee, which we say is an abstract class, leave is an abstract class, because you can see that there are several specialisations of this which we have separately discussed earlier that will result, so these are the 2 basic concepts.

(Refer Slide Time: 26:39)



Identification of Properties

Properties of the two abstract class of LMS

Employee (Abstract)

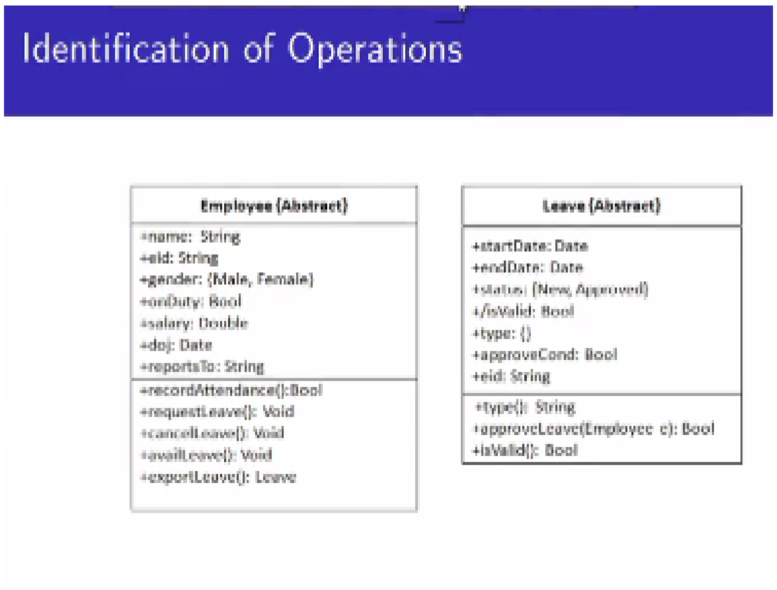
```
+name: String
+eid: String
+gender: {Male, Female}
+onDuty: Bool
+salary: Double
+doj: Date
+reportsTo: String
```

Leave (Abstract)

```
+startDate: Date
+endDate: Date
+status: {New, Approved}
+/isValid: Bool
+type: {}
+approveCond: Bool
+eid: String
```

Beside that, there would be several other classes, like there would be a class for the (()) (26:22), there is a class for the documentation, there could be class for (()) (26:27) actors in the system like doctors who prescribed different documentation and so on. So with that we will try to complete the or; try to make a complete list of the identified classes. For classes that we identify, we have different attributes, and we had seen the extraction of these earlier.

(Refer Slide Time: 27:02)



Identification of Operations

Employee (Abstract)

```
+name: String
+eid: String
+gender: {Male, Female}
+onDuty: Bool
+salary: Double
+doj: Date
+reportsTo: String
--------------------------
+recordAttendance():Bool
+requestLeave(): Void
+cancelLeave(): Void
+availLeave(): Void
+exportLeave(): Leave
```

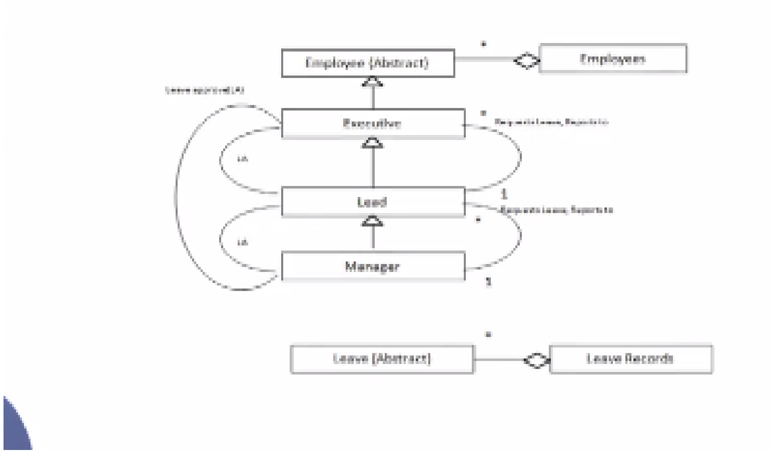
Leave (Abstract)

```
+startDate: Date
+endDate: Date
+status: {New, Approved}
+/isValid: Bool
+type: {}
+approveCond: Bool
+eid: String
--------------------------
+type(): String
+approveLeave(Employee e): Bool
+isValid(): Bool
```

So for these abstract classes, which we can add these attributes, these properties and we can keep on adding more and more, I am not saying that these are the exhaustive list and I am just indicative one, which you can see how to put together. Then you on top of this attributes or properties, you have add different operations that that class do, so we had started with out of these 3 compartments you started first with this, which is the mandatory.

(Refer Slide Time: 27:32)

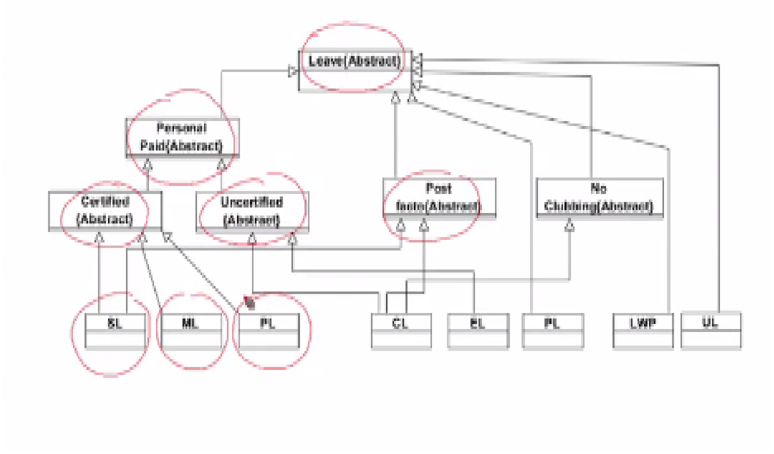
Identification of Associations



The class identity, then the attributes, properties, then the methods of operations, so this becomes a more completed class representation for the employee as well as this one is for the leave. So some of the associations again we had earlier identifies requires leave report to so, the different use case will often see, the different use case is actually manifest themselves as in terms of operations in classes.

(Refer Slide Time: 28:04)

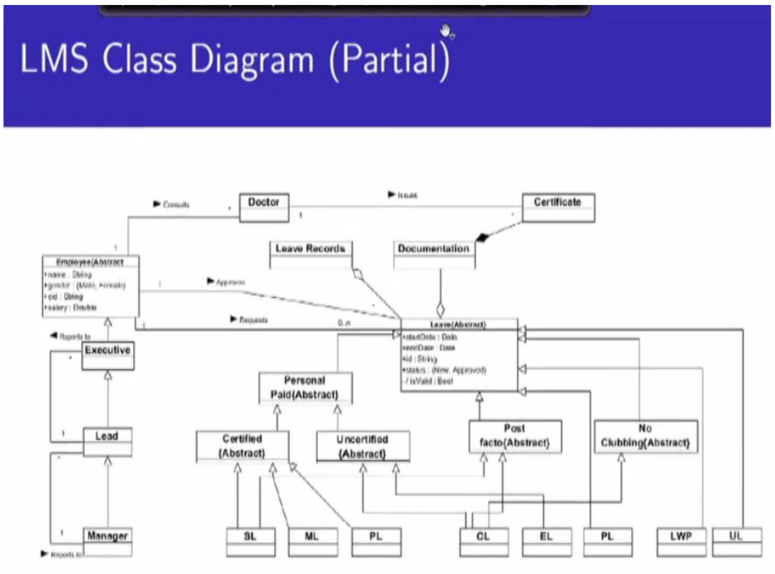
Identification of Generalizations



Because the classes, different classes will have to actually provide that service to other object or they manifest actually in terms of different association that exist between the classes because that is how the classes interact. Then we have seen the generalisation specialisation so in terms of the formal representation this is what to have the leave abstract leave and we had earlier shown this.

We have personal or paid leave, we had post factor leave, certified leave, uncertified leave, all of these are basically abstracts because you the organisations does not as such have something called as the deferred leave or you cannot directly apply for it as the desired leave, what you apply for other at the leaf level, the different concrete class instance that are possible.

(Refer Slide Time: 28:46)



So this diagram now fits into our class diagram notation, so I if you put things together and I remained you this is just a partial diagram, this is not a complete one, we will share the complete one as a solution of LMS, so here you can see the actors have come into one hierarchy, which was originally captured in the use case diagram but in addition to that there is an association identified.

Which is report to executive reports to the lead with the multiplicity of one on the lead side and any on the executive side which mean that many executives could report to a single lead and it is possible that at a point of time, a lead does not have any executive reporting to us. Similarly, the similar relationship exist association exist between lead and a manger. We could identify several others.

Here we have the abstract leave class on this side as a whole we have this whole hierarchy, generalisation hierarchy from the leave which will all actually inherit the semantics, association that the leave makes, so the critical is the employee extraction and the leave extraction, say as an association request, which is request leave. So we want to mean that the employee request leave, so employee has to initiate that and leave will get requested.

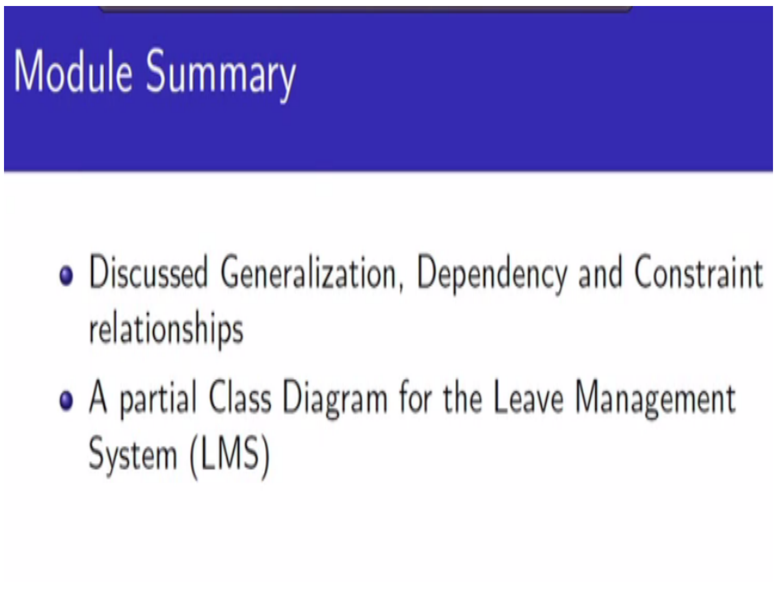
As a multiplicity of one on this side, and the multiplicity of 0 to n, 0…n, on the other side. because if you see that any leave will have an unique employees who has applied for it. So on employee side, the multiplicity is singular but on the leave side, an employee may not request for any leave, so the association could be backwards, or could request for 1 leave, 2 leave, different kinds of leave and soon.

So this is the multiplicity on here and based on that it will reflect as you propagate down the relationship on the generalisation hierarchy, so this association on the generalisation hierarchy, will get different kinds of specialised form of this association but the most fundamental is this association between the 2 abstract classes and the other association of approval between the 2 abstract classes.

When we say this you can see that between this approves and reports to between these 2 associations, you need to set some constraints. I have not done that, just left it opens so that you could take it up and complete the diagram further because certainly the leave of an employee is approved by another employee that is because it is an abstraction, so an executive is an employee, so a lead is also an employee.

So if you read it that way, then the approval basically means an employee is approving a leave which belongs to yet another employee. Now certainly what will lead to rule out, you lead it to rule out that in this approval process, it is possible that if the executive reports to the lead, only then the lead approves the leave for that executive. The executive will not approve leave for other, I am sorry; the lead will not approves leave for other executives who do not report to that particular leave.

So between whenever an approval has to happen then that reporting relationship will also have to be valid. So that is a constraint, I will leave it, you discovered more and more associations, aggregations, and generalisations, possibly generalisations has been extensively done already, the constraints and so on and try to complete this diagram, let later point of time will come up with the solution also.
(Refer Slide Time: 32:40)



Module Summary

- Discussed Generalization, Dependency and Constraint relationships
- A partial Class Diagram for the Leave Management System (LMS)

So this conclusion on discussion on the class diagram as a UML diagram and now you should be in a good position, to be able to represent any system prescription in terms of a class diagram. I would request that for that the assignment management system, the MS system, you take it as your exercise to carry out the class diagram and get convinced that you are understanding all of those very well.