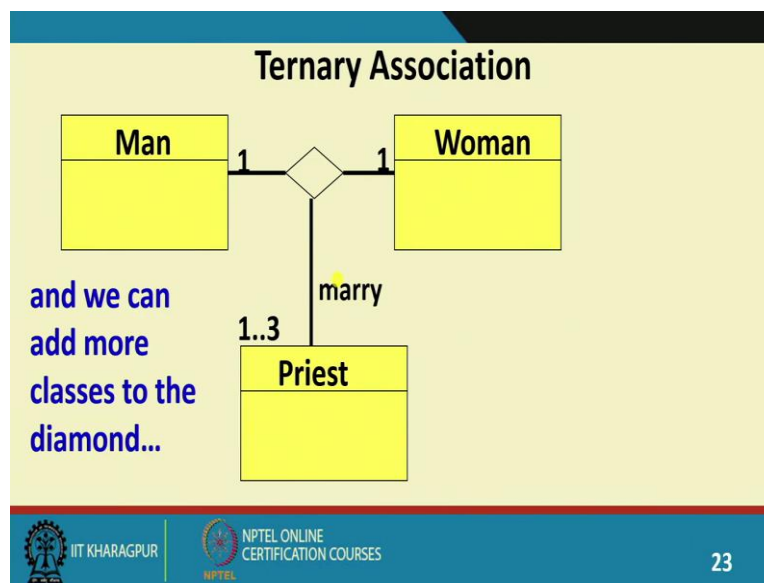


**Object – Oriented System Development Using UML, Java and Patterns**  
**Professor. Rajib Mall**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Kharagpur**  
**Lecture 13**  
**Qualified Association**

Welcome to this lecture.

In the last lecture, we are just started discussing about Ternary Association.

[Refer Slide Time: 0:24]

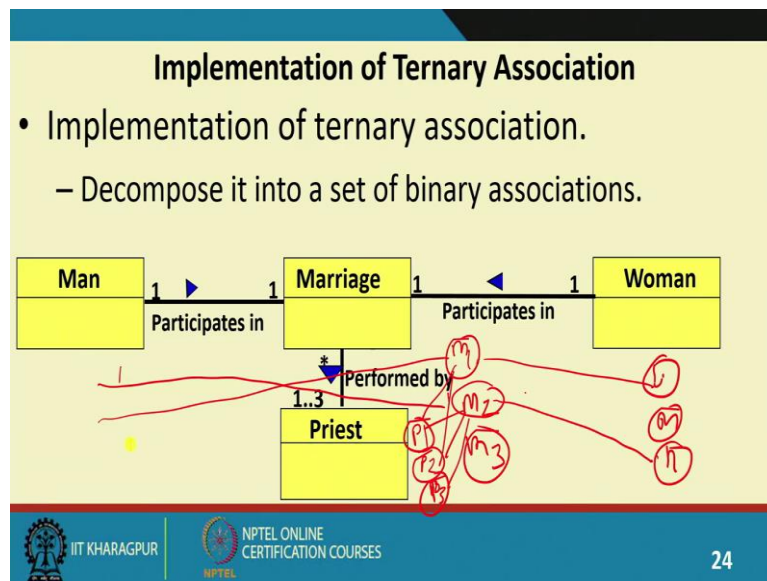


We have so far discussed about the association class, the Unary and Binary association. Those are very common. But occasionally, we also need to use the Ternary Association, Quaternary Association etc. which is less common but in many modelling problems, you might see Ternary Association that 3 classes are associated. Generally, using a diamond symbol we represent ternary association.

In the above slide example, we can see name of the association is 'marry'. So, the name of the association is marry and man marries woman, marriage performed by up to 3 priests. We can have Quaternary Association, where we can add another class to this. For example, we can add another class 'Witness'. The marriage was witnessed by specific persons. So, we can have Witness and '\*' is the cardinality in Witness side.

But now the question is how Ternary Association implemented?

[Refer Slide Time: 1:57]

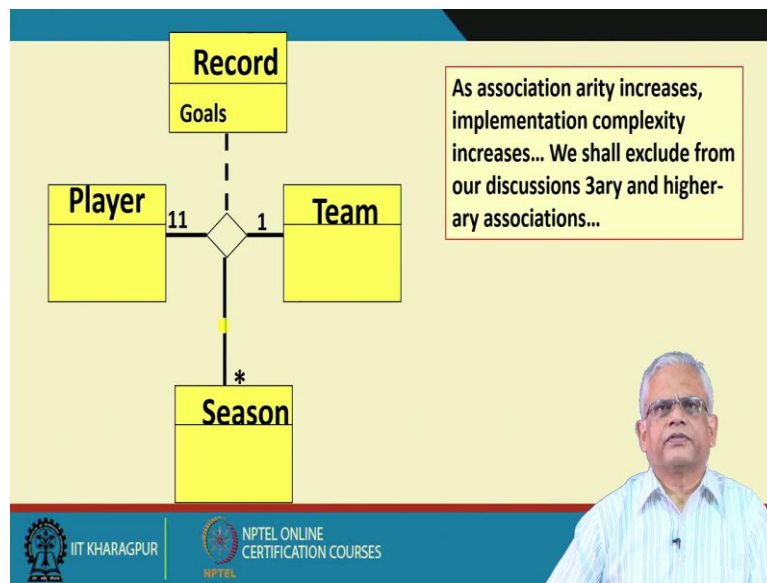


To implement the Ternary Association, we decompose it into a set of binary associations (in the above slide). For the example, we discussed just now, we have this name of the association ‘marriage’ which is made into a class here. marriage and on the object for the man class has 1-1 link. And woman to marriage, has 1-1 link. Priest and Marriage has 1..3 to ‘\*’ link.

Now, we can have several objects under man let say A, B, C and we can have women, L, M, N (As shown in above slide with ‘red link’). And we have different marriage objects created: m1, m2, m3. Let say A married N. And we have 1-1 link, with the m2 marriage object. m1 marriage object, is between let say B and L. Then, let say there are 3 priests objects: P1, P2, P3. And we will have links between Marriage and the Priest. For example, May be two priests married off B and L. So, there will be link to 2 priests here. So, that will be the object diagram for this. So, this is the way we can implement object diagram.

We have so far discussed about implementation of unary and binary association. We can very well implement that. Now let’s look at the quaternary association.

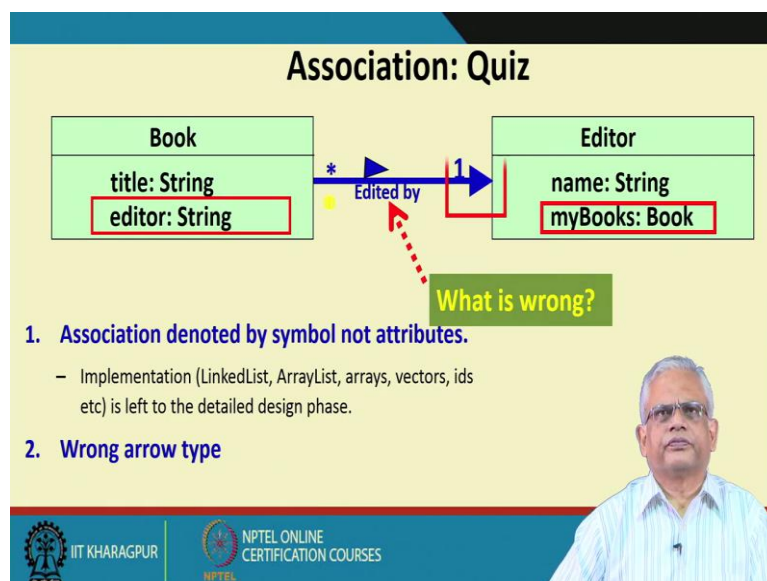
[Refer Slide Time: 4:52]



Here is the quaternary association (in the above slide). A football team has 11 players. And the team and the player association is 1 to 11 that is one team has eleven players. The players play in many seasons but the players for different seasons may vary. For the same team, may be different players play in different seasons. But given a team and the set of players that play there, for the different seasons, we have different records of goals.

We would like to maintain all these. And then we will use a quaternary association and we will implement this, in the form of a 4-binary association. Since these are not very common, we will not spend time on this.

[Refer Slide Time: 6:11]



Let's move on to the other types of relations. But before we move on, let's have a small quiz. Let say we represented an association relationship between a Book class and Editor class (as shown in the above slide). Title, editor are the attribute of the Book class and we have the name and myBooks attribute in the Editor class. And we represent the association relation with the name of the association 'edited by'. The same editor can edit multiple books. Now, the quiz is, given this situation, please identify are there any notational problems in this UML class diagram?

We have so far looked at the class diagram, involving association relationship, binary, unary, ternary and so on. Let's identify what is the mistake in the modelling here. Because if we know the mistake, we will not commit those mistakes. The first mistake here is that, these are implicit attributes. And we do not explicitly write here, the editor is an implicit attribute of the book. We do not explicitly write that. Once we implement, those will appear in the code. The second problem is the arrow type is wrong, it should not be a filled arrow, it should be an open arrow. So, these are 2 major problems here, in this association class diagram.

[Refer Slide Time: 8:40]

**Qualified Association**

- A qualified association allows us to express uniqueness
- Implemented by hash tables, maps, dictionaries.
- How to read?
- There exists upto one file for each instance of filename in the directory .

The slide contains three UML diagrams illustrating qualified associations:

- A simple association between **Class** and **Student** with the association name "belongs" and a multiplicity of "\*" at the Student end.
- A qualified association between **Class** and **Student**. The association is labeled "1" at the Class end and "0..1" at the Student end. A pink box labeled "Roll" is attached to the Class class, representing the qualification.
- A qualified association between **Directory** and **File**. The association is labeled "1" at the Directory end and "0..1" at the File end. A pink box labeled "file name" is attached to the Directory class, representing the qualification.

The slide also features a portrait of a man in the bottom right corner and logos for IIT KHARAGPUR and NPTEL ONLINE CERTIFICATION COURSES at the bottom.

Now, let's look at Qualified Association. Sometimes, in an association relation, we would like to be more expressive, than what we have been doing so far. Let's just look at that. Let us say, we have a simple association between a class and a student. The student belongs to a single class, but the class can have multiple students. The binary association is simple. But let say, we want to be more expressive.

We want to express that, there exists up to one student, with a given roll number in a class. Given the roll number, we can uniquely identify the student of the class. Or a directory has many files, but given a file name, there can be at most 1 file in the directory.

Let's first see how to read this. Then we will see slightly more details of this.

In the qualified association (in the above slide, file and directory example), There is up to 1 file for each instance of the file name in the directory. There can happen that file name exists but there is no file associated with it: it is dummy. But then, given a file name, you can uniquely identify the file. There can be at most 1 file, with that file name.

Now another example (in the above slide class and student example), given a class, for a given roll number, there can be at most 1 student with that roll number. May be for a roll number, there is no student. But then, given a class, there exists up to 1 student, for a given roll number or each instance of the class, we will have at most 1 student with a given roll number. Let me just read again, (follow the way of reading) for first example, there exists up to 1 file, for each instance of the file name in the directory and for second example, there exists at most 1 student, for each instance of roll number in the class.

[Refer Slide Time: 11:57]

**An Example**

- Consider an order has many Order lines.

Class Diagram 1: **Order** (1) — **Has Line item** — (\*) **Order Line**

- Let us express: There is at most one Order Line in the Order for each instance of Product.

Class Diagram 2: **Order** (1) — **Product** (1) — **Line item** — (0..1) **Order Line**

The slide also features a video inset of a man in a light blue shirt and a footer with logos for IIT KHARAGPUR and NPTEL ONLINE CERTIFICATION COURSES.

Now, let us look at another example (in the above slide). Let consider that an order for a company has many order lines. Each order line identifies the product that has been ordered for the company and the quantity and so on. A class diagram shown on above slide for the same. For this, an order has many order lines. We just drawn here Order as the class and

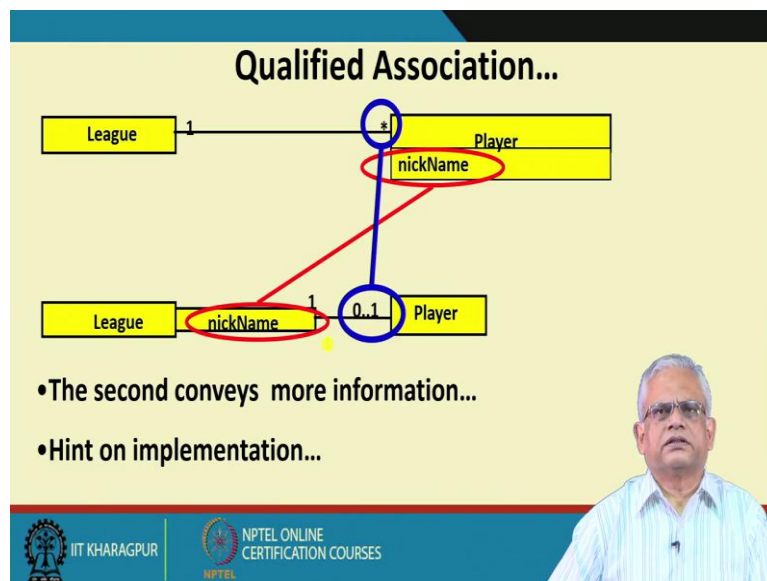
Order line as another class and each order consists of many order lines. Each order has a line item and multiple line items.

Now, we want to be more expressive. We want to express that there is at most one Order Line in the order, for each instance of the product that is ordered. That means that there will be no two instances of the order line, for the same product. Given a product, there is at most one Order Line in the order. How do we express that?

We will use the qualified association. See here, the '\*' in Order Line side has become 0..1 when we expressed as qualified association.

In the normal binary association, it is understood that, an order has multiple Order Line. But then, just see here in the qualified association, in the order we have at most one Order Line. There is at most one Order Line in the order, for each instance of the product or given a product, there is at most one Order Line in the order.

[Refer Slide Time: 14:11]



If we observed the above slide that given an association like '\*' at player side becomes 0 to 1 when we expressed as qualified association. The '1' which uniquely identifies the object instance on the '\*' end. For example, a league has many players and the player is uniquely identified by the nickname. That is in qualified association will be given a nickName, there exists at most one player in the league.

As you can see, that this is more expressive. Compared to the other diagram, we are expressing here, that given a nickName, there exists at most one player in the league. That is



no two players can have the same nickname. So, we can understand during implementation we should avoid somehow, that there are no duplicates in the nickname exist in the implementation. We should preclude or avoid any chance of having duplicates. But how do we do that?

[Refer Slide Time: 15:39]

The slide is titled "Qualified Association: Implementation". It features a UML class diagram and two code snippets. The UML diagram shows a class "League" with a "nickName" attribute, associated with a class "Player". The multiplicity is "1" for League and "0..1" for Player. The "League" class code includes a private Map of players, an addPlayer method that checks for the existence of a nickname before adding a player, and a private League attribute in the Player class.

```
classDiagram
    class League {
        nickName
    }
    class Player {
    }
    League "1" -- "0..1" Player
```

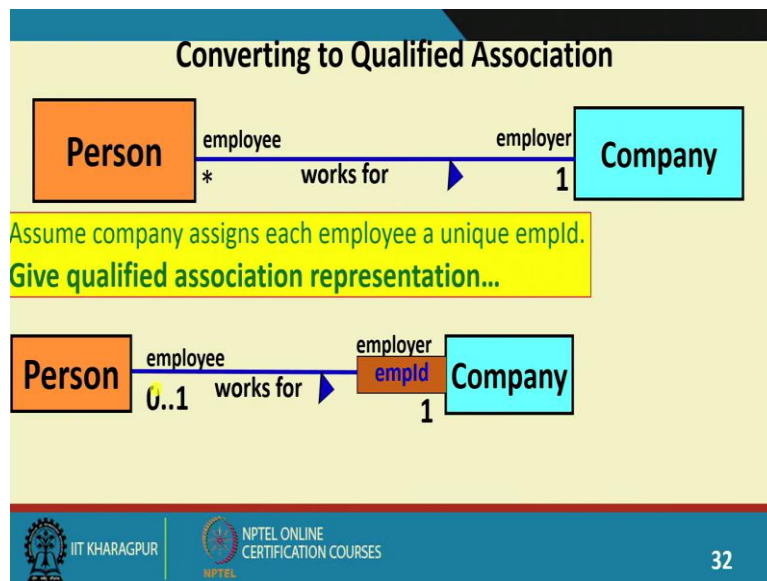
```
public class League {
    private Map players=new HashMap();
    public void addPlayer
    (String nickName, Player p) {
        if(!players.containsKey(nickName))
            players.put(nickName, p);
    }
}

public class Player {
    private League
    league;
}
```

Let see the implementation part here (in the above slide). To implement a qualified association, we use a map that is a collection class. Given a key, it uniquely identifies the object and we can ensure that the keys are unique. So, to implement this situation, that given a nickname, there exists at most 1 player in the league, on the league class, we need to keep track of all the players on the league. So, we use a map here. And now, once we want to add a player to the league, with a given nickname, we use 'addPlayer' method with the parameter nickname and the Player object. The Player object and the corresponding nickname, we first check whether the nickname exists or not using 'players.containsKey()'. If there is no player with a given nickname, then we add that player using players.put().

If there is a nickname exists already, we will not add that and this ensures that there is at most one player with a given nickname. For the player class, we don't have to do anything special, there is just a simple association, each player plays in a league. So, we keep track of the league that he plays.

[Refer Slide Time: 17:46]



Now, let's just do this exercise (in the above slide).

A company employs many persons. And the role here, company is the employer and person is the employee. Or we can also read here, person works for a single company. But then, let's try to be more expressive. We want to express that given an employee id, the person can be uniquely identified. That is, for a given employee id in the company, there is at most one employee in the company. We will use the qualified representation. On the company end, we will have the employee id and we have 0..1 here on person side instead of '\*'. And given an employee id, there is at most one person in the company with that employee id.

Refer Slide Time: 19:17]

### Qualified Association

Bank (1) — accno — 0..1 Account

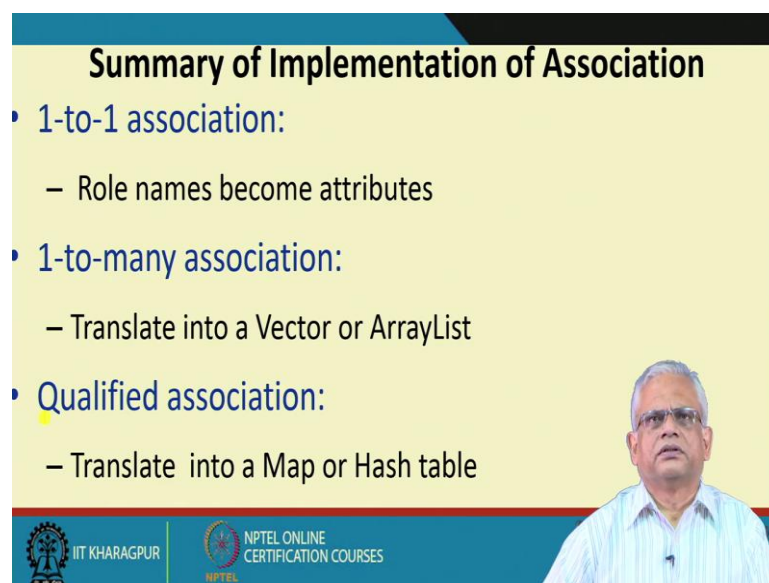
- Qualifier hints at setting up efficient access to linked objects:
  - For example, access accounts based only on the account number;
  - Also to avoid a linear search through all accounts.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES



There are other examples (in the above slide). While solving a problem, we might come up with more situations, where we need the qualified association. A bank has many accounts. But given an account number, the account is uniquely identified. So, here it is more expressive than the bank has many accounts and each account has account number. But we want to be more expressive, saying that given an account number, there is at most one account in the bank with that account number. And to implement the qualified association, we will use map. We will check once the account number is generated, while adding an account, we will check whether already such an account number exists or not. And also, since the implementation is a map, while accessing account using the account number, the access mechanism is very efficient, because it is not like an ArrayList, where we need to check through all the accounts in the bank. There may be thousands of accounts, we do not have to do a linear search or something. We just use the map and given the account number; we automatically identify the corresponding account.

[Refer Slide Time: 20:59]



**Summary of Implementation of Association**

- 1-to-1 association:
  - Role names become attributes
- 1-to-many association:
  - Translate into a Vector or ArrayList
- Qualified association:
  - Translate into a Map or Hash table

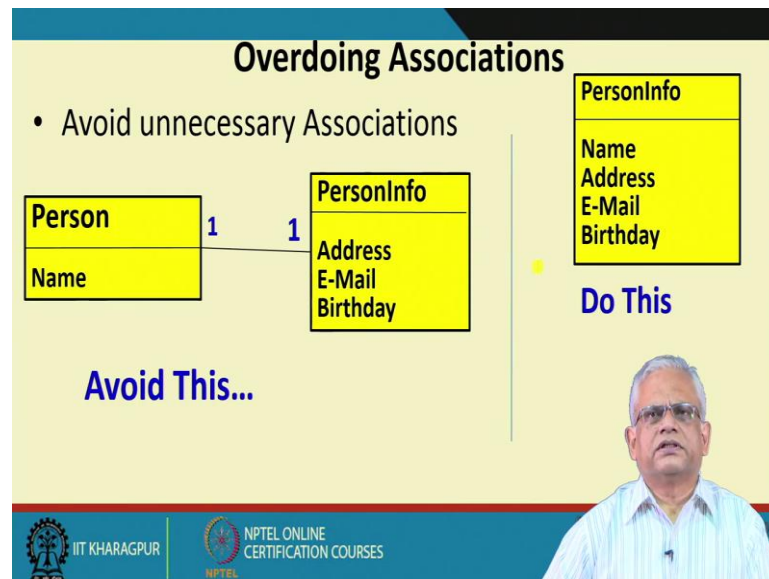
The slide also features a small inset image of a man in a light blue shirt and glasses in the bottom right corner. At the bottom, there are logos for IIT KHARAGPUR and NPTEL ONLINE CERTIFICATION COURSES.

So far, we looked at the implementation of association over last few sessions. And also, we looked at the Ternary and the Quaternary Association. We looked at how to implement a 1-1 association. And we said for 1-1 association, an implicit attribute on both objects that are linked due to this association is enough. And we said that the role names actually become the attribute names, the attributes that are implicit.

And then, we looked at 1-to-many association. And we said that this case tool is there to translates code from diagram. And while we write manually the code, we would use even a 1-

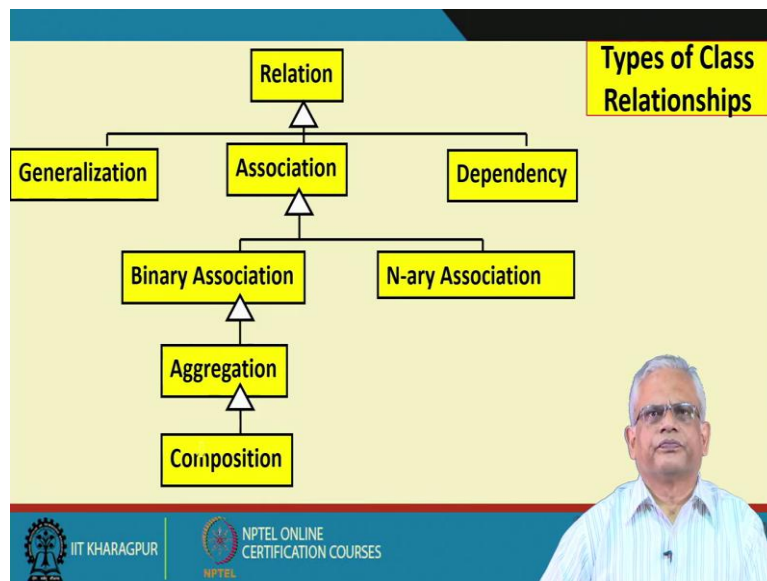
to-many association. We will use a Vector or ArrayList for that case. And for the Qualified association, it will translate to a Map or a Hash table.

[Refer Slide Time: 22:19]



But before we conclude our discussion on associations and look at the other relations between classes, the thing is that once we know a technique, we tend to overdo that. For example, a person and the person's information (in the above slide). Person has name and person's information, like address, e-mail, birthday et cetera. We should not create two classes here. The person's information and person, is a 1-1 association but we should have these in the same class, because these are uniquely belonged to this person. It's inefficient, more complicated to have a 1-1 association in between this rather we would have that in the same class. So, let's not overdo the association.

[Refer Slide Time: 23:25]



So far, we looked at Generalization that is the inheritance relationship. And we found that it was the simplest implement in Java code, using the extends keyword. And then, we looked at association. We looked at the Binary Association and N-ary Association (in general the Unary, Ternary and so on) and then, we are trying to be more expressive. We looked at association classes, Qualified association and so on.

Now, let's look at a special type of Binary Association, which is Aggregation and Composition which is a special type of Aggregation.

Let us now start discussing about Aggregation and Composition. And after that, we will look at Dependency, which is another different relation. We said that there are 4 types of relations between classes that are possible. We said that Generalization and Specialization, Association, Aggregation and Composition, and Dependency. But as we will discuss now, Aggregation is a special case of Binary Association, or can be say something more than Binary Association. We will see what is more. And composition is a special type of Aggregation. Composition is an Aggregation but then, slightly more than that. We will see that, what's the different.

[Refer Slide Time: 25:17]

The slide is titled "Aggregation Relationship" in a yellow box. It contains a list of bullet points and a class diagram. The diagram shows three classes: Company, Person, and Club. Company is connected to Person with a line ending in a diamond and an arrow, labeled "employs". Person is connected to Club with a line ending in an arrow, labeled "memberOf". Multiplicity "\*" is shown near Person in both relationships. A small yellow dot is next to the second bullet point. In the bottom right corner, there is a video inset of a man with glasses speaking. The footer contains logos for IIT KHARAGPUR and NPTEL ONLINE CERTIFICATION COURSES.

- Represents whole-part relationship
- Represented by a **diamond** symbol at the composite end.
- Usually creates the components:
  - But often indistinguishable from plain association.
  - **Except, aggregate usually invokes the same operations of all its components.**
- Usually owner of the components:

The Aggregation Relationship represents a whole-part relationship that is, given an object, it might contain other objects. Aggregation relation, represented by a diamond symbol, like this (as shown in the above slide). Given a company, it employs many persons. And a person is a member of one club. A club has many persons as member and a person is employed by a single company. One of the differences between an Aggregation and plain association is that the aggregate object is responsible for creating its component objects.

As we proceed, we will see this in more detail. But this approx of the difference between an association and an Aggregation. Just see here, the person and the club: there is also 1 to "\*" association but here, the club does not create the person. The person is created by the company. The new is called by the company method to create person. But other than that, an aggregation is also an association and often indistinguishable. This is a very subtle distinction between aggregation and association. And off-course, we can identify one more difference, that typically the aggregate object invokes the same method on all its objects, in a loop. The same operation is invoked on all objects (in above example for person objects). For example, let say, a company wants to print name for the employees, or maybe disburse salary and so on. It will invoke on all person elements. And we also say that since the company creates the persons in the code, we say that the company is the owner of the person (for the above example). The club is not the owner of the person, does not create it. Here, we say that the company is owner of persons. Even though it is owner, but it shares this person with other classes.

We are at the end of this lecture. We will stop here and from this point onwards, we will continue.

Thank you.