

**Object Oriented System Development using UML, Java and Patterns**  
**Professor Rajib Mall**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Kharagpur**  
**Association Class and Ternary Association**  
**Lecture 12**

Welcome to this lecture.

In the last class we were looking at the association relationship and the ways to implement the association relation in Java. We started in simple cases where the cardinality of association is '1'. For cardinality '1' we could use an implicit attribute variable. But when we have fixed cardinality like 10, 50 or something, a simple array was sufficient because there was a fixed upper bound.



But many times, we use '\*' as a cardinality and we said that any number of objects of one class can be associated with 'n' object of another class. And we saw that the collection classes come useful here where there is no upper bound and the number of objects to be associated with an object.

We also discussed little bit about the collection classes in Java. Collection class is the framework and we saw the different class types in the collection classes and we saw that array list and vector are typically used to implement '\*'. And then we are discussing about the association class.

Let's start this session with the association class.


(Refer Slide Time: 02:05)


# Association Class



13

- An association relation between classes often has its own attributes and methods:
  - **Example 1: A teacher teaches a subject:**
    - A teacher may teach the same subject multiple times over many semesters
    - Each time the students may be different
  - **Example 2: A person works for a company**
    - The date he joined a company
    - Pay scale and grade at which joined, promotion, so on
    - He may even join the company multiple times

**Association Class**





An association class is a powerful modeling mechanism and any non-trivial problem association classes come handy in constructing the class model. We were just discussing in the last session that a teacher teaches a subject. It appears to be a very simple association model where we have a 1-1 association between a teacher and a subject. But then we can have a non-trivial situation that a teacher may teach the same subject multiple times over different semesters. In this case, a simple 1-1 association between the teacher and the subject because it does not capture different aspects like the teacher over different semesters has been teaching the same subject, each time

the students who register for a subject are different, their grades are different, roll numbers are different and so on.

We are looking at another example. Person works for company. It's a simple association. A person works for a company, between the person class and the company class there is a 1-1 association. As it drawn in above slide. Here, the association is 1-1 association and 'works for' is the association name and the reading direction is from Person to Company.

This is a simple association example. There is implicit attribute the person side to keep track of the company he works for and the company side also has the implicit attribute to keep track of the person that works for that company. But in a non-trivial situation, a person starts working for a company on certain date, gets a promotion, joins at a higher post or maybe left the company for joining another company then again came back and joined in a different designation in the company. Now the simple association model we discussed does not capture all these non-trivial cases. Let see how we can use the association class to for this purpose.

(Refer Slide Time: 05:49)

The slide features a yellow background with a blue header and footer. In the top right, a yellow box with a pink border is labeled "Association Class". The main content includes a list of bullet points and a UML class diagram. The diagram shows a class "Teacher" connected to a class "Subject" by a solid line with an arrow pointing to "Subject" and an asterisk "\*" at the arrowhead. A dashed line connects the "Teacher" class to a class "Teaching" below it. The footer contains the logos for IIT KHARAGPUR and NPTEL ONLINE CERTIFICATION COURSES, along with a small inset image of a man in a light blue shirt.

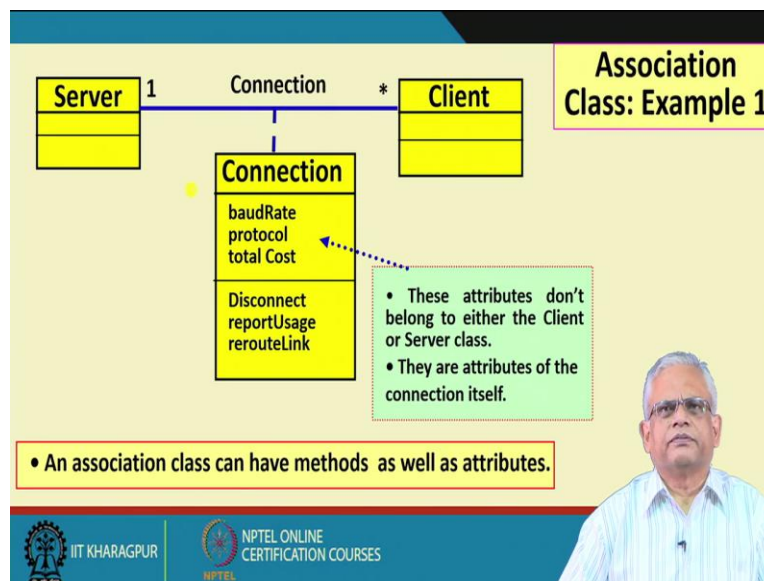
- An association class is shown as a class symbol:
  - Attached to the association symbol by a dashed line.
- An association class:
  - Same name as the association relation because...
  - It represents the association!

A teacher teaches a subject or a teacher teaches multiple subjects. But then the different subjects that he teaches maybe he teaches multiple times, in some semesters he teaches object-oriented systems and then after couple semester again teaches object-oriented systems, meanwhile might be teaching other subjects and so on.

We modelled this situation by drawing teacher and subject as in an association (as shown in the above slide). The example is the unidirectional association in the sense that from the teacher object we can identify what are the subjects taught but not vice versa. From the subject we cannot determine the teacher name but you can have a bidirectional association here that we had seen earlier. Here, the association class is modelled with a dotted line and the association name that normally appears on the association line, we just make it a class. This class captures all the aspects of the association, for example when he taught, which semesters, it can have a link to the students he taught and so on.

The name of the association class is usually the same name as the association. Because it captures all the details of the association like its attributes, methods and so on. These is the first example of the association class and this association class is a powerful mechanism. Many situations where we cannot just model with a simple association, we would use an association class there.

(Refer Slide Time: 07:59)

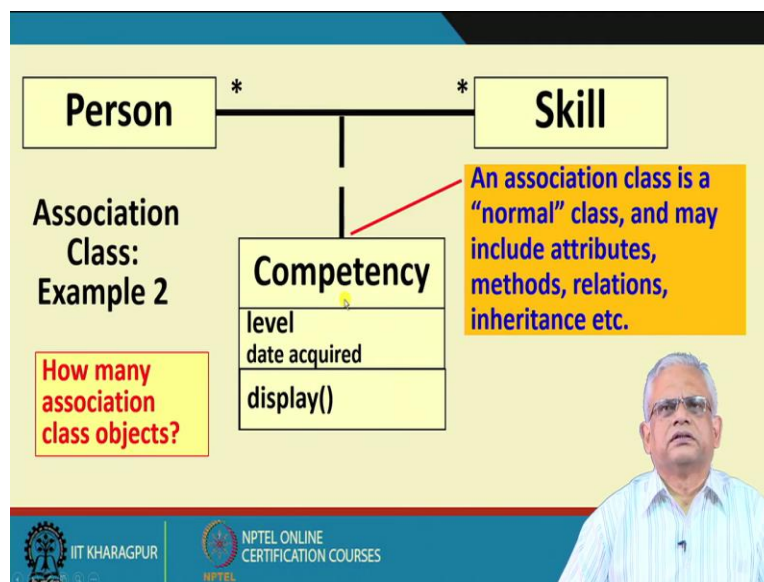


This is another example of association class (in the above slide). A server connects to multiple clients over time and each time it connects to a client there is a billing information associated with it and also the connection is characterized by the baud rate (the speed at which it connects), the protocol it uses whether FTP, HTTP, et cetera and the total cost incurred for that connection

(that would depend on the total time of the connection and the baud rate and the protocol). The connection will also have methods like Disconnect, ReportUsage, routeLink, etc.

In the above slide, we can see association class 'Connection' is attached with a dotted line. Association class is a normal class, it has its attributes and methods and most of these characterize the association itself. These attributes here with the association class don't belong to the client or the server classes as they are not characteristics of the baud rate with which the client connected. We cannot store all these attributes in the server side. These attributes are characteristics of the association. In general, an association class can have both attributes and methods.

(Refer Slide Time: 09:56)



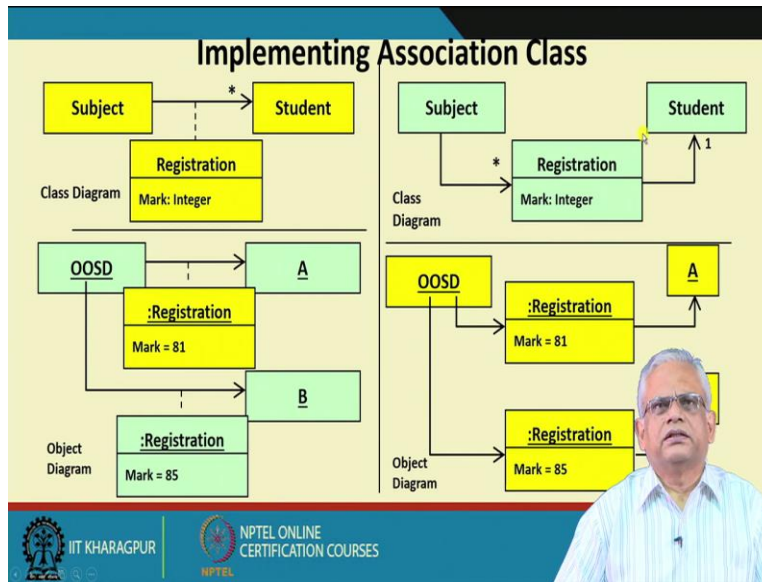
Now let's look at another example (in the above slide). Here, a person acquires many skills and the skill is acquired by many persons. Each link that gets established between a person and a skill is characterized by a competency. A person might acquire painting or maybe let say singing or music and so on and for each gets a competency. There is the level he completed which indicate is it preliminary, expert and so on. There is date to indicate the competency was acquired and we have these methods here display competency. A person over the year might get different skills, so skill objects are there. The skills are music, painting, drawing, et cetera. Also, there are many persons and the same skill might be acquired by multiple persons A, B, C, D etc. And a person can have multiple skills and each skill will have a competency and that competency is captured

in the 'Competency' association class. As we said earlier that the association class actually is a normal class, it has its attributes, it has its methods, there can be association inheritance, et cetera with other classes.

The association class is mainly used to capture the association between two classes. But a question that may arise that given a situation like there are 3 instances of the person A, B, C and there are 2 skills music and painting. So how many association class objects will be there?

I hope you can figure out that for every link that gets established there is an association object, that is an instance of the association class. If person A acquires skill drawing then there is a competency that, the level to which it is acquired, the date acquired, etcetera. So, given that there are 3 objects here and 2 skills here, assuming that in the case where all the person objects acquire all the skills there will be totally 6 links ( $3 * 2 = 6$ ) links will be there and therefore 6 competency objects will be required.

(Refer Slide Time: 13:47)



Now let's get more insight into the association class. Let's discuss how to implement this and that will give us a better insight into the association class. Let say, in a subject multiple student register and the same subject is start over semesters and therefore maybe a student registers for multiple times: first time failed, second time passed and so on. Also, may so happen that each time the subject is referred, there are different sets of students. So, there will be different semester on which the student will get different marks. And the students who registered for subject has the details of the student like name, department and so on.

Now if we can extent the discussions, for every instance of the subject there is a link created with the corresponding object (in the above slide, 2<sup>nd</sup> diagram of left side) and here there are 2 students A and B, and OOSD Object Oriented Software Design is the only subject we are displaying here (there can be other subjects also). And for every subject, there is a link established to the student. It is a unidirectional link. There is registration object for each student and this subject mark is stored in this instance of the registration object. For example, between the OOSD and the student B there is registration object where the Mark = 85.

We can also model the same association between the student and subject by two binary associations (in the above slide, 1<sup>st</sup> diagram of right side). The subject is associated to a specific registration and the registration is associated to a specific student. So, if we have '1' (at Registration side) here cardinality and '\*' (at Student side) here then we have the same subject

has multiple registrations over years and each registration identifies one specific student. And if we can think these association class and two binary associations is equivalent (one is between subject and registration, the other is registration and student) then the implementation becomes straightforward. It becomes the normal association implementation between subject and registration, and registration and student. So that is also another way association class can be implemented.

(Refer Slide Time: 17:44)

The slide displays the following Java code:

```
public class Subject {
    private ArrayList <Registration> reg=new ArrayList<Registration>();
    public void enrol(Student st) {
        reg.add( new Registration(st) );
    }
    ...
}

class Registration {
    private Student student;
    private int mark;
    Registration(Student st) {
        student = st; mark = 0;
    }
    ...
}
```

The class diagram shows a **Module** class associated with a **Registration** class (indicated by an arrow with an asterisk). The **Registration** class is associated with a **Student** class (indicated by an arrow with a '1' at the end). The **Registration** class has an attribute **Mark: Integer**. A speaker is visible in the bottom right corner of the slide.

Annotations on the slide include:

- Pass the Student to Registration.** (pointing to the `st` argument in `reg.add( new Registration(st) );`)
- Maintain the link to Registration** (pointing to the `reg` array list in the `Subject` class)
- Keep track of the Student reference.** (pointing to the `student` attribute in the `Registration` class)

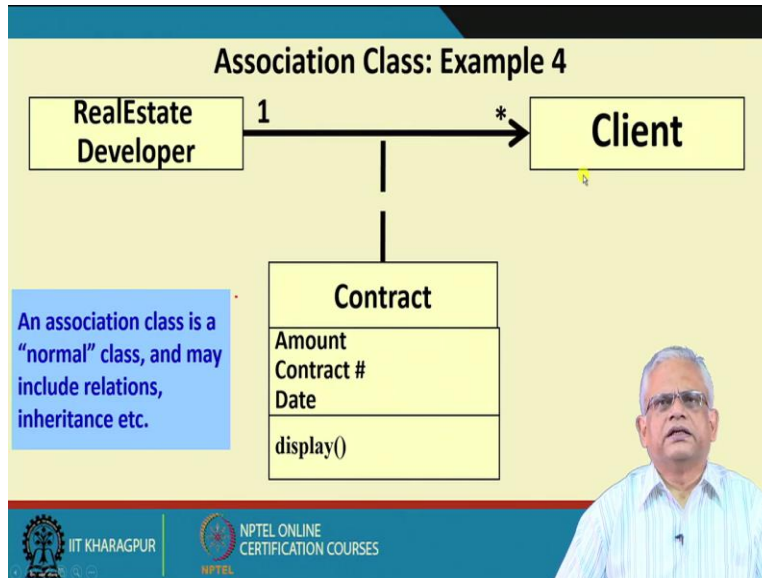
The slide also features logos for IIT KHARAGPUR and NPTEL ONLINE CERTIFICATION COURSES.

Now if we agree to that, then the class code is straightforward (in the above slide). We have the class subject and the subject is associated with multiple registrations. We use an array list here to keep track of the registration and call it as 'reg'. 'reg' is an array list of registrations. Once a student enrolls on a subject we create a registration object (`new Registration(st)`). Here, with the student reference 'st' we create a new registration object and we store it into the 'reg' array using `reg.add()`. And we keep track of the registration using that 'reg' array.

We have also this constructor '`Registration(Student st)`' which takes the students reference as the argument and stores the student and this is a 1-1 association between registration and student and therefore a simple attribute is sufficient here.

(Refer Slide Time: 19:24)





In many practical problems we need to use association classes. This is another example, a real estate developer has many clients and each client signs a contract with the real estate developer.

So, the contract between the real estate developer and the client is the name of the association within the client and the real estate developer. It also a unary association that is the real estate developer maintains the details of all clients but the client does not keep track of the real estate developer. If we want the client side, the client object also to have the reference for the real estate developer for that case we need to use a bidirectional association here. The contract is identified by the amount for which the contractor has signed, the contract number, the date on which it was signed and we can have methods in the contract like display the details of the contracts and so on. And we can implement this as binary associations between two binary objects that is real estate developer and contract that is 1 to '\*' here and between contract and client there is a 1-1 relation.

(Refer Slide Time: 21:31)

```
public class RealEstateDeveloper{
    private Vector <Contract> contracts= new Vector <Contract>();
    public void buy(Client c){contracts.add(new Contract(c));}
}
public class Client{
    private Address address;
    public Address getCurrentAddress(){}
```

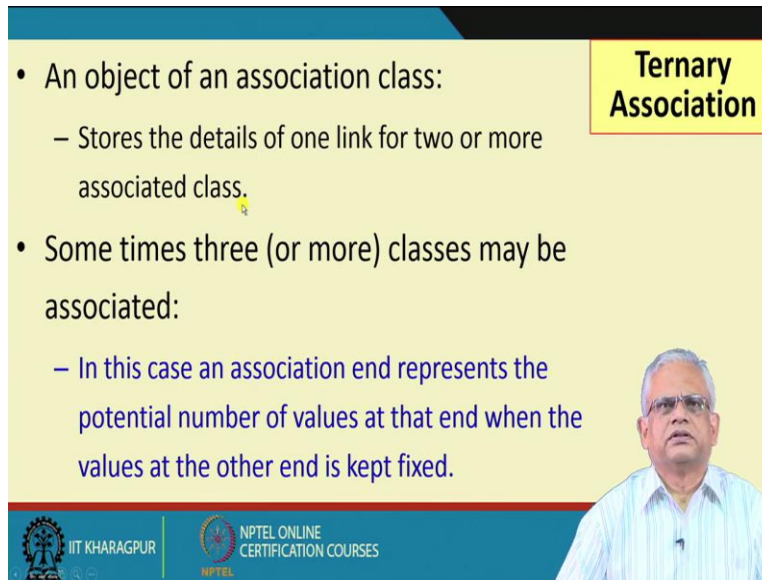


```
    }
    public class Contract{
        private Client client;
        private int contractNo;
        public Contract(Client c){ client=c;}
    }
}
```

Then the code for the two binary association becomes straightforward (in the above slide). We can use the array list or a vector to keep track of all the contracts in RealEstateDeveloper class. The real estate developer only keeps track of the contracts and once a client buys it create a new contract in the 'buy' method. Here we can see, 'Client c' buys a property and then keep track of the contract here in the contracts attribute (which is an array list of contracts), it creates a new contract and pass the client to the constructor of the contract class. In the contract class, the constructor set the attribute client = c, so unidirectional association between the contract to the client is maintained.

And in the client side we do not keep any implicit attribute, it just has its normal attributes like address, et cetera.

(Refer Slide Time: 23:01)



**Ternary Association**

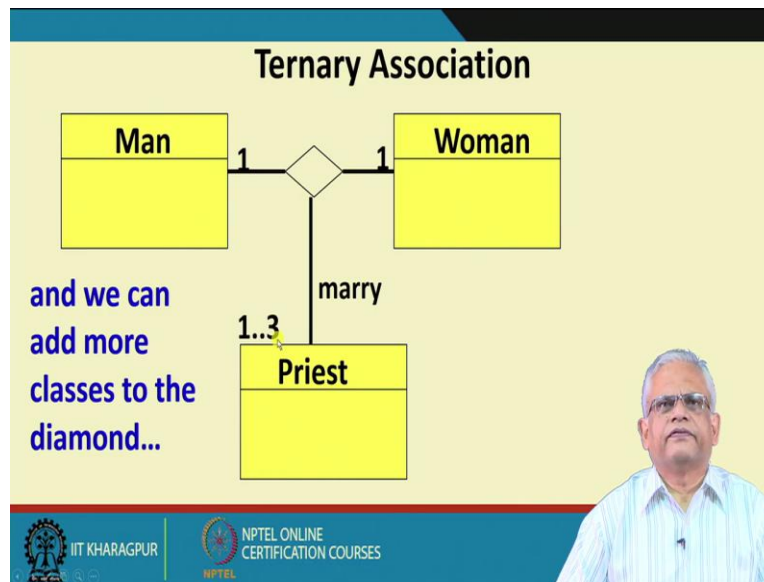
- An object of an association class:
  - Stores the details of one link for two or more associated class.
- Some times three (or more) classes may be associated:
  - In this case an association end represents the potential number of values at that end when the values at the other end is kept fixed.

IT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

We can also have ternary associations. So far, we looked only at the binary association between two classes, the unary association, association defined on a single class and then we looked at the association class which is basically two binary associations. We also said that the association class is an elegant modeling mechanism.

We can also have ternary association where the association exists between 3 classes and under some situations, we can also have more classes associated for example 4, 5, etc. We will just look at some examples of ternary association.

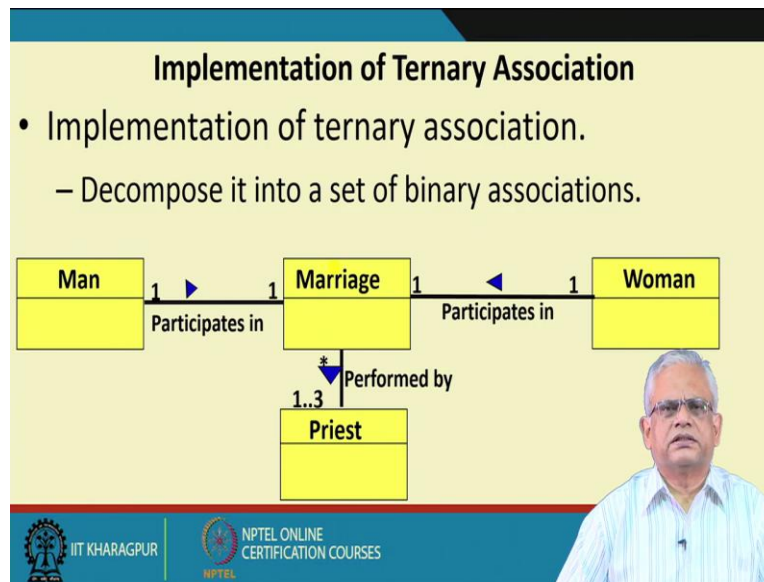
(Refer Slide Time: 24:04)



Let see this ternary association example there are 3 classes that are associated (in the above slide). A man marries woman and the marriage is done by 1 to 3 priests. If you consider the link between an instance of man and woman then we can identify up to 3 priests who performed the marriage.

The diamond is the symbol used for representing ternary association. We can add more classes to the diamond to represent quaternary association and so on. And the cardinality here is between man and woman and also cardinality in Priest side to identifies how many priests performed the marriage.

(Refer Slide Time: 25:22)



Now the question is how do we implement a ternary association?

Again, it is a set of binary associations. In a ternary association we have 3 pairs of binary associations between Man and Marriage (man participates in marriage), Woman and Marriage (woman participates in marriage) and between Marriage and Priest (marriage is performed by priest). So ternary association is implemented by set of binary association. In this example, three binary association required.

Similarly, a quaternary association can be implemented as a set of binary association in that case we will have one more binary association between the other class and this class.

We are almost at the end of this lecture hour.

We will stop here and continue from this point in the next lecture.

Thank you.