Object Oriented System Development using UML, Java and Patterns

Professor Rajib Mall
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Implementation of Association in General Case
Lecture 11

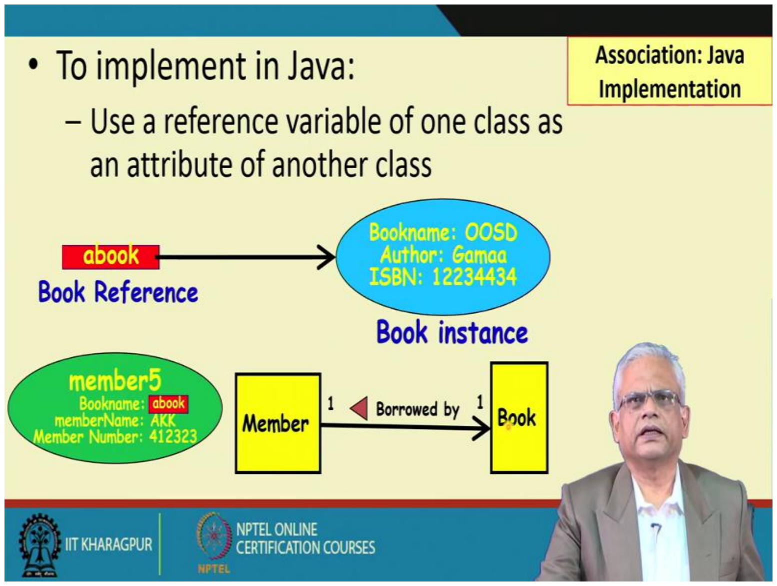
Welcome to this session.

In the last session we were discussing about the association relations between classes. We found that association was a powerful mechanism to model relations between classes. A single class can be related to itself and we called it as unary association. The binary association is the most common where two classes are associated. For binary and unary association, we discussed a number of examples and then we also discussed little bit on ternary association, quaternary association, and so on.

We were towards the end discussing about implementation of the association relationships in Java. If we draw a class diagram with association relationships between classes, how will the Java code be generated?

For inheritance relationship between classes it was straight forward but for association relationship we had to do some bit of programming. A case tool like Argo UML is thereto generate the code in Java, C++, etcetera from the association diagram. Even the tool is there but we must understand that what is the implication of the association relationship in terms of the Java code or given a Java code we should be able to identify the relationship that's represented. Now let's proceed from that point onward.

(Refer Slide Time: 02:19)



Association: Java Implementation

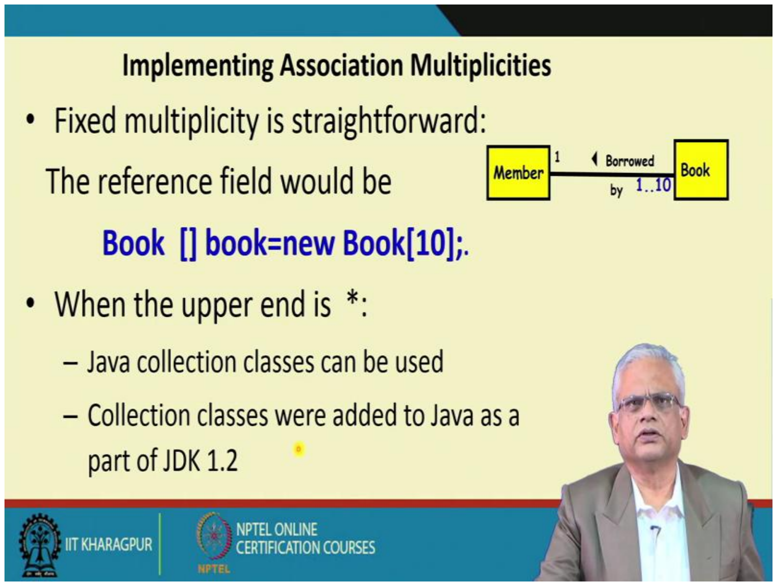
- To implement in Java:
  - Use a reference variable of one class as an attribute of another class

We had seen that the association relationship between a member and a book is a unidirectional association (in the above slide). A book is borrowed by a member. And we had the name of the association written here which is ‘Borrowed by’ and also cardinality is represented here in the Book end and Member end. And to implement in Java we use a reference variable of one class as attribute of another class. For example, the book reference needs to be stored in the member object. This is a unidirectional association in the sense that from the member we will be able to reach the book. The book object methods can be invoked from the member object but not vice versa that is indicated by this unidirectional association (unidirectional because an directed arrow from Member to Book class in the diagram).

If we have the book object, like ‘abook’. ‘abook’ has book name OOSD and author is Gamaa and other attributes of the book (as shown in the above slide) and then we have the handle here with which you refer to this object ‘abook’. If we have association here with Member and Book then on the member object a book reference need to be stored as an attribute. In the slide wee can see we have member object (green circle in the slide) and then in this attribute ‘abook’ which is the book reference is stored here the book name and then we have the other details of the member here. And once we have attribute here abook, then we can invoke various methods of the book class, for example abook.display_name(), abook.find_duration for which borrowed and so on. This was the basic idea of implementing an association in the simplest case.

If we have bidirectional association, then the reference of the member needs to be stored on the book side as well and from the book object we can also invoke the methods of the member objects.

(Refer Slide Time: 05:26)



Implementing Association Multiplicities

- Fixed multiplicity is straightforward:

  The reference field would be

  Book [] book=new Book[10];.

- When the upper end is *:
  - Java collection classes can be used
  - Collection classes were added to Java as a part of JDK 1.2

Member
1
Borrowed by
1..10
Book
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES

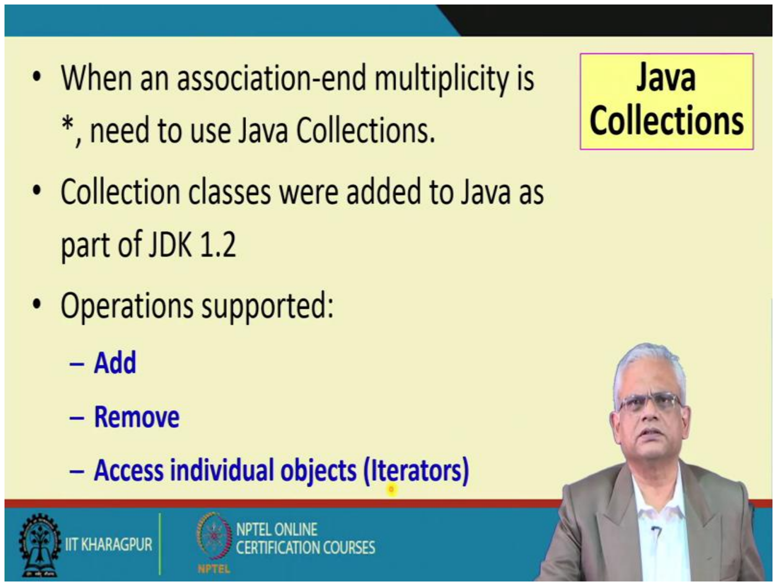
Now let see more general cases. In the previous case we had a 1-1 cardinality or multiplicity which was the simplest case. But what if each member can borrow up to 10 books?

So, a member object would have link with 10 book objects (as shown in the above slide). This also is not very difficult; we use simple arrays here. In the member object we will have this attribute book and here we will have 10 book objects.

But what about the cardinality ‘*’ instead of 1 to 10. ‘*’ denotes any number of instances, any number of books the member can borrow. How can we implement it in Java?

We can use the Java collection classes which is powerful feature of the Java language and we can implement the multiplicity ‘*’ just to add object in the collection attribute. The collection classes were not there in the initial versions of JDK but JDK 1.2 onwards we had the collection classes.

(Refer Slide Time: 07:24)



Java Collections

- When an association-end multiplicity is *, need to use Java Collections.
- Collection classes were added to Java as part of JDK 1.2
- Operations supported:
  - Add
  - Remove
  - Access individual objects (Iterators)

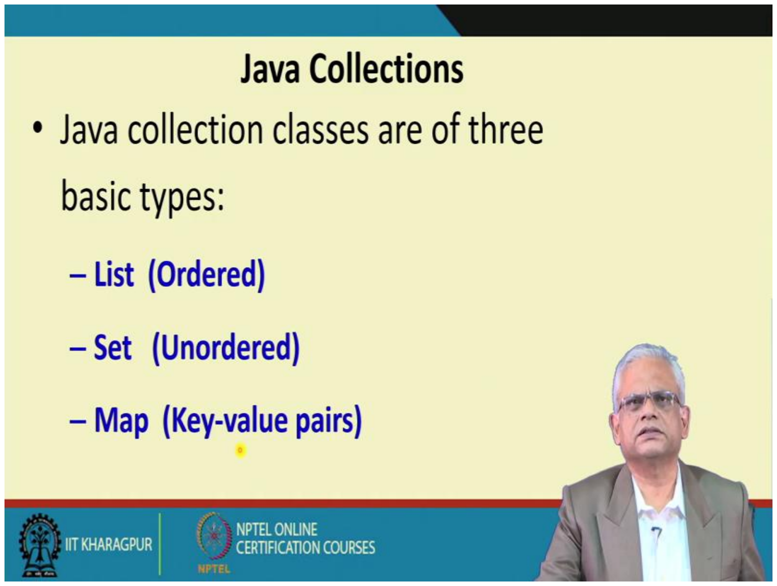
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
NPTEL

The Java collections are used when we have the association-end multiplicity '*'. Fixed multiplicities like 1 to 10 we use simple arrays but we cannot do that when we have '*'. For '*' we do not have any upper bound.

So, for '*' we need to use the Java collections. The collections were added from JDK 1.2 onwards. In the Java collection classes, there are some standard methods like add, remove and also, we can access the individual objects of the collection classes by creating iterator objects.

As we proceed in this course, we will see that iterators are one type of pattern, we will discuss about the iterator pattern which has been implemented in the Java collection classes. Right now we will just use the collection classes but later, once we start discussing about the design patterns, we will discuss about the iterator pattern.

(Refer Slide Time: 08:53)



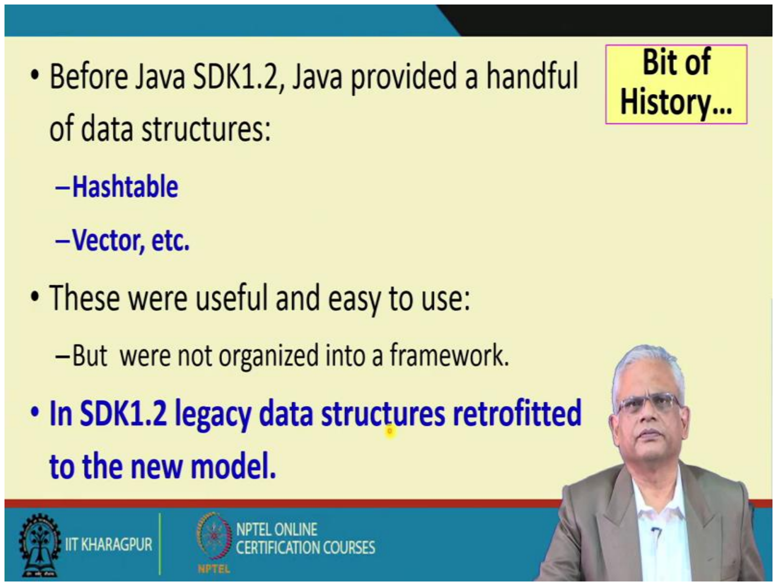
Java Collections

- Java collection classes are of three basic types:
  - List (Ordered)
  - Set (Unordered)
  - Map (Key-value pairs)

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
NPTEL

There are basically three basic types of Java collections. First one is the list which is the ordered set of objects. Second one is the set which is unordered set of objects. In the list we can say that which is the previous object of an object, which is the next object but, in a set,, it is unordered we can only say that whether it is there in the set or not or add to the set, delete from set etcetera. In set, we do not have any notion for the preceding object or succeeding object. The third one is the map. A map on the other hand is a key-value pair or key-object pair. Based on the key we can find from the map whether the corresponding object exists, whether it does not exist, and so on.

(Refer Slide Time: 09:55)


Bit of History...

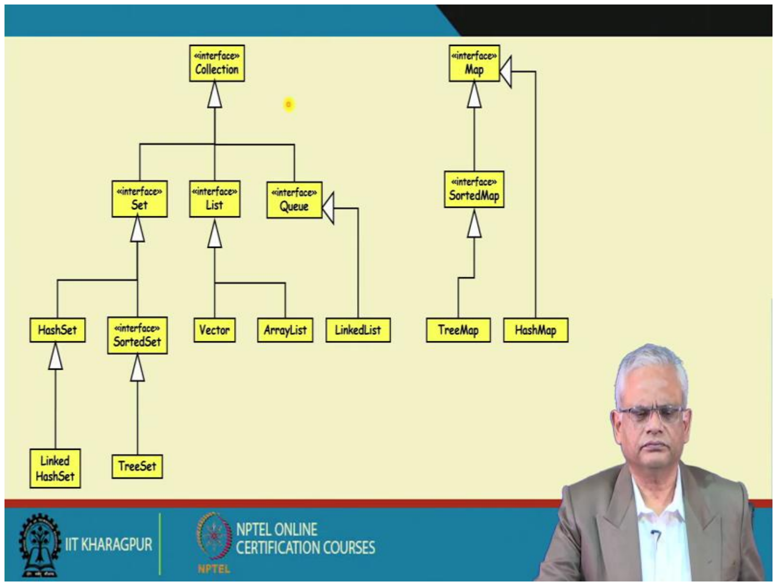
- Before Java SDK1.2, Java provided a handful of data structures:
  - –Hashtable
  - –Vector, etc.
- These were useful and easy to use:
  - –But were not organized into a framework.
- **In SDK1.2 legacy data structures retrofitted to the new model.**

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
NPTEL

It will not be out of place to just mention a little bit of history that before SDK 1.2 there were only a handful of data structures supported in Java for example hash table, vector et cetera. These were useful and easy to use but were not a framework basically.

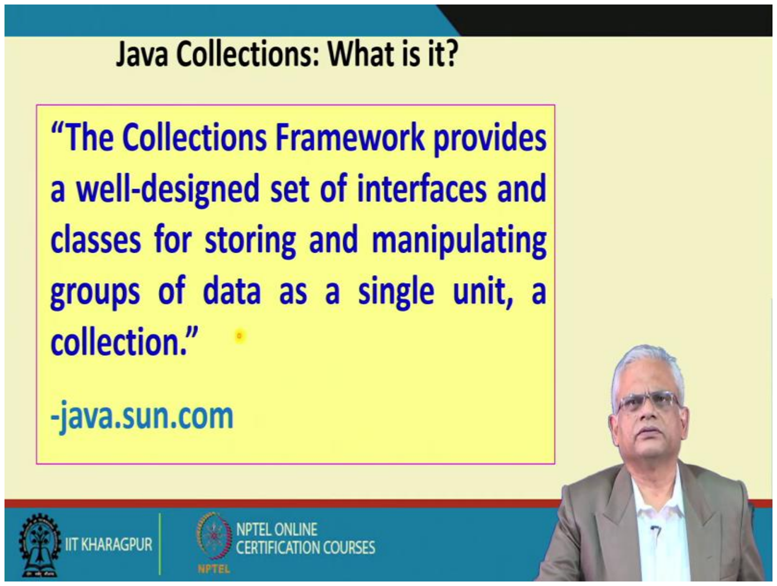
A framework is one which the programmer can extend easily just add few methods, et cetera and most of the ones are already available and we have sophisticated tool for using the framework. Since hash table, vector, etcetera already existed in the early stages of Java these were retrofitted into the new model.

(Refer Slide Time: 11:02)



If we show this in the form of a diagram (in the above slide), we can see here the Java collection classes, the vector was retrofitted here in the list and in the queue, we have linked list. We had array list which is inherited from List class. We can also observe that the map is a separate class hierarchy. It is not really the part of the Java collections in the sense that we don't have the iterators for map but for all the java collections we have iterator.

(Refer Slide Time: 11:51)



Java Collections: What is it?

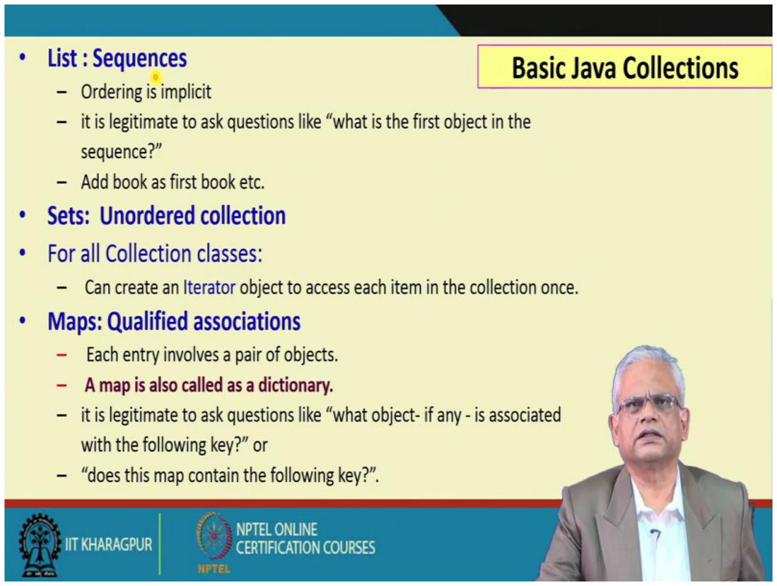
"The Collections Framework provides a well-designed set of interfaces and classes for storing and manipulating groups of data as a single unit, a collection."

-java.sun.com

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
NPTEL

In the java.sun.com website, we can find that the Java collections as it is described. the collections framework provides a well-designed set of interfaces and classes for storing and manipulating groups of data as a single unit and this also provides a dynamic data structure where we can add any number of elements unlike arrays where the size is fixed.

(Refer Slide Time: 12:36)



Basic Java Collections

- List : Sequences
  - Ordering is implicit
  - it is legitimate to ask questions like "what is the first object in the sequence?"
  - Add book as first book etc.
- Sets: Unordered collection
- For all Collection classes:
  - Can create an Iterator object to access each item in the collection once.
- Maps: Qualified associations
  - Each entry involves a pair of objects.
  - **A map is also called as a dictionary.**
  - it is legitimate to ask questions like "what object- if any - is associated with the following key?" or
  - "does this map contain the following key?".

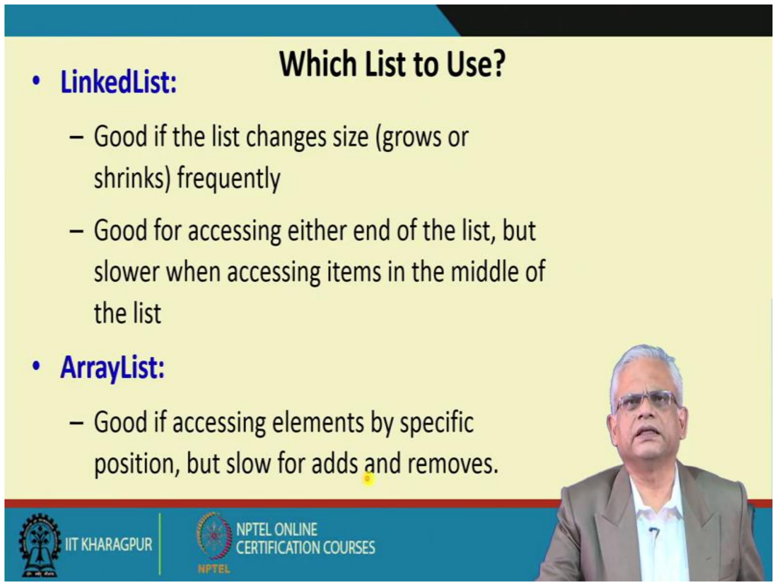
The basic Java collections are lists and sets. The lists are the ordered sequences; the sets are unordered collection. And for all collection classes we can create an iterator object just saying that this is an important pattern which has been implemented in Java.

Later as we proceed with the course, we will look at the iterator pattern which is a standard way to iterate or to display the details, access all the elements of a collection etc. In the sequences, the ordering is implicit and we can always ask questions like what is the first object in the sequence, what is the next one and so on. We can also add elements as the first object or the next object and so on.

We have the maps which required for qualified associations. The maps we have a pair, one is the object and another is called as a key. A map is also called as a dictionary. Based on the key we can look up a map and find the corresponding object or maybe the object does not exist, so we will say that the key does not exist.

We will implement some types of associations using maps but typically we will use the sequences that is array list for implementing associations with multiplicities of star which is any number of objects.

(Refer Slide Time: 14:42)



Which List to Use?

- **LinkedList:**
  - Good if the list changes size (grows or shrinks) frequently
  - Good for accessing either end of the list, but slower when accessing items in the middle of the list
- **ArrayList:**
  - Good if accessing elements by specific position, but slow for adds and removes.

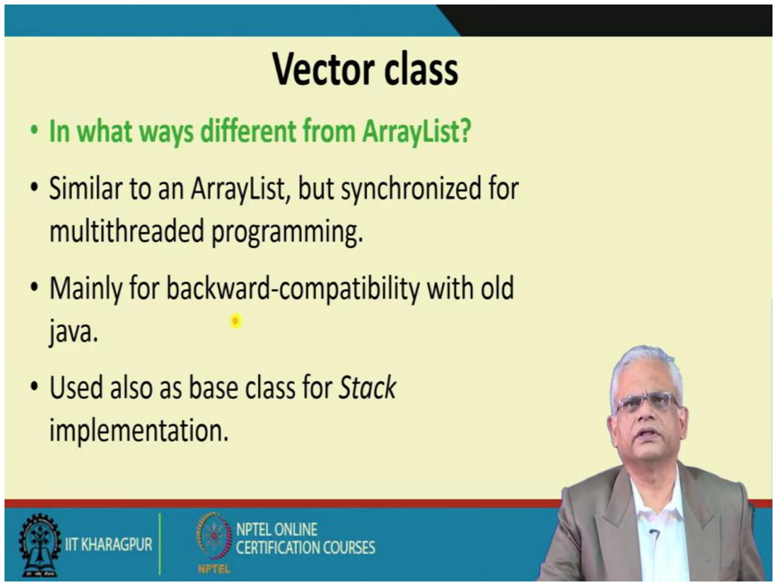
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
NPTEL

If we compare the different collection classes between linked list and array list, which one should we use?

A linked list is good if the list changes in size that is it grows and shrinks frequently. We need to access either end of the list but if we want to access something in the middle, we need to traverse through the list and therefore it is slower.

In our implementation of association, we must keep this in mind that if we are always accessing one end of the objects in the list, then we would rather use linked list. But array list is the most commonly used collection for implementing associations because here we can access the elements by specifying a position arbitrarily, but then here it is slower in add and remove because if we add at arbitrary position and delete arbitrary elements at arbitrary positions it takes time.

(Refer Slide Time: 16:12)



Vector class

- In what ways different from ArrayList?
- Similar to an ArrayList, but synchronized for multithreaded programming.
- Mainly for backward-compatibility with old java.
- Used also as base class for *Stack* implementation.

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
NPTEL

The vector class and the array list, these are very similar actually. These are linear dynamic arrays of objects. But how does a vector class differ from an array list?

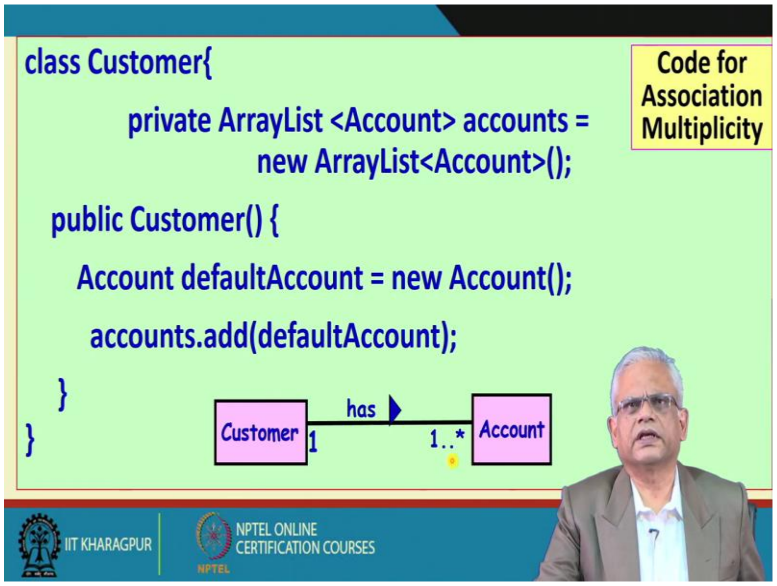
The vector class is similar to an array list but the only difference is that it is not synchronized for multithreaded programming.

Vector class as we are saying that it existed before JDK 1.2 and array list was added later since JDK 1.2 and the array list is synchronized for multithreaded programming and typically in our implementation of associations, we will use array list. But otherwise there is not much difference between an array list and vector.

If you are not doing multithreaded programming, we can use either of these. We might have a question, why do we have two collection classes that do almost similar things?

The answer is that, mainly they are for backward compatibility with the old Java where we had vectors. And also, the vectors are used as the base class for stack implementation.

(Refer Slide Time: 17:57)



Code for Association Multiplicity

```
class Customer{
        private ArrayList <Account> accounts =
                    new ArrayList<Account>();
  public Customer() {
    Account defaultAccount = new Account();
     accounts.add(defaultAccount);
  }
}
```

Now with this brief introduction to the collection classes, let’s try to implement an association like this (in the above slide). So here just observe that we read it as a costumer has one or more accounts that means ‘n’ account belongs to a single costumer. On the account side we need to store the specific customer with which this account is associated. On the other hand, on the customer object side we need to keep track of many accounts, there is no upper bound here. The customer can have many accounts as many as he creates. And naturally we will use a collection class like an array list to implement this kind of multiplicity.

So let’s write the account side code, I am sure that all of you will be able to write, we had written this kind of thing where the multiplicity was 1. It will be something like:

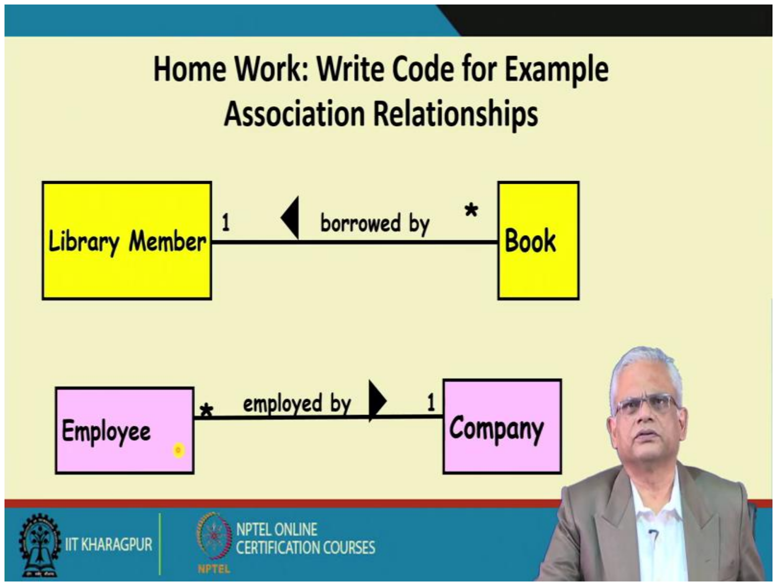
```
class Account {
        Customer customer1= new Customer();
        …………..
        …………..
        }
```

Now let us write this code where the customer object needs to have reference of many accounts and there is no upper bound in the number of accounts that a customer can have.

The class Customer has ‘Private ArrayList<Account>accounts’ -- it is new array list. So here we are creating an array list accounts of the type Account. The type Account class we are creating accounts as an array list and when we create a customer. There is a constructor for customer where we create account for customers. We create at least an account because the multiplicity is 1..* that means a customer has at least one account. Therefore, in the constructor itself we create one account and later we might add more accounts. First, we add the first created account which is a default account and later we can add more account in a customer accounts list. So this default account we add to this array list accounts to keep track of the customer accounts.

And we can have new accounts, create account for the customer and we will keep on adding on accounts using ‘accounts.add()’. So this is the basic mechanism of implementing ‘*’.

(Refer Slide Time: 21:19)

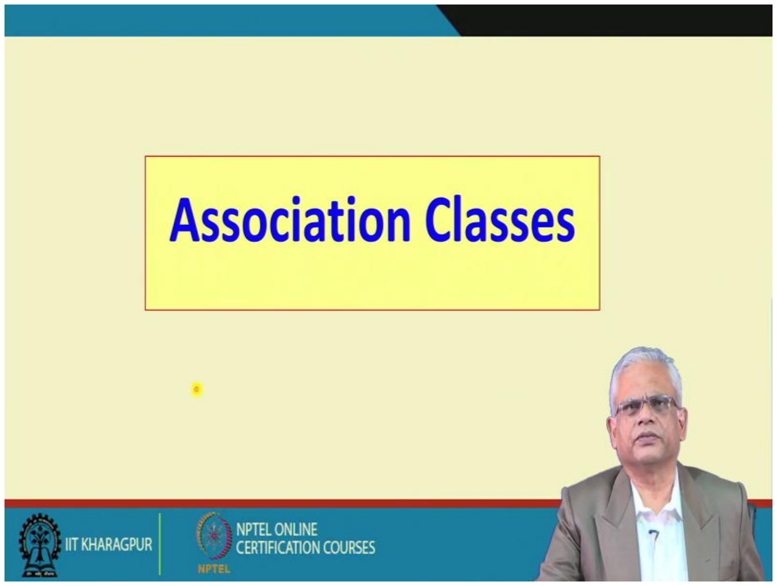


Home Work: Write Code for Example Association Relationships

As a homework please do above slide exercises. Here two examples of an association with multiplicity of’*’. A library member borrows many books or we can read a book is borrowed by exactly one library member. And also, please write Java code for this example, an employee is employed by a single company but a company employs many employees.

Please write code for these two as a homework.

(Refer Slide Time: 22:11)

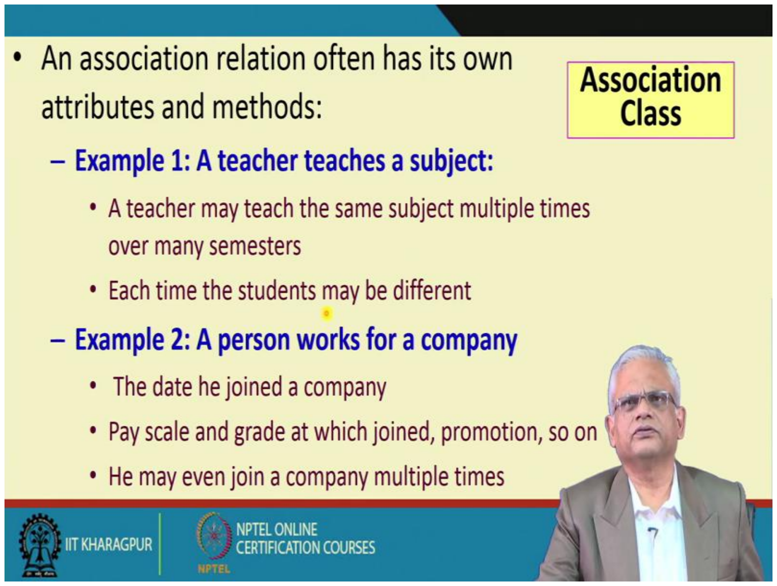


Association Classes

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now let’s see the association classes. Many times, we have problem descriptions where the simple association concept that we have so far discussed becomes inadequate to model some real-world problems. Let’s look at some examples.

(Refer Slide Time: 22:41)



Association Class

- An association relation 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 a company multiple times

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

In a real-world problem often an association relation has its own attributes and methods. For example, a teacher teaches a subject, this is a simple association. The teacher class and the subject are associated with the teachers relation but then the teacher teaches a subject over many semesters.

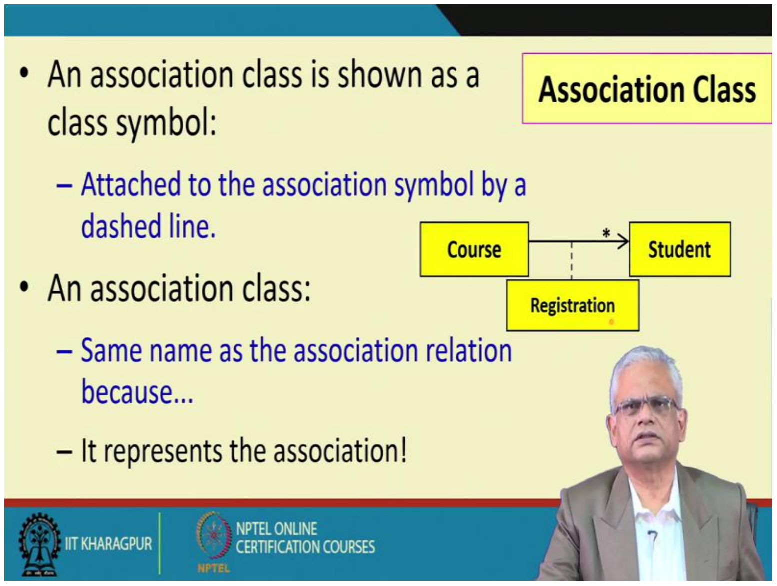
He can taught once in the year 1999, 2005, 2007 and each time he taught, he has taught the same subject multiple times over many semesters. And each time the students who have registered are different. So just by having this association relationship between teacher and subject, we are not able to capture this issue that the teacher has taught it many times this subject, in different years or in different semesters. And each time the students who registered were different and maybe the same student registered couple of times and each time he got different marks. How do we model this?

Another example is that a person works for a company. A simple association to look at it that a person and company are two classes of association relationship.

But then just remember that the person joined a company on certain date and at that time he had some grade and on a later date he joined on a different designation, his pay was different and maybe he left the company for a while and again joined back. So how do we model this?

A simple association relationship between a person and a company does not capture that he joined the company multiple times on different dates, on different pay scales, different grades and so on. In this situation, the simple association is not enough. In this case, we need to use the association class.

(Refer Slide Time: 25:46)



Association Class

- 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!

The association class is given the same name as the association relation and is a separate class by itself (as shown in the above slide figure). It is drawn with a dotted line. Here a course is taken by many students and let say in different years, different students take the course and there is registration information. So, the course is registered by many students and each time different students register for the same course.

So, this is represented by association class (Registration is the association class here) where we use this dotted line (as shown in the above slide) to represent the associated class. The association class which is basically a normal class like any other class but it captures the attributes of the association. For example, which year registered, what was the grade obtained by the students all are used as attributes of association class. The association class has the same name as the association and therefore we don't need to write the association name on this arrow. We can write association name but if we write there is no problem, but then the association class itself represents the association and therefore we need not separately write down the name of the association, it is implicit as long as we have a dotted line and associated class.

The association class is a powerful mechanism to model many real-life situations. We will use it in many problems.

we will stop here and continue in the next lecture.

Thank you.