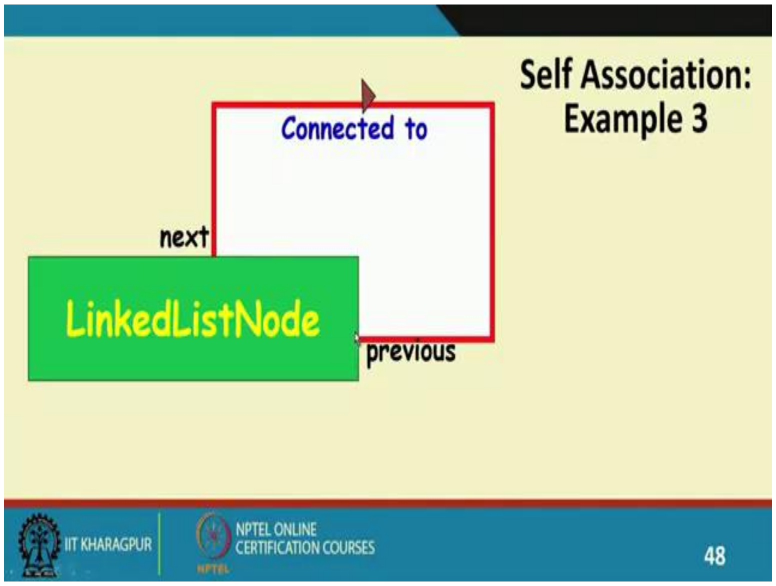
Object Oriented System Development using UML, Java and Patterns
Professor Rajib Mall
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur
Implementation of Association Relation in Java
Lecture 10

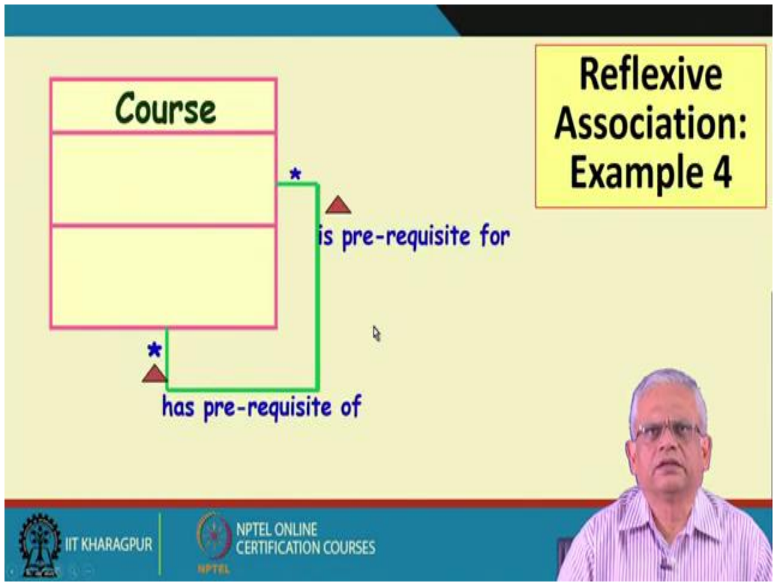
Welcome to this session. In the last session we were looking at the Association Relationship. We looked at the basic syntax, the association relationship. We looked at the binary association between two classes and we looked at the unary association. The unary association is defined on a single class that is the links get formed in this association between the objects of the same class, remember that we call the association between classes but in the object space we say that links get formed between the object of the class. We looked at few examples of unary association. The unary association is also called as reflexive association or self-association. We looked at some examples like linked list, friend-off etcetera. Now let us proceed from that point onwards.

(Refer Slide Time: 01:34)


Self Association: Example 3
Connected to
next
LinkedListNode
previous
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
48

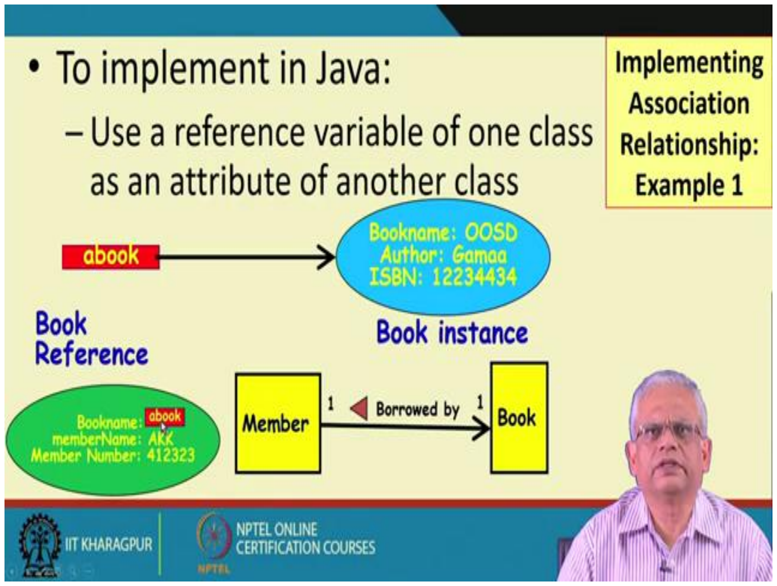
We were looking at the linked list representation. In a linked list, the association represent the association relationship is on different nodes of the linked list. So a node is connected to previous node and also a node is connected to the next node.

(Refer Slide Time: 02:04)


Reflexive Association: Example 4
Course
*
is pre-requisite for
*
has pre-requisite of
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES

Now let us look at another example of the reflexive association or the self-association or unary association, these terminologies are used interchangeably. Now here a course is a prerequisite for many other course okay. So A course has a prerequisite of many other course and A course is a prerequisite for many courses. A course has prerequisite of many courses and A course is prerequisite for many courses. Of course we should not write here twice this is the other reading direction but just for our reading convenience I have written here but just one is enough, either one of the reading directions should be enough. A course has prerequisite of many courses and A course is prerequisite of many courses.

(Refer Slide Time: 03:28)

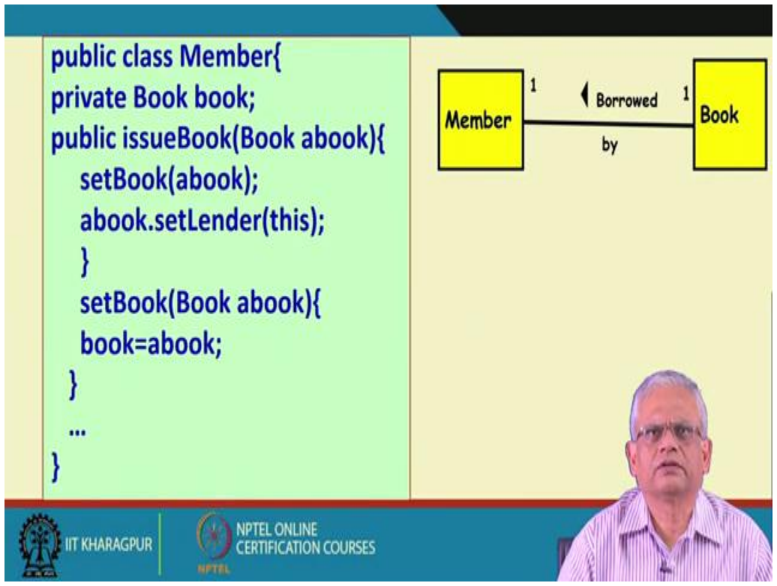

- To implement in Java:
  - Use a reference variable of one class as an attribute of another class

But now the question comes that suppose given a problem description we could identify the classes and the association relationship correctly, we could identify the multiplicity or cardinality of the association correctly, roles and so on. But now how do we write the Java code for that? Let us just take an example. A member borrows one book or A book is borrowed by exactly one member. Now how do we implement it in Java? We had been saying so far that if there is an association relationship then an object of the member class should be able to invoke the methods of the books class to which it is associated or linked, that is a member borrows any book the member can find out what is the tittle of the book that is has borrowed or what is the due date of the book.

So here the member has to invoke some method of the book class and to be able to invoke the method of the book class, the member class has to store the object reference or the ID of the book object. Remember that when a book object is created then it has a handle here which we called as the reference for the book, abook is the reference for the book. Now somehow when a member borrows a book, the book reference must be stored in the member class so that the member can invoke the methods of the book class and here it is a unidirectional that means only the member needs to store the reference of the book class and not the other way that is book does not have to store the reference of the member class.

But if it is a bidirectional association like a straight line without arrowheads or say arrowhead on both sides then we need to have the ID store at both ends. So somehow we have to program such that for a member number this that is a member object who borrows a book, abook we should have the abook stored as the book name. The book name should be an instance variable here and abook must be stored there. Now let us look at the code.

(Refer Slide Time: 07:04)

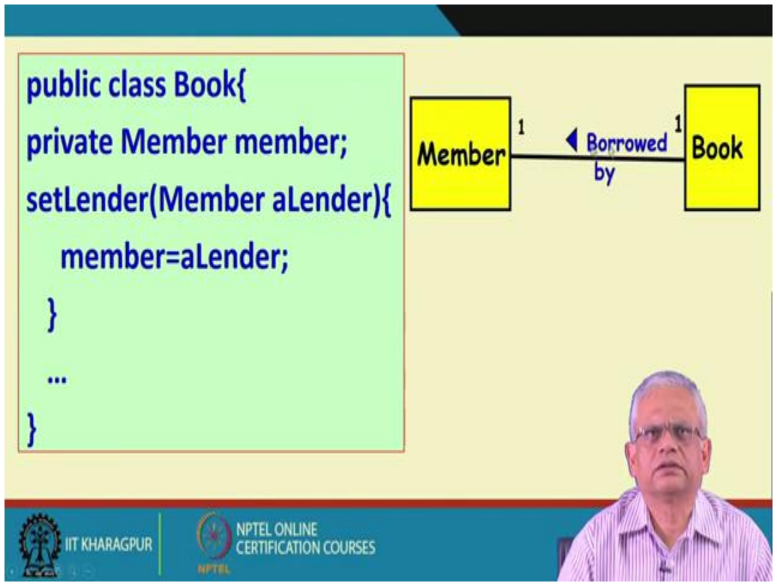


```
public class Member{
private Book book;
public issueBook(Book abook){
    setBook(abook);
    abook.setLender(this);
    }
    setBook(Book abook){
    book=abook;
  }
  ...
}
```

So this is a bidirectional association user base that means we need to store the book reference on the member side and also the member reference on the book side. So we do that here first we write the code for the member and here we have this instance variable, book and once the book is issued abook that is the book is issued the reference for the book is here and then we say setBook(abook). And in setBook() we make book = abook and also we set abook.setLender(). In setLender(), we must have a member attribute on the book side where we set the member reference this is the member reference.

So abook.setLender() and this should store the member reference on the book side. So this is the code for the member we have the book as attribute and whenever a book is issued the reference of the book is applied to this issue book with book reference and the book reference is stored here by calling the setBook(). In the set book we set it to abook and also during the issue we write abook.setLender(this) and this forms the link on the other side. Let us see the code for the book class.

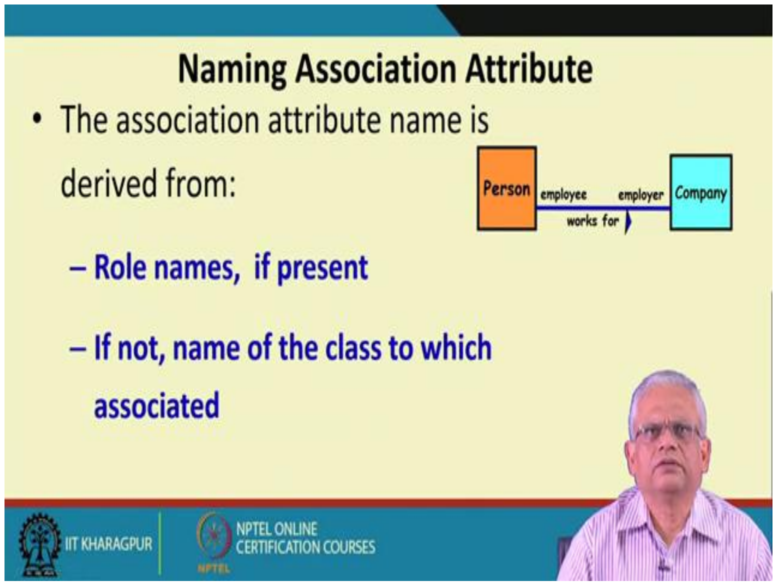
(Refer Slide Time: 08:58)



```
public class Book{
private Member member;
setLender(Member aLender){
   member=aLender;
  }
  ...
}
```

In the book class we have member as attribute and when the setLender() is called then the lender reference is stored in the book class as an attribute. So that forms the bidirectional association link between member and book and the book and member. The implementation is not very difficult but not as simple as the inheritance where you could just use the keyword extend and inheritance relationship between classes could be easily implemented. Here we need to have this implicit class variable and also carefully set the ID or the reference of the other object so that the link gets formed.

(Refer Slide Time: 10:07)

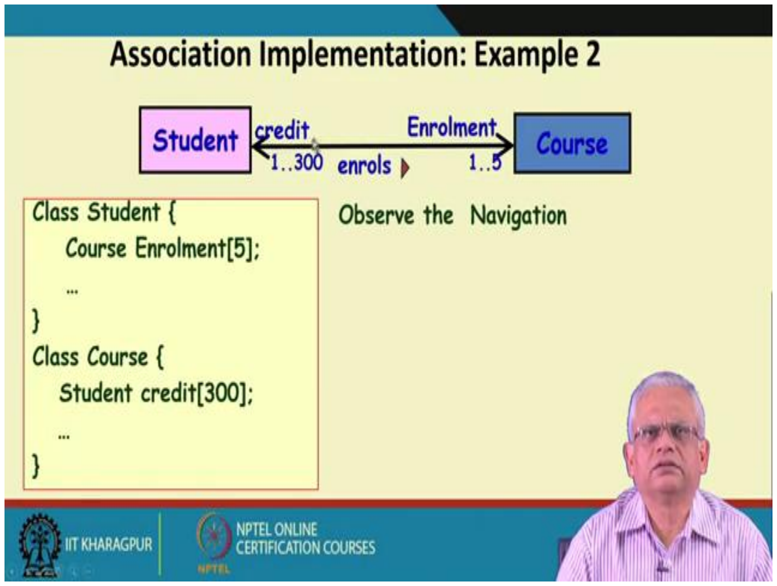


Naming Association Attribute

- The association attribute name is derived from:
  - Role names, if present
  - If not, name of the class to which associated

But is there any way that we can standardize the naming of the attributes? Standard way is that the role names if they are present they are used as the attribute, the implicit attribute. For example, on the person class we will have employer as the name of the attribute and we will set the specific company on the employer attribute and on the company side we will have employee as the implicit attribute and for the specific person we will set the employee here. But we had mentioned that the roles here are optional sometimes we may not have the roles specified in that case we use the name of the class as the name of the attribute. Let us look at some examples how do we name the implicit attribute in an association relation.

(Refer Slide Time: 11:24)



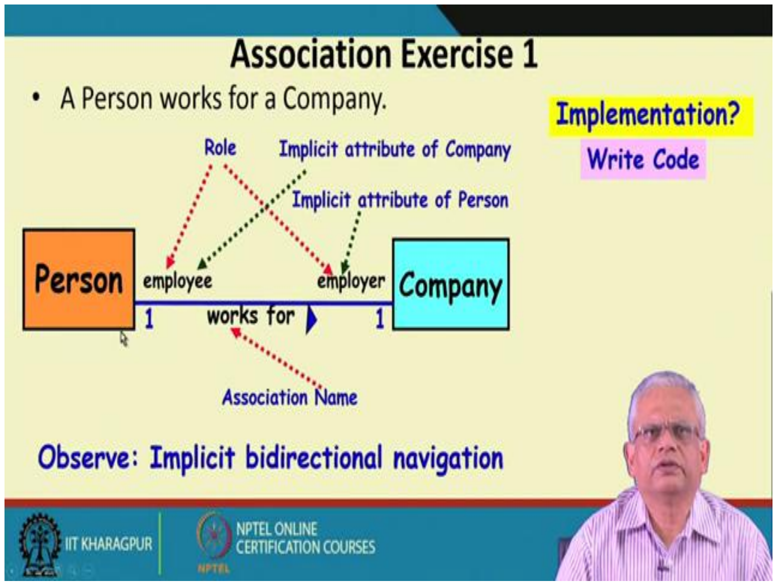
Association Implementation: Example 2

```
Class Student {
    Course Enrolment[5];
    ...
}
Class Course {
    Student credit[300];
    ...
}
```

Observe the Navigation

So let us look at this example between a student enrolls in 1 to 5 courses and a course has enrollment of 1 to 300 students. Now how do we write the code here and the name the attribute? Here, observe the navigation is bidirectional that same as just a straight line we has drawn arrowheads on both sides that is equivalent to just straight line. Now on the student side, student object can enroll in up to 5 courses, 1 to 5 courses. So we write here an array of 5. On the course side a course can have enrollment of 1 to 300 students. So we write here a student credits 300 sorry a course has 300 students up to 300 students crediting the course. So we have used this role here naming the attribute on the students side and the role here on the student side we have used that in naming the attribute on the course side.

(Refer Slide Time: 13:05)


Association Exercise 1

- A Person works for a Company.

Implementation?

Write Code

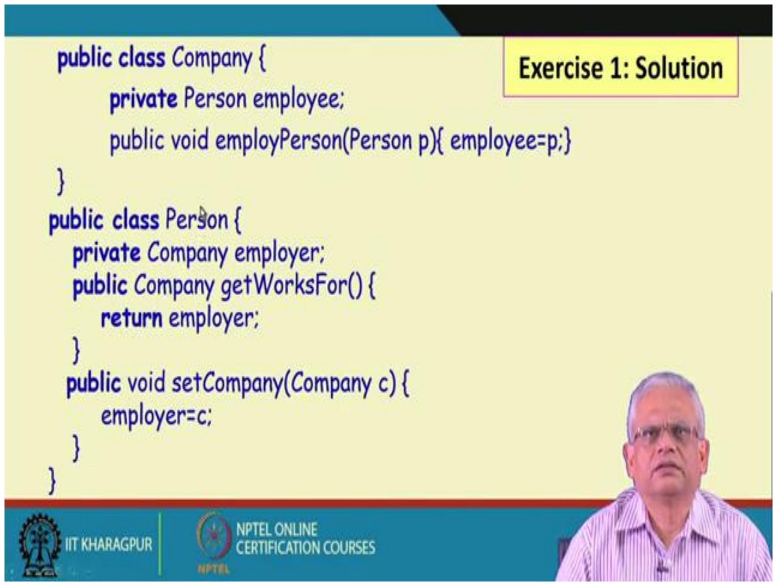
Observe: Implicit bidirectional navigation

Role
Implicit attribute of Company
Implicit attribute of Person
Person
employee
employer
Company
1
works for
1
Association Name
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES

Now let us do one exercise. A person works for a company and we draw the association relation, we drew the association relation, name the association relation as works for and it is bidirectional association, this is the reading direction, a person works for a company and the role of the person is employee and the role of the company is employer in this association. So the association name is the works for, employee is the role of the person in the association, the company's role is employer and these two become the implicit attributes i.e., employer is the implicit attribute on the person class and employee is the implicit attribute on the company class.

Now let us write the code for this, can we write the Java code for this assume that it is 1-1 that a company has one person working for it and a person works for one company. Can we write the Java code for this, so that is the cardinality here, the multiplicity or the cardinality is 1 on the both side. A company has one person working for it and A person works for one company. Please try writing the Java code for to implement this association between a person class and a company class.

(Refer Slide Time: 15:11)



Exercise 1: Solution

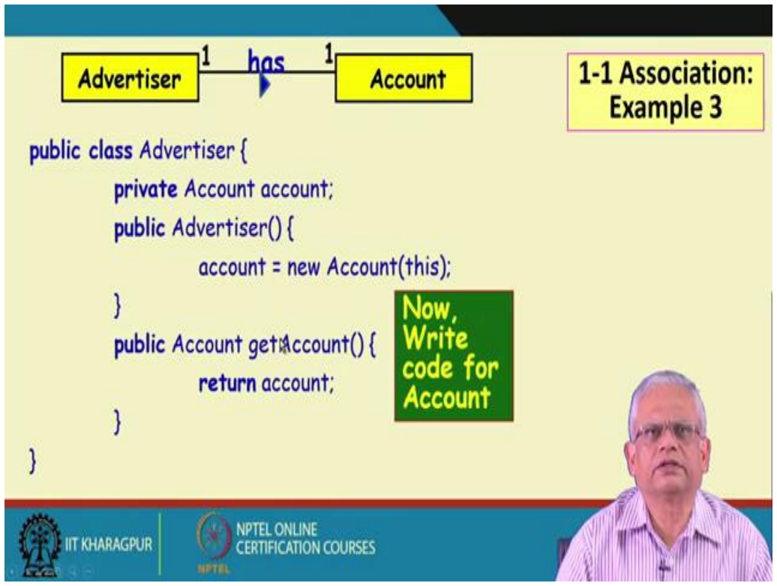
```
public class Company {
        private Person employee;
        public void employPerson(Person p){ employee=p;}
}
public class Person {
   private Company employer;
   public Company getWorksFor() {
      return employer;
   }
   public void setCompany(Company c) {
      employer=c;
   }
}
```

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES

We display the solution here, please compare with your solution. Here on the company class we have employee as the attribute, implicit attribute. We call it implicit because in the class diagram we do not show it by virtue of the association relationship this attribute appears here. And here as the company employs person, employPerson() method of the company is called with a person p. Then we set the employee to p and the attribute is set to p. Now on the person side we have the employer as the implicit attribute and we can invoke a method like getWorksFor() that for which company the a person works we call the getWorksFor() method for the person object and it returns the employer.

The employer is the implicit attribute storing who is employer of the person. And on the person side we can set company and here as you call the setCompany() with a company reference c, the c is stored here in the employer attribute. So the simple bidirectional association 1-1 association between a company and person can be represented in this Java code. Many case tools actually generate the Java code if we draw the class diagram. For example, the Argo UML, you can install it on your computer, draw this class diagram and observe that such code gets automatically generated and these case tools generate code in Java, C++ etcetera as per requirement.

(Refer Slide Time: 17:30)



1-1 Association: Example 3

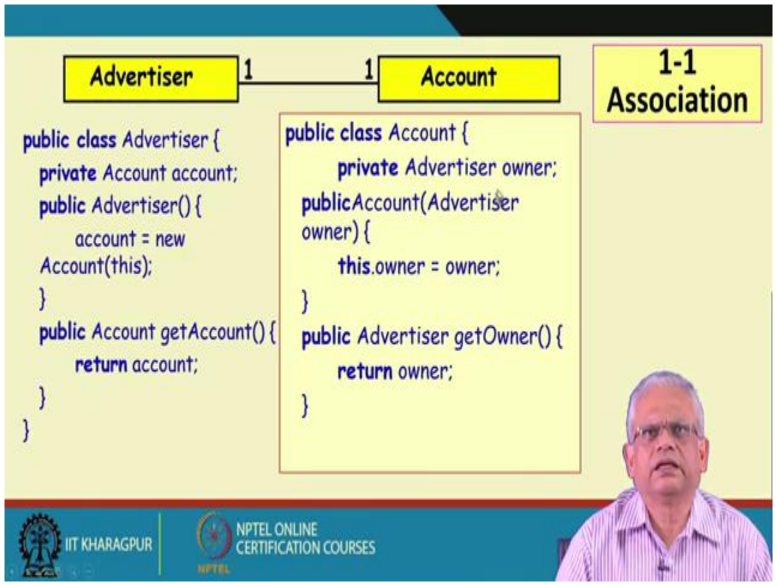
```
public class Advertiser {
        private Account account;
        public Advertiser() {
                account = new Account(this);
        }
        public Account getAccount() {
                return account;
        }
}
```

Now, Write code for Account

Now let us look at this association. An advertiser has one account and an account is associated with a single advertiser. Now we can write the code for the advertiser side and since this is the cardinality here is 1 that means when an advertiser is created we have to establish a link with an account class. And that we do here this is a possible implementation that when we call the constructor we create an object in the constructor itself an account is created, new account an account is created and this account is stored here. So the association is the link between the corresponding objects is created as soon as advertiser object is created.

The corresponding account is created and we have this 1-1 association and if we call the getAccount(), it returns the account. Now what about on the account side for the account class what code do we write? Please try to write the code for the account class based on the code we have written for the advertiser class. Please write the code and compare with the one that we display here because only by yourself writing the code and then comparing you can find out end mistakes that you commit.

(Refer Slide Time: 19:41)



Advertiser 1 — 1 Account

1-1 Association

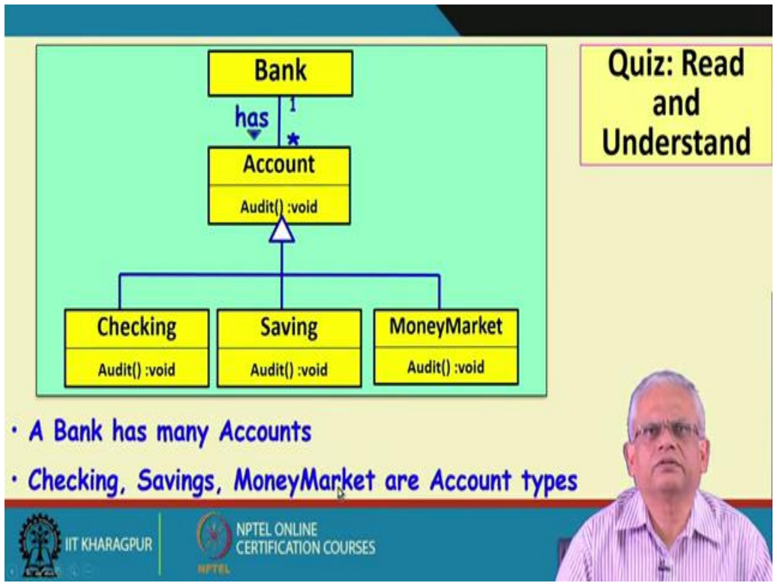
```
public class Advertiser {
  private Account account;
  public Advertiser() {
      account = new
  Account(this);
  }
  public Account getAccount() {
      return account;
  }
}
```

```
public class Account {
      private Advertiser owner;
  publicAccount(Advertiser
  owner) {
      this.owner = owner;
  }
  public Advertiser getOwner() {
      return owner;
  }
```

So this is the code that we have written for the advertiser class. The account since we did not have a role here we have use the class name itself as the implicit attribute and we created the account object whenever we created an advertiser object implicitly by through the constructor here. Now let us do that for the account side. On the account side we will have the advertiser as the owner or we could have written here advertiser and then we have this account once we call the Account() with the advertiser object we store that.

That is basically this here, look at this that we call the Account = new Account(this). So in the new Account(this), we are passing the reference for the advertiser object here to the account object that we are creating. And that is here the constructor for the account and we have this advertiser is owner and we are storing the owner for that and then we can have methods like getOwner(). So the 1-1 association gets formed in the constructor itself. The constructor once we call we pass the reference of the advertiser object and then the new object account object gets created and it is linked to the advertiser object.

(Refer Slide Time: 21:50)

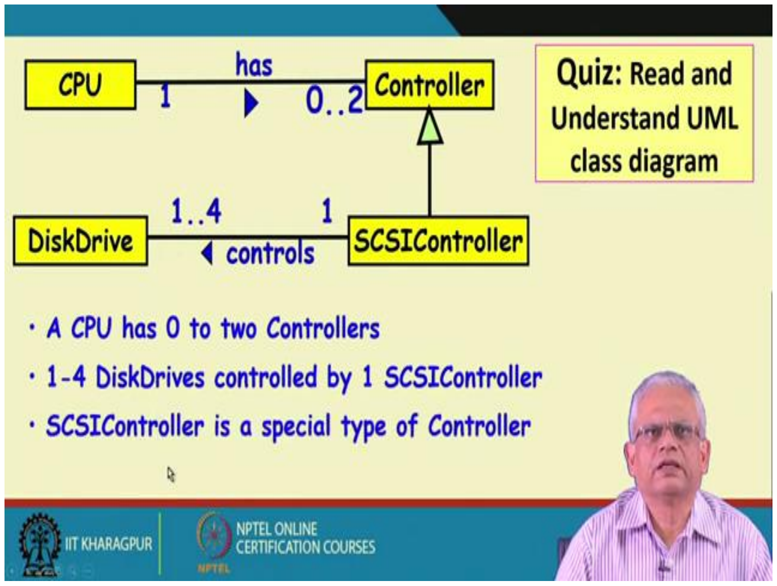


Quiz: Read and Understand

- A Bank has many Accounts
- Checking, Savings, MoneyMarket are Account types

Now let us display one diagram, class diagram with class relations. Please try to read and understand. How do we read this class relation where there are association and inheritance relation? So we can read here that a bank has many accounts. An account is with only one bank, an account is there with one bank and the accounts are of various type. The account can be a checking account, account can be a saving account, account can be a money-market account. A bank has many accounts and an account can be a checking account, saving account or money-market account.

(Refer Slide Time: 22:58)

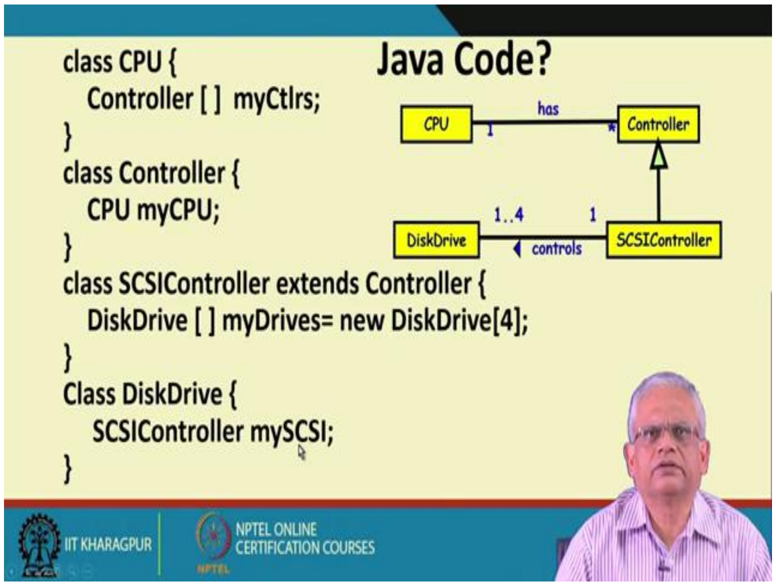


Quiz: Read and Understand UML class diagram

- A CPU has 0 to two Controllers
- 1-4 DiskDrives controlled by 1 SCSIController
- SCSIController is a special type of Controller

Now let us have another UML class diagram. Let us try to understand this diagram and read this diagram. How do we read this diagram? We will read this diagram let us start from here. A CPU has up to 2 controllers and a controller is there with exactly with 1 CPU. A SCSI controller is a special type of controller. A SCSI controller controls 1 to 4 disk drives but a disk drive is controlled by exactly 1 SCSI controller. A CPU has 0 to 2 controllers that is up to 2 controllers. 1 SCSI controller controls 1 to 4 disk drives. A SCSI controller is a special type of controller.

(Refer Slide Time: 24:11)

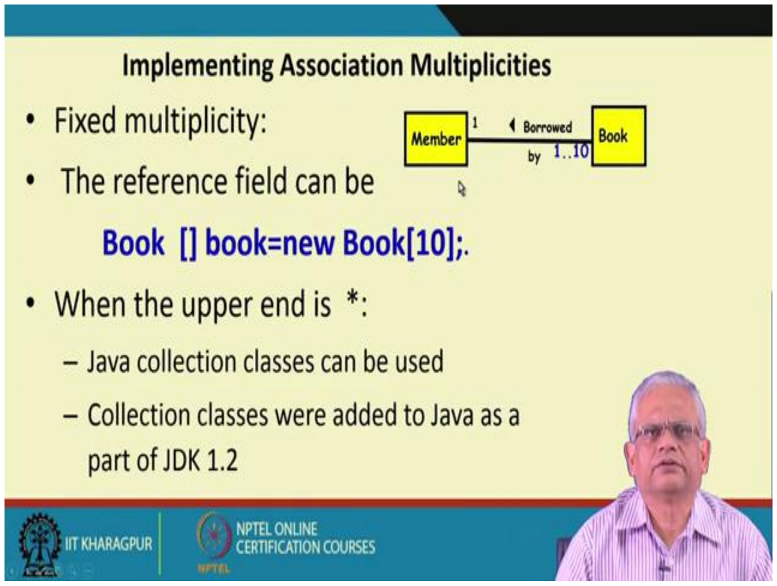


Java Code?

```
class CPU {
   Controller [ ] myCtlrs;
}
class Controller {
   CPU myCPU;
}
class SCSIController extends Controller {
   DiskDrive [ ] myDrives= new DiskDrive[4];
}
Class DiskDrive {
   SCSIController mySCSI;
}
```

Now assuming that we have understood and we can read the diagram, how do we write the Java code, how do we implement this class diagram in Java? If we have case tool, we draw this a case tool like Argo UML you can download, install on your computer, draw this diagram and see what code it generates. So here a CPU has many controllers and a SCSI controller is a special type of controller through the extends relations. And a controller is used in exactly one CPU. So this we remember here the CPU object to which it is connected. And a disk drive is connected to only 1 SCSI controller so that we saw on this side that a disk drive has is connected to exactly 1 SCSI controller.

(Refer Slide Time: 25:35)


Implementing Association Multiplicities

- Fixed multiplicity:
- The reference field can 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

But how do we implement the association multiplicities? Let us take an example that we have a member who can borrow up to 10 books and a book is borrowed by exactly one member. So on the member side we should be able to store the object reference for 10 books and that we store in an array like this. If it is 1 to 10 it is not very difficult but what about star? Instead of 1 to 10 we had star here that is we can borrow as many books as we like 0 to many books, many can be any number may be 10, 100 etc. So we cannot use an array because we cannot given an upper limit to the number of books we can borrow.

And fortunately we have the Java collection classes that can be used in this situation when we have the array size as variable. This is a conventional array but Java supports the collection classes and the collection classes were added to Java as part of the JDK 1.2. We are almost at the end of this session, in the next session we will see how the Java collection classes, various types of collection classes can be used to implement the multiplicities in an association. We will stop here. Thank you.