

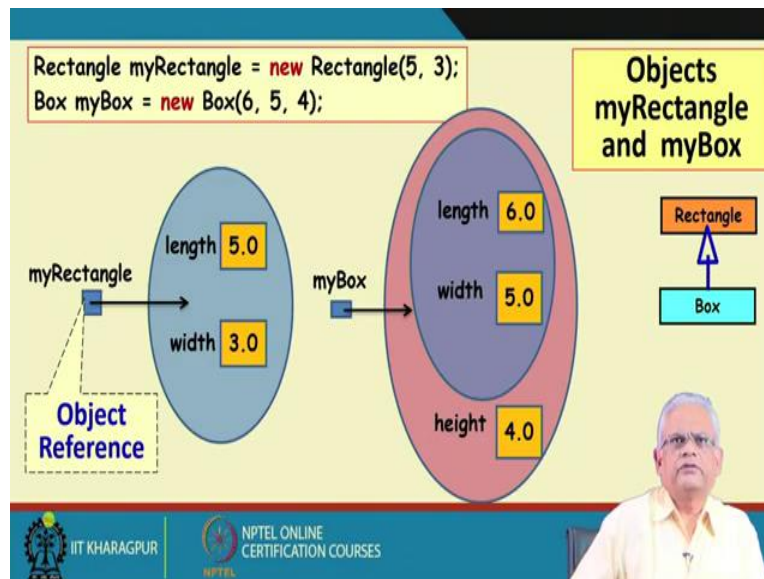
Object Oriented System Development Using UML, JAVA and Patterns
Professor Rajib Mall
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur
Lecture 8
Class Relations

Welcome to this lecture.

In the last lecture we had discussed about the class, the syntax, and the UML syntax. How it is represented in UML and also how it's instantiated into objects and we were trying to look at some examples of instantiation. We were discussing about the inheritance relation and we said that inheritance is very easy to program in Java.

We use the extends keyword to inherit a base class. Now, let's just proceed with what we are doing last time.

(Refer Slide Time: 01:10)

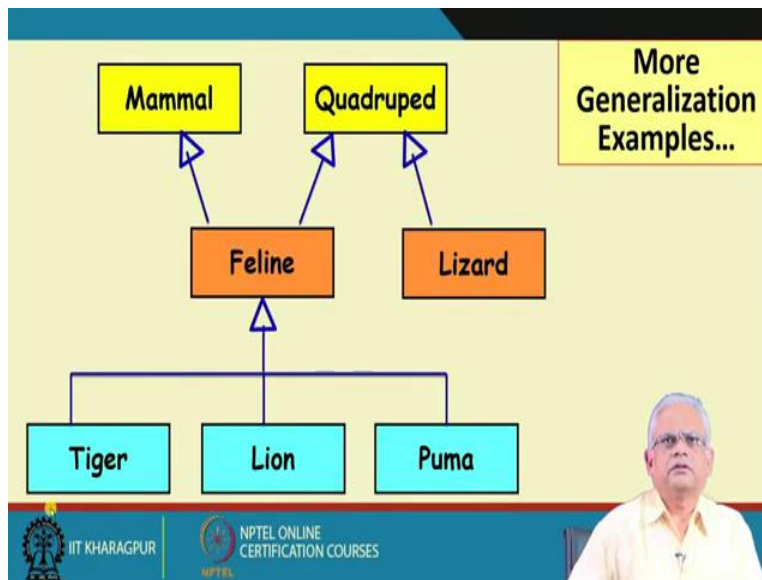


We were discussing the example of rectangle and box. We said that rectangle has length and width and as we define the inheritance relation with the keyword extends implicitly the box inherits the length and width and also we define a new attribute height and once we create an object, we create `myRectangle` using the keyword `new` and with the parameters 5 and 3 which are basically the length and width.

And we create a box using a new keyword and the object that we get the reference to that is myBox. But if we really visualize that will see that the actual object has its attribute length and width (as shown in the above slide). We have also got convenient handle when we create an object. We call it as the object reference. Here one of the object reference is myRectangle and we carry out all operations and the object through this object reference.

Similarly, when we instantiate the box with the new keyword, we get the object reference myBox which just points to the box object. And the box object internally has length, width which are the inherited attributes set to 6 and 5. In box object we have also a new attribute height which is defined here and this value is set to 4.

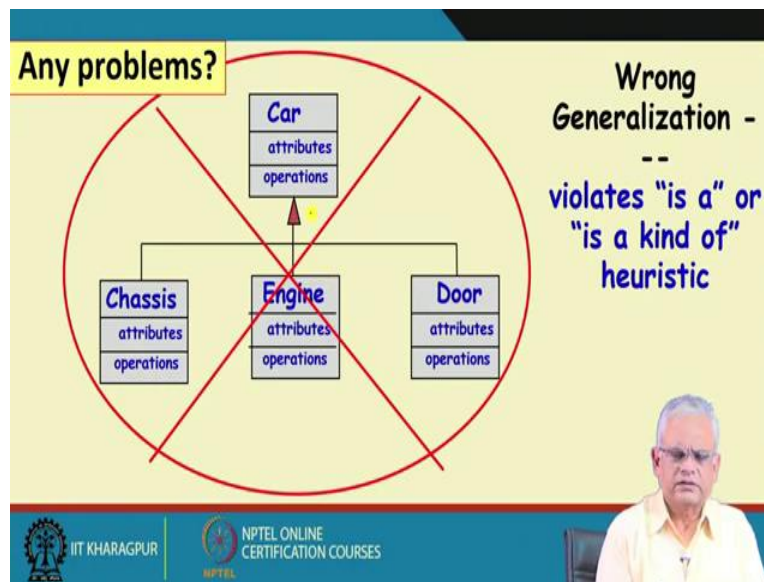
(Refer Slide Time: 03:53)



Now, let's look at some more examples of inheritance. This is an example of a Feline inherits both mammal and quadruped whereas lizard is animal which is derived from Quadruped. Tiger, lion and puma are the derived classes from the base class Feline. But, we can see here Feline is used as multiple inheritance it has two base classes, the mammal and quadruped. And we had said that, programming multiple inheritance is a bit troublesome.

In C++ we use things like virtual base class when we have multiple inheritance to avoid the problem of repeated inheritance where we get multiple copies of the same attribute. But, in Java the things are bit simpler in that multiple inheritance is not allowed, only single inheritance and you can implement an interface class.

(Refer Slide Time: 05:19)



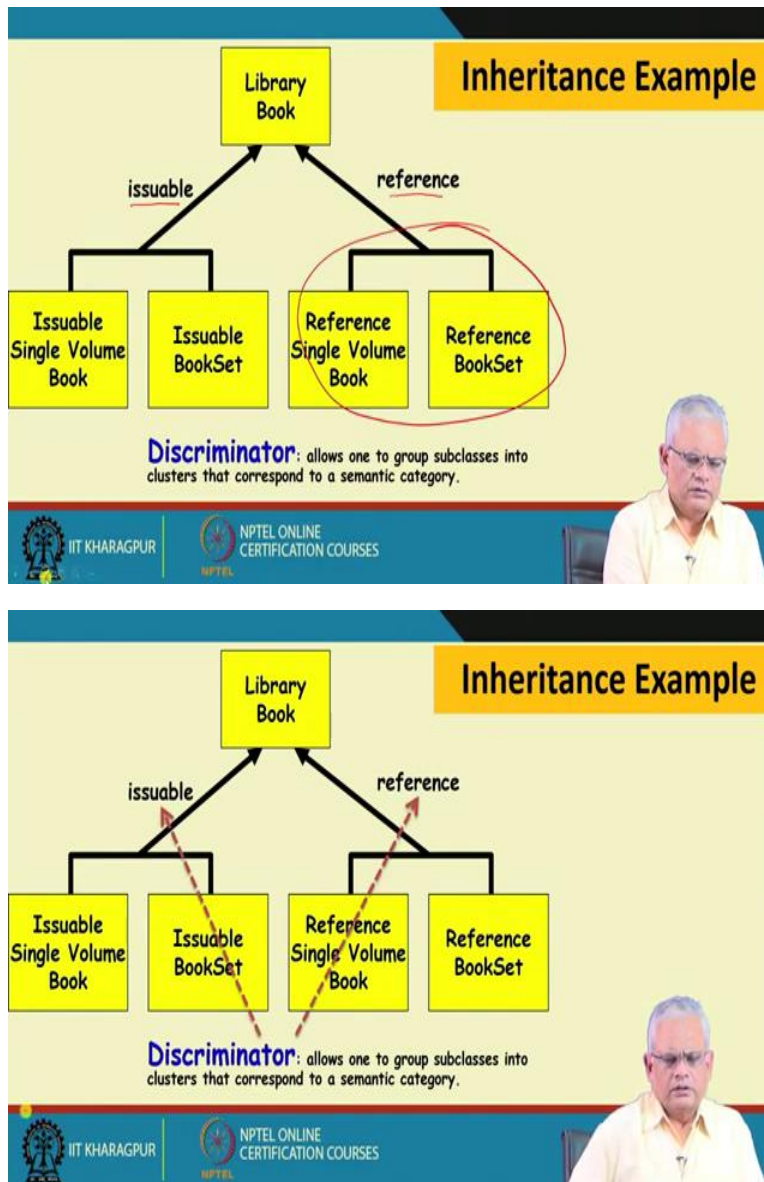
Now, let's look at this example (in the above slide). We have car as a base class. It has some attributes and operations and then we have derived class chassis, engine and door.

Is this okay, class relation? Does it appear correct that we have car as the base class and chassis, engine and door are the derived classes. Do you find it acceptable relation or is there any problem?

Okay, it is not really correct because we had said that whenever there is a generalization, we should be able to express this in the form of a 'IS A' relation. We should be able to say chassis is a car, engine is a car and door is a car which is looks odd, it's not correct. So, it's a wrong generalization. It violates the 'IS A' or 'IS A' kind of heuristic that we had mentioned earlier.

Whenever we want to test inheritance hierarchy that, we check whether can express that using the 'IS A' heuristic to find that whether it's a correct generalization. This is not correct; the classes here have different relation. A car consists of chassis, engine and door, it is not inheritance relation, it's an aggregation.

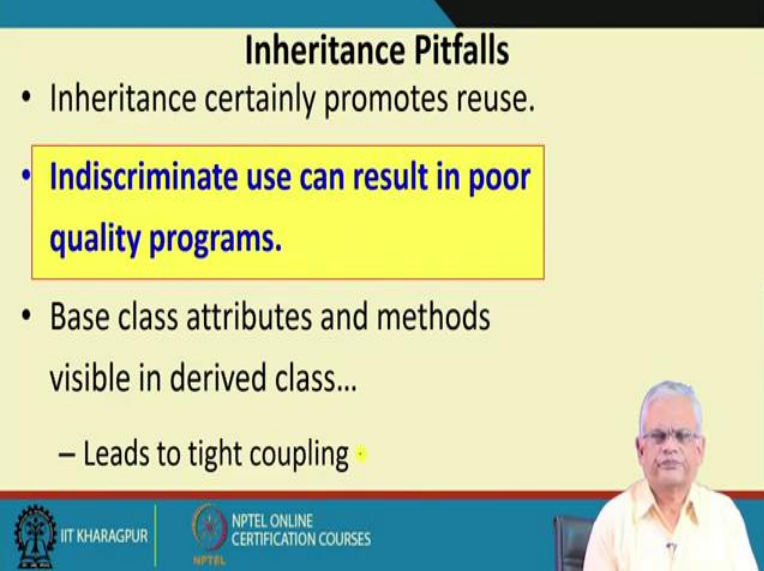
(Refer Slide Time: 07:38)



Now, let us look at some more inheritance relation (in the above slide). The base class here is the Library Book and there are derived classes which are Issuable single volume, Issuable BookSet, Reference single volume book and Reference BookSet. But we can use a discriminator to help us organize this class hierarchy in much better by using these discriminators issuable and reference (as shown in the above slide). The discriminators are here issuable, and reference. For Issuable Single Volume Book, and Issuable BookSet ‘issuable’ discriminator used and for Reference Single volume Book and Reference BookSet ‘reference’ discriminator used. To represent

inheritance, we used inheritance arrow. We write the discriminator which allows us to group classes based on some semantic category.

(Refer Slide Time: 09:17)



Inheritance Pitfalls

- Inheritance certainly promotes reuse.
- **Indiscriminate use can result in poor quality programs.**
- Base class attributes and methods visible in derived class...
 - Leads to tight coupling *

The slide features a yellow background with a blue header and footer. A small video feed of a man in a light blue shirt is visible in the bottom right corner. The footer includes the logos for IIT KHARAGPUR and NPTEL ONLINE CERTIFICATION COURSES.

So, we understood that inheritance is simple, easily programmed using the extends relationship in Java. It promotes reuse we define it once and reuse in the derived classes. But, let's be aware that if we use the inheritance relationship too often it can result in poor quality programs.


As we proceed further will have several examples will see what problems actually arise. We will examine this problem in more detail. We will see that as the class hierarchy becomes deeper the attributes and methods are inherited in the lowermost or the leaf level classes has too many methods and attributes typically happen in a library class once we inherit.


We do not know too many methods, too many attributes. So, that's one part of the problem, there are other problems as we proceed will see. But, at this point will just remember that inheritance is good, it is simple, promotes reuse. But we need to be careful in using the inheritance and the problems that arise we just now have a hint that it leads to tight coupling of the methods. Large number of methods and attributes breaks encapsulation. But the exact problems will realize as we proceed in this course.


(Refer Slide Time: 11:48)


- How implemented in program?
- Enables objects to communicate with each other:
 - One object must “know” the ID of the corresponding object in the association.
- Usually binary:
 - But in general can be n-ary.

Association Relationship




IIT KHARAGPUR


NPTEL ONLINE
CERTIFICATION COURSES



Now, let's look at the association relationship between classes. We said that there are four types of relation between classes that can exist. The first one is inheritance relation between classes. We just use the extends when one class is derived from another class for the implementation of the inheritance relationship.

Now, let's look at the association relationship. The first question is that, can you give some examples of association relationship and then will address this question that how it is implemented in program. Generalization is through the extends keyword but how do we implement the association relationship. Is there a keyword?

No, we don't have a keyword but then we can program it. We will just see how to program it. Let's first understand the implication of the association relationship, when one object is associated with another object. For example, let say a student registers in a course. So, the student and the course are associated because the student has registered. And in this case in the student class, we should be able to identify or printout what are the courses he has registered, what are the names of those courses, what are the credit structures, what are the syllabus of these courses, and so on. We should be able to printout all these.

And to be able to do that we should be able to communicate with the classes with which the student is associated and also on other hand for each course we should be able to see the student's information who registered for the course.

The student class is associated with the course. We represent in a straight line and we write that the student registers and we write a reading arrow to say that, the student registers in a course. The meaning of this is that from a student class we should be able to identify all the courses he has registered. We should be able to have some way to identify the courses, courses syllabus, credit structure and similarly, on the course side we should be able to identify who are the students who have registered in this course. And to be able to do that, that the course should be able to identify who are the student and the student should be able to identify what are the courses he has registered, we should have the reference, the object reference of the students towards in the course class. Only then, we will be able to identify who are the students registered. Similarly, on the student side we should have the course object references stored inside the student class. Then only we will be able to identify, what are the courses with which the student has registered.

This student and course association is an example of binary relation. We can think of many more examples of association relation but we will see that in general it can be n-ary. That is, it can be 1-ary that is one class associated to itself, can be a 2-ary or binary, we can have 3-ary or ternary, 4-ary or quaternary, and so on.

(Refer Slide Time: 17:00)

Association – An Example

- In a home theatre system,
 - A TV object is associated with a VCR object
 - It may receive a signal from the VCR
 - VCR may be associated with remote
 - It may receive a command to record

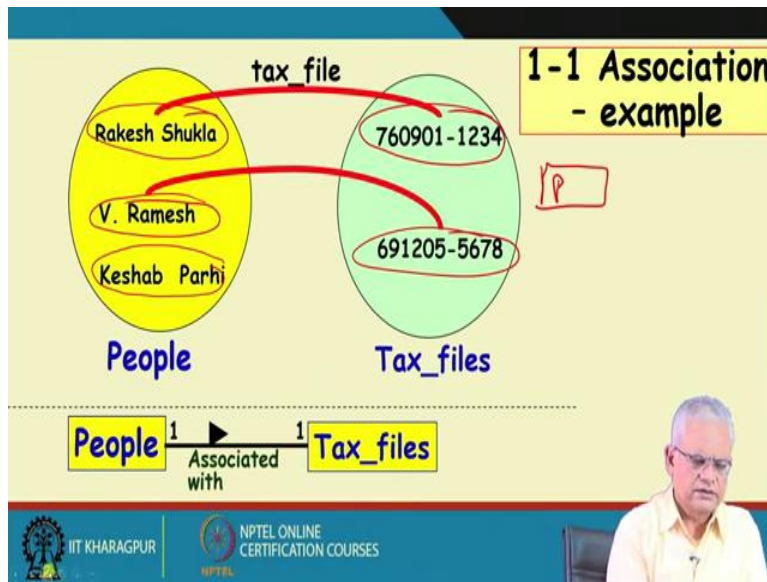
```
graph LR; Remote[Remote] -- commands --> VCR[VCR]; VCR -- Connected to --> TV[TV];
```

IT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, let's look at more examples of association (in the above slide). This is a home theatre system. Here, we have a TV object. The TV class is associated with the VCR class and to be able to program it. The VCR must contain the reference for the TV class, to be able to invoke the methods of the TV class. Similarly, maybe necessary for the TV class to store the reference of the VCR class and similarly the VCR maybe associated with the remote because the remote commands the VCR. The remote is associated with the VCR and the VCR is connected to the TV. Representation shown in the above slide.

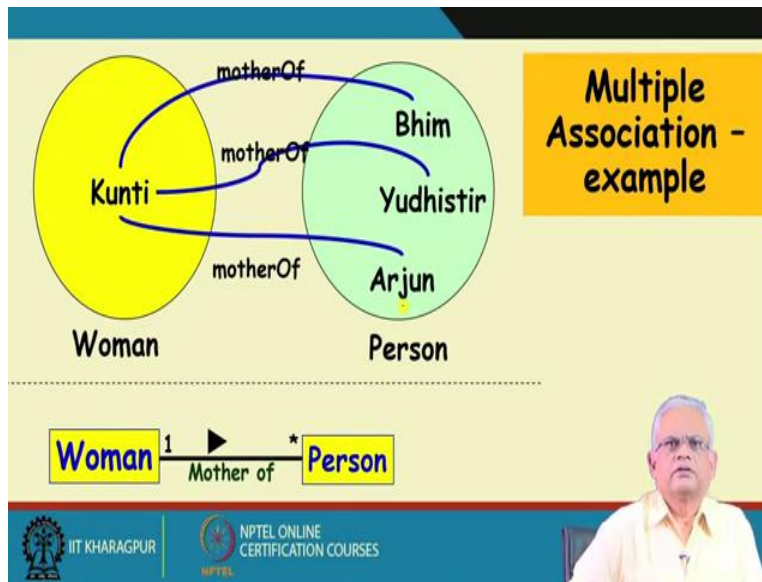
Name of the association between VCR and TV is connected and the reading direction shown. We read as VCR connected to TV and we will look at the multiplicity 1-1 that is written near the class. We can see it is a unidirectional association. Similarly, The remote is associated to VCR but not vice-versa and the name of the association relationship is commands and reading direction is given in the form of arrow and we read as the remote commands VCR.

(Refer Slide Time: 18:51)



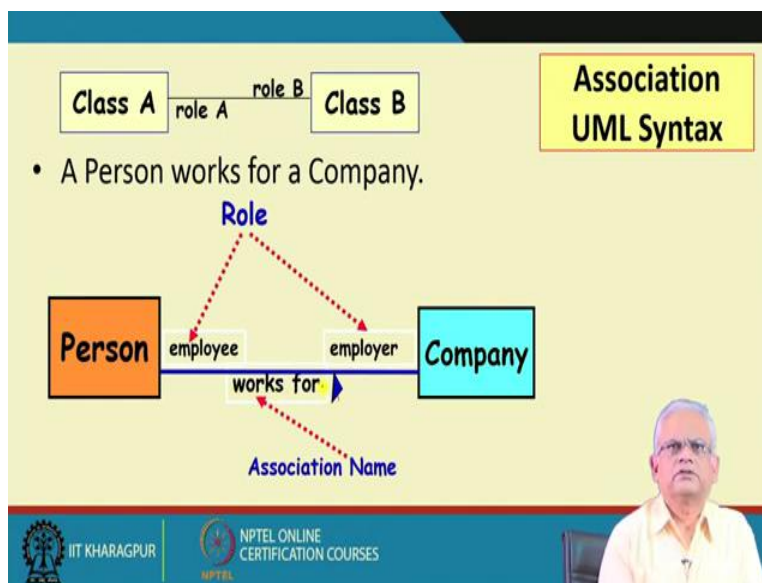
When the classes are associated the objects of another class they are actually linked to the object of another class. The People class is associated to the tax file (in the above slide). The meaning of this in the object space is that, for every object here on the people class we have a corresponding tax file here to which it is associated. For example, for V.Ramesh object we have the tax file 691205-5678 but it may happen that for some objects, we do not have a corresponding tax file. For example, Keshab Parhi object does not link to any tax file. So, if we have this kind of relation between two classes it may so happen that some of the objects here do not have a corresponding object here but, at most one object of People class is associated with one object of Tax_files class.

(Refer Slide Time: 20:27)



Now, we may have also multiple association. In multiple association, an object of one class is associated to multiple objects of another class (as shown in the above slide). To express multiple association, we use the '*' relation here. One woman can be mother of many persons and if we look at the above slide example, o the persons Bhim, Yudhistir and Arjun is associated with Kunti who is the mother of these persons here.

(Refer Slide Time: 21:10)

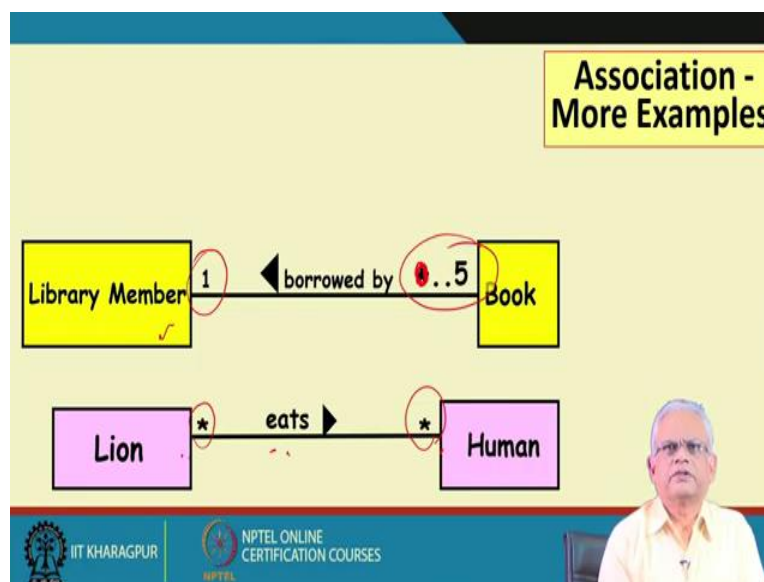


Now, let's elaborate the UML syntax of association a little more. We said that a straight line indicates association relationship between Class A and Class B (as shown in the above slide).

But we can also write the role of the class A in this association relationship and role of Class B in this association relationship.

Now, let's elaborate that with the one example that, a person works for a company (in the above slide). Person is a class, company is another class they are associated. So, a straight line drawn between them, where the name of the association is 'works for' and the reading direction shown. And we also have written the roles of these two class in this association: we have company as the employer and the person as the employee. Role of the person is employee; the person works as the employee and the company is the employer in this association. So, in the UML syntax we can optionally write the name of the role in addition to the name of the association relation and the reading direction.

(Refer Slide Time: 22:57)



Now, let us try to read this, how do we read this?

The reading direction is from Book to Library Member. We can read as a book is borrowed by a member. When we read it we start with a single book. Remember when read from one class to another class, our first-class object will be referred as 'a' or 'an'. It represents single. Single book is borrowed by a member or a book is borrowed by a member.

Now if we read in reverse direction, we will read as a library member borrows 0 to 5 books. A library member borrows 0 to 5 books in reverse direction and a book borrowed by single member

in in reading direction. Let repeat this that in the association relationship, we represent it by multiplicity (in the above slide, side of class the number represent multiplicity) that each book here is associated with how many objects of the other class 'Library Member'. We always read a book borrowed by a library member in reading direction. We can read it in the other direction by reversing this phrase here. We say that, a library member borrows 0 to 5 books (because, in Book class multiplicity is 0..5).

Now, let's look at another example of Lion and Human. How do we read this?

Please try to read this diagram. As we said earlier to read this diagram, we need to first read in the direction of the arrow (reading direction), and we identify one object of this class is associated with how many objects of this class.

So, we read in reading direction as a lion eats many humans (because in Human side '*' multiplicity used) and on the other direction, we read a human is eaten by many lions (because in Lion side '*' multiplicity used). We will have more examples, but then I hope that the multiplicity that given here is clear. The multiplicity is to identifies one object is associated with how many objects of another class.

We will have few more practice problems and also, we will look at how the association relationship is implemented in Java. And also, we will look at the unary and ternary associations with more examples.

We will stop here; we will continue in the next class.

Thank you.