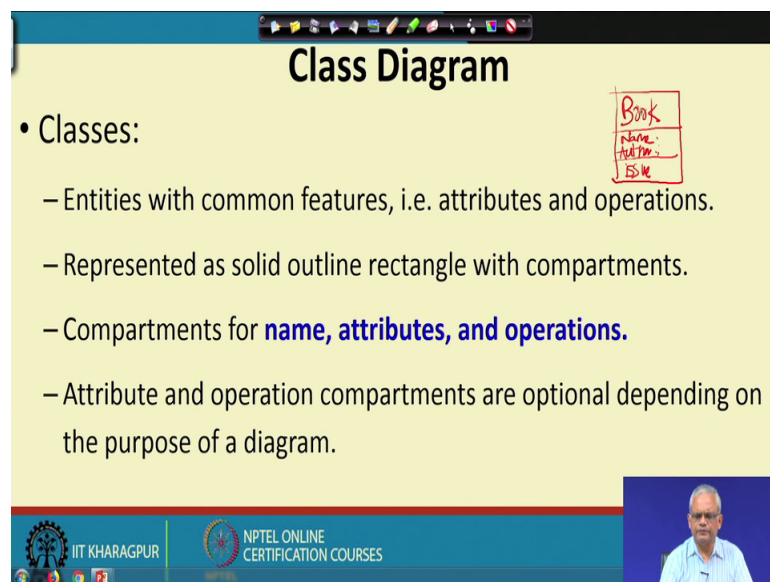


**Software Engineering**  
**Prof. Rajib Mall**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture - 34**  
**Inheritance Relationship**

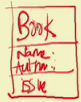
Welcome to this lecture, in the last lecture we had discussed how to develop use case diagrams and given a specific problem how does one go about developing use case. And we said that we need to practice that with several examples and exercises, to get some insight into how to develop the use case model and to document. In this lecture we will discuss about the class diagrams.



(Refer Slide Time: 00:52)




**Class Diagram**

- **Classes:**
  - Entities with common features, i.e. attributes and operations.
  - Represented as solid outline rectangle with compartments.
  - Compartments for **name, attributes, and operations**.
  - Attribute and operation compartments are optional depending on the purpose of a diagram.



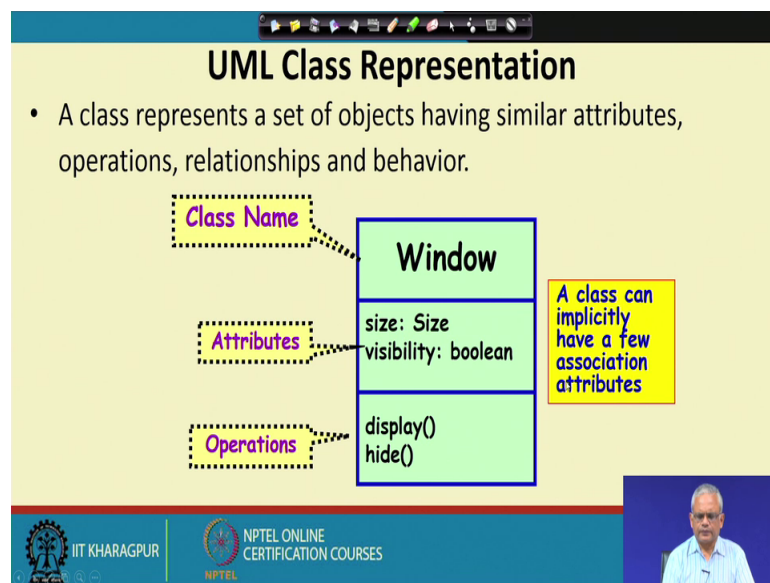
 



In the class diagram with document the entities with common features, that is all the objects in a problem which are similar attributes and operations the constitute a class. For example, in a library system the book objects have similar attributes and operations for example; each book has a name and ISBN number and so on. And each book has similar operations like issue a book, returned the book and so on; create the book and so on. And each class is represented using the solid outline rectangle with compartments, we will use the solid outline rectangle with compartments and we will write the name of the class like book at the top compartment.

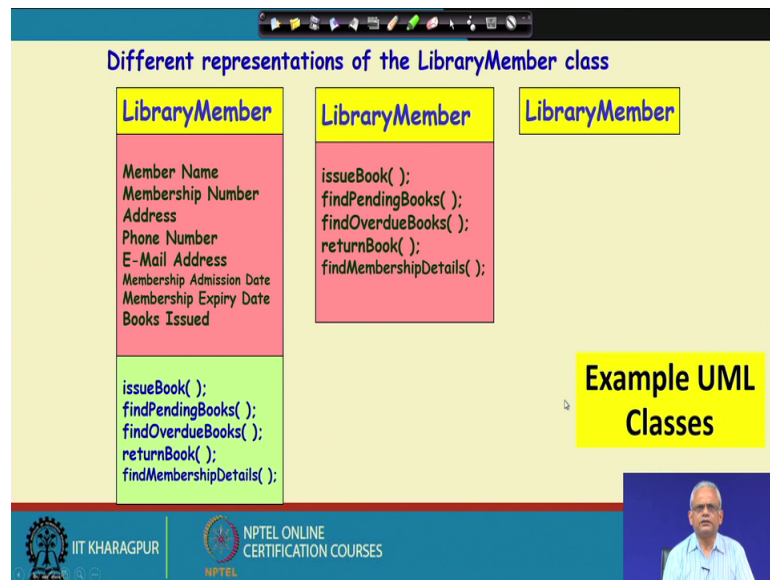
We will write the attributes here for example, the book name, the author, the ISBN number etcetera and then in the last compartment will write the operations for example, create book, issue book, returned book and so on. But then we do not show all the three compartments always we might, in some cases write only one rectangle containing the name of the class that is book. Just notice here that the name of the class is singular we do not name the class books. So, we can have just one compartment just the class name, we might have the class name and the methods or the operations that are applicable on this book or we might have all of them together with three compartments.

(Refer Slide Time: 03:19)



So, this is the representation, name of the classes window, the attribute name is size which is an instance of the size class visibility is a Boolean. The operations are display at hide the class name is at the attributes and these are the operations, but later we will see that there are some attributes which are not shown, but there present we called them as association attributes as we proceed it will become clear.

(Refer Slide Time: 04:03)



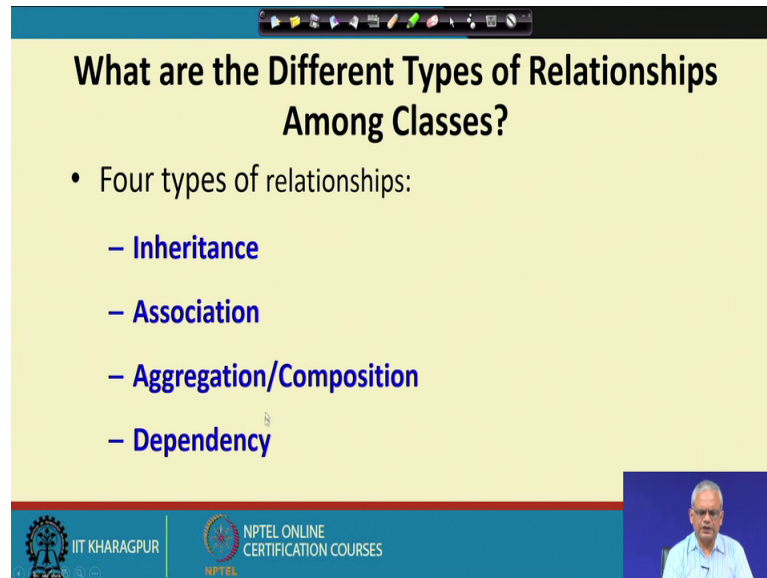
As we were saying that the same class can be represented with different details, we might just represent the class name sometimes we might represent the class name along with the operations and that class or we might represent the class name along with the attributes and the operations. But, then one would ask that when does one use this representation, which is the name of the class. When does one use the second representation, which is the name with the operations and the full representation using the name attributes and the operations. To answer this question when we start the object oriented design process as will be seeing shown that how to go about doing the object oriented design, initially we identify only the class names that is which are the classes we identify the class.

So, in the initial stage of the design process we represent the classes using this representation, as we proceed with the design steps we assign the operations to the classes and then we know about which classes or which operations and in that stage of the design we represent using the class name and the operations. And towards the end of the design we identify the attributes that the classes should have and that is why these are more complete and can be easily translated to code, the class name what are the attributes of this at the class member and then the methods of the class.

So, to start with the design process as we start we use the simplest representation, because we identify only the classes and much later in the design process we identify the

operations that for each class. And we use this representation at towards the end of the design we have this complete representation and this can be easily translated to code.

(Refer Slide Time: 06:59)



**What are the Different Types of Relationships Among Classes?**

- Four types of relationships:
  - **Inheritance**
  - **Association**
  - **Aggregation/Composition**
  - **Dependency**

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, let us answer this question that in any object oriented implementation there are many classes, many objects which we from which we construct the classes. Now a typical software might have a does not classes or maybe several does not are 100's of classes. But the classes are not independent, the classes are related. What are the possible relations that can exist among the classes? Actually there are 4 types of relations that may exist among the classes in a system. The first type of relation is inheritance, one class may inherited from the base class.

So, the base class can from the base class you can inherited derived class. Between two classes and association relationship may exist please give an example of a association relationship between a between two classes, couple of minutes we will just see some examples of association relationship between two classes. There may be an aggregation or composition relationship between two classes if you can give some example regarding aggregation or composition relationship between two classes.

Some realistic examples that will be really nice otherwise in couple of minutes we will see, some examples of aggregation and composition relationship and what is the implication of this with respect the implementation of the software. The fourth type of relation between classes is dependency, so one class may depend on another class we

will see short while from now, that when does a class is said to depend on another class we will see that there are various reasons why a class may be depended on another class. Now let see this 4 type of relationship in some detail.

(Refer Slide Time: 09:35)

### Inheritance

- Allows to define a new class (derived class) by extending an existing class (base class).
- Represents generalization-specialization
- Allows redefinition of the existing methods (method overriding).

```
graph BT; Undergrad --> Students; Postgrad --> Students; Research --> Students; Faculty --> LibraryMember; Students --> LibraryMember; Staff --> LibraryMember; style LibraryMember fill:#ffff00,stroke:#333,stroke-width:1px; style Faculty fill:#ffff00,stroke:#333,stroke-width:1px; style Students fill:#ffff00,stroke:#333,stroke-width:1px; style Staff fill:#ffff00,stroke:#333,stroke-width:1px; style Undergrad fill:#ffff00,stroke:#333,stroke-width:1px; style Postgrad fill:#ffff00,stroke:#333,stroke-width:1px; style Research fill:#ffff00,stroke:#333,stroke-width:1px;
```

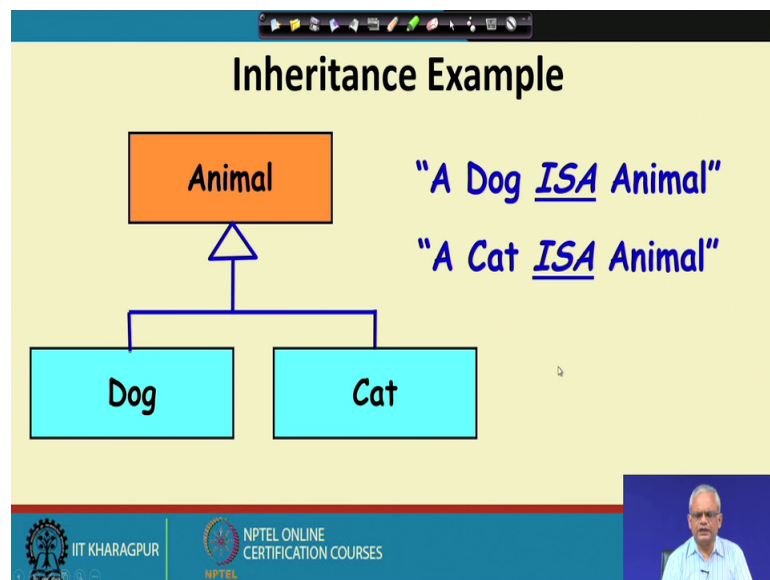
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

First we will look at the inheritance relationship, inheritance is a powerful mechanism in object orientation it allow just to define a new class called as a derived class by extending the base class. Using the inheritance we can represent the generalization specialization relationship. The base class is the general class and the special class the specialization classes they add few more attributes are methods to the general class. While, we can add few more attributes to the general class, but we can at the same time redefines some of the existing classes that is called as overriding sorry.

Some of the existing methods of the base class and that is called is overriding. Just look at this example the library member class is a base class, there are different types of library member, but some characteristics are common among them for example, all of them has a have a library membership number. All of them can issues certain number of books, they can return book and so on. So, the common characteristics among different members here are represented as the base class and the faculty, student and staff these are the child classes are the derived classes. And each of these add few more attributes and operations to the base class and you have another hierarchy here, that the students can be undergraduate student, postgraduate or research students.

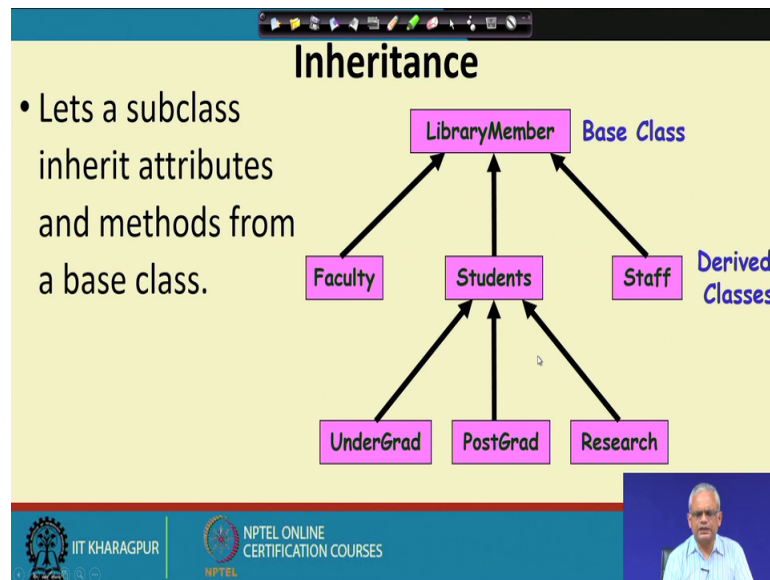
So, the characteristics that are specified here in the student class that is the method and attribute are all shared or all of these classes here they have the same characteristics the attributes and methods, but each of these add few more attributes and maybe methods, this is an example of inheritance.

(Refer Slide Time: 12:14)



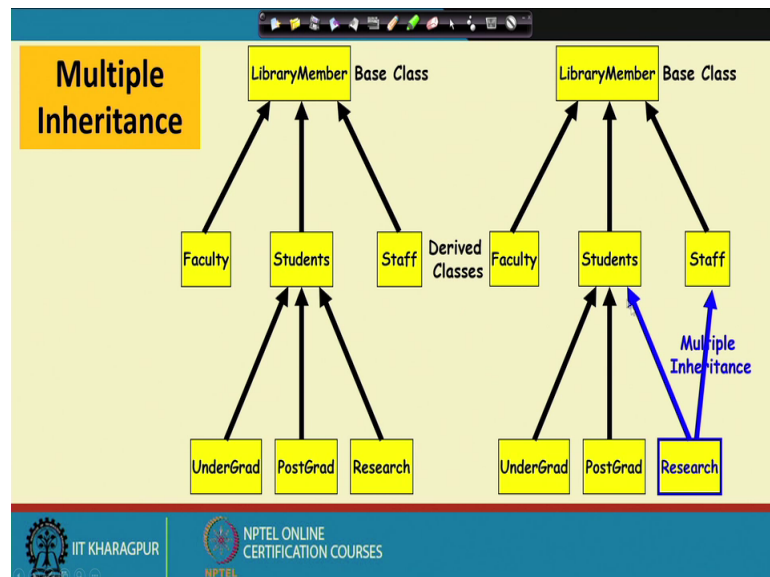
A dog is a animal, a cat is a animal we have underlined the ISA here to denote that sometimes the inheritance relation is also called ISA relation. Animals we know they have some characteristics, there living, they take food, the die etcetera and those characteristics are all said by the different animals. So a dog is a animal, a cat is a animal and therefore, sometimes the inheritance relationship is called as a ISA relationship.

(Refer Slide Time: 13:11)



A derived class inherits the attributes of the base class and this helps in reusing the code the library member may have some attribute and method, we do not have to repeat those rewrite those again in all this classes these are automatically inherited. When we use the inheritance mechanism and that helps in code reuse, you can have multiple inheritance.

(Refer Slide Time: 13:51)



In which a class may inherit from two base classes, so the research student has the characteristic of a student and also a staff so this is called as the multiple inheritance.

(Refer Slide Time: 14:10)

### Inheritance Implementation in Java

- Inheritance is declared using the "extends" keyword
  - Even when no inheritance defined, the class implicitly extends a class called Object.

```
class Person{
    private String name;
    private Date dob;
    ...
}

class Employee extends Person{
    private int employeeID;
    private int salary;
    private Date startDate;
    ...
}

Employee anEmployee = new
```

The diagram shows two class boxes. The top box is labeled 'Person' and contains attributes '- name: String' and '- dob: Date'. The bottom box is labeled 'Employee' and contains attributes '- employeeID: int', '- salary: int', and '- startDate: Date'. A red arrow points from the 'Employee' box up to the 'Person' box, indicating that Employee inherits from Person.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, let us see how inheritance relationship can be implemented in Java, of course, if you are familiar with other object oriented programming languages that will be very similar to here just a small change in the keywords and so on. But the concepts are very very similar let us assume that this is the class hierarchy that we have in our design. Person is a class and we have name and date of birth is the attributes and employee is a derived class, employee is a special type of person. So this is a general class, this a special type of person is an employee which has not only the name date of birth it has few other attributes as well for example, employee id, salary, start date for the employment and so on.

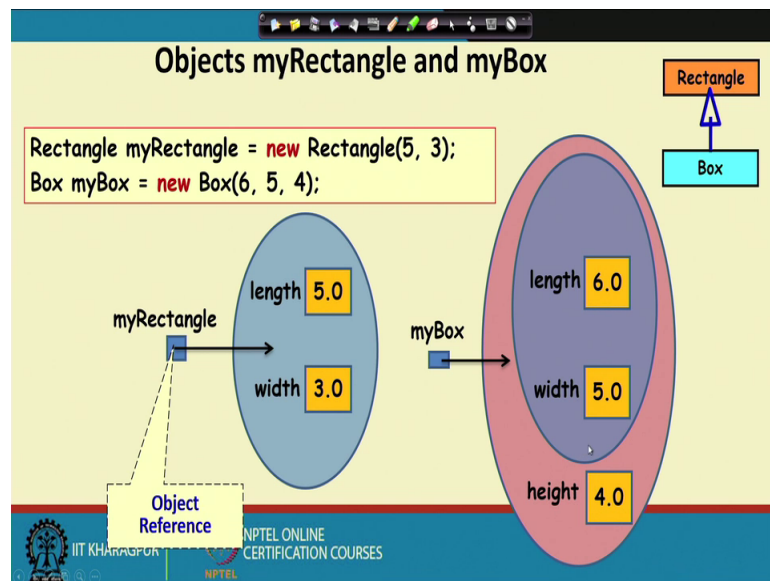
How do we write the java code for this? That is very straight forward we write the definition for the class person which is private string name, private date of birth. So, straight away we can translate this into the code similar is with C plus plus or any other object oriented language and for the derive class we use the keyword extends in Java. The class employee extends person and if you write extends then automatically name and date of birth are inherited by the person class and we have few attributes that are defined here we write them here private employee id, private int salary, private date start date.

So, inheritance implementation is straight forward we have a keyword extends in Java and we can very easily implement such a relationship in Java by simply using the extends relation, but we will see that the other type of relationship between classes like



association, aggregation and so on. We do not have single keyword using which you can implement those relations we need to do something else, but it would have been nice are very simple if we just had a keyword like a extents these the simplest implementation of a relationship by just using a keyword. And once we create the two classes here employee class we can instantiate an employee class into objects write we write employee, an employee is equal to new employee. So, the new keyword helps us create the object of employee class and then name of this object is an employee.

(Refer Slide Time: 17:41)

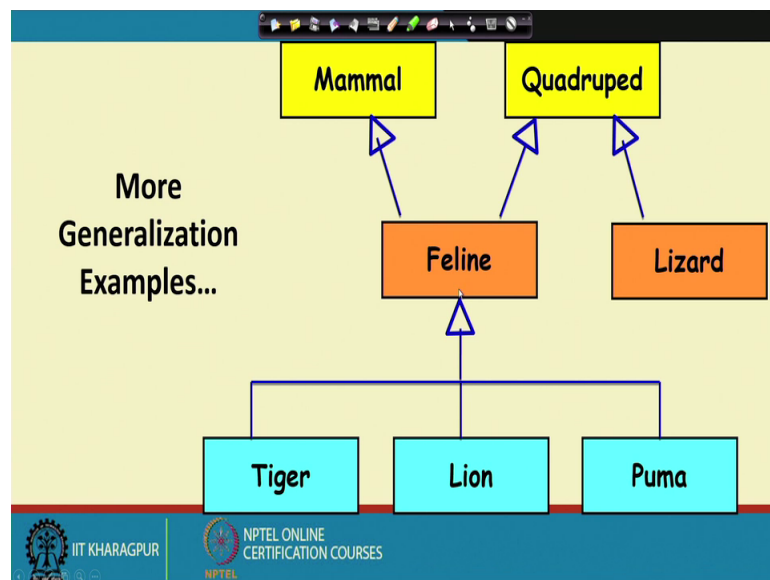


Let say we have we want to create objects, my rectangle and my box. We have the classes rectangle and box now we want to create objects my rectangle and my box. We write rectangle my rectangle is equal to new rectangle and we give the dimensions of the rectangle 5, 3 as the argument to this constructor and box my box is equal to new box and we give the arguments here.

So, when we create this we write new rectangle 5, 3 the attributes in this object are assign the value 5 and 3 and once we write the new operator that is we create an instance of this class we get handle here that we called as the identification or the identify of this object. Normally we call it as the object reference, so my rectangle once you say new rectangle 5, 3 my rectangle is the object reference or the reference for this object that we created. Similarly once you say that my box is equal to new box, my box becomes the object reference for the box object, but the box object is a inherited object, it is a derived

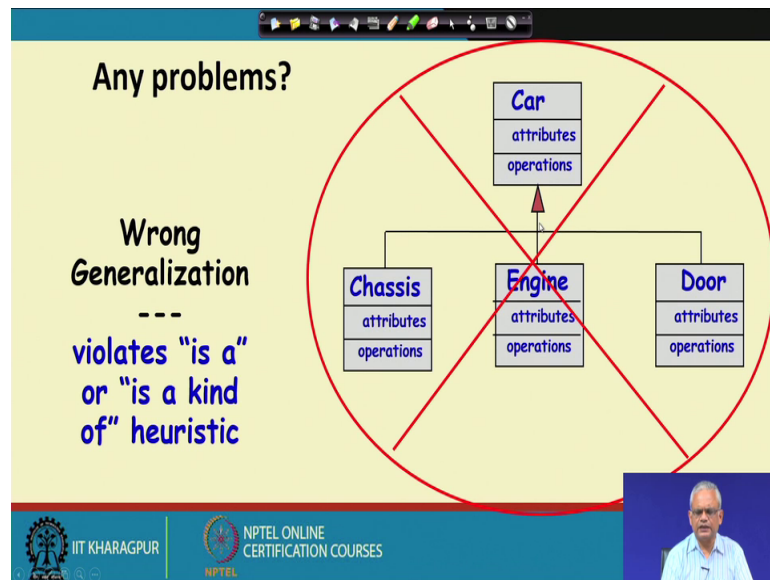
class and it inherits the attributes and methods of the base class. And therefore, implicitly we can think that the attributes length and width defined in the base class arrival represent in the derived class and you have this new attribute which is added here is the height which is also present.

(Refer Slide Time: 19:54)



So, this is another example of a generalization, specialization. There is an example of a multiple inheritance the feline class inherits from both mammal and quadruped and then from the feline class we have the tiger, lion and puma has the derived classes, lizard is derived from quadruped.

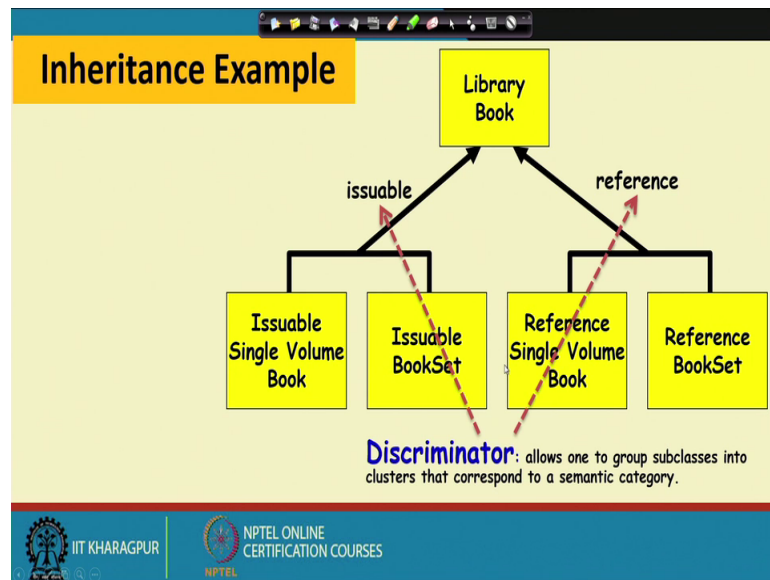
(Refer Slide Time: 20:31)



Suppose someone has drawn this inheritance relationship chassis, engine and door are derived from class. Is this correct, does it appear all right, this inheritance hierarchy that chassis, engine, door a derived from the car class? No this is not correct it is a wrong generalization, we have said that the inheritance relationship we characterized by ISA relationship we cannot say that chassis is a car, car is not chassis, engine is a car is not a proper generalization and similarly door is a car is not a proper generalization. So this is not a proper generalization specialization relationship and we cannot represent it using an inheritance relationship later we will see that this is actually an aggregation relationship.

So, to check whether the relationship is correct or not we can use this test that we can say that whether a chassis is a car, engine is a car, if it is a general class and these are special class then the easier relationship hold. For example, library member UG member PG member and staff member and always a UG member is a library member, PG member is a library member and staff member is a library member. So, the ISA relationship holds, here it does not hold so this is a wrong construction of generalization specialization relation between these classes.

(Refer Slide Time: 22:41)



We often have many classes derived from a base class, but it is a good idea to use a discriminator based on which these derived classes are defined for example, this the library books are can be issuable books or reference books. So it is a good idea to write the discriminator based on which this set of classes are derived and this set of classes are derived that helps in readability and the supported by eoml to write the discriminator while constructing a complex class hierarchy, that helps us to understand the hierarchy better.

(Refer Slide Time: 23:34)

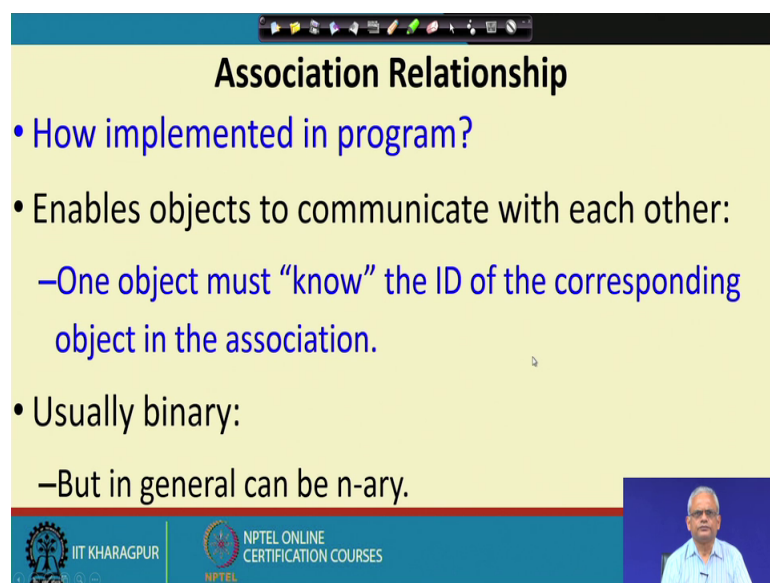
**Inheritance Pitfalls**

- 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

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Inheritance is a powerful mechanism promotes reuse, simplify the design and also easily implemented in code just by using a keyword, but in spite of its many advantages if indiscriminately use inheritance it will result in poor quality of programs. For example, if you have a very deep class hierarchy then it is has poor coercion and coupling because a large number of methods are visible the in the leaf classes and that leads to tight coupling and it makes it very difficult to understand the leaf classes, because they have too many methods and we need to exam in what are the methods it has inherited. So, inheritance should be used with some question.

(Refer Slide Time: 24:57)



**Association Relationship**

- 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.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, let us look at the second type of relationship between classes, this is the association relationship we need to discuss some examples of association relationship and also we need to answer this question that how are these implemented in a program. In case of inheritance we saw that there are simple keywords in all object oriented languages whether it is Java, C plus plus or any other language the inheritance relationship is straight forward implement.

But how does one go about implementing the association relationship, one thing that we must remember about the association relationship is that, when two classes are associated the corresponding objects can communicate with each other as we will look at the examples it will become clear. One class can invoke the methods of another class, but to able to invoke the method of another object, the object must somehow store the id of the

other object otherwise how will it invoke, the method of the other class. And that holds the idea of how to implement an association relationship will seen soon look at that. The association relationship is usually binary, but it can be higher degree association like ternary or in general can be n-ary let us look at an example of a association relationship.

(Refer Slide Time: 26:41)

**Association - example**

- In a home theatre system,
  - A TV object has an association 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

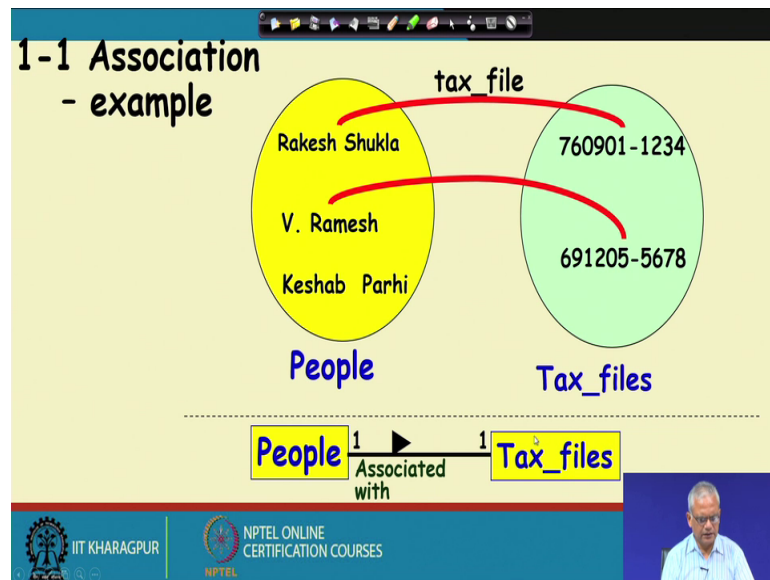
The diagram shows three objects: Remote, VCR, and TV. An arrow labeled 'commands' points from Remote to VCR. A line labeled 'Connected to' points from VCR to TV. Multiplicity '1' is written at the end of each line.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Let say VCR a video recorder is the associated with the TV and we can have a remote which is associated with the VCR. So we can the remote object can give command to the VCR the VCR can record what is being played on the TV. So, the TV object is associated to the VCR object, because the VCR object receives signal from the TV object and the VCR main term be associated with a remote and it can receive command from the remote. The way we represent the association relationship is by drawing line, there are different ways we can draw the line sometimes, we draw an arrow sometimes, we just write this line. We will see when we draw an arrow and when we draw a line and we write here the multiplicity and we write the reading direction here.

We will see the details of how to represent the association relationship what are the implication of just drawing an arrow, just drawing a line and then we write the name of the association here on this line or arrow. And we write the reading direction say that the VCR is connected to TV it helps us to read or we can read the TV connects to the VCR. The remote commands to the VCR or we can read as the VCR receives commands from the remote.

(Refer Slide Time: 27:57)



So, we have reach the end of this lecture and we will continue in the next lecture to discuss about how to represent the association relationship, the multiplicity and so on; as per the eoml syntax. We will stop here and continue in the next lecture.

Thank you.