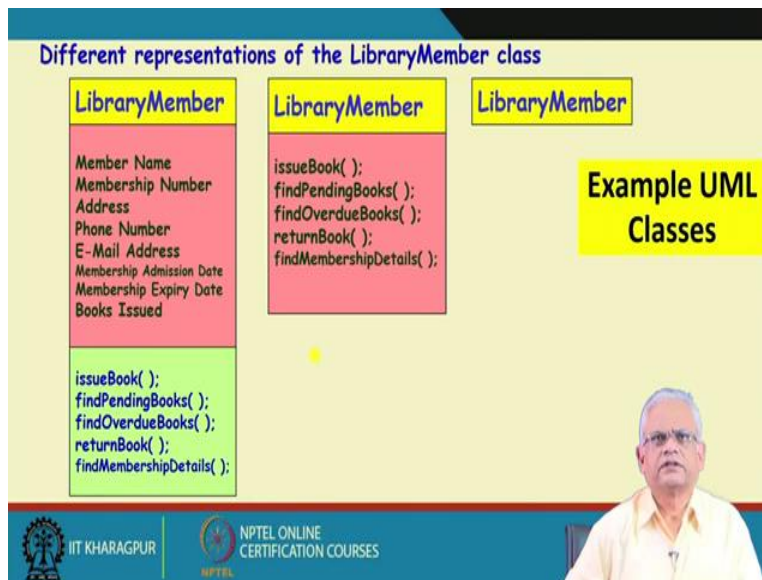


**Object Oriented System Development Using UML, Java and Patterns**  
**Professor Rajib Mall**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Kharagpur**  
**Lecture 7**  
**Class Diagram**

Welcome to this lecture.

In the last lecture we were discussing about the class diagrams. We were discussing about the representation of the class and we had said that, the classes can be represented in three different notations.

(Refer Slide Time: 00:32)




In one notation we can just have the name of the class (the right one of the above slide). In another notation we can have the name of the class and the operations supported by it (the middle one of the above slide). And another more detailed representation where we have the name of the class, the attributes and the operations (the left one of the above slide)..

But then what about the visibility of the attributes and the operations?

From our knowledge of object orientation, we know that there are different visibility rules for the operations and methods. Those who have some preliminary programming knowledge of C++ and Java they would be aware about the visibility rules.

(Refer Slide Time: 01:26)

Class Attribute Examples	
Java Syntax	UML Syntax
Date birthday	Birthday:Date
Public int duration = 100	+duration:int = 100
Private Student students[0..MAX_Size]	-Students[0..MAX_Size]:Student




IT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

We look at the visibility rules. We first look at that the class attributes (in the above slide). In the Java syntax, if Date is a class and birthday is a type of an instance of date then in UML we can write 'Birthday: date'. The public visibility in Java we represent in UML as '+'. In the above slide, java syntax 'Public int duration = 100' is represented in UML Syntax as '+duration:int = 100'. Private visibility, we represent by '-' sign and here java syntax 'Private Student student[0..MAX\_Size]' is represented as '-Students[0..MAX\_Size]:Student'.

(Refer Slide Time: 02:55)

Visibility	Java Syntax	UML Syntax
public	public	+
protected	protected	#
package		~
private	private	-

Visibility Syntax in UML



IT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES

In Java there are different types of visibility (as shown in the above slide). The public visibility is visibility to all classes for that UML syntax is '+'. Protected visibility in Java, we write protected class or protected attribute using the keyword protected. But here we use the '#' symbol in UML. In Java if we do not specify the visibility of a class, method or attribute, then by default it is package wide. But in UML we use '~' to specify package wise visibility. The private visibility inside a class the Java syntax is private but UML syntax is '-'.

(Refer Slide Time: 04:10)

● **Methods are the operations supported by an object:**

- Means for manipulating the data of an object.
- Invoked by sending a message (method call).
- **Examples:** calculate\_salary(), issue-book(), getMemberDetails(), etc.

**Methods vs. Messages**

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES


What about methods and messages, are they the same thing?

We know that the methods are the operation supported by the object. We use the methods for manipulating the data of an object. We can access the data use it and also, we can modify the data of an object. A method is invoked by sending a message. We send a message to the method and we also call it as a method call. Sending a message is the same as a method call. For example, we may (call) calculate\_salary(), issue-book(), getMemberDetails() et cetera. All these are methods in Java and we invoke these methods through a method calls or sometimes we say that we send a message to the method.

(Refer Slide Time: 05:31)

### Method Examples

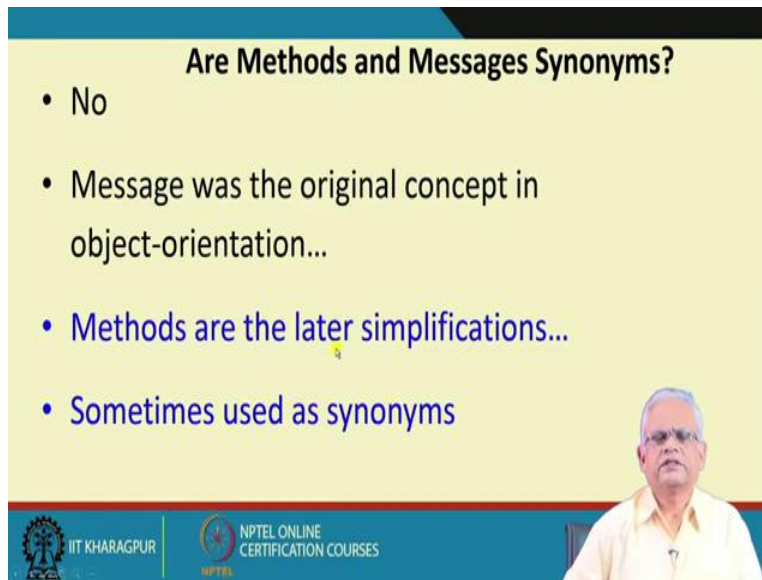
Java Syntax	UML Syntax
<code>void move(int dx, int dy)</code>	<code>~move(int dx,int dy)</code>
<code>public int getSize()</code>	<code>+int getSize()</code>



The slide features a blue header with the title 'Method Examples'. Below the title is a table with two columns: 'Java Syntax' and 'UML Syntax'. The first row shows 'void move(int dx, int dy)' in Java and '~move(int dx,int dy)' in UML. The second row shows 'public int getSize()' in Java and '+int getSize()' in UML. At the bottom of the slide, there are logos for IIT KHARAGPUR and NPTEL ONLINE CERTIFICATION COURSES, along with a small video feed of the presenter.

The method syntax there is a close correspondence between the Java syntax and the UML syntax. In the above slide, for ‘void move(int dx, int dy)’ the visibility is default. We have not specified the visibility neither public nor private, just omitted that and that is package wide and we write in UML syntax ‘~’ to represent package wide visibility for the move. For ‘public int getSize()’ we write ‘+’ in UML syntax to represent visibility.

(Refer Slide Time: 06:50)



**Are Methods and Messages Synonyms?**

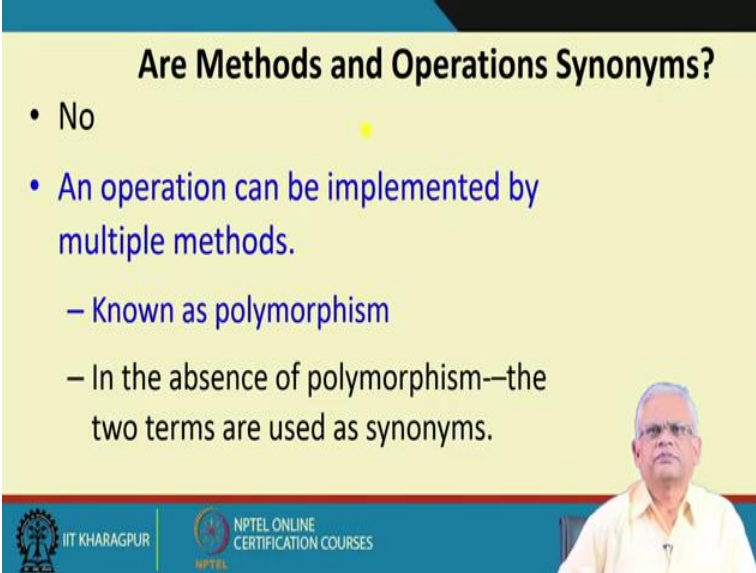
- No
- Message was the original concept in object-orientation...
- Methods are the later simplifications...
- Sometimes used as synonyms

The slide features a yellow background with a blue header and footer. A small video inset in the bottom right corner shows a man with glasses speaking. The footer contains logos for IIT KHARAGPUR and NPTEL ONLINE CERTIFICATION COURSES.

Now a question is that are methods and messages synonyms? Are they used interchangeably? The answer is, yes. Methods and messages are sometimes used interchangeably but if we look at the historical development methods and messages are not synonyms. Even though at present, they are used sometimes interchangeably but if we look at the historical development, they had different purposes. Methods and messages were entirely different. The message was the original concept in object orientation to have loose coupling between the objects. One object can get services of another object by sending messages. Messages are asynchronous. For example, an object can send a message to another object but the another object may reply sometime. But methods are the synchronous invocation. An object invoking a method of another object waits for it that is synchronous. In the asynchronous messages it need not wait for this, this is a loose coupling whereas synchronous method calls lead to tighter coupling. Even though the initial developers are object orientation wanted loose coupled system through messages but, then it did not become very popular and later simplified that into synchronous messages. But in nowadays many times we use messages and methods as synonyms, method calls and sending messages as synonyms, even though originally, they were not really supposed to be synonym. There were two different concepts.

Messages were used for loose coupling. The first object-oriented languages like smalltalk use messages rather than method calls but from C++ onwards the messages were simplified into method calls.

(Refer Slide Time: 09:30)



**Are Methods and Operations Synonyms?**

- No
- An operation can be implemented by multiple methods.
  - Known as polymorphism
  - In the absence of polymorphism--the two terms are used as synonyms.

The slide features a video inset of a man in a light-colored shirt speaking. At the bottom, there are logos for IIT KHARAGPUR and NPTEL ONLINE CERTIFICATION COURSES.

But what about methods and operations, are they synonyms?

Not really. A class supports many operations and each operation can be implemented by multiple methods. The same operation is implemented by multiple methods and this is known as method overloading or polymorphism. In method overloading the name of the methods is the same but they may take slightly different arguments. But, in the absence of method overloading that is a single operation is implemented by a single method, then methods and operations are actually synonyms.

So, there is a finer distinction between a method and an operation. If an operation of a class is implemented by multiple methods, we say that, this is a case of method overloading and we use the methods and operations differently in this situation. But, if an operation is implemented by a single method, then we can say that the method and the operation are the same in this situation.

(Refer Slide Time: 11:11)

The slide features a yellow background with a blue header and footer. The title 'What are the Different Types of Relationships Among Classes?' is in a yellow box at the top right. A list of four types is on the left, and a small video inset of a man is at the bottom right. Logos for IIT Kharagpur and NPTEL are at the bottom left.

- Four types:
  - Inheritance
  - Association
  - Aggregation/Composition
  - Dependency

Now one question to think for. Many of us have written programs in Java and C++. We have defined classes there and solved problems using classes. But, for solving a problem the classes need to interact and also some of the classes are related to each other. Now can you identify what are the different class relations that you have used in your programming? What are the different relations that can exist between the classes in an application solution?

There are actually 4 types of relations among classes. One is the inheritance relation between classes where one class is derived from another class. Another type of relation is association relation between two classes where one class is associated with another class. The third type of relation is aggregation and composition where the objects of one class are aggregated in the object of another class and similarly is the composition. And the fourth relation is dependency where one class may be dependent on another class. We will see that under what situation dependency arises. When two classes are dependent if we change something to one class, the other class also needs to change or at least needs to be requiring recompilation.

Now can you think of any programming example of inheritance relation, as well as association, aggregation and composition, and dependency?

Let's look at some examples of this. I hope that you have got some examples. Now let us look at some examples of these classes and let's look at slightly deeper into these relations. These



examples will help you to know the kind of relation between classes of an application can have and it will also be going to help you to know how will the code look like for each of this relation.

(Refer Slide Time: 15:10)

**Inheritance**

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

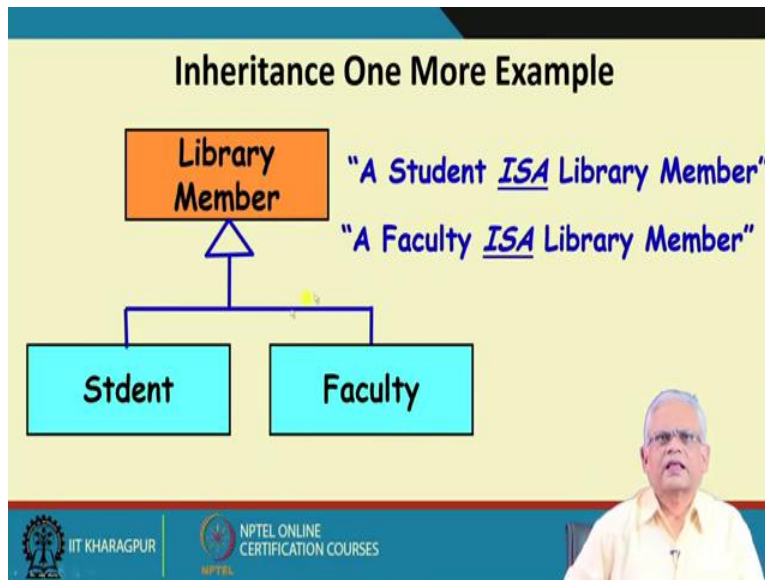
```
graph BT; Individual --> Faculty; Full_time1[Full-time] --> Faculty; Faculty --> LibraryMember; Full_time2[Full-time] --> Students; Research1[Research] --> Students; Students --> LibraryMember; Research2[Research] --> Staff; Staff --> LibraryMember;
```

IT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Inheritance as we know from an existing class which we call as the base class, we can obtain a new class which we call as the derived class. This is also called as a generalization-specialization relation in the sense that the base class is a general class and the derived class is a special form.

For example, the generalization may be a member but then there can be many times sub-special members like student members, faculty members, staff members et cetera. The derived class inherits the existing methods implicitly. The derived class has all the methods implicitly that the base class has. But the derived class may override some of the methods that are inherited. In the above slide we can see, LibraryMember is the base class and Faculty, students, staff are the derived from LibraryMember.

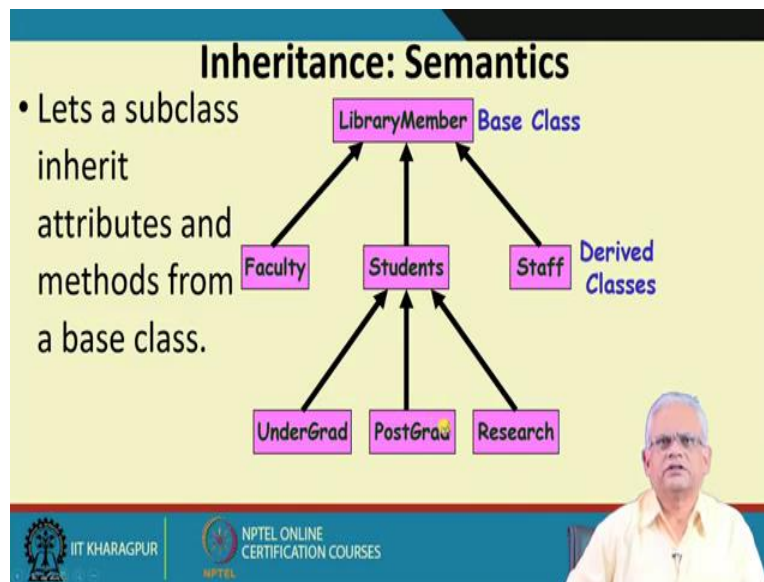
(Refer Slide Time: 16:40)



We can see one more example of inheritance in the above slide. Here, student and Faculty derived from LibraryMember which is a base class. The inheritance relationship is showed through an arrow. The arrow head here is empty. In the above example, the student and faculty are a special type of library member. That means the library member is a general class and a faculty is a special type of library member and a student is a special type of a library member. Maybe the faculty can issue 20 books and a student can issue only 5 books. Often, we call the inheritance relation as a "IS A" relation.

We call it as "IS A" relation because whenever we have an inheritance, we can always phrase it by saying that a special class is a general class like, a student is a library member, a faculty is a library member etc.

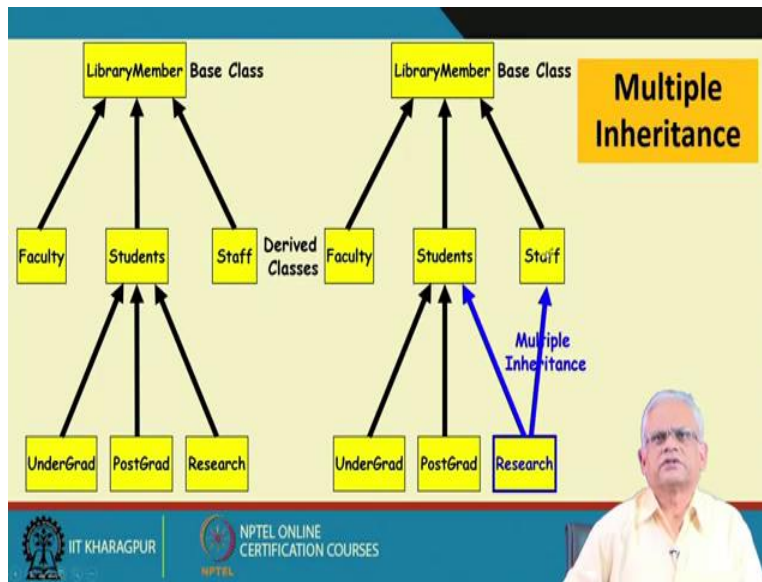
(Refer Slide Time: 18:26)



Now let's look at the semantics or the implication of inheritance (in the above slide). The derived class or the subclass inherits the attributes and methods of the base class implicitly. Here library member is a base class and the faculty, students and staff are the derived class. Whatever attributes and methods defined in the base class LibraryMember are implicitly available in the faculty, students and staff. Similarly, all the attributes and operations of the LibraryMember class and student class are also available for the undergrad, postgrad and research.

Now we will see about multiple inheritance.

(Refer Slide Time: 19:29)



When a derived class inherits from multiple base classes it is termed as multiple inheritance. In the above slide, we can see that the research class inherits the operations and attributes of the staff class as well as the attributes and operations of the student class. But multiple inheritance in programming is problematic. The main reason for this is that the derived class can have duplicate methods and attributes. For example, in our above slide, Research class get attributes and methods of LibraryMember once through the staff and again through the student. So, the research class has multiple instances of attributes of the LibraryMember and also multiple methods. This is a problem because we update one of the methods, one of the attributes the duplicate attributes that have been obtained from once through student and other staff can behave inconsistently. We update one of the attributes of base class and then later in our program we print another duplicate attribute and say that it is behaving inconsistently. Whenever there is duplication of data, we get into problem, we get into inconsistency.

So, we can understand that multiple inheritance leads to programming complications. In C++ it is handled through virtual base class. When we say virtual base class, it can inherit only a single attribute in the derived class. But in Java we don't allow multiple inheritance. A class can inherit from a single class but it can implement multiple interfaces. But remember that interfaces don't contain attributes, they only contain methods and therefore there is no problem of inconsistency.

(Refer Slide Time: 22:04)

## 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;
    ...
}

```

```

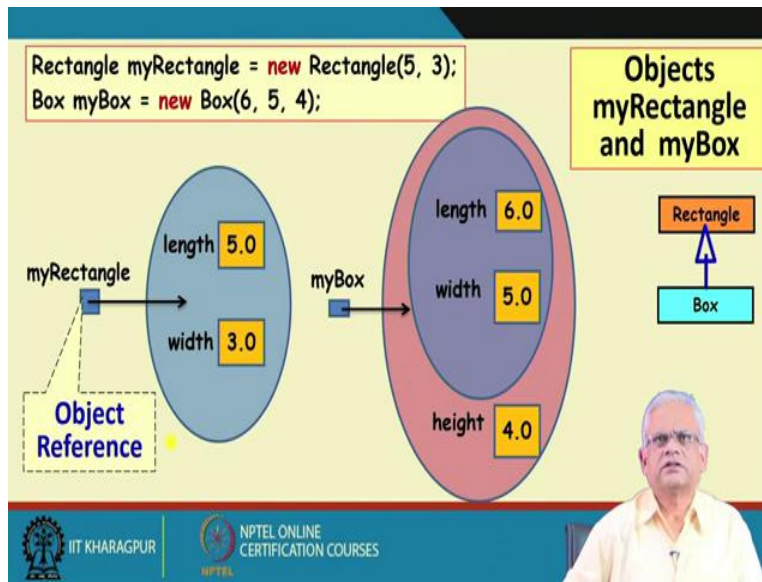
Employee anEmployee = new Employee();

```

Now, let see how inheritance is implemented in Java. It can be easily achieved by the extends keyword. In the above slide we can observe that Person is the base class and Employee is a special type of person derived from Person class. In the above slide we can see the class definition of Person class and corresponding UML syntax. We have this keyword extends we just write ‘class Employee extends person’ and that is an implementation of inheritance. So, implementing inheritance is straight forward.

If class Employee extends Person then Employee class implicitly inherited Person class and also we can define the new attributes and methods in Employee class. Using ‘Employee anEmployee = new Employee()’ we instantiate the employee and create an object in the form of an employee.

(Refer Slide Time: 23:55)



To look deeper into it let us take another example (in the above slide). let say we have Rectangle as the base class and then we derived the box class from it. Now, we can instantiate the rectangle, the rectangle takes two integer parameters denoting the length and width of the rectangle. The dimension of the rectangle is 5 (length) and 3 (width) and we get myRectangle as the object of Rectangle class, and we can create an instance of box which is myBox is an instance of box.

We use the 'new' keyword in Java to create the instance of box and we give the argument 6 (length), 5 (width), 4 (height) which are the dimensions of the box. Now, we can see for myRectangle object height set to 5 and length set to 3 and we have a handle (the arrow line from myRectangle object) onto created object. Later we refer to this object by its handle which we call as the object reference and myRectangle is a reference to this object.

Now, when we create the box reference, the box implicitly has the length and width because the box is inherited from rectangle. The length and width which are set to 6 and 5 and also it defines a new attribute called as height. The height is set to 4 based on the 6, 5, 4 and then again once instantiate, we get this handle from myBox to created object. Later we can access this object through this object reference myBox.

Now, this object reference is important because later we will use this in our further discussions that we create objects by the new operation in Java. And then we get the object reference and

later we carry out all operations on the object through its reference. We can create many objects and will get different object references for those.

We are at the end of this lecture, will continue from this point in the next lecture.

Thank you.