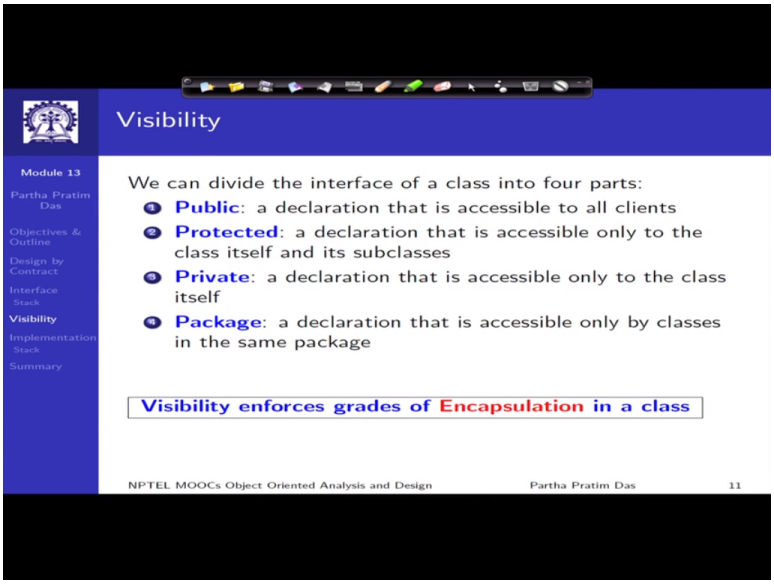
Object-Oriented Analysis and Design
Prof. Partha Pratim Das
Department of Computer science and Engineering
Indian Institute of Technology-Kharagpur

Lecture – 23
Nature of a Class: Interface and Implementation (Contd.)

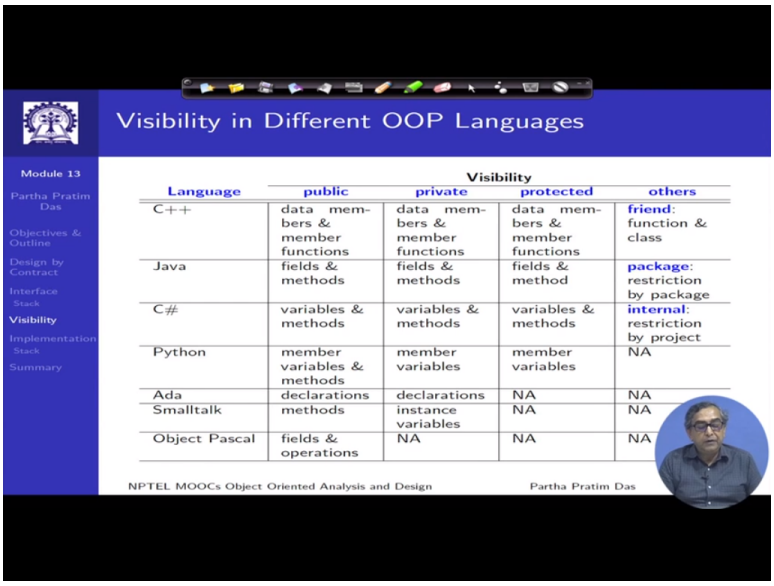
Welcome to module 13 of object-oriented analysis and design. We have been discussing about nature of classes. We talked about design by contract an how that can be used in designing interfaces of classes and took an example of the stack class, of a stack class to explain the whole idea. Then we also took a look into the visibility aspects of different components of a class, different declarations and noted that primarily there are 4 kinds of visibility commonly used in object-oriented systems the public, protected, private and package.

(Refer Slide Time: 01:09)



Continuing on this I would just like to draw your attention to the visibility as is available in different object-oriented programming languages.

(Refer Slide Time: 01:12)



Now certainly this course is not directly dealing with object-oriented programming as I had mentioned earlier. But certainly you all are trying to learn object-oriented analysis and design to be able to at the end do programming of the system on some of the common object-oriented languages.

So here I have compiled for some of the common languages which are mostly people use. So if you are using C++, C++ has primarily 3 different visibilities available to public, private and protected both data members as well as member functions. Data members means basically the state variables and the member functions or methods can be into any one of this visibilities. But in addition C++ has another kind of visibility which comes through friend.

It can declare some function to be a friend or some class to be a friend. A friend will have a private like visibility into the inner of the class and therefore it is very important to choose the friends very carefully. What is if I if we move to java, we again have the public, private and protected visibility for the fields and methods but java does not have a friend but it has something similar which is a package where if you put a set of classes within the same package, they can they are visible to each other.

Many of you may be programming in c sharp which has a very similar visibility structure, public, private, protected for variables as well as methods but it has an interesting actually it has 2 more

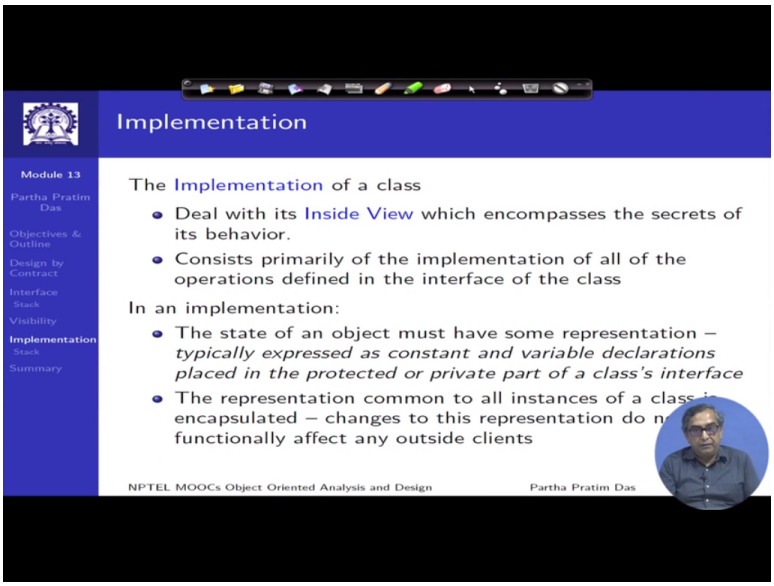
interesting visibility options one is known as internal, another is known as protected internal. These relate to these are restricted to the projects in C++ c sharp that is c sharp as you know is it is managed by dot net.

So any C sharp application you want to develop, you have to define a project so all that all the class you put inside one project has an internal visibility or protected internal visibility. Python on the other hand has only public visibility for its methods. Its methods cannot be encapsulated out from others. It is always public but the member variables can be public, private, protected. There is no other visibility in python.

And some of the other languages just for the sake of completeness I have mentioned at a small talk and object Pascal for example object Pascal is an interesting case where all fields and operations in object Pascal are public. So it is another extreme of an object-oriented language which has no encapsulation at all. So there is a lot of different grades so it is depending on which language you are using it will be good to first quickly study what is a visibility

Or encapsulation grades that the language support and then understand how should you define your class, how should you design your interface using this visibility information.

(Refer Slide Time: 04:54)

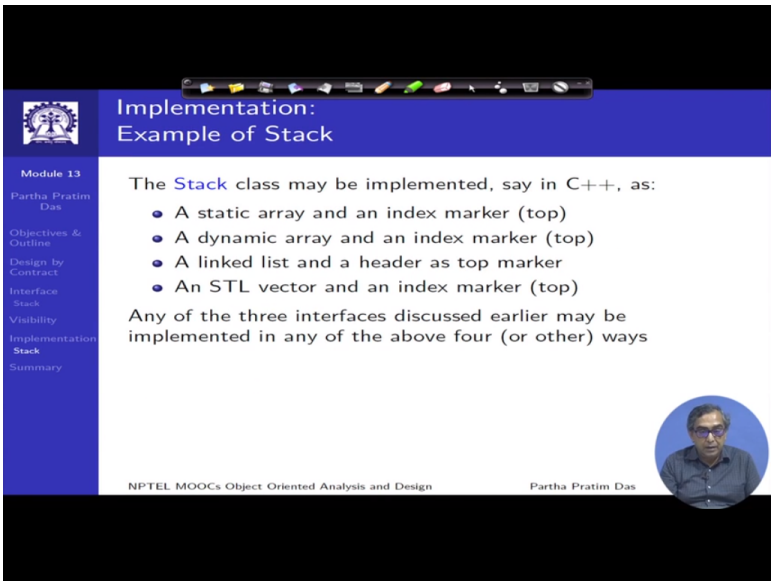


Let us move on to the other part of the class. We have been looking at the outer view now let us consider the inside view, the inner view. The inside view that is where all our secrets are very carefully protected, kept is known as the implementation of a class.

So I can say that the implementation of a class is basically the implementation of all the operations of all the methods that the class supports. Now naturally if we want to implement something we have known that the class will give rise to instances as objects. Objects will have state behavior and identity. So behavior is the exposed methods which is basically what we are carrying in the interface.

So state is what the class will need to represent in terms of the implementation. So state of an object must have some representation in terms of the implementation of the class that will typically have inter occur in terms of constants in terms of variable declarations and typically occur in the protected or private part of a class interface because if it if this representation of the state is put in the public part then it will understand that outsiders can come in any time.

And change those values and your state will become indeterminable. So even though some object oriented languages might allow your variable declarations or data members or fields to be put in the public view, you should be very careful in terms of doing that while you are designing an interface and the corresponding implementation of the class.
(Refer Slide Time: 07:01)



So given that let us look at go back to our stack class. We have already seen different interfaces for it. But then suppose you are given to implement this stack class in say C++, all languages will have similar examples. But if you are given to implement this stack class in C++, you could do that using a static array. You can statically define an array and you can use an index marker which will keep an index of the array to show how much the array has been filled up by the stack operation.

So what you simply do you start with a marker being placed at minus 1 which is outside of the array and then every time you push, you increment the marker and put the element in that array location. Every time you have to pop, you simply decrement the marker. So that could be one way to implement the stack but this will certainly that your stack will be able to take only a fixed only up to a fixed number of elements at a time because the array is static,

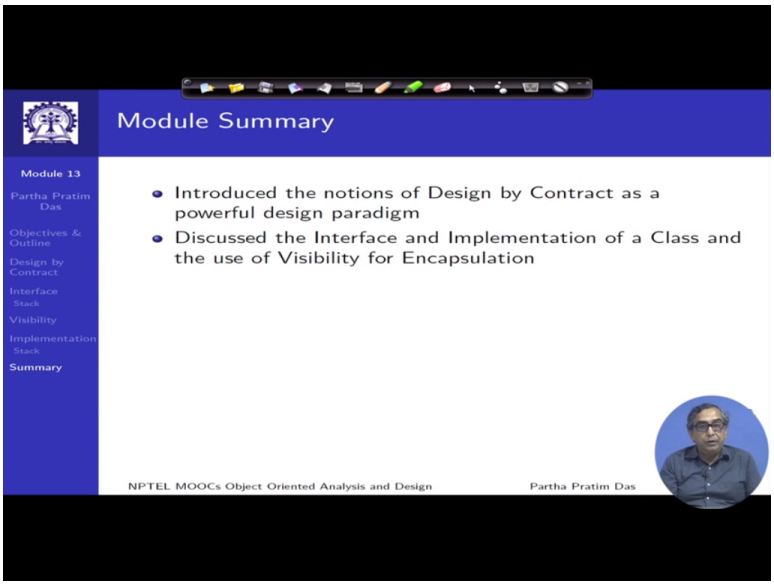
It will have a fixed size so the alternate could be that you could have a different implementation which we will use a dynamic array so that you could if require have a bigger array when you have more elements being pushed in and of course an index marker. Alternately you could use a linked-list if you use a linked-list and you will need a header to hold that list and that header will act also as a top marker.

If you are more advanced C++ programmer, you could use a stl vector and an index as a top marker and achieve the same results. There could be other, several other alternate implementation designs in terms of the variables, in terms of the data members as well as in terms of the details of the algorithm that use but any of the three interfaces that we discussed earlier, remember we discussed three interfaces based on three different contract schemes.

That the stack object may have then any of them can be implemented in any of the above 4 methods or may be even other methods. So you can see that on both terms, you have a wider set of choice and but when you go to implement, you will have to make sure that your interface is completely honored in terms of the methods that you are in implementing as well as the contract that has already been given.

But the contract will tell the implementation as to what needs to be done to fulfill the contract and the implementation will need to be completely hidden in terms of the private protected part of the class so that if in future from the time that you have implemented the class for the first time sometime later, you want to make a different change in the implementation for some reason of the other, interface will not get affected, the methods will not get affected, the contract will not get violated.

And in that process, the user would be able to continue using your object without any difficulty even though the implementation is not maintained to be the same.
(Refer Slide Time: 10:33)



So in this module, we have introduced the basic notion of design by contract as a powerful design paradigm we will see that at some certain later points will again keep on re referring back to this and we have discussed the basic concept, nature of a class in terms of its interface and implementation and we have reiterated the fact that the implemented chill and interface needs to be segregated, it needs to be separate.

And this process of separation can be achieved in various graded manner by using various kinds of visibility that we may use and certainly we will hope to discuss more about this and get more inside when we have introduced to the other different relationships of classes and particularly the inheritance and specialization, we will see how visibility will control the encapsulation at multiple level to provide you, further refinements on the interface and finer control on the implementation.