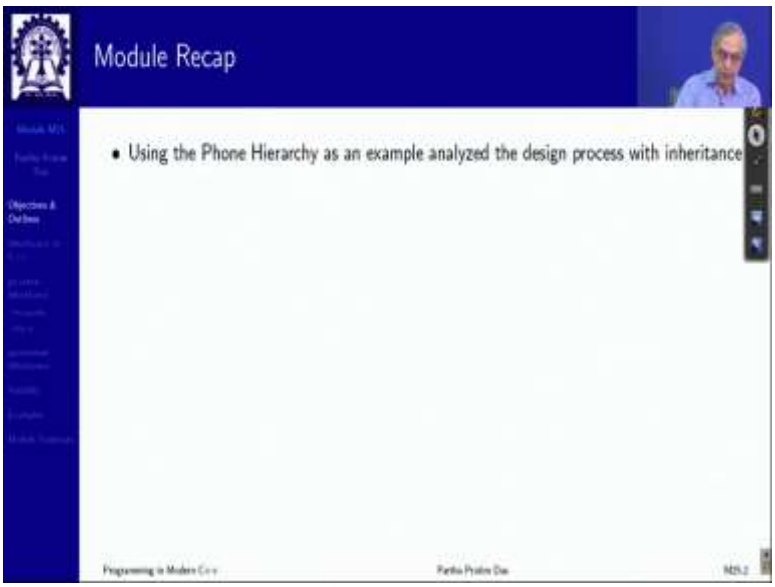
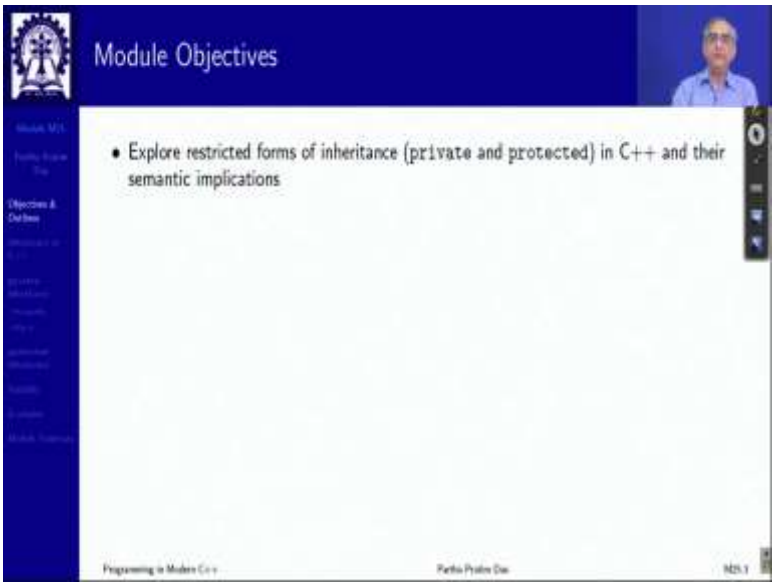
Programming in Modern C++
Professor – Pratha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology – Kharagpur
Lecture – 25
Inheritance: Part 5: Private and Protected Inheritance

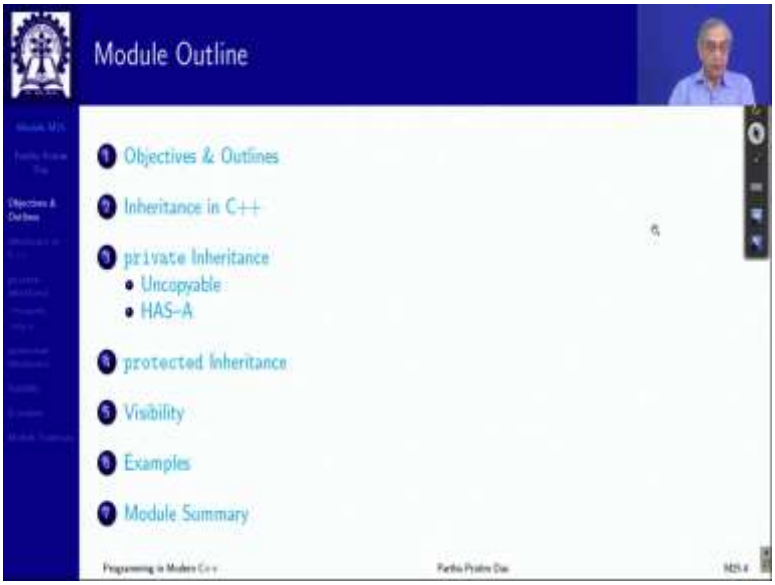
(Refer Slide Time: 00:32)

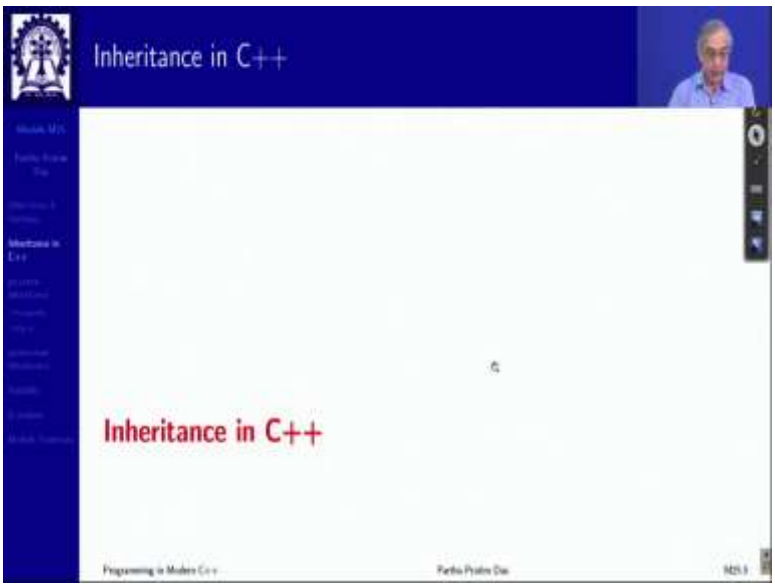


Welcome to Programming in Modern C++. We are in week 5 and I am going to discuss module 25. In the last module, we have taken a look at the detailed design process for a very simple phone hierarchy, which to give you kind of a complete integrated feeling about the different inheritance features that C++ has provided to be able to model, implement and use ISA relationships of object orientation, inheritance in object orientation.

(Refer Slide Time: 1:08)



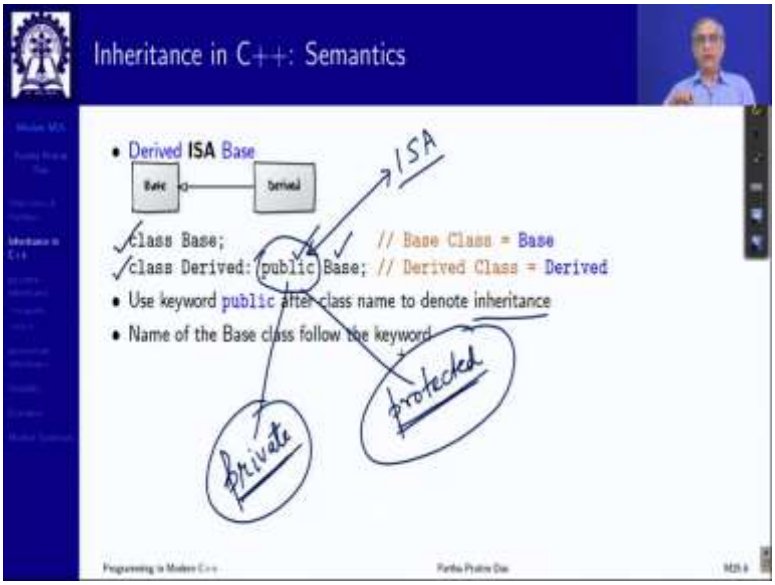




In the present module, in this module 25, I will talk about certain more forms of inheritance or rather it is even questionable as to whether I should call them inheritance but they are available in C++ they have very limited use. But unless I tell you what they mean, it is quite possible that you might use them and get into trouble.

So, kind of this module will describe two other forms of so called inheritance by private and protected form and with the example, with the kind of caution that if you need to use any one of them then you need to be really really thorough and confident that you need them. Any normal good C++ hierarchy design should never need this. So, it is kind of a little bit of a negative module or negative knowledge I should say but every knowledge is important.

(Refer Slide Time: 2:28)



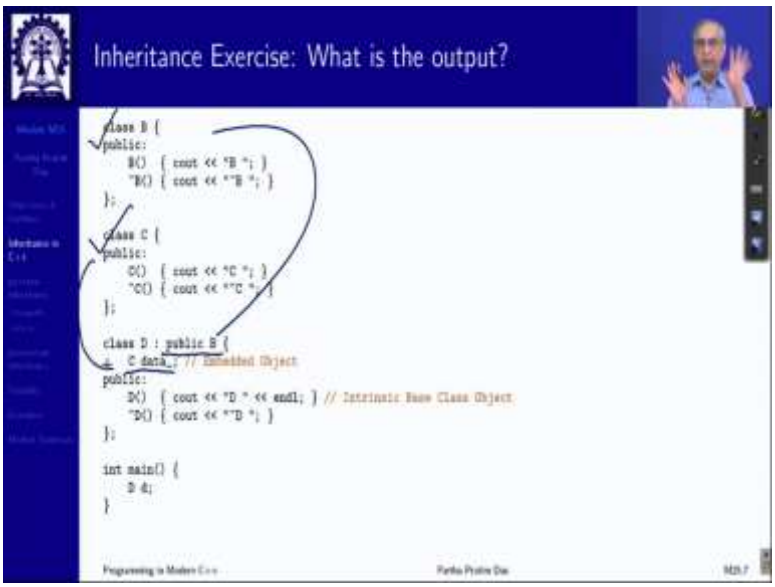
This is the outline, so I start here, inheritance in C++, this is what you have seen derived is a base class and you model that using the base class and the derived specialized class using this keyword public and using the name of the base class. Now, this denotes the inheritance, let me tell you in very clear and loud terms that using the keyword public here is the only way to do inheritance as we understand in the object oriented manner.

Now, the point is C++ does allow you to use the two keywords private and protected as well. As you know these are originally access specifiers public, private and protected came in for inheritance. Now, when you are inheriting, you have also tried to, the language has also tried to give us these options that I can say public or private or protected. But what I want to specifically as your mentor for C++ design and programming, I want to caution you that they are known by inheritance in C++ but they are not the inheritance of object oriented analysis

and design.

The OAD principle, ISA relationship that OAD, object oriented analysis and design puts forth for handling related generalization and specialized concepts, is only possible through C++ public inheritance. These are available, they have some semantics and we will learn what those are but have to be very very cautious in using them.

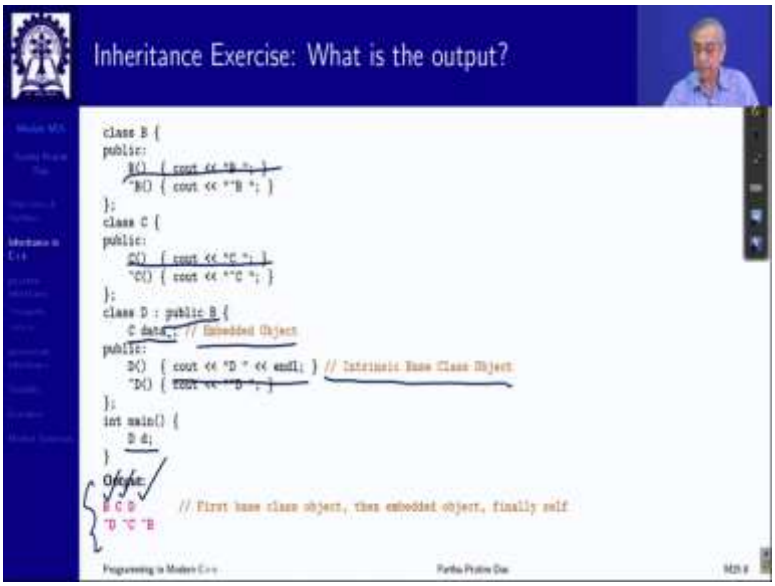
(Refer Slide Time: 4:53)



So, with that introduction let me get into the details, say this is what we have seen, again an example I am repeating just to illustrate the point that there is a class B, there is a class C, both of them are two independent classes. There is D, is derived from B, it is a public inheritance so it is a specialization and it uses an object of class C inside it.

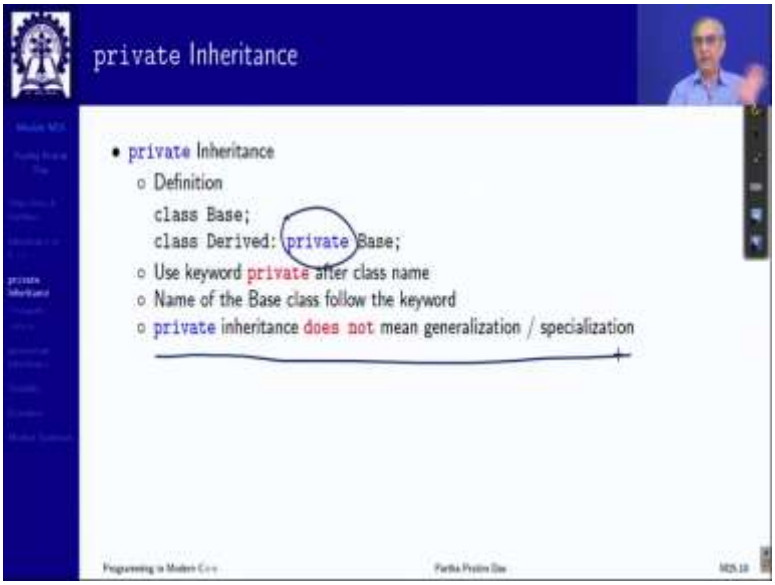
So, this is the entire thing you can have, in a very small terms as I said, every unique concept in one slide. So, this is where you have the possibilities of the objects that you can have in your layout, a base part and the embedded objects. So, I am just showing you this because I am going to go forward and show you contrast it with when public is something else, the public inheritance is something else.

(Refer Slide Time: 5:58)



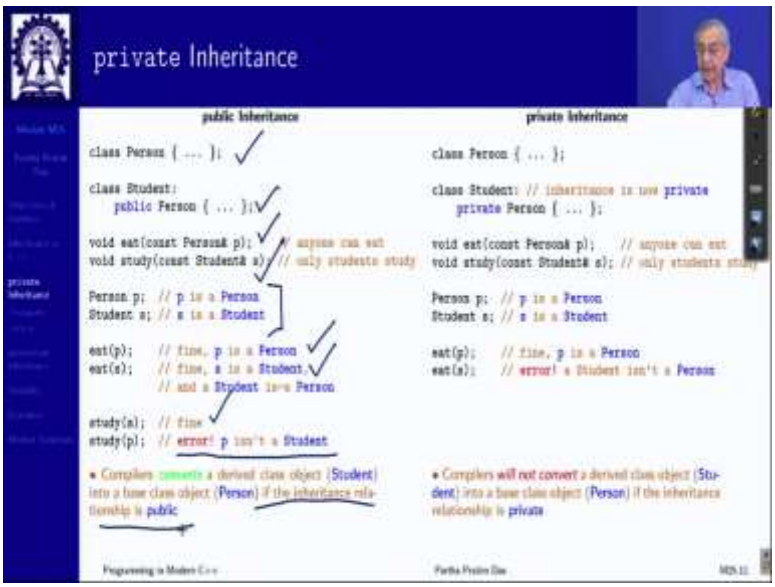
So, if I go on to the next, this is nothing but other than the same example with the actual run output, so you can you can try it out yourself and then see this slide. Obviously if I run this, I will get this because as I said for creating a D object, you have to first create a B object. So, this is what happens first the base part of it. Then now, it creates the embedded object so it has to create the C data which is the C class constructor and finally completes itself which is the D class constructor. So, this is how the constructors are called objects are created in reverse order they are to be destructed.

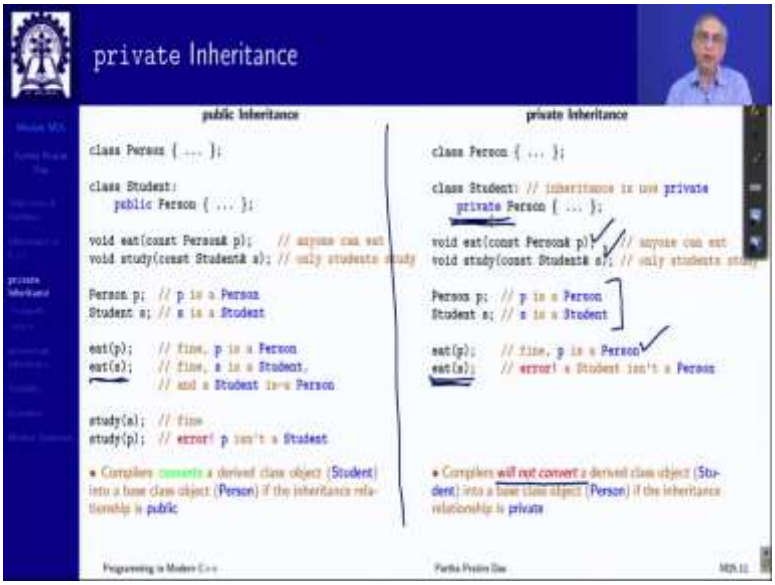
(Refer Slide Time: 6:53)



Now, let us talk about private inheritance. So, what we do in private inheritance, the only difference is we put this word private in place of public. I do not know how strongly I would like to tell you this that private inheritance does not mean generalization specialization. It is just a language feature, it is just a language feature, keep this in mind.

(Refer Slide Time: 7:27)





So, what is that you get through this, if I write code it in terms of C++ I have a Person, I have a generalization, I mean specialization of that Person as a Student, this ISA, Student is a Person. So, then if I have a member function to eat, a member function to or a function to eat, a function to study. Eat should take a Person because everybody eats, study should take a Student because only Student study, others do not.

I make objects of both of these classes, eat can take the Person, eat can also take the Student s, because a Student also eats. Concept wise that is what this means. The study can be done by a Student, its fine but naturally I cannot pass a p or a Person object to study because a Person in general is not a Student, does not study. So, the compiler converts a derived class object Student into base class Person, if the relationship is public, inheritance relationship is public that is the reason that eat expects a Person but it is ok to pass it a Student.

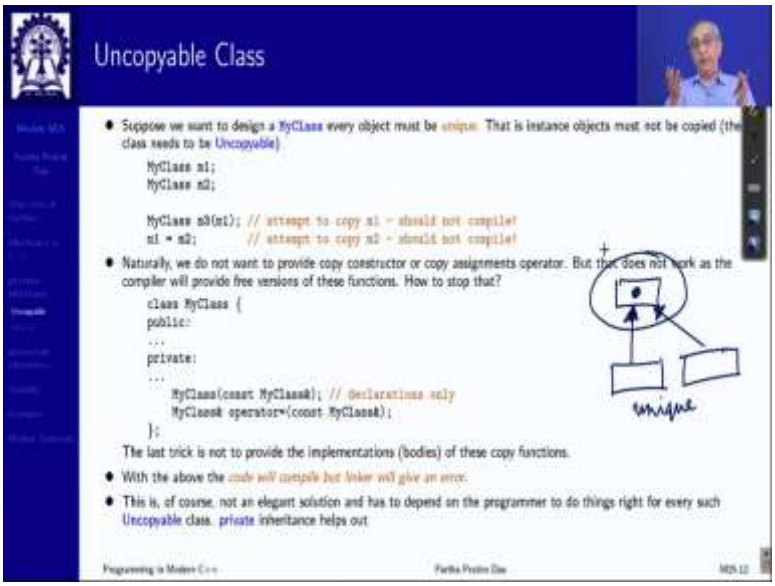
Again, remember what is expecting a Person and Student is a generalization of a Person. So, Student has at its part, a Person based class component which can be passed on here and the eating habits can be checked out. This is, I mean reemphasis of what we have already seen. Come to the right side, change it to a private inheritance. So, called, this functions are the same, the creation of the instances are the same.

Now, a Person p can eat. So, this is fine but since this inheritance is private, the compiler does not convert, compiler does not convert a specialized object to a generalized object. So, eat s, look at the eat s between these two, eat s was fine on the left because first Student is a specialization of Person. Here it is merely a matter of syntax and certain semantics but it is not modelling specialization. So, by this Student is not a Person.

It is just happens to be programmatically related to the Person class and compiler honours that by not allowing this conversion. So, poor student he or she does not eat. So, this is the basic difference between public and private. I think through this example, I am trying to again reiterate the point that ISA will be applicable only when you do public and the compiler honours that, knows that.

And here it is merely a language syntax with associated semantics and never confuse it for an inheritance in the object oriented sense or generalization specialization that we have been discussing.

(Refer Slide Time: 11:40)



I will show you an interesting application of private inheritance. This is a example is courtesy Scott Meyer's from his effective C++ or effective modern C++ book. The idea is nice, that suppose, like we have the general idea of the classes and objects and so on and we have seen that there are in the real world certain situations where I want to design classes having a certain property.

One that we have discussed in depth, you remember when we are doing the static discussion, on the static data members and member functions is the concept of Singleton Class. Can I design a class which can create, allow creating only one object which has a global point of access and we showed how to do that. So, here I talk about a class which is, I want to design my class and the property that I want is the objects must be unique.

Suppose I am just saying, Scott Meyers book use a different real world example but I am just

saying that, suppose I am trying to model a boutique of say dresses, or fashionable dresses. So, a good boutique designer like Shweta Pal or Sabyasachi will say that every piece that is designed is unique.

You will not get a second piece because if you get a second piece, that becomes like the kind of shirt that I am wearing. You will get probably hundred thousand same pieces of the shirt all across the country. But when you go to a fashion boutique, you pick up items and you pay for that because it is unique. Besides it is good, it is nice looking, it is durable and or whatsoever.

So, this uniqueness is very important. What does that mean? Think in terms of, we are trying to model real world, in terms of the C++. So, I have my class, I have written some my class. I can create objects of my class so these are unique, different unique dresses in the boutique collection of say Shweta Pal. Now, given the class we know that I can make a copy of m1 into m3 using the copy constructor.

If I write a class that is always possible or if I have two objects m1 and m2, I can assign m2 to m1. What does that mean in the real world? In the boutique world what does that signify? If I am able to do this, then I am basically able to copy one unique design into another dress. So, this models plagiarism basically. This basically models plagiarism and will destroy the uniqueness of the boutique.

If I assume assignment then what I am meaning in this? That one dress can be tailored to be exactly like the other dress, m1 can be tailored, retailored modified to be like m2, again the uniqueness is gone. So, if I use a simple design to model this, then it will not satisfy the real world condition. So, in my C++ terms what I do not want is, I do not want copy constructor. I do not want copy assignment operator.

So, what I will do is, I will be silent. I will not provide them. There is a risk and there is an inherent risk that the compiler is too friendly with me and if I do not write a copy constructor for my class or copy assignment operator the compiler will provide the free versions and these codes will still compile and run, of course using the, those concepts we have seen using the shallow copying, bit copying concepts and so on, but it will be possible.

So, the question is how do I stop that because unless I can stop that I cannot maintain the uniqueness of the real world. Do not get impatient, I am not telling this story to go away from

private inheritance that we are discussing but this background is very very important. So, what I can do is simple, I say okay I do not want the compiler to provide the free functions, so what I will do? I will provide the free functions but I will put them into private.

If I provide this functions and put them in the private then it will be fine. So, what will happen, if someone outside the class wants to use it then there will be an error, anybody tries to write this outside the class will get an error because these are provided and private member functions. Some member function within the class, suppose some developer does not understand and within the class is putting up a logic which is copying this certainly will not get a compiler error.

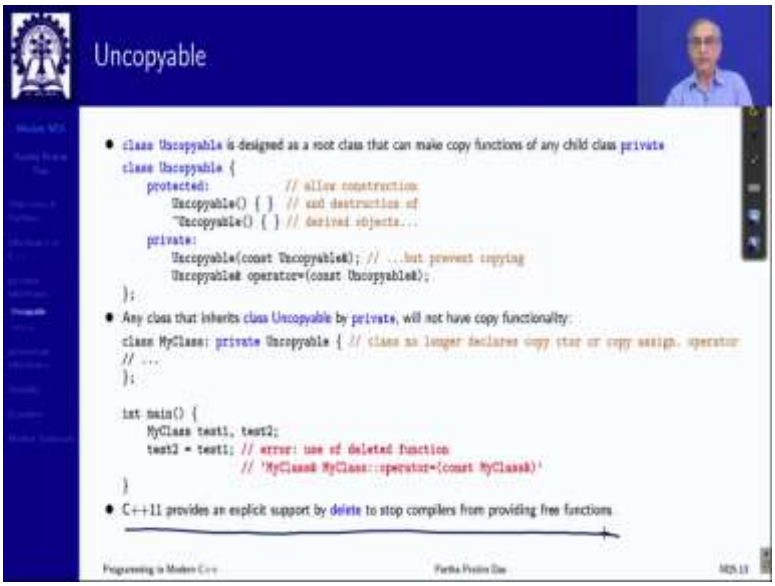
Because the functions are given but what I will have to do is not only put them in private but I will not provide their implementation because I do not want that copy. So, I do not know what the copy means so there will be nobody. There will be no implementation for this functions. So, if a member function of my class tries to write a code like this, the compilation will go through fine because the functions are there, the signatures are there.

But what will fail, the linker will give you an error because it did not find the linkable implementation of this function. Mind your linker does not bother about not finding the implementation of a function which you have never called. So, in general the linker will not shout but if a member function of my class happens to attempt to make a copy either by construction or by assignment then the linker needs a body for it, implementation for it and the linker will shout.

So, in a lot of roundabout way you are finally able to ensure that neither the external world nor the developers of this class or those who can make changes to the member functions in this class will be able to make a copy of any object. You can only construct object, that is all. So, uniqueness is guaranteed now, if you look into this entire design and I am sure, rather I am not sure that whether you have understood the whole design process here if you are not just rewind back and listen to it again try this out, is quite cumbersome, quite complicated.

So, what is, the idea that we want to do is, could I have, could I have a base class so that any object any class that I want to have uniqueness, any object that I want to have uniqueness will simply inherit from that. So, the way I am trying to see this is in this base suppose I have a property that objects cannot be copied and then I am saying that any class I create, I inherit from that base and inherit that I cannot copy.

(Refer Slide Time: 21:08)



Now, this kind of inheritance will not work with the normal public inheritance. So, let us see what could be the design solution. So, we design a, such a base class called uncopiable and in that uncopiable base class, I put exactly the design paradigm that I said it has constructed destructor, which are dummies nothing the trust will not do anything, does not have data members, does not need to.

And it puts the copying functions into private so the compiler does not give anything and the copying functions will not have any implementation. So, all that I told in abstract terms or maybe rather not in abstract terms but all that I prescribed for my class is now, done in terms of a separate tiny minimal class called uncopiable which stays here uncopiable. Now, I say that my design paradigm is that my class will simply inherit from uncopiable by private.

It will inherit uncopiable by private. So, the class if it inherits by private then even if compiler provide, I will not provide naturally copy functions for my class, the compiler will provide the copy function for my class has to, but the compiler provided function will not be usable because to be able to copy say copy construct my class object, it needs to call the copy construct of the base class object. And that is private, that is private in the base class, so it cannot call.

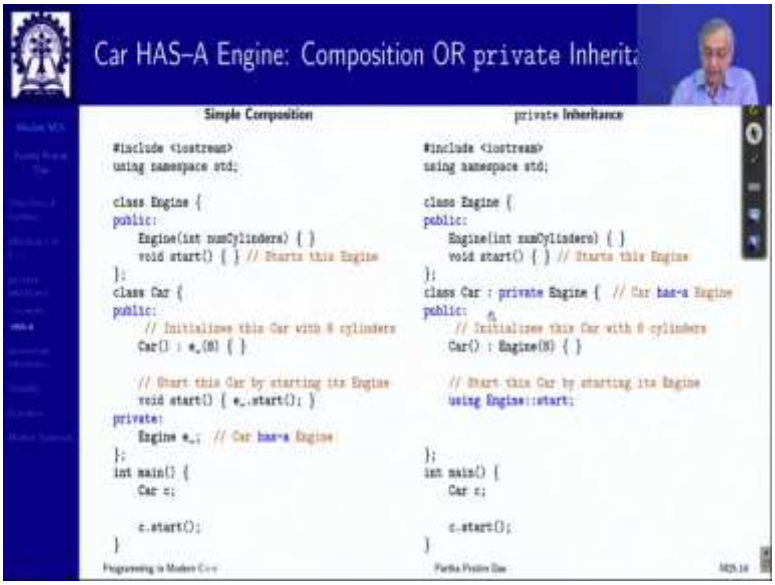
So, if there is any attempt to copy objects for my class, the compiler will try to give me or compiler will give me the free copy constructor but that copy constructor will not compile because it cannot use the copy constructor of the base class, problem solved. And this is where the private inheritance is one situation where the private inheritance is important. It has

a good use because it is not a case of public inheritance.

Uncopiable my class is not a generalization or specialization of uncopiable. Uncopiable is just a implementation paradigm that I want which is what this private inheritance is providing me. So, this is kind of, I mean I just possibly Scott Meyer did that and I am also trying to do that, is kind of support private inheritance to still stay there to find some use for it and this use is actually very very important.

Particularly if you are programming in C++ 03 there is no easy way that you can stop compilers from giving you the free copy functions. So, you can use this, just in the passing I will make a note and we will come back to this when we do the modern part of the language that C++ 11 solves this original problem by providing explicit support for delete that if you do not want the compiler to give you the free function you can declare the free function and then say delete on that signature of the free function and the free function will not be provided by the compiler.

(Refer Slide Time: 25:28)



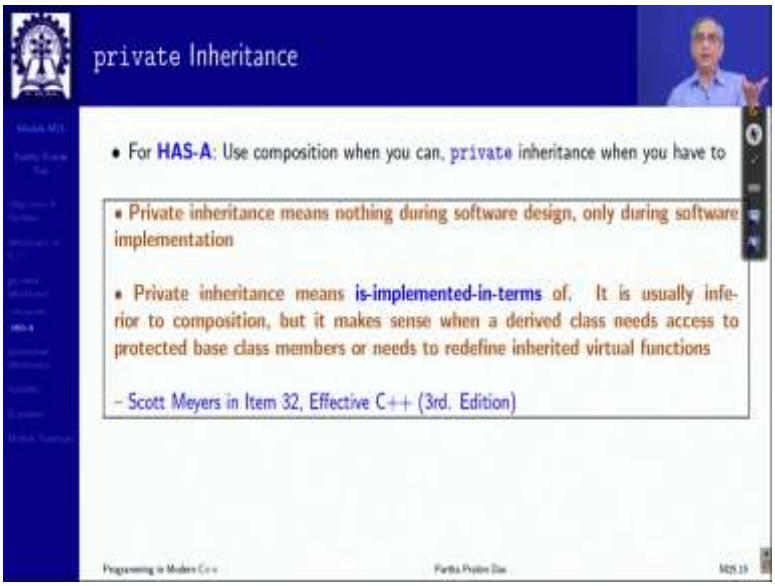
Now, the other way to look at private inheritance is in terms of HAS-A relationship, HAS-A is basically something is a component of that, ISA specialization, HAS-A is a composition. So, I have a car, car has an engine. So, it is basically having an embedded object. So, I have an embedded object so car has an engine is modelled by composition. You can also model this by private inheritance, you can say that I have a engine and private engine.

Why two questions? One is why this is not public inheritance because car is not engine rather

the car is implemented with an engine but you can get that similar paradigm because all that you need in composition is the object you are being composed of is a part of your object. So, engine object must be a part of the car object. So, composition directly gives you that. So, does this inheritance mechanism also gives you that.

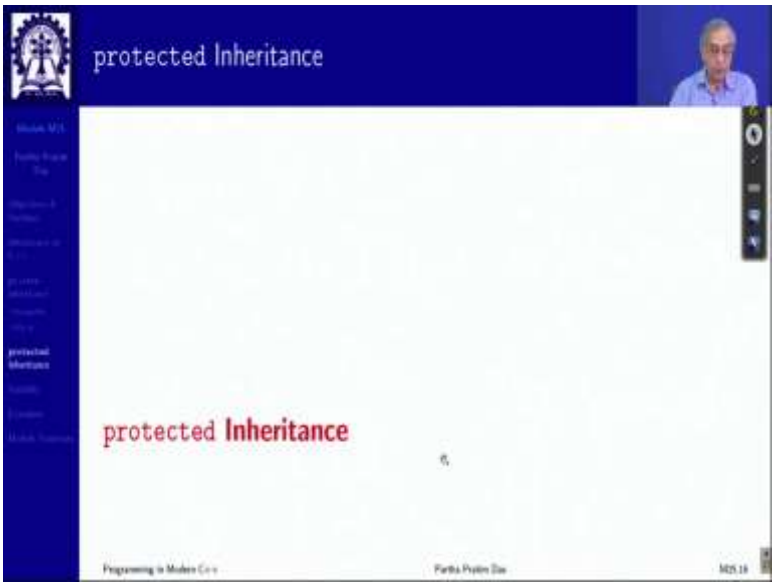
So, that part of HAS-A is a component of is common between doing the embedded object and doing an inheritance object. But I cannot do public here because that is not the concept that I am trying to implement. I am just trying to use this feature that engine object will be a part of the car object. So, I can use private inheritance for that another use case for private inheritance. But mind you, none of this are very strong use cases but these are the places where private inheritance have been used from time to time.

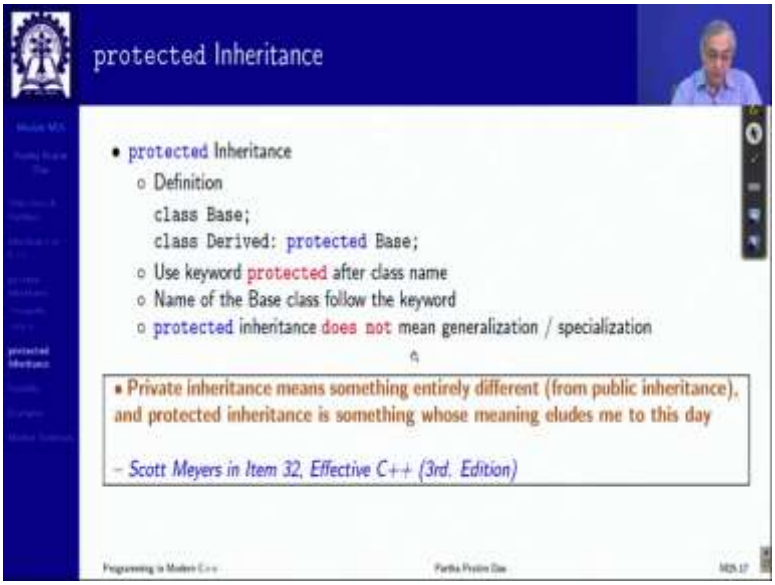
(Refer Slide Time: 27:31)



So, even after this possibility for HAS-A relationship never use composition whenever you can use private inheritance only when you have to and in in terms of object oriented analysis private inheritance does not have a name, it is rather Scott Meyer suggest that we think of it as is implemented in terms of like car is implemented in terms of the engine and so on.

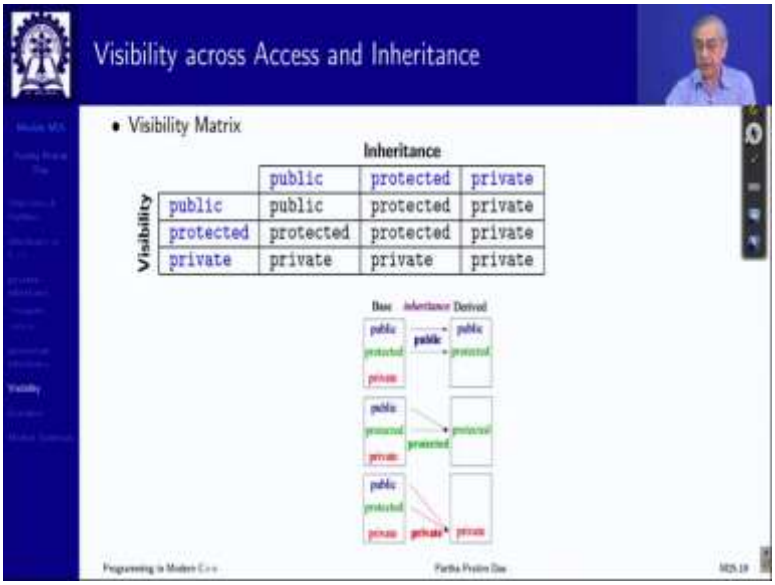
(Refer Slide Time: 28:02)





So, that was a story of the private inheritance paradigm. You can similarly have public a protected inheritance. Now, the question is, by symmetry it is there but it is neither means generalization or specialization, it is not the ISA and to be very frank nobody has actually found a use for this protected inheritance, it is just possibility was given for symmetry and then no use case was actually found. So, bottom line, do not use it.

(Refer Slide Time: 28:38)

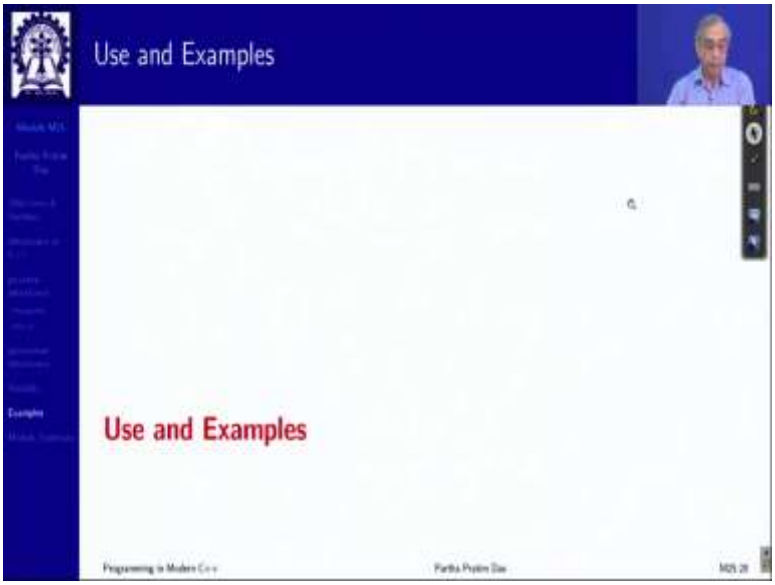


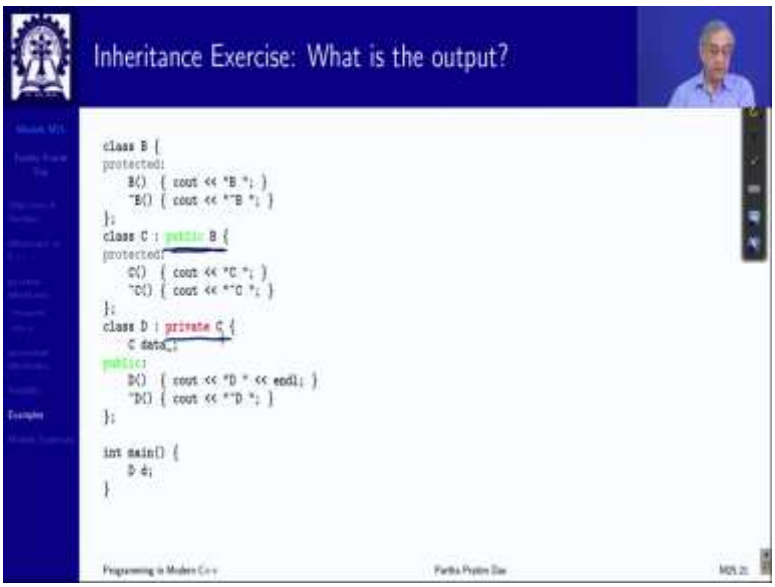
Now, the question is, there is a consequent question of what happens to the visibility of data members when you have different kinds of inheritances. So, your original visibility of data members in the base class are like this public protected or private and there are inheriting by different means now, possible means, 99.99 recurring percentage of time you are doing this because you are into ISA relationship.

That is your basic hierarchy so that does not change the visibility, public is the most relaxed so here the visibility does not change. But if you happen to inherit by private then in the derived class the visibility of all members become private irrespective of what was their visibility in the base class. So, it is like putting restriction naturally you can make out that protected has a similar restriction.

This is the basic common diagram you will find many C++ books, in early days, I should say 20 years back or so people used to spend lot of time discussing this visibility restriction under inheritance but I would not do that because time has shown that basically hardly have any use in C++. So, all that you are doing is in public and even if you do there is a little bit in private.

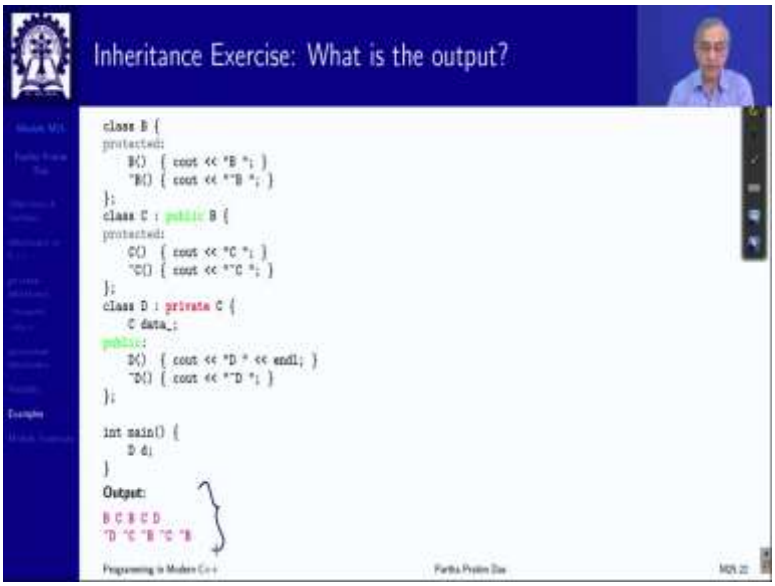
(Refer Slide Time: 30:31)





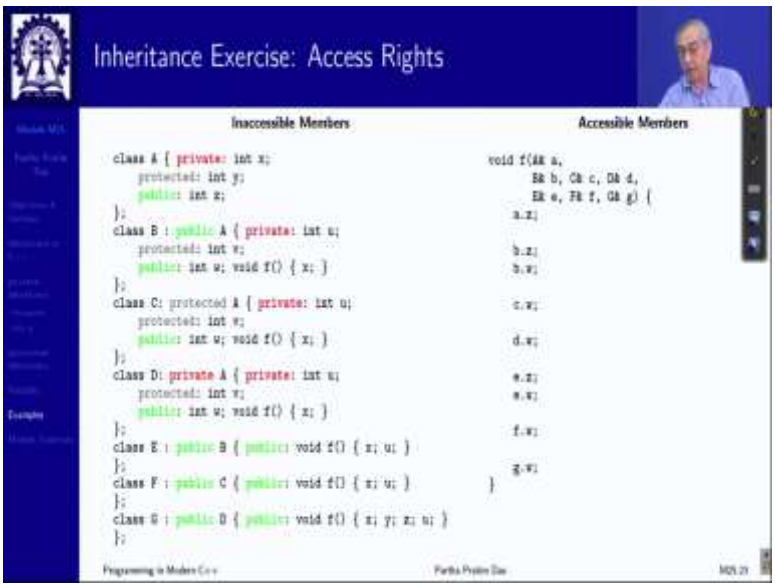
So, its most often you will not have to really bother about this visibility aspect except for the examination and interview where you will be given questions on this and you will be asked questions just to know whether you understand. But I am talking more from the engineering perspectives. So, before I close, there are a couple of example slides that have put this is C is a B by public inheritance. Here just D is implemented in terms of C as we have learnt private inheritance and if we do this what will be the output.

(Refer Slide Time: 31:12)



This is what you will get to see, it happens in this way. So, this is just shows you the order in which things happen with private inheritance and so on. It is not semantically very meaningful but it is good to know these examples, you can practice that.

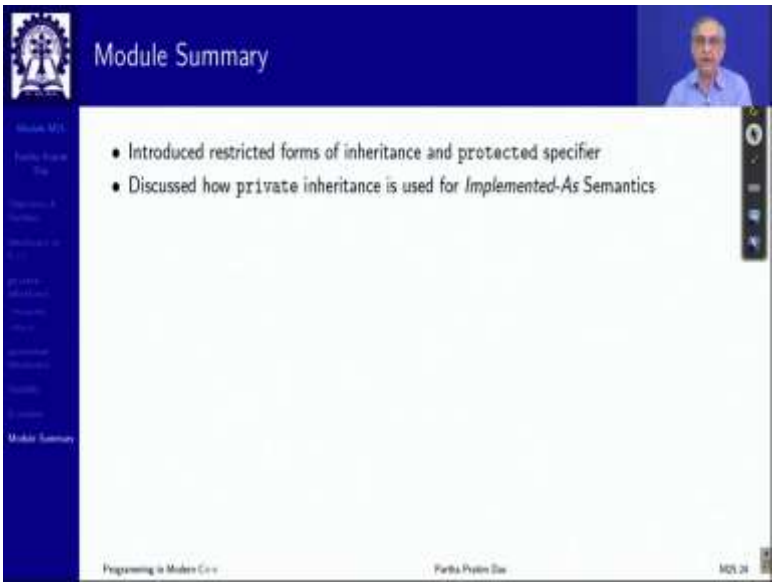
(Refer Slide Time: 31:28)



And finally just as an illustration on the way the access rights or the visibility change based on the inheritance type. Here I have created an exhaustive example where class A has three types of members, class B is a is public inheritance from A and has three types of members and its inherited so it will also have the members of A, class C is also inherited from A but in the not in the public manner but in the protected manner.

Class D does that in the private manner and then class E, F, G are pure, is an inheritance from B, C, D. So, this is just you know the visibility matrix that I showed you, this is just a code equivalent of that and along with that these are the test points you can run this and see really what is the accessibility that it is working and what accessibilities will not. As I said, I will not like to spend a lot of time on this because to me what really matters is a public inheritance which is the modelling of the relationship.

(Refer Slide Time: 32:47)



So, that brings us to the end of this little queer kind of module that I discussed where two main features of private and protected inheritance I have discussed but with the strong recommendation that protected possibly will never use private if you are having to use or you are planning to use just think over 10 times.

So, thank you very much for your attention and I think in this week we have covered a very very strong second principle of object oriented analysis and design which is inheritance or specialization generalization hierarchy. In the coming week we will build up further on that look forward to it and thank you again.