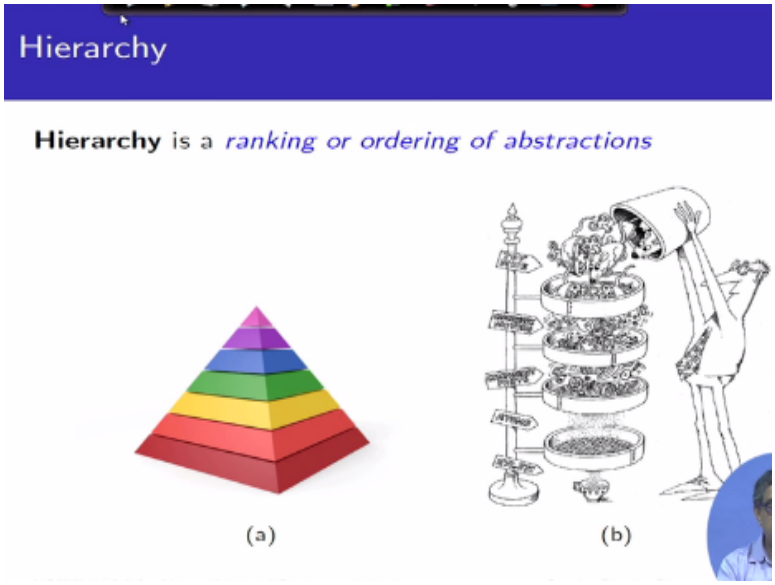
Object-Oriented Analysis and Design
Prof. Partha Pratim Das
Department of Computer Science and Engineering
Indian Institute of Technology- Kharagpur

Lecture-15
Elements of the Object Model (Major): Modularity and Hierarchy (Contd.)

Welcome back to module 9 of object oriented analysis and design. We have been discussing about elements of object model, last module had discussed about abstraction and encapsulation and in the earlier part of this model, we have talked about modularity and different aspects of what a modularity should try to achieve and what are the basic principles of intelligent modularisation.

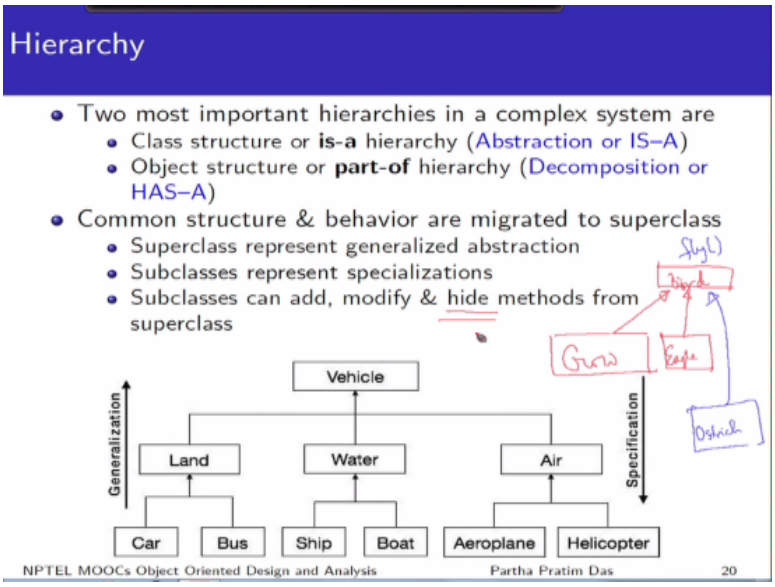
(Refer Slide Time: 00:51)



Next we take up the last of the 4 major elements that of hierarchy. Hierarchy is a ranking or a ordering of abstractions so you would have seen a glimpse of this in module 8 when we talked about abstractions and we said that abstractions could be hierarchal as well, so you got initiated to that, but hierarchy by its own requirement is eligible to be in other important element for object models and is described as such.

So it is a ranking of abstraction so which mean that the different abstraction that the system has that goes from details to less details to, even less details to absolutely top levels so it is kind of in an artistic view, it will give you a pyramid like structure to work with. So if you are at the top of the pyramid, then you get to see only very simple the whole thing in very simple terms, you use the abstractions only at that level.

(Refer Slide Time: 02:09)



Then as you go down, you see more details as you go down you see further details and different kinds of hierarchies play role and let us take a look into those. What we are talking about here again you have already had some glimpses of it, so when we talked about specifically the canonical form of complex systems. We exposed you to 2 work through orthogonal hierarchies that are common most important for complex system.

The hierarchy of class structures which is typically called as abstraction hierarchy or IS-A hierarchy and the hierarchy of object structure which is called the part of or decomposition hierarchy or HAS-A hierarchy. The class structure hierarchy shows you how over the hierarchy the common property is a share between one layer to the next and the object structure show you a hierarchy of containment.

How one object can be thought of having been composed of different components or different component objects and this in turn can be thought of in terms of sub components and so on. So in this context, the common structure and behaviour are migrated to what is known as super class, so what we say is in a hierarchy, it class that occurs higher in the hierarchy, hierarchy is a typically drawn in terms of inverted freeze.

So that classes or the abstraction which occurred at a higher level in the tree and known as super class and the related lower level abstraction is called as sub class. So if we look into the diagram below, this is we say is a root abstraction, so at this root, a system is being defined in

terms of one single concept vehicle, so if you say a vehicle all of us understand something. So that could be lot of different as you say specialisation.

But vehicle we can say is combines all of them together by one unified concepts that it is a mechanised gadget to transport. A vehicle is a mechanised gadget to transport. Therefore, if this is the general concept then we can always say that vehicles at least are of, can be of free types that which transports on land, the land vehicles which transports over water, water vehicles and which transports through air, air vehicles.

There could be other kinds of vehicles also like vehicles going for space, vehicles which can go over more than one medium and so on we will not go to those kind of complexity but when we say this, this is what is a concept hierarchy or abstraction hierarchy, so this is what we say is a super class and let me use the different colour, these are the subclasses, now please understand that the super class and sub class are relative notions.

So vehicle is a superclass, relative to land, water or air vehicle, but if I just to focus specifically on the water vehicles, let me pick up red again, if I specifically focus on the water vehicles, then the ships, the boats, they are the subclasses of this water vehicle. So this now becomes the superclass, this is the subclass. So it is in the hierarchy, if 2 concepts are related in terms of generalisation and specialisation then we will use the term superclass for more abstract or more generalised concept or abstraction.

We will use subclass to represent specialised more specific class or more specific abstraction. Now as we go for this hierarchy, then on the 2 sides, there are 2 directions showing that certainly if you go from below to lower level to a higher level, then you are actually generalising that you are losing special properties so as you go from cart to land, you lose the property that cart is far limited transportation.

Transportation of limited number of people or limited volume of goods, whereas necessarily is for transportation of large number of people. The commonality that you can out here, as you go up is both of them run on land and similarly you have certain commonality with water and when you go to vehicle you lose even more, you lose even the fact that on which medium the vehicle transports, you just retained the generalisation that the vehicle indeed transports.

So this is what would keep on happening, so when we go the other way, that is when we go from generalisation to specialisation, then a subclass can add, modify or hide methods from the superclass. So vehicle says that I can transport. When I come to water vehicles, then it will add that I can transport of course, I have got that from the super class vehicle but we will add that I can float.

This is not the requirement of vehicles in general but water vehicles will have a requirement of float. I will add different other factors like I can possibly go with much less loss of power because water provides far less resistance than land. So I have more and more properties coming in here and then when we go for the down, then ship has everything that water does, water vehicle has but it has something more.

It says that it is a big water vehicle which is typically motorized and so on. Whereas when I talk about boat, then I will typically have water vehicle which are manually driven with will have oars and all that ships will usually, at least these days will never have oars, they are driven by underwater propellants and so on, so as we go along the specialisation from top to bottom, we can add and modify different methods.

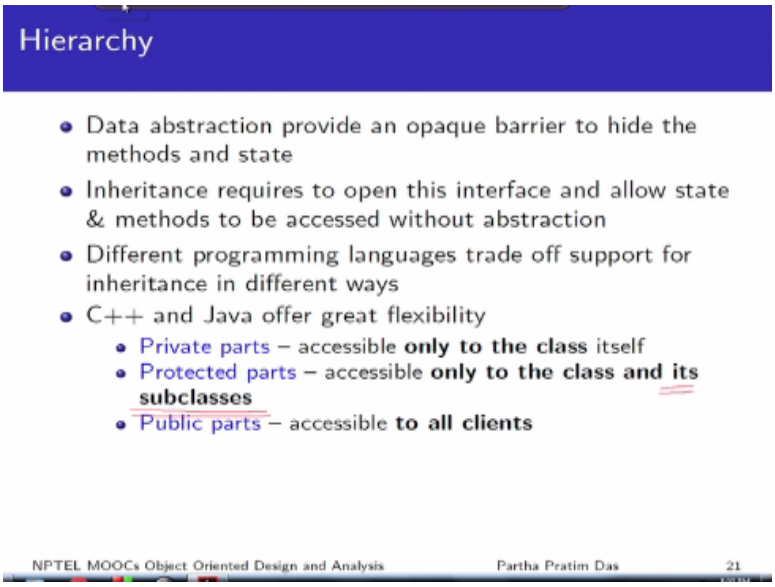
In fact, there could be instances where we main into hide a method. For example, I will just by the side show you an example, suppose I am talking about a concept bird, so I can say that crow is a bird, because anything the bird can do, the crow can do that, may be crow can do something more, for example crow can creates a kind of sound in a certain way. I may have eagle, is a bird, it can do anything that the bird can, but it can do something more.

It can go to really very high heights but let us now look at that I want to put here say ostrich, all of us know ostrich is a bird, we know it is a bird. But we also know that a method or a property after being a bird is to be able to fly, ostrich we know cannot fly so we get into certain kind of contradiction that our hierarchy rule say that a subclass will get all the methods, it can add new methods, it can change that methods.

But it will get all the methods at the superclass has. The superclass has a method called fly, birds can fly, crow inherits that, crow gets that, crow can fly, eagle gets that, eagle can fly, eagle can add other method that it can glide which crow cannot but I come to ostrich, it

cannot fly, so this is the specific example to show that on a hierarchy, we my often need not only to add new properties to specialise.

(Refer Slide Time: 11:27)



Not only to, may be change the way the method works, change the way the birds fly, they change the way birds reproduce and so on but you may need to actually omit certain methods which has superclass has, so you need to look at all of add, modify and hide methods from super classes. So hierarchy basically provides data abstraction and I mean hierarchy provides on top of the data abstraction it tries to provide a mechanism by which the properties can propagate from one layer to the other.

We can see that once we have done abstraction so we have already done 3 major elements so we know that here is abstraction, so I have abstracted a concept then we know on that abstraction where putting encapsulation, so moment we have put in encapsulation, we have put an opaque barrier to hide the methods in the state, we said implementation is not feasible. Because it is encapsulated. Now I want to say that now I am coming and saying that here, I have vehicle.

It can transport and then I have land vehicle then I have car and so on. Now if I have done my abstraction and encapsulation in the right way then this is encapsulated. The properties of vehicle are encapsulated. Then how does land vehicle define how transport happens? Because vehicles in general cannot say how transport will happen, it is just an abstract concept of transportation, land vehicle has to do transportation using certain physical laws.

All vehicles need to do transportation using certain physical laws, land vehicles use friction or (()) (13:12) of friction, gravity and all that, water vehicles use buoyancy, third law, etc. Now so there is an inherent contradiction in terms of the way we have abstract and encapsulated when we want to create hierarchies, where we want properties methods to propagate from a superclass to the subclass. Now this is something which is specifically handled in different ways by different implementation languages, different programming languages.

Particularly, one of the reasons that C++ or Java gets very popular in terms of object oriented programming is a fact that these languages allow you a great flexibility to provide mechanisms by which you can honoured the abstraction and encapsulation, at the same time, create hierarchies, so though we are not going to discuss in depth about the language constructs in this course.

But it will be good to understand that there are concepts like visibility which say that you can, of your encapsulation you can create different grades of encapsulation, different porosity of encapsulation, certain encapsulation are known as private which are totally encapsulated only if you are inside that class, only if you are inside that abstraction, you will able to use those properties, use those methods.

The other extreme is basically to provide the exposing part of the encapsulation to honour that contractual interfaces that you wanted to provide which is a public part, the public visibility which anybody and everybody can have a look at. Now these 2 do not solve your hierarchy problems, So C++, Java these kind of languages introduce a very interesting kind of intermediate visibility or intermediate encapsulation granularity.

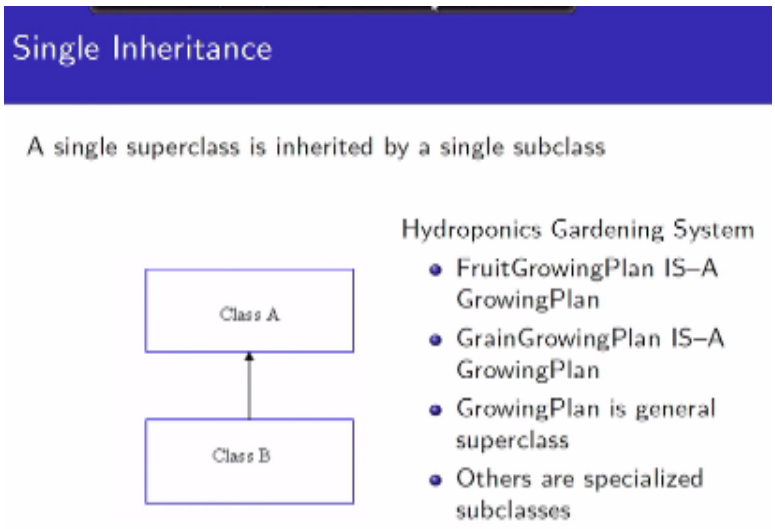
Where you say that you are different implementation parts are protected, by which they are available to the class as the private ones are, as the public ones are as well, but it is specifically available to this sis the main part of protected, it is available to the subclasses, but these are not available to public, so protected parts do not become the part of the contractual interface.

They are just an extension of the implementation of the structure, but they can be propagated along the hierarchy to make that realisation of hierarchy is possible, so it is kind of, little bit

started you know my great in between the conceptual aspects of object oriented analysis and design and slowly taking you closer to how it will look in the implementation, so I just included this part to be discussed as a part of the hierarchy to give you a glimpse.

It is not that everything in this whole paradigm of things are fits into together exactly seamlessly, certain concepts which we need to impose like hierarchy, which is very critical to manage complexity contradict some of the other assumptions you made for abstraction and encapsulation, but this is just to highlight that do not get worried because the languages when you finally come to system implementation, the languages will be able to take care of that.

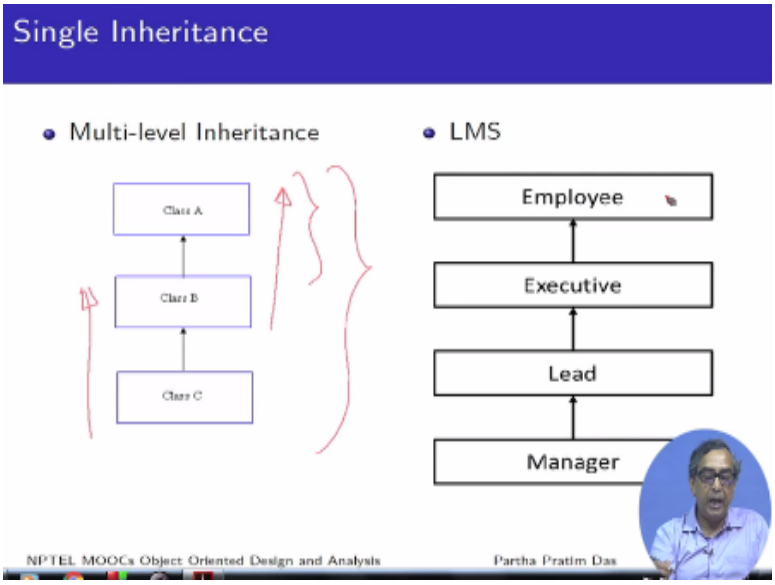
(Refer Slide Time: 17:17)



We will see when we talk about UML that we translate all these different elements of the objects models into UML language, we will able to express this varied kind of visibility and porosity of encapsulation over the UML hierarchies. Now in terms of this there are these are more like of terms you need to understand and remember. On a hierarchy we say the term single inheritance is referred to when there is one superclass.

There is a specialised subclass, so we will say, [No video or audio from 17:41 to 19:47] here in a single inheritance, we will say that this is a single super class and this the above on we will write it as B is a A. So that is the simple presentation so that is called the single inheritance and if we just refer to the Hydroponic gardening system then fruit growing plan IS-A growing plan.

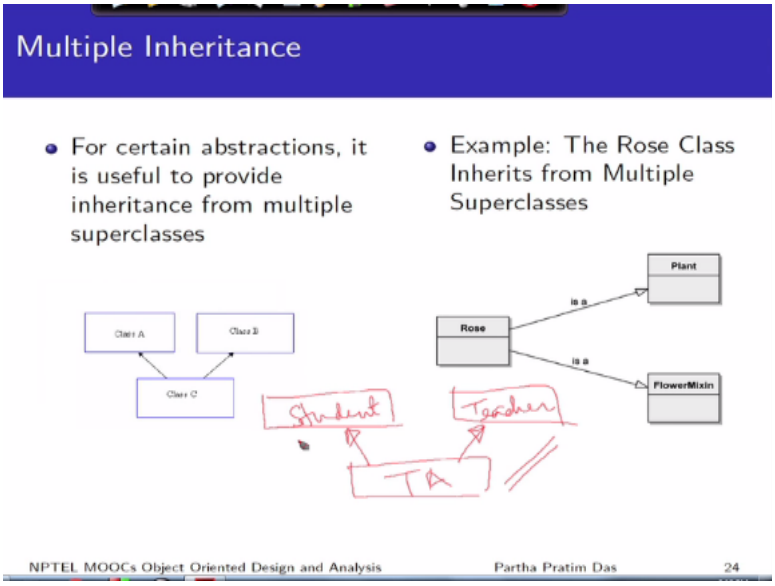
(Refer Slide Time: 20:24)



So these are all specialisations and these are specialisations and these are generalisations and growing plan may be the most general super class that it will have. Single inheritance can also be extended it multiple levels, so here we have the same, we have B is a A and then we have C is a B, we have already seen instances of this in terms of the vehicle example, so when this happens then we say it is multilevel inheritance.

That is inheritance is happening not between just 2 levels but it is more than 2 levels, so a manager is a lead LMS, leave management system a manager is a lead, can do anything that the lead can do, lead is an executive, the same property is be an executive as an employee, so employee happens to be the most generalised super class and manager happens to be the most specialised subclass in the whole hierarchy of multilevel inheritance.

(Refer Slide Time: 21:14)



A different concept come in terms of multiple inheritance, in multiple inheritance, you basically have more than one super classes which is jointly specialised into common subclass so whenever that happens we say that we have multiple inheritance because when this will happen then C, the class C will enjoy the properties, methods of class A and as well as class B and multiple inheritance is a regular reality of art life.

Multiple inheritance come from the different roles that we keep on playing all the time, for example, let us think about just I have been regularly, I am getting regularly assisted in this course by 3 teaching assistance (TAs) Himadri, Srijoni and Tanwi and so think about the classes of students and teachers and TS and what will happen, we will have students, what is the distinguishing property of a student? Why it is a key abstraction? Because, student study.

We have teachers, what is the distinguishing characteristics of the teachers, they should also study but distinguishing characteristics of the teachers, they should also study, but distinguishing character is a teach. What is a TA? The TA has to teach because she is a teaching assistant but to be a teaching assistant to be a student, so a teaching assistant a TA is kind of specialises from, it gets certain attributes certain properties from the students, certain properties from the teacher.

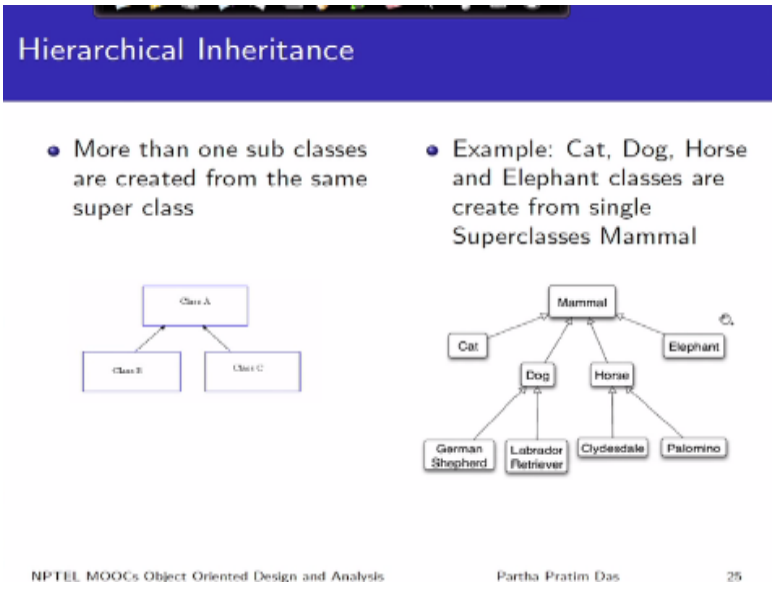
So this one very direct example of multiple inheritance we will need and if you think little bit more, you will find that you will need all different kinds of properties of rather power of subclass, subclass TA to add, modify and hide methods. For example, TA can assistant the codes, TA can evaluate assignments but TA cannot award the final grade, which teachers can, so if I say TA is a teacher as I am saying here.

Then TA will inherit the method to be able to award grade to students which it has to hide so but at the same time it will from the student it will get the lot of properties which will possibly, it will possibly will be able to enjoy altogether, so this is a basic context of multiple inheritance it is a, I would like to warn you this there is an another different example here which you can make out by yourself.

But I would like to warn you multiple inheritance is one of the very difficult areas of design because is not only easy to identify multiple inheritance there are lot of fundamental theoretical conceptual issues in dealing with multiple inheritance. Some of them relate to that

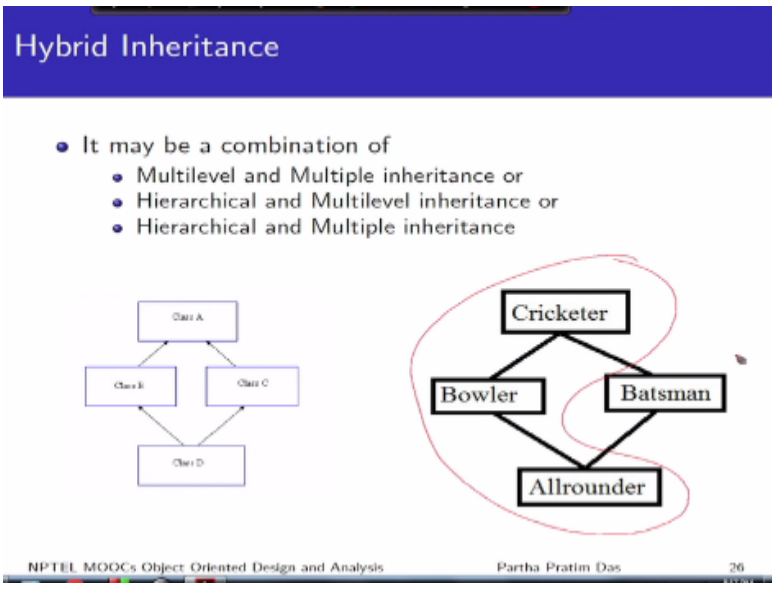
if you inherit the same method the same behaviour from 2 different super classes then how should you behave in the sub class and so on.

(Refer Slide Time: 24:53)



So there are different design principles relating to in terms of dealing with multiple inheritance and we will talk about some of those when the time of actual email modelling comes. Then next is hierarchal inheritance, which basically is when you have more than one subclass specialising from one common super classes which is what we have already seen quite a lot in terms of the vehicle example we just took.

(Refer Slide Time: 25:28)

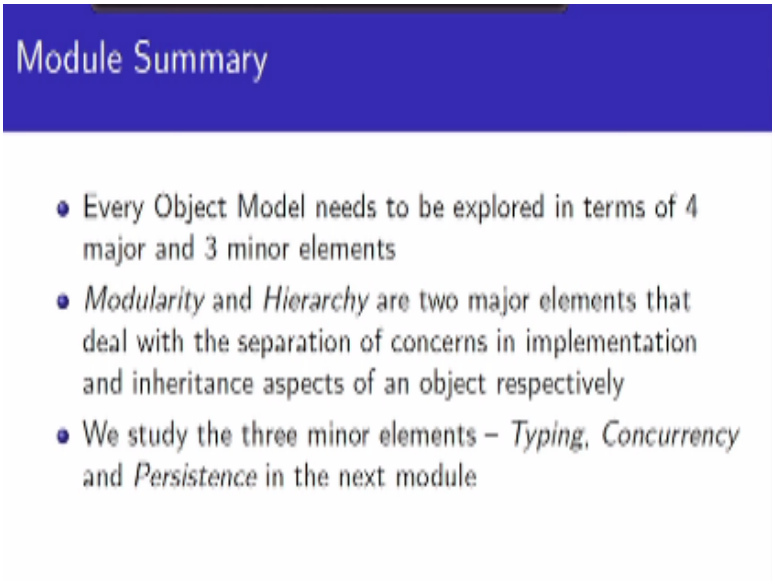


Here I have shown another example based on mammals so this mammal is a most generalised super class then we have cat, dog, horse, elephant all different subclasses, they in turn have other subclasses so when we have this kind of inheritance structure we say we have

hierarchal inheritance, now in reality what happens is, you will not have, often you will not have pure single inheritance or single multilevel inheritance or hierarchal inheritance or multiple inheritance.

You will typically have a mixture of them and therefore often you will have a situation of hybrid inheritance as you can see here, so in this hybrid inheritance, you can see here you have a more pure form of hierarchical inheritance, here you have a more pure form of multiple inheritance and if you look into this part of the structure then you have multilevel inheritance.

So hybrid could mix up different kinds of inheritance structure in this and the reason in object oriented design some on needs to understand the hierarchies and inherited structure is this is one great tool which go a long way to restrict complexity of systems because as you can understand that once a subclass can inherit a whole of a superclass with very minimal changes additions and you know hiding then you have a great opportunity for use of code.
(Refer Slide Time: 27:04)



We have great point as to a how should module arise you have a great use of the whole notion of abstraction that you have been working with. So a major consequence of hierarchy is these different forms of inheritance that it results. So to summarise in this module, we conclude on the major elements of object models, we had 4 of them, 2 were done earlier and here we have talked about modularity and hierarchy.

Two of the other major elements that deal with modularity certainly has deals with the separation of concern, the better separation of concern, you can do in terms of code organisation, we have better modularity implementation and we talked about several principles for that and hierarchy deals with the inheritance aspect of an object and these 2 together will go a long way supporting with abstraction and encapsulation in doing good design of object oriented systems.

Now in the next module we will take up the minor elements which are not essential but often turn out to be very useful elements to provide the actual related functionality of the system that will be in module 10.