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

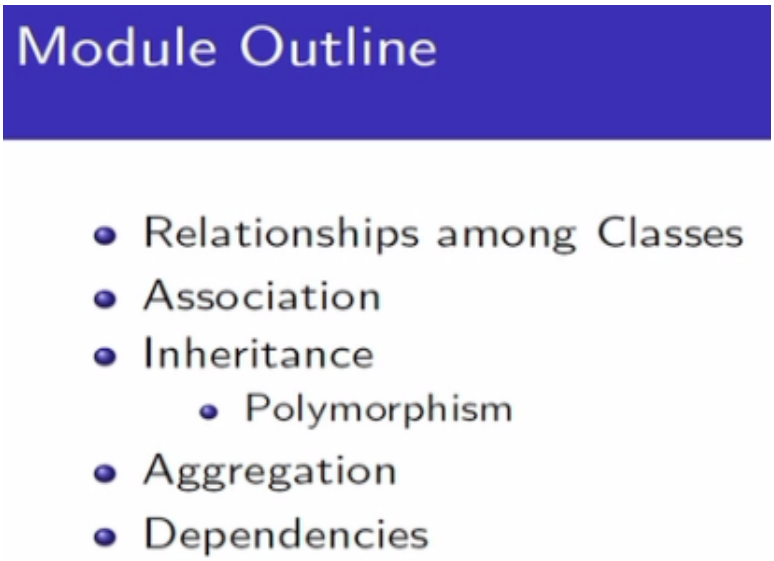
Lecture - 24
Relationships among classes

Welcome to module 14 of object-oriented analysis and design. For the last couple of modules, we have been discussing about the nature of objects and the nature of classes, we have discussed the relationship amongst objects and in this current module we will discuss about the relationship among classes.

(Refer Slide Time: 00:46)

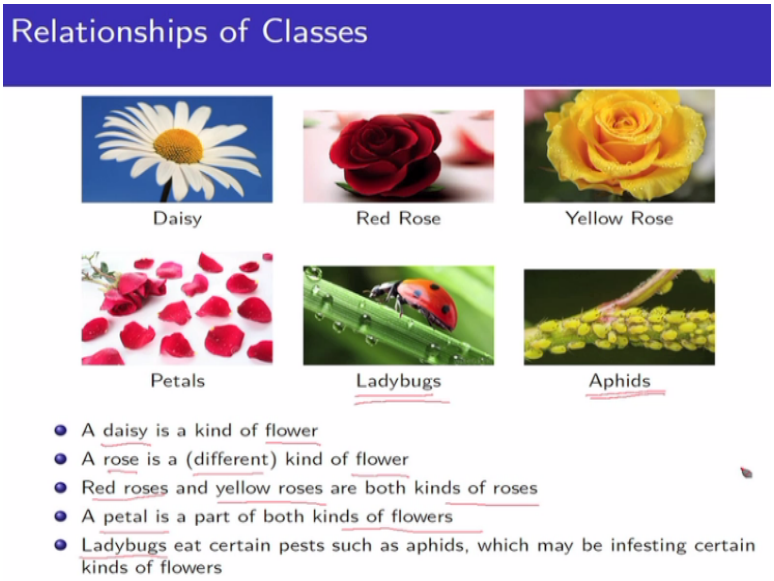
So, the single objective of this module would be to understand a variety of relationships that may exist between different classes.

(Refer Slide Time: 00:58)



So, we will briefly talk about what we mean by relationship among classes and we will discuss about 4 different kinds of common relationships that are frequently found amongst different classes of an object-oriented system, the relationship of association, inheritance, aggregation and finally dependence.

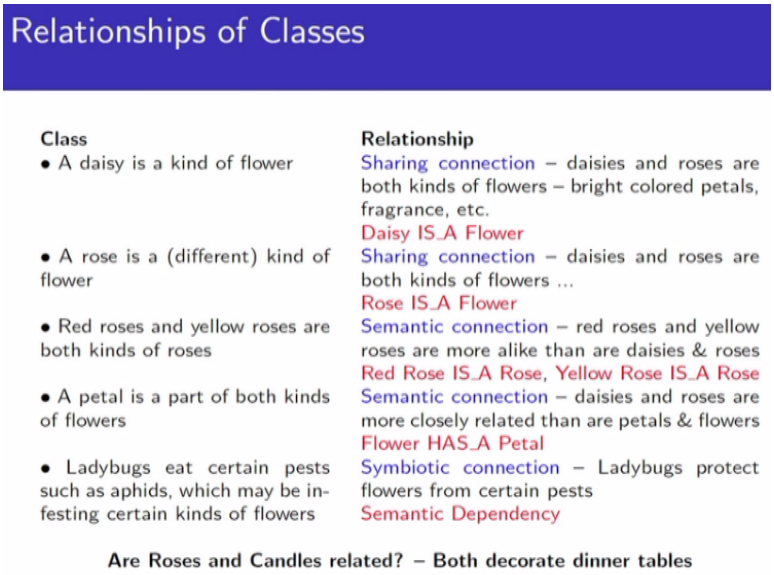
(Refer Slide Time: 01:29)



So, let us start with illustrative example. So, here the pictures are for your reference. The actual statements are below, what we are trying to say is daisy is a kind of flower. This is a statement that we are making that is what we know. Rose is a kind of flower but it is different from daisy. Red roses and yellow roses are both kinds of roses. So we can just look at the pictures for reference as to how these similarities or relationships exist.

Petal is a part of both kinds of flowers. Be it daisy or be it rose, petals are part of them. Ladybugs, these are ladybugs, eat certain pests, this is the most typical one, aphids and save the plant. So, this is a kind of real world scenario that we are trying to model, that we are trying to represent.

(Refer Slide Time: 02:39)



So, if we look into these 5 statements and first what we may do is we may quickly indentify the different classes that we have. Certainly flower is a class which has all kinds of flowers that may happen. Daisy is a class. It is all daisy flowers that we can think of. Rose is a class. Red rose and yellow rose they are classes. Petal is a class. Ladybug is a class. Aphid is a class. So, at least these classes we can identify and we have talked about the nature of classes in the last module.

So, following these concepts we can easily say these are the classes. Now, what is important, what we are trying to focus on now is, each of these statements actually describe some kind of relationship between these classes. So, the first is kind of sharing connection. We can say that daisies and roses, if we just take these 2 statements that daisy and roses are both kinds of flowers. They are kinds of flowers in the sense they share bright color, they have fragrance and all that.

So, we will often express this connection, this relationship by saying that daisy is a flower or saying that rose is a flower. Similarly, when we get into the third statement that red roses and yellow roses are both kinds of roses then the semantic connection is that they are more closely

related compared to what a rose is related to with a daisy. Certainly the similarity between the red rose and the yellow rose is lot more than what it is between a red rose and a daisy.

So this will get again expressed, this semantic connection will again get expressed in some kind of a relationship I can say that red rose is a rose, yellow rose is a rose and so on. In contrast, when you look into the fourth statement, the petal is a part of both kinds of flowers. Then we have a semantic connection which is kind of describing a whole part relationship that is hardly we will come across flowers which do not have petals and if we see some petals somewhere we know that there was a flower from which the petal has been fallen off.

So, we will model that kind of a relationship with say that the flower has a petal that is a flower that is the whole part which will have petals. Finally, in the fifth statement you have some kind of a relationship such as ladybugs eat certain pests such as aphids. So, if you look into these 2 classes, particularly these 2 classes, ladybugs and aphids or specifically aphids will be pests.

If we look at that then there is no kind of earlier kind of relationship of one and sharing the properties of the other or being similar to that or one being a part of another and so on but these are also some kind of important relationship because I can say that ladybugs protect flowers from certain pests. So, in my object module of this reality, we would like to model this kind of semantic dependency as well.

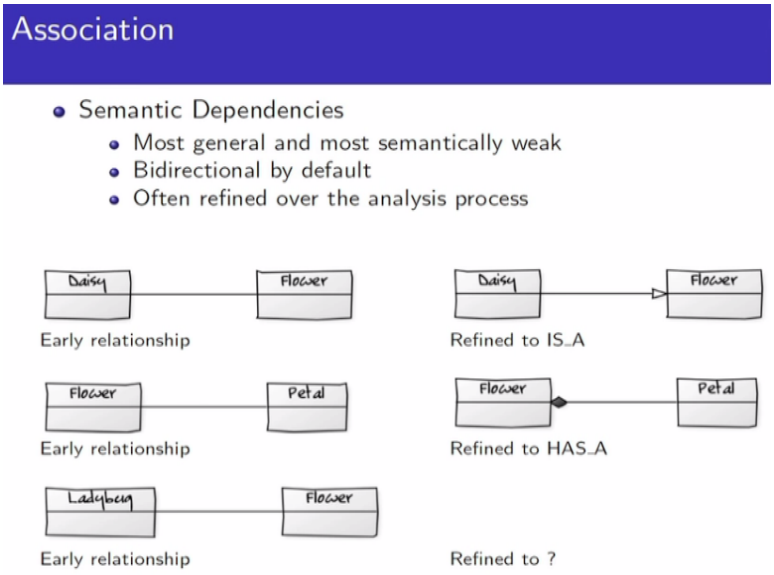
And just before we move on, just to put the question to you, are roses and candles related? Of course between the similarities between the ladybugs and aphids are more than the similarities between roses and candles because ladybugs and aphids you can say both are dependant for their livelihood and dependant on the plants. But we can also find that in some context rose and candles can be related because both of them happen on the dinner table.

So the relationship, I mean what can be the relationship and what may not be a relationship certainly has a wide range, a wide variety and therefore what we will study is some of the specific types of the relationship which are very frequent and they occur very frequently in many designs and I would just like to remind you and refer you to module 12 in this connection, where

we discussed about relationships between objects and we will see the different relationships between objects as we have observed.

They will occur in a very similar manner between the respective classes as well.

(Refer Slide Time: 07:38)



Moving on, let us talk about the first kind of relationship that we want to focus on. This is called association. Association does not have a very strict definition. It is one of the most general and kind of the most semantically weak, most general and most semantically weak kind of relationship. Semantically weak in the sense that it does not often need a very strict definition. By default, it is bidirectional in nature.

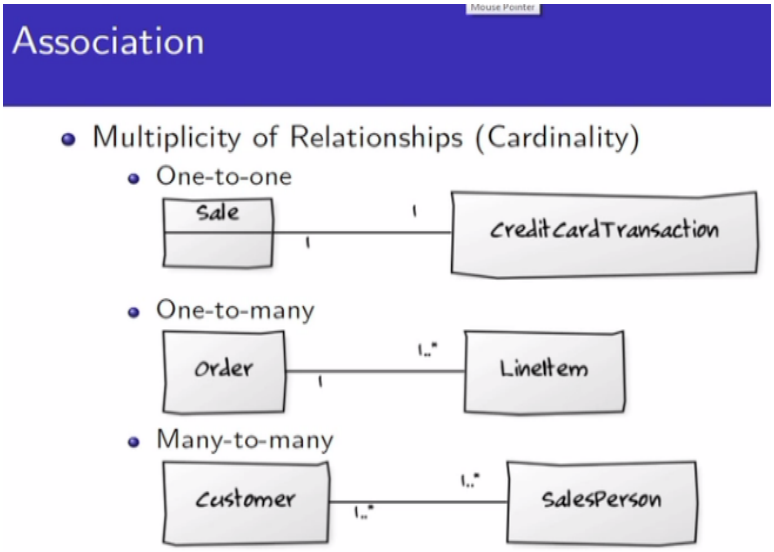
So, we are saying that there is a daisy and a flower so there is a relationship between them. In a flower and petal there is a relationship between them. Now, associations often happen in all the stages of the design. So, when we have come across a daisy, a flower, a petal, a ladybug and so on then we will just initially try to identify what are the different ways they can, they may be semantically associated and we just draw these early relationships in terms of the bidirectional links which does not have any arrow heads.

But as we go over the design, in later stages we have more and more information analyzed we may be able to refine those relationships which for association here will later on turn out to be in

more refined relationship and we will see that this kind of elevates to an inheritance relationship or whatever was just an association here between flower and petal will elevate to an aggregation kind of relationship whereas a relationship between ladybug and flower may just continue to remain equally identified.

So, association is a very generic form of a relationship which you should use whenever you find two classes have some way to interact, some way to be related to each other, whenever you find some kind of semantic dependency you should put an association between them and then look forward to further information called the analysis in the stages of object-oriented design, where you may be able to refine some of those and find stronger more structured relationships between them.

(Refer Slide Time: 10:01)



Now associations often will have multiplicity of relationship and these multiplicities known typically as a cardinality. So what is said that suppose you consider that you are buying an item using your credit card, so what will happen that on the buying side there is an order that you place and on the credit card side there is a transaction. So if this order is for Rupees say 1000 then you will need to have Rupees 1000 transaction on the credit card.

So if we identify the order to be on the sale object which the merchant has sold you and the credit card transaction is a transaction object through which you have actually made the

facilitated the payment to the merchant then you will identify that for every order, every sale that you do, every buy that you do that sale object must have a corresponding credit card transaction object.

For every credit card transaction there has to be a sale, for every sale there has to be a credit transaction. So if this is the scenario then you say that this is an association which is one - to – one, that is between this class and between this class, one instance of this class will be associated with one unique instance of this class and vice versa, whereas in some cases it could be different. For example, consider an order you are again buying, you are ordering, now often we do not buy one item.

We will buy item 1, then we will buy item 2, item 3 like this. So we will pick up all of those put them together in the cart and then buy that whole thing together. So if I look, if I model by Line item class, if I model each and every item that are placed in the order and by the order class if I model this whole order, then the relationship between them is one to many, that is one order object instance will have one or two or more Line items.

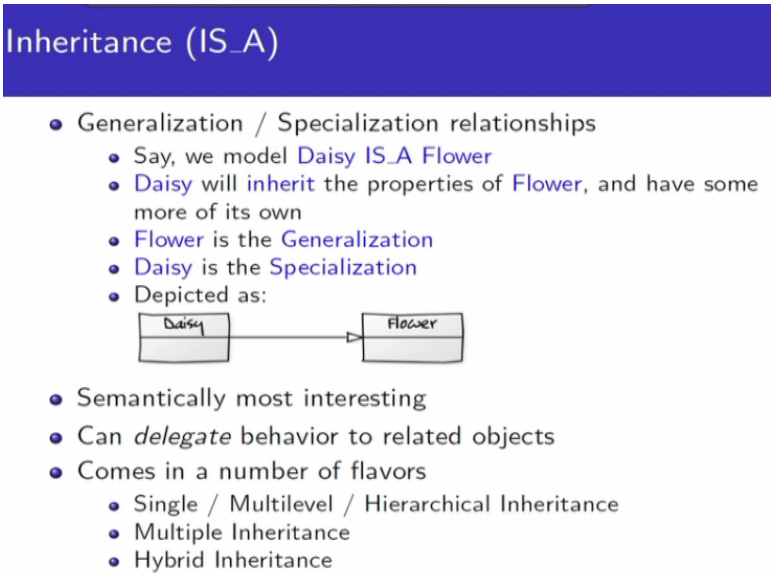
And that is what is depicted in the picture. Here like this side the number is 1 which say that for one order and this side the number is, if you are not being able to read it properly what is return is basically this. So it means may range that is one to as many as you want. So it could be 1, 2, 3, 4, 5, whatever. So these are typical notations, for example you could have written 1 to 5 something like this, where it will mean that it is 1 to 5 range.

This is a particular one which says it is one to as many as you wish; such associations would be called One - to - many associations. And then finally there could be Many to many associations, think about any sales scenario and having different customers and different sale persons, naturally every customer maybe dealing with number of different sales person. Somewhere is buying groceries, somewhere is buying dresses, somewhere is buying electronic goods and so on. Similarly, sales person is also dealing with multiple different customers.

So if I take an object here and if I take an object here then there may not be any unique relationship. One customer object may be related to one or more sales person object and one sales person object in turn would be related to one or more customer objects. That is why this is designated in this manner, these are the many to many association relationships.

So whenever we depict an association if possible we will try to put the cardinality of the relationship on both sides because and of course in this case it is not explicitly mentioned in the slide but I would like to make a note that here we are just talking about simple binary relationships. It is always relationship between two classes but there could be ternary associations also that could be, that is three different classes could be together related to each other.

(Refer Slide Time: 14:50)



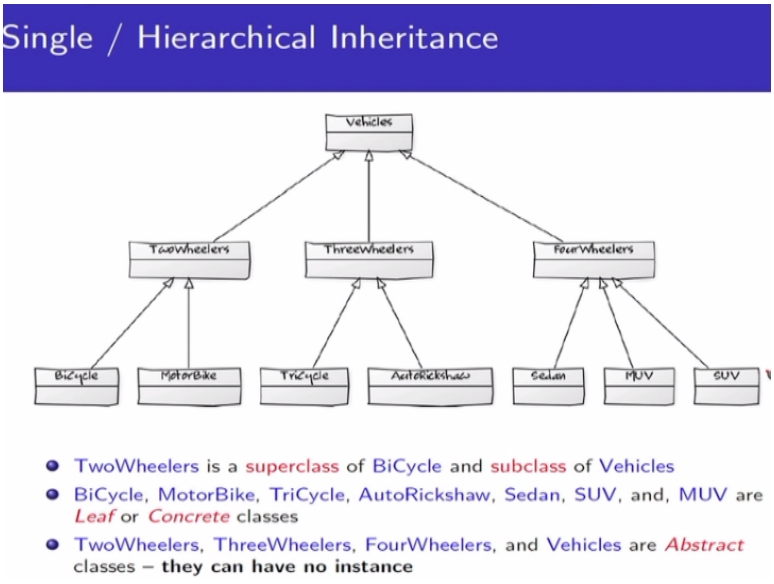
Let us move on the next relationship that we talk about is already known to you. We have talked about this, since almost the, since the beginning of the course. We have talked about inheritance or hierarchy as a general mechanism of a managing complexity and then we have talked about this in length while discussing about relationships of objects. So I will be very brief this time, because the concept of inheritance as you learned in terms of objects is very similar in terms of the classes from which the objects actually arise.

So inheritance as you know is a generalization or specialization relationship we have just seen if you model Daisy is a flower then it means that daisy will inherit all the properties of the flower. But Daisy may have some specific properties, some additional properties of its own, so that is a basic principle of inheritance, that you inherit everything from the superclass and you may add some more properties of your own and terminology wise we will say flower is a generalization often also referred to as superclass and Daisy is a specialization referred to as subclass.

If you are coming from experience of some of the OP programming languages, then you will find other equivalent definitions. For example, Superclass in C++ is often referred to as base class. Subclass is often referred to in C++ as derived class and so on. And this is the diagram that typically depict, this is a UML diagram and you can see this hollow arrowhead which depicts that daisy is a flower.

Now semantically certain inheritance is the most interesting relationship to find amongst classes and it can be used to delegate the behavior to related objects and we have seen in while discussing relationship amongst objects that inheritance could be of various different kind. Single, multilevel, hierarchical kind of inheritance or it could be multiple inheritance or it could be hybrid forms of heritance.

(Refer Slide Time: 17:25)



So if we quickly take a look in terms of the single hierarchical inheritance example, we have representing a hierarchy, here were in the top at the root are vehicles. So vehicle is something some mechanism by which people and goods can be transported you can say. Whereas so vehicles can be described as a class with very limited set of properties like, it is a mechanism to move.

It should be able to carry something and the implicit here, which is not separately mentioned but implicit here is a vehicle moves on land. We are not talking about ships or aero planes here and certainly vehicle can be driven. The vehicle will have a mechanism of transportation and so on. So these are the general properties that this class will have. But then we say that there are specialization subclasses like this.

We said that two wheeler is a vehicle, because two wheeler will satisfy all these properties of vehicle that we have just discussed. It can move it can transport people; it can transport goods and so on and so forth. But in addition to all that properties of Vehicle two wheeler has additional property that it has, it moves on wheels first of all. Second it has two wheels, it needs balancing and so on. So there are certain additional properties that a two wheeler must have.

Then we can say that two wheeler in turn could be specialized into a bicycle or a motorbike. Both of them have lot of things similar two wheels, balancing required and all that but the two wheeler does not specify as to the source of energy that is required to move the vehicle or the two wheeler. Here when you come to this level the source of energy basically distinguishes us to between a bicycle two wheeler and a motorbike two wheeler.

Bicycle two wheeler is manually driven and a motor bike two wheeler is driven by some kind of fuel, some diesel, petrol or something like that. So this is how properties continue to get specialized as you move down a hierarchy and this is exactly what we are designating here and in this some more terminology that you should be comfortable with. All these classes, a set of classes are known as leaf classes or concrete classes, in the sense that these classes do not have any further specialization.

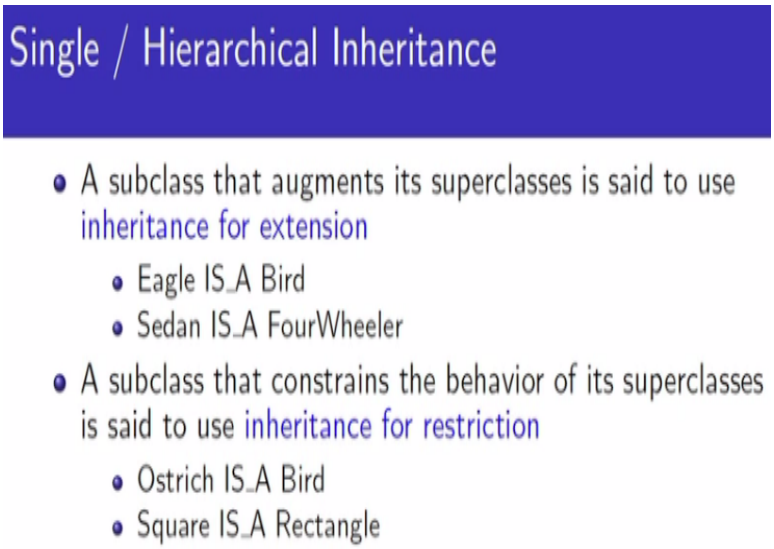
So certainly these classes exist because there are objects that will exists for these classes. But very interesting thing is these other classes, these four classes which are not leaf, the kind of, we will refer to them as kind of abstract classes. And we will go with this concept of a class being abstract more and more but just to introduce the idea, let me say that a class is abstract if it is not possible to instantiate an object for it.

Now do not get shocked in this statement because in the last module itself we had mentioned that classes have instances that are objects but here we are saying that the class, some classes are possible and these classes occur only on a inheritance hierarchy which are called abstract that is they have certain properties but they are cannot be an instances of that class, that I cannot say that specifically I have an object of a three wheeler class.

But I can say that I have an object of a bicycle class because it is a leaf node, leaf class and I can also say that the bicycle, the object of the bicycle class can be thought of as an object of the two wheeler class because the bicycle class is a specialization of two wheeler class. So the bicycle class satisfies all the properties of the two wheeler class. Therefore, a bicycle object, the specific bicycle that I own say will have all the properties of a two wheeler class as well.

But if I do not consider the leaf level if I just want to create an instance of the two wheeler class I will not be able to do that, that is what is a meaning that we are trying to say. We will come to more refinements of this concept and soon I will explain with this example only I will show you why such kind of abstract notion is required for the classes. But one point that I would like to highlight here is, in this example all leaf classes are concrete which will always be the case.

And here all non-leaf classes are abstract. Now this part is incident, in non-leaf class is not necessarily abstract and abstract class will always have to be non-leaf but a non-leaf class may also be a concrete class. That is we will explain later on, that is not a very good style of design but there are many hierarchies where non leaf classes are also concrete classes. But you can never have a leaf class which is not a concrete class, which is an abstract class.
(Refer Slide Time: 23:17)



So in terms of this let me also introduce two more notions that of inheritance for extension and inheritance for restriction. We have said that as we go from generalization to specialization, this is a hierarchy as we go from the generalization to specialization, we inherit all the properties and then add some of the subclasses additional properties. These added properties could be adding own to what we inherited but it barred it could restrict in terms of what we have inherited.

So as an example let us say if we have a superclass bird and I say eagle is a bird then certainly eagle has some properties which generically is not added in the bird class. For example, eagle can stay port without actually flipping its wings on top of a stream of hot air which generically birds cannot do. So we are adding on that property to the bird class when we come to the eagle, so this is an extension.

We can say sedan is a four wheeler is an extension because sedan has everything that a four wheeler has, in addition to that it has some more properties of having a nice (()) (24:40) and so on, possibly a nice air conditioner and all those which is not necessary for a four wheeler. On the other side, there are some cases like we say that ostrich is a Bird we all know and we know that ostrich cannot fly but a bird can fly.

So when we say ostrich is a Bird, I am actually restricting some of the properties that bird has. So this is an inheritance for restriction. Similarly, we will say square is a rectangle, but what is a

rectangle? Which has all its internal angles are right angle 90 degrees. Square satisfies that, so Square is a rectangle. But when we say I have a square then I have the additional restriction that the sides will have to be equal which is not so for the rectangle.

So here when you specialize, we are imposing further restriction on the subclass. So whenever you look at the hierarchy, it is possibly the inherence sense most of us keep on getting is, the inheritance is for extension that I am hiding more and more and more properties but please keep in mind that the addition of properties could actually mean extension or it could mean restriction in many cases as well. We will stop here in this lecture and we will discuss about the polymorphism inheritance hierarchy in the next part of this module.