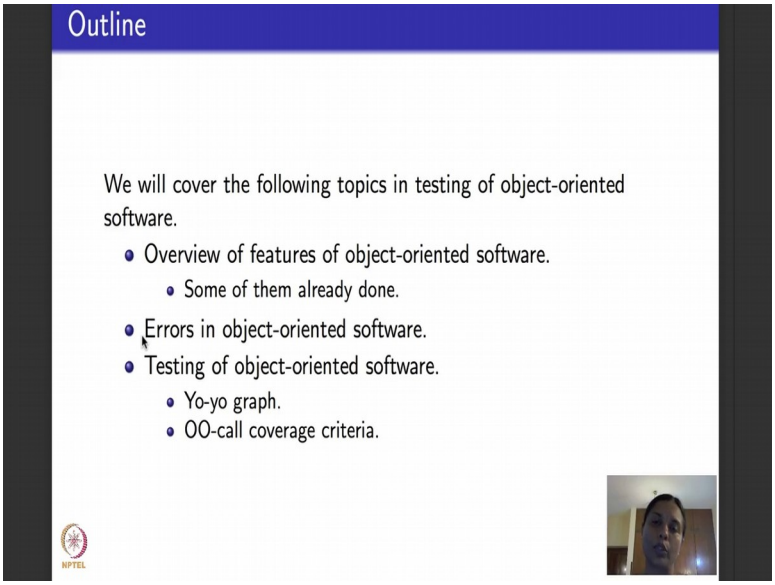
Software testing
Prof. Meenakshi D'Souza
Department of Computer Science and Engineering
International Institute of Information Technology, Bangalore

Lecture - 48
Testing of Object-Oriented Applications

Hello again, we continue with week 10's lecture. Last three lectures we discussed about if I have a web application and I want to test for functionality at the system level where the client software and the server software is put together, what are the properties and techniques we can test for web applications.

Now moving on, what I would like to begin with this lecture is testing of object oriented applications. If you see object oriented programming languages, the most classical of them being Java or even C++ are extensively used and even, in fact, Android code could also be object oriented.

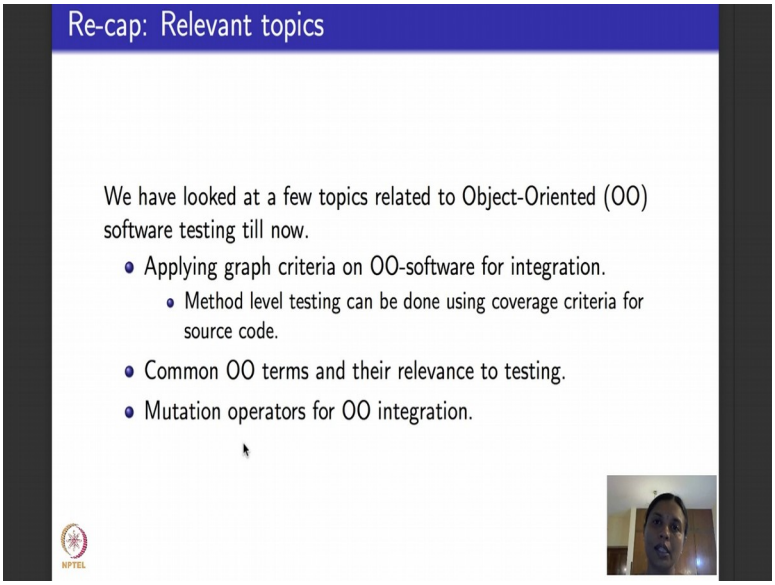
(Refer Slide Time: 00:58)



So, we will see generic testing of object oriented applications over the next few lectures beginning with today. So, what is the outline that we are going to cover in the next few lectures? We will cover the following topics I will begin with giving overviews of features of object oriented software.

If you remember when we did mutation testing, I had already given you several features of object oriented software. If you missed that I urge you to go back and look at those lectures because I will not be recapping them in these lectures following which we will discuss about errors or anomalies that arise specific to features of object oriented software especially inheritance and polymorphism. And then we will discuss testing of object oriented software very unique graph models which are labeled Yo-Yo graph after the toy Yo-Yo, arise when testing object oriented software. So, we will see that model today and we will also define several call coverage criteria specific to object oriented software.

(Refer Slide Time: 01:52)

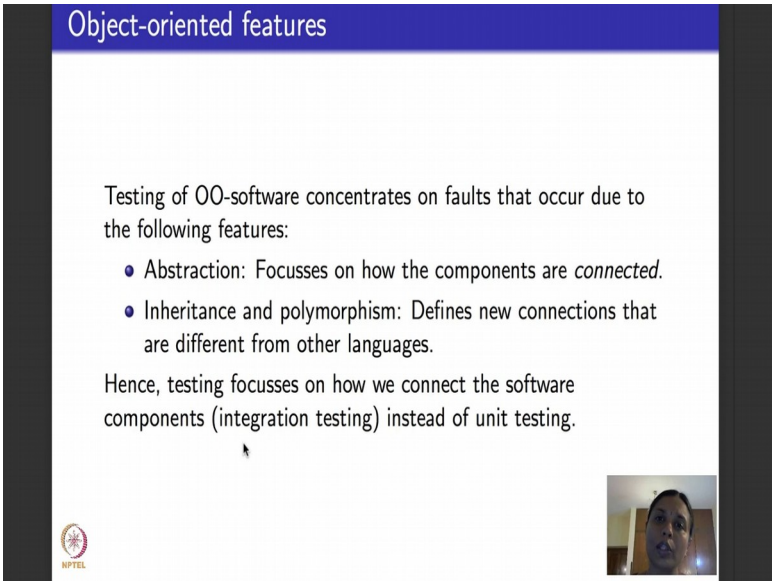


So, what have we done till now in all the lectures that we have covered when it comes to object oriented software. So, if you remember when we did integration testing, we specifically saw how to apply graph based coverage for object oriented software integration.

And in that, if you consider unit testing which means testing inside each method that can be done by using any traditional testing that we saw--- like graph based testing, logic based testing, all of them apply equally well to unit test. And then we also I told you we saw common object oriented terms and introduced them as they relevant to testing. To reiterate it was not meant to be a thorough introduction to object oriented programming, but only just a list of features as we would needed for testing and when we did mutation

operators, specifically integration testing mutation operators we saw operators for object oriented integration also.

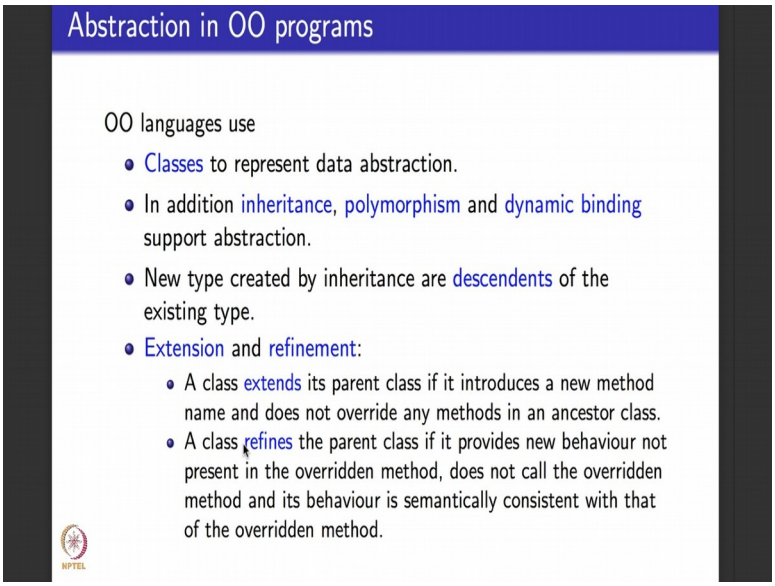
(Refer Slide Time: 02:49)



So, now what I will do is I will introduce and recap some of the relevant object oriented features that we will need for this lecture in the next couple of lectures. So, what do object oriented software do? Once a central features that object oriented programming provides is that of abstraction. So, abstraction let us you abstract the kind of information that you want typically as classes and you focus on how the components are connected, that various abstract components are connected. We also discussed features like inheritance and polymorphism when I introduced them last time. They basically define new connections that these abstractions offer and these are typically not found in of non object oriented languages.

So, our module as we will see for object oriented programming please remember we will not focus on testing inside a method what we are instead going to focus on is integration testing. I will show you the four levels of object oriented testing and also show you which part that we are going to focus on.

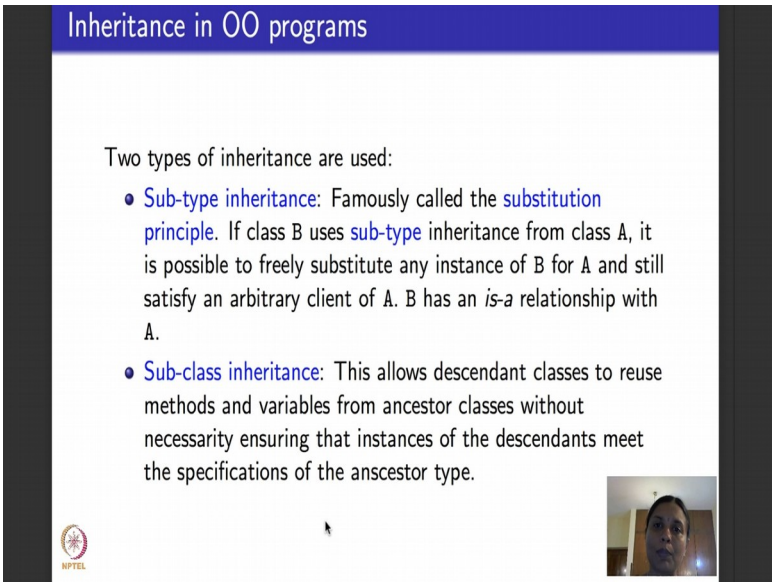
(Refer Slide Time: 03:54)



Before we move on what is abstraction and object oriented programs look like? The central entity in abstraction are that of classes, classes represent data abstraction. In addition to classes we have features like inheritance, polymorphism, dynamic binding, that also support abstractions, and how is a new type created? New type could be created by inheritance and as the word says, if it is a new type created by inheritance then whatever is created is called a descendant of the existing type of abstraction. Now there could be two kinds of new types that could be created. You could have extension or you could have refinement.

So, what is an extension? We say a particular class extends its parent class if it introduces a new method name and it does not override any method from its parent or ancestor. A particular class, on the other hand, refines a parent class if it provides new behavior that is not present in the overridden method and it also does not call the overridden method and in now along with that its behavior is semantically consistent with that of overridden method.

(Refer Slide Time: 05:07)

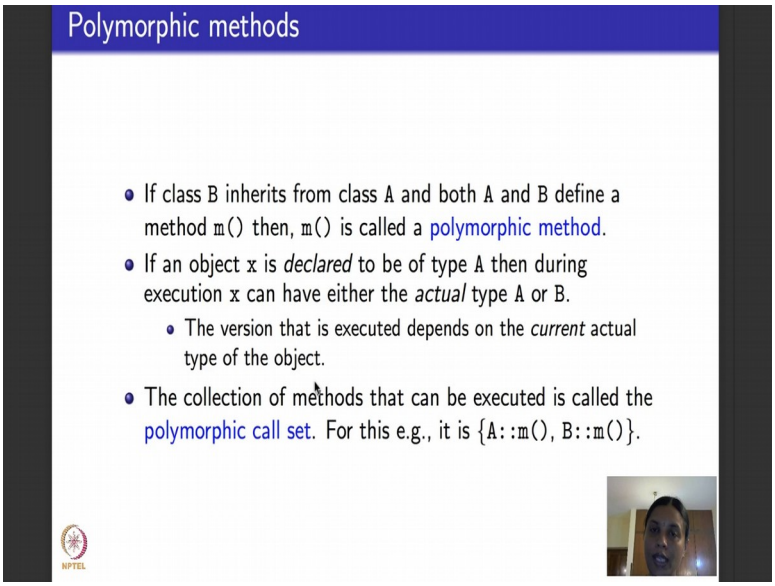


Moving on we considered two types of inheritance. Please remember all the terminologies that I am defining currently now is irrespective of the specific object oriented programming language that I use. These are generic object oriented features that. We will consider towards testing of object oriented software. So, there are two kinds of inheritance one is called the subtype inheritance, this is famously called substitution principle invented by this computer scientist called Barbara Liskov, who subsequently won the Turing award for this.

So, we say a particular class, let us say class B, uses subtype inheritance from class A, if it is possible to freely substitute any instance of B for a and still satisfy an arbitrary client for class A. So, we say B because it can be freely substituted for any A. So, we say B has what is called "is a relationship". What do we mean by that? That any instance of class B is also an instance of class A. So, B can be freely substituted for class A. The next kind of inheritance is what is called subclass inheritance, where descendant classes reuse methods and variables from ancestor classes without necessarily ensuring that the instance of the descendant classes meet the specifications of the ancestor class.

So, two kinds of inheritance: subtype and subclass. Subtype means B is also class A, can be freely used like that, subclass means descendant classes reuse methods and variables, but they do not have to meet all the specifications of a parent class.

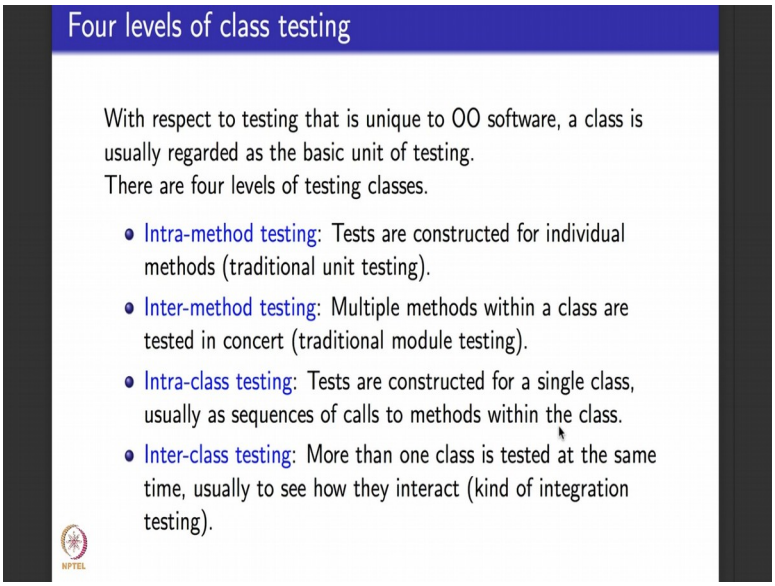
(Refer Slide Time: 06:47)



Now, just to recap what polymorphic methods are, because we would need this extensively today. We have already done this in one of the lectures, but I will recap it once again. So, consider two classes A and B and let us say class B inherits from class A and let us say both the classes A and B define a particular method m. So, m is defined in A and m is also defined in B and in addition B inherits from A. Such a method m in object oriented programming is called polymorphic method.

So, what happens now with polymorphic methods? Suppose there is an object X that is declared to be of type A then during execution, because B inherits from A the actual type can be A or B. Whichever the type that takes is completely dependent on the execution that happens right. This collection of methods which are polymorphic that can be executed when a particular object of a particular type is called is called a polymorphic call set. For example, in this case where there are two classes A and B and both define a method m, the polymorphic call set is A:: m (), as it occurs in class A and m as it occurs in class B.

(Refer Slide Time: 08:03)



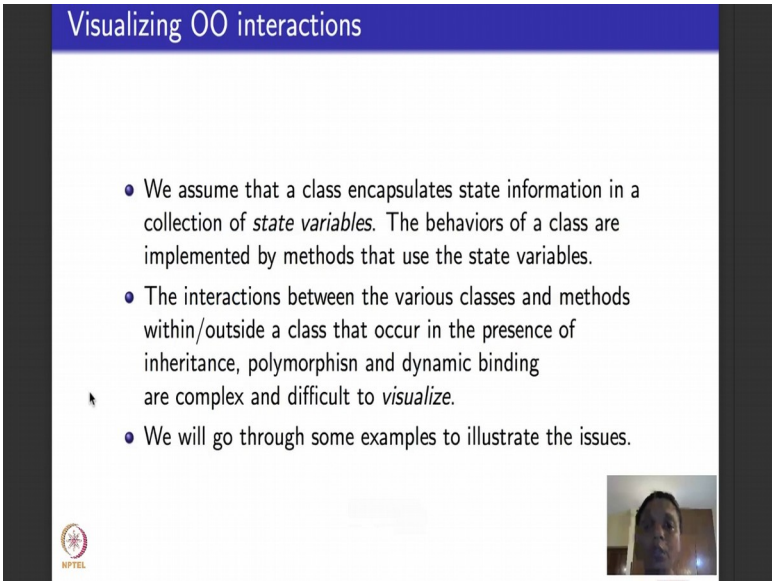
So, when while testing object oriented software we consider four levels of testing. We consider class as the basic unit of testing not a method as I told you when we consider a method as a basic unit of testing you could use any of the earlier coverage criteria that we learnt graphs, logic, mutation testing to be able to do method level testing. For us now focuses on a class level testing. So, with reference to class level testing for object oriented software, they could be four categories of testing. The first most elementary category involves intra method testing; that means, testing a particular method.

As I just told you use any applicable condition that we have learnt till now to test whether the method behaves as per its functionality, this is traditional unit testing. Moving up one level up from a method the next level of testing is inter method testing where interactions between methods are tested. This is traditional module level integration testing we have checked this when we did design integration testing based on graphs. In fact, we saw specific object oriented integration operators even for mutation.

The next is intra class testing that is testing within a class, what happens here? Tests are conducted for a single class that fixed to be tested at a particular point in time usually the tests are a sequence of calls to methods within a class. So, this class will define several methods a test case for the whole class will involve all the calling each of the methods, a select set of methods and so on. Finally, the full blown object oriented testing feature is a inter class testing where more than one class is put together and tested at the same time

usually to see how they interact. We will be considering inter class and intra class testing in these lectures.

(Refer Slide Time: 09:58)



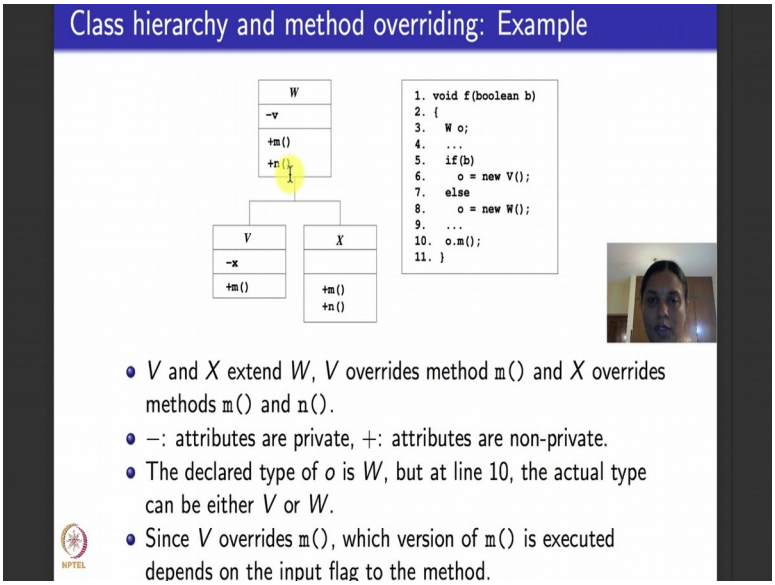
So, now before we move on, actual testing we will see a couple of lectures later. What I am going to tell you for the rest of this lecture and in the next lecture is what are the problems what are the difficulties, what are the anomalies, what are the faults that occur due to the specific object oriented features of inheritance, presence of polymorphic methods and dynamic binding. What happens in these cases? What are the issues that can come when we do inter class and intra class testing.

So, now before we move on, the most difficult thing is when you have a large many classes each defining its set of methods it is very difficult to visualize the actual sequence calls, the actual interactions that happen amongst these classes. I will show you examples of how difficult it is. So, we want to be able to do that. So, to do that we assume that a class encapsulates all the state information that I need. This is a traditional way, there is nothing related to object oriented software in any piece of program, what is the state of a program? State of a program defines the values of all its variables along with the location counter where the program decides.

So, in object oriented programming we assume that the class as an entity, encapsulates all the state information as a collection of state variables as it is done in other programming language. So, now, the behaviors of a class; how do they come? Class is a

static entity not executable. So, the behavior of a class is implemented by a set of methods that use these state variables. The interactions between various classes and methods, methods could be inside the same class or it could be outside a class will occur in the presence of all these features, inheritance, polymorphism, dynamic binding and those are the features we want to understand and visualize how the interactions happen.

(Refer Slide Time: 11:49)

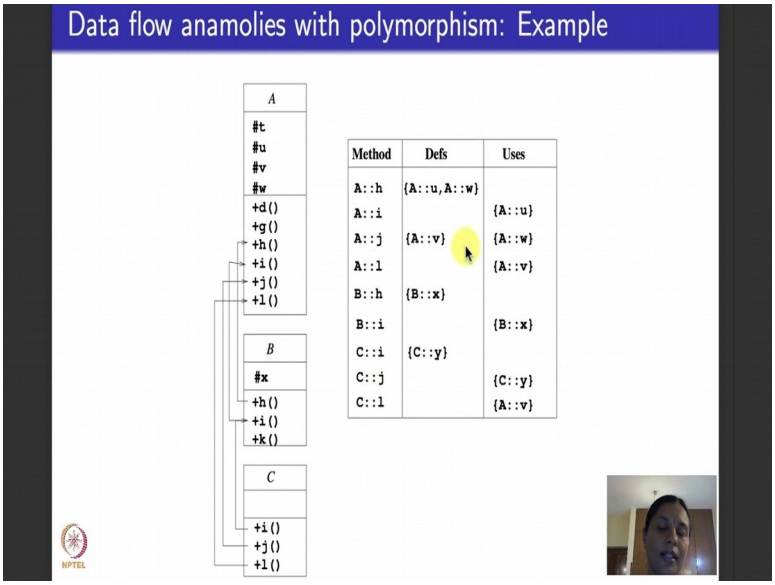


So, here is the first example consider this situation look at the left hand side of this figure. There are three classes W V and X and as per this picture V extends W, X also extends W. W has a private variable called small v and it has two methods m and n, V also has a method m which means this method m in V overrides the method m in W. Similarly the method m; methods m and n in X override the methods m and n in W. Minus sign indicates as I told you earlier the attributes are privates and plus sign indicates of the attributes are non private.

Now let us look at this piece of code on the right hand side. It is talking about some function f that takes a Boolean argument V and it begins by saying that the declared type of this object o is W and then there some piece of code dot dot dot read it as some piece of code and then let us say there is a new statement which says if this boolean variable b is true then you make o as what type, V. If this Boolean variable is false then you retain o to be of type W and later after some point let us say you call the method m for the object o. Now we know that the method m and V overrides the method m in W.

So, at this point in line number 10, we do not know which version of the method is called. Is Vs version of m called or its Ws version of m called that is not clear. It will purely be dependent on what the value of B is, this part here described by dot dot dot did it change B what happened to B. So, we need to know. So, usually this kind of analysis provides challenges because m overrides, the m in V overrides the m and W at this point in time we do not know which code of m got executed. So, this is the first kind of difficulty that we find in visualizing.

(Refer Slide Time: 13:51)

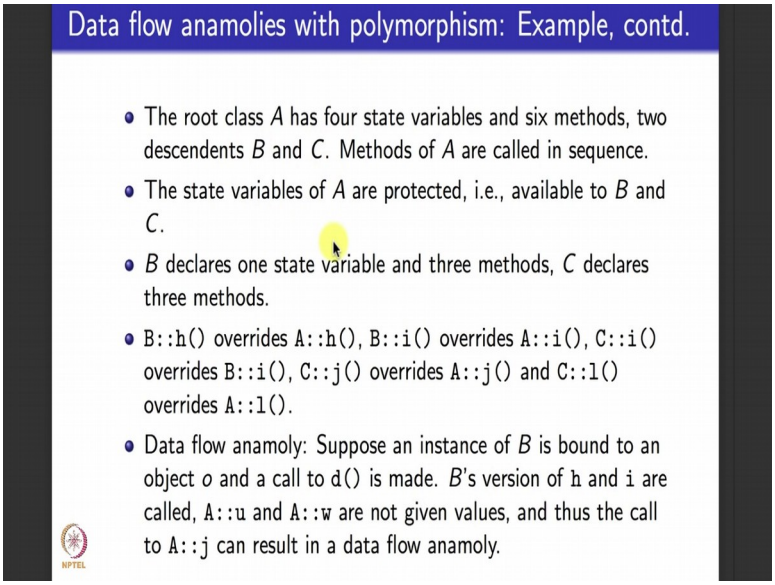


Moving on here is another kind of difficulty that you find with polymorphic methods this is a slightly bigger piece of program. What does it have? It has three classes A B and C, I have put them one below the other. A has how many variables four variables t, u, v and w, and it also has six methods d, g, h, I, j and l. Just for simplicity sake we have retained them a single letters they could be any method names. And then class B has one variable x. It has two three methods h, i and k. And what do these arrows indicate? These arrows indicate that the method h in B overrides the method h in A. Similarly the method i in B overrides the method i in A.

Now there is one more class C which has three methods I, j and l. And again there are arrows from this i pointing to the i of B, going by the same interpretation read this presence of this arrow as the method i in C overrides the method i in B, the method j in C overrides the method j in A, the arrow goes all the way up to class A and the method l in

C if you go trace the arrow and go up all the way, this method, overrides the method l in A. Is this clear that is what I have written here in the next slide.

(Refer Slide Time: 15:19)



The root class A has four state variables six methods we saw that right, four state variables six methods. It also has two descendant classes B and C as I told you. Methods of A are called in sequence which means what you assume that in the code of there is some piece of code in a where method d calls method g, which in turn calls method h, which in turn calls method I, which in turn calls method j and j in turn calls method a that is what I have written here.

Now the state variables away as you can see in the notation are protected which means what, they are available to the descendant classes B and C. B declares one state variable three methods, C declares three methods. And we discussed this overriding which method of B overrides which methods A, which methods of C overrides which method of A and B.

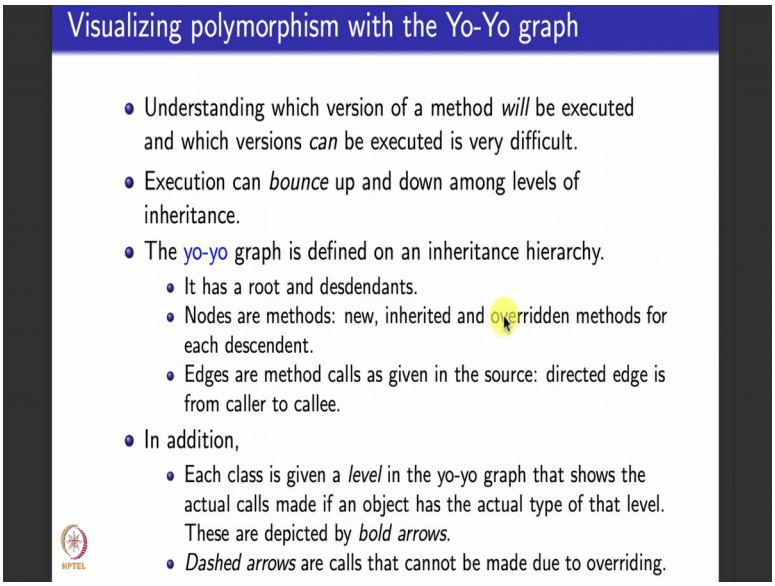
Now, consider a situation like this. What does this say? This says how to read this table this says in the method h as present in class A, two variables are defined u and w. And the variable u that is defined in the method h for A is used by the method i for A. Similarly the variable v is again defined in the method j present in A and method j also uses the variable w. The method a variable v defined in j gets used by the method i in A. Now B also defines a variable x in its method h B uses the variable x in its method i; C

similarly defines variable y in its method i and uses variables y and v in its method g and f. Is this clear?

Now I claim that because there are these polymorphic methods i and k and h and the j and l there is a problem with dataf low. How do you understand the problem with data flow? Here is the anomaly, suppose an instance of B is bound to an object o and let us say a call to d is made, a call to d is made right. Now B's d, what will d do - d will call g g will call h, but which version of h is called, B's version of h is called right. B's version of h is called which means what - B's version of h and B's version of i is called. If B's version of h and i is called then you look at it here. B's version of h, what is the variable that it defines it defines the variable x, B's version of i defines nothing. But then what do we want we finally, want A j because a in sequence I told you right g calls g sorry d calls g which in turn calls h and i, but h and i are the B's versions that are called and I further calls, what does j need ? j needs A w because it is going to use A w, but as version of I and h were not called only B's version of h and i were called and w was not defined B's version of h defines only x, B's version of I defines nothing.

So, that is what is said here B's version of h and I are have called A u and A w are not given values because only as version of h gives them values which means what the call to a j can result in a dataflow anomaly. It does not present at all. So, how will it use W there will be a problem. So, these are the kind of faults that we want to recover, identify and test for when we do object oriented programing.

(Refer Slide Time: 19:15)



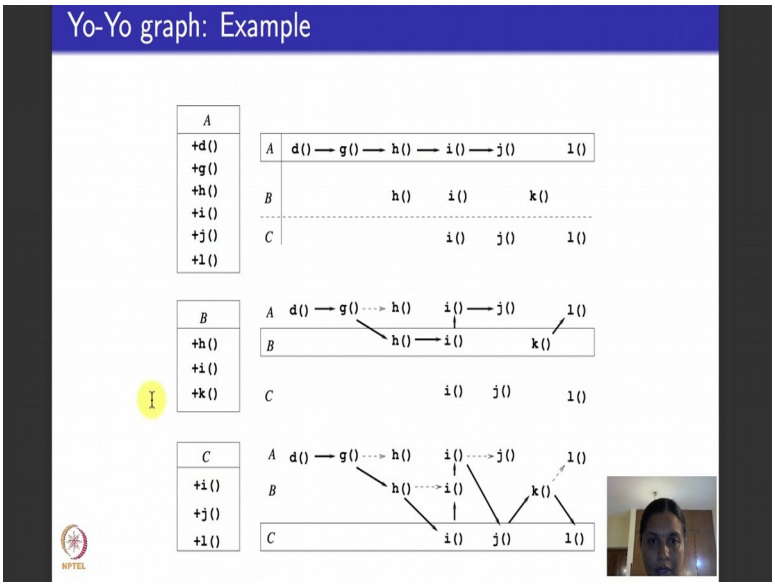
Now one last example before we move on where I introduced Yo-Yo graph formally to you. So, Yo-Yo graph deals with a problem of which is the version of the method that will be called. It is a very difficult problem to understand because the methods that can be called can bounce up the calls of the methods can bounce up and down like in a Yo-Yo that is what we will illustrate. So, what is a Yo-Yo graph? You all might be familiar with a Yo-Yo toy right this is string and then this top like thing where you can rotate the string and when you release the rotating entity from the string, it bounces up and down up and down right.

So, what we trying to say is that the method call graphs across classes can bounce up and down like a Yo-Yo. We will illustrate it for the example that we saw in this slide. So, before that what is a Yo-Yo graph. Yo-Yo graph is more like a tree it has a designated vertex called a root and then it has some descendants. What are the vertices of the graph? Vertices of the graph are not statements or anything like that, they are individual methods because we are focusing on method calls we focusing on call graph that deal with methods. So, nodes are methods, methods could be new methods, inherited ones or overridden methods for each of the descendant.

What are the edges? As I told you edges are the method calls that are given in the source code. A directed edge is present from the caller to the callee. In addition we have two other things there is a level that is given to each class in the Yo-Yo graph that shows the

actual calls made for an object if an object has a particular type of that level these are depicted by bold arrows. The Yo-Yo graph also has dashed arrows which are lighter which talk about the calls that cannot be made because the methods have been overridden. So, for the example that is given here, if you remember this method, class A six methods class B three methods class C three methods here is how the graph looks like.

(Refer Slide Time: 21:18)



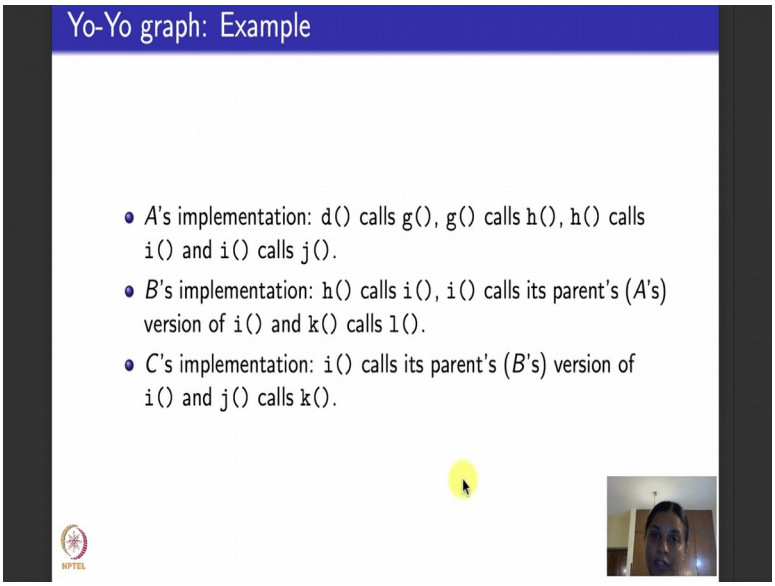
So, I have just done the left hand side here that I am tracing with the cursor, I have just copied the same thing, but I have omitted the arrow is corresponding to overriding just not to clutter the figure too much. What have I depicted here? Again it is the same information: classes A, B and C are given here this is a list of methods that are present in the class A, this is list of methods that are present for class B and here is the list of methods that are present for class C. As I told you methods and class A call each other in sequence from top to bottom, which means d calls g, g calls h, h in turn calls i, i calls j. There is no problem everything works fine because these are all methods within the class A and the calls are absolutely fine.

But let us say you get this kind of visualization, what happens here? Method d is called in A, d calls g, g calls h, but which version of h B's version of h and B's version of h in turn calls B's version of i which calls A's version of i and then calls j and independently k to l call is present. Now if you see there is a d-link because I do not know what to do.

now this makes it even worse. So, say we start similarly we say d cause g, g calls h and h does not even call B's version it called C's version of I and C again calls B's version of i which in turn calls as version of i and As version of i now goes all the way down and calls j's C which calls B's k which calls l C and so on.

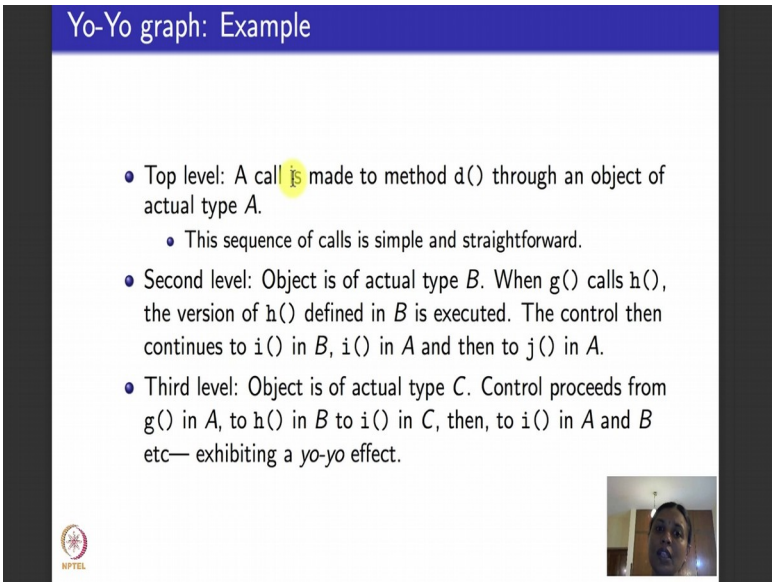
So, you can see this you can think of as a Yo-Yo unrolling itself, this is a Yo-Yo bouncing back, this is a Yo-Yo bouncing back and unrolling itself. So, the pattern of calls can grow up and down like a Yo-Yo. The dashed lines as I told you here are calls that cannot be made due to overriding, but if they had been made it could have been anomaly free that is what we are trying to say here.

(Refer Slide Time: 23:25)



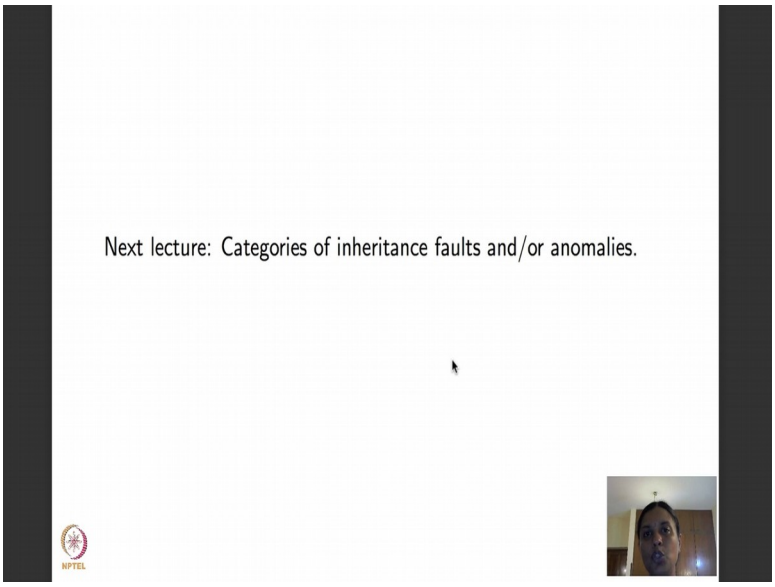
So, what I explained to you is what is written here. In A's implementation there is the sequence of method calls, in B's implementation there is the sequence of method calls h calls i and i calls its parent version of i as I told you here right I calls its parent version, parent is A so it calls its parent version of i and k calls l. In C's implementation I calls this I of C calls its parent version of i which is B's i which in turn calls A's i and then the call to j is made. So, the calls of methods go flat, bounce up, bounce down like a Yo-Yo hence the term Yo-Yo graph.

(Refer Slide Time: 24:06)



Now, if the top level call is made then there is absolutely no problem. For second and third level you could have data flow anomalies as we saw in this example. So, this is another kind of visualization that we will like to work on.

(Refer Slide Time: 24:17)



I will stop here for this lecture. In the next lecture I will discuss with you about specific faults and anomalies that can occur due to inheritance and polymorphism.

Thank you.