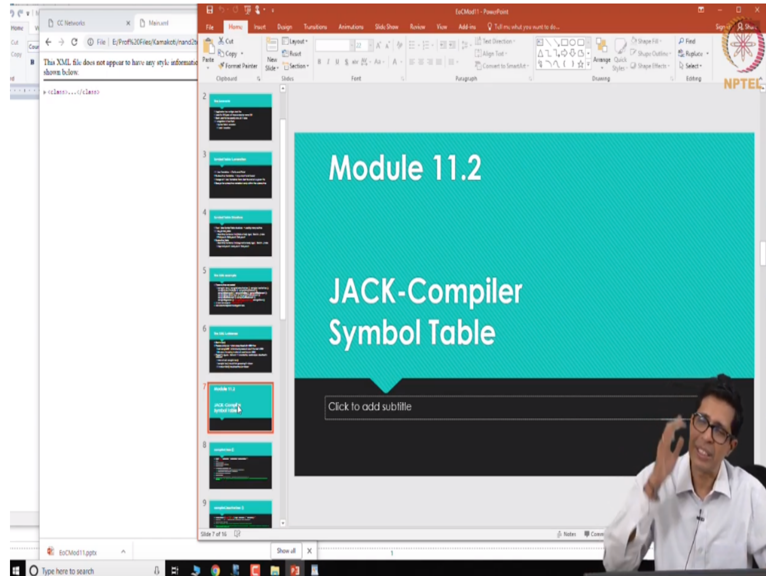
Foundations To Computer Systems design
Prof. V. Kamakoti
Department of Computer and Engineering
Indian Institute of Technology, Madras
Module11.2
The Jack Compiler Backend: Populating the Class and Subroutine Symbol tables

(Refer Slide Time: 00:16)



Welcome to module 11.2 in this particular module we will now go ahead and explain how the symbol table is getting populated right so there are two symbol tables as I had mentioned there is a global simple there is a class symbol table which is global to all the subroutine and every time a subroutine is compiled we need to have a subroutine come symbol table since the scope of that symbol table is restricted to the subroutine.

Once you finish that subroutine and go to the next subroutine the whole all the entries can be erased and we can use the same structure for the next subroutine where we populated it again so the class symbol table is populated once for the entire file that subroutine symbol table is populated every time you start fresh compiling one a subroutine again and again.

(Refer Slide Time: 01:08)

compileClass()

- Class: 'class' className '{' classVarDec* subroutineDec* '}'
- Begin
- Check for keyword class
- Check for identifier className
- Check for symbol {
- If it encounters <classVarDec> then
  - call compileClassVarDec(); on return you see </classVarDec>
  - Repeat above if again you see <classVarDec>
- Similar treatment for subroutineDec
- Check for symbol }
- Should initialize class_symbol_table : static_count, field_count, total_count = 0
- Initialize a global string currentClassName to the className read above. This is needed during compilation of m functions.

So this is how we will so we will now see how the symbol tables are populated where do we fit in the code so we have 13 routines the first two routine will be main we have 15 routines the main will call compiled class, compiled class will call other routine so each of these routine will contribute towards building the entire execute file the VM file and what will happen in each of these subroutine.

What is their contribution that is what we will be presenting the first thing is that we need to construct the symbol table we will see which routines are going to contribute for the symbol table now let us go into this now class, so the class basically has, so if you just see a class has class name this is a rule for class let us go back to this.

Rule for classes that the class has the keyword class then a class name then this symbol then class where next our subroutine deckstar then this closing symbol so what will compile class do it will first go and check whether keyword class is there yes it is if it is that then it will check for what is the class name right then it will check for the symbol this then it will go and see if it is encountering class vardec because class where next star means class vardec is optional so I made have class variable I may not even have class variable my class may not have variable.

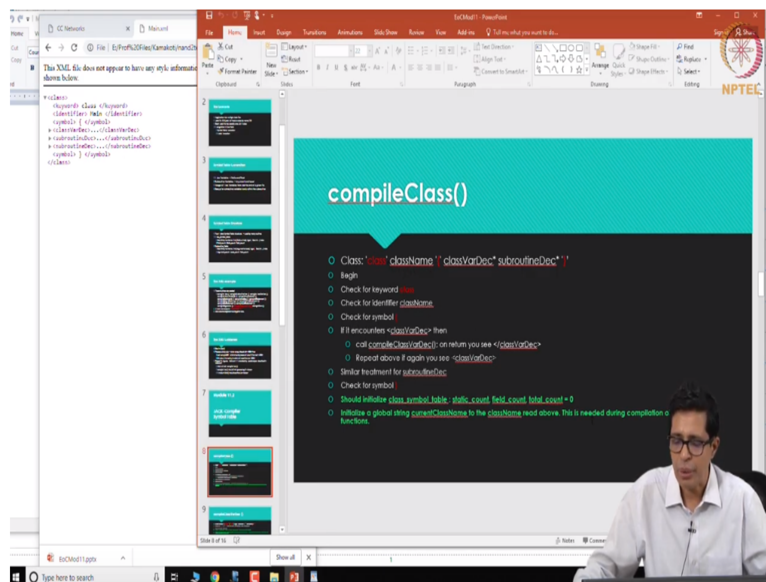
So this class where next star is optional so there can be zero or more class where declaration, so if I encounter class vardec then what we will do what is the rule I encounter XXX immediately I should call compile XXX, so if I encounter classifier Dec immediately I will call compile class

vardec and when this return I should see slash class vardec, now I will repeat the same two steps again and again till as long as I am going to see class vardec in the first one right.

So when I see classify a Dec I will call compiled class vardec then when control comes back I will see whether I am having slash class perfect again I will go and check if classifier Dec is that if it is there again I will do again I will do and I will repeat it once this is over then I have subroutine decks again that is star subroutine Dec star means I could have multiple subroutine declarations zero or more I may have a class where there are no subroutine also right.

So again the similar treatment is done first subroutine deck, so if I see less than subroutine Dec greater than that symbol then I will call compiled class a compiled subroutine Dec and on return I will see whether slash subroutine is that and I will keep on doing it at the end of this I will go and check for symbol the closing braces, so this is how compiled class will work.

(Refer Slide Time: 04:20)



So if you just go back to this routine so first AC class so main C's class so compiled class comes as I told you first it will check yes classes there then main is that this is the class name then the opening simple is there then there is one class vardec please note that you are seeing class vardec so immediately compiled class vardec will be called on return it will check for slash class vardec then you see subroutine Dec repeating two times a subroutine take then subroutine this will again you will call compiled subroutine Dec which will process the remaining taken here and on return I will see slash subroutine Dec similarly again I will call subroutine Dec.

And again I will get back to this then finally I will go and check if this symbol closing parenthesis fail no I will return back control to main the main will now see this slash class there, so this is the overall structure so all the tokens between say for example class vardec to slash vardec deck will be consumed or compiled by compile class well Dec all the tokens between subroutine Dec to slash subroutine Dec will be consumed by compiled subroutine Dec similarly this also will be my compulsive routine Dec, so this is how the whole thing works right now what will classes do at the beginning here, so actually.

So this is about checking the grammar but what will class do in addiction, so the compiled class once it is doing all these checking intermediately whatever I have put in green and henceforth in this entire module whatever I am putting in green is what is the contribution of the routine to the ultimate virtual machine code that I am generating so what will class name so this will initialize the class symbol table.

So what is the class symbol table where symbol table as we associated integers one is the static count we shall tell you how many static variable are there filled account which will tell you how many field variable are there and total count which will tell you how many total variable total of static plus filled, so this all these things are initialized to zero that means your class symbol table does not have any entry right zero variable are that now it will also initialize a global string called current class name and it assign the class name that you are seeing here.

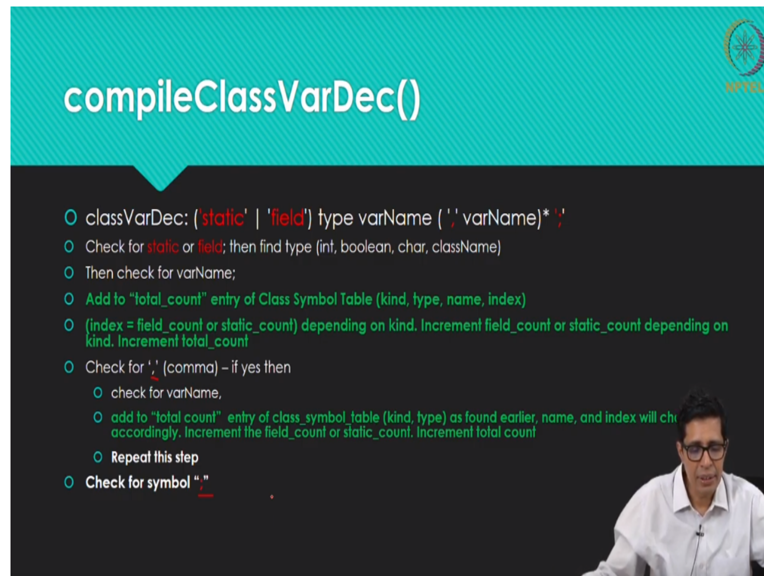
So you are seeing the story that the class name you are seeing here and I am just moving the mouse there the class name that I am seeing here that will be, so the class name that I am seeing here that will be assigned to this current class name because we may need we will need this for some other purpose which I will tell you later.

So two things the complied class has to do in addition to doing all those checks and calling compiled class where they can compile subroutine calling them specifically for the code generation purpose it has to initialize the class symbol table what do you mean by initializing it the static count and the total count should be set to zero then it has read the class name that it assigns to global variable called current class name.

It assigns the value class name to that current class name right it assigns the string class name to the current class name substring, so these two are only thing that the class does with respect to

generation of the code other than that it only checks the grammar and calls the corresponding, subroutines namely a compiled class vardec and compiled subroutine Dec.

(Refer Slide Time: 08:21)



Now let us go to compiled class vardec after this after your class finishes these things the first thing it will call is compiled class vardec, now what will saw the class vardec rule is that I should see a static or filed then there should be a type name type then a variable name followed by multiple variable names, so there will be one variable for sure but there can be multiple, so for example I could have static int I comma J comma K right.

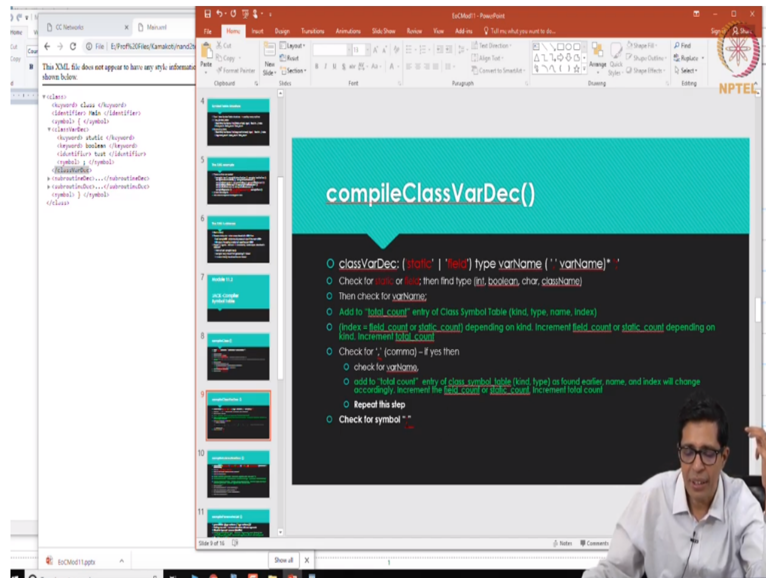
So immediately what you do is we check for static or field first thing then find the type so it should be int, Boolean, char, class name then you find the where name right now add to total count entry of class symbol table, so initially total count will be zero that means add to zero entry in the class symbol table, so class symbol table is array of structure each entry will store the kind type and name and index of the variable in this case the kind will be static or field.

Whatever you have seen here than your type will be whatever type you have seen here in Boolean care or some other class name the name of the variable of course and the index will be if you are if your static variable index will be static count if it is a field variable it will be field to come right, so you make this entry the moment I see static type where name something so immediately you do this entry so index is either filled counter static on depending on the kind increment field counter static on depending on which our kind and increment total counter right.

So these two are the things that you do here so that means an entry is created in this class symbol table for that variable static, so if I have static int I then I will be entered as a static variable inside okay, so if I have a static int I comma J then the comma will be checked there as you see here and then again you check for where name, so add to total count entry of class symbol table, so total count is incremented.

So the next entry will come there you add the kind and type that you have seen earlier itself and the name and index will change accordingly because now you have the name and the index will be if it is a field to counter static on that will be an then you increment the field counter static count and also increment total count, so and keep repeating step and ultimately then you have to see a semicolon here so this is how class vardec will work and what is the role of class vardec it will populate the class symbol table completely.

(Refer Slide Time: 11:35)

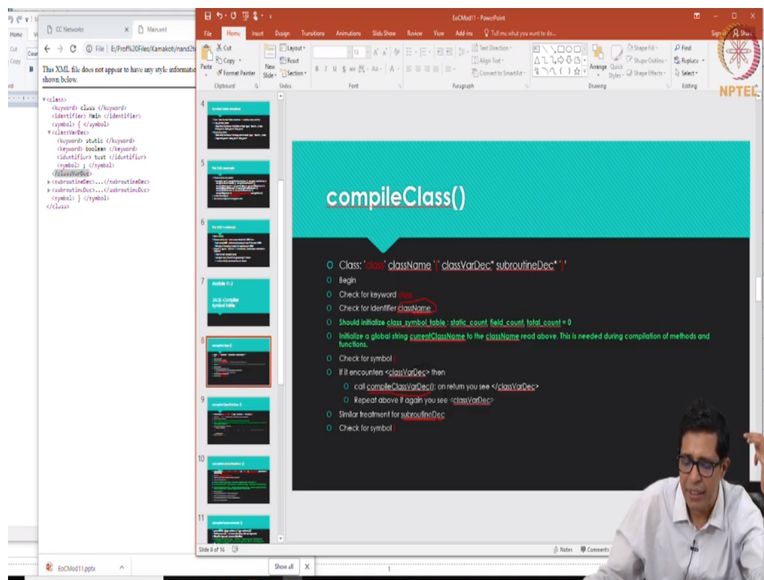


Now let us see a simple-simple thing here, so ultimately when your class compels it finish a class main this thing then it will come to class for a Dec now you see what is going to happen in class for that it will first see it is static, static or field it is a static then type is Boolean name is test, so and then semicolon immediately this will be entered in the table as kind static type Boolean name test index zero and the static count will become one like that for your field variable also this can be right.

So this is and then at the end of this it comes out after checking for the symbol semicolon then the calling main root routine we will see whether class vardec ended right slash class verdict, so this four tokens are consumed by compiled class verdict and when the return comes back to compiled class which is the bounding class we have compiled class it will check for slash class verdict, so this is the basic way by which all the class variable.

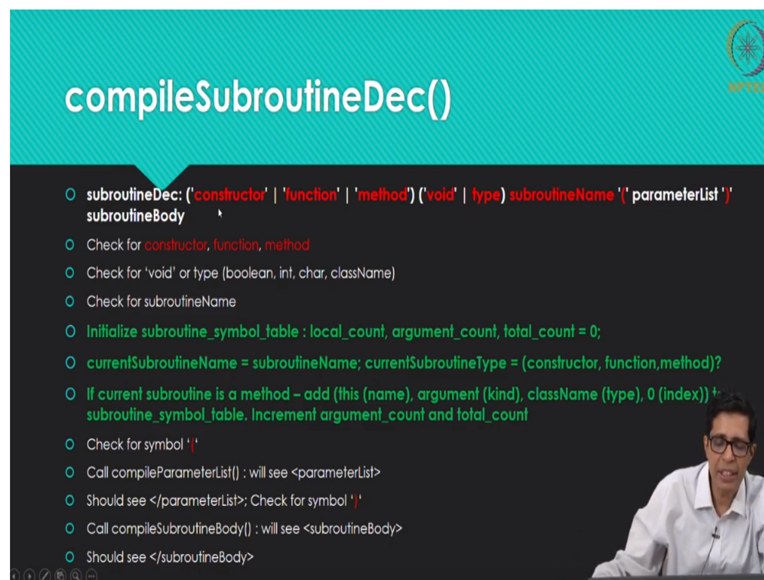
So if I put done as in if this class vardec can be called multiple times and each time it is called one more entry gets added and that is why we had the class symbol table as a global structure so that multiple times your class vardec star cells it will keep on updating that global structure, so to sum up the compiled class vardec is actually responsible for populating the class symbol table and this is exactly that code that you need to write which will go and populate your class symbol table right.

(Refer Slide Time: 13:30)



Now after this if you go over to compile class after your compiled class vardec this will now start calling called subroutine Dec, so now the next thing that we want to see here after class verdict is the subroutine.

(Refer Slide Time: 13:46)



So the subroutine Dec rule if you see first it will check for constructor function or method then it will see if it is wide or some type can be Boolean care integer class name then it will have a subroutine name then followed by the starting parentheses then parameter list then the right parenthesis and this is the followed by subroutine body, so this is the rule so what will happen first it will go and check for constructor function method then it will check for wide or type can be Boolean in then it will check for subroutine name.

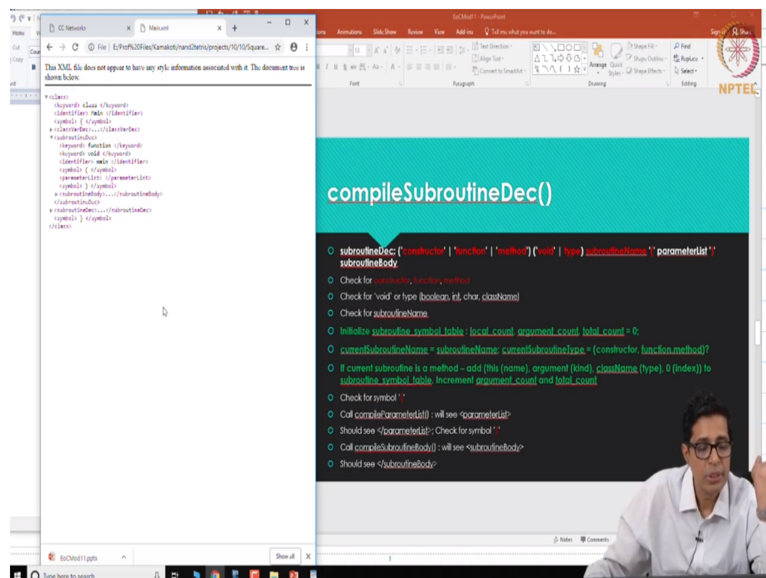
Now once I check for subroutine name at this point these are the things that we do initialize your subroutine symbol table that means your local count argument count and total count which is local plus argument should be set to zero that means the subroutine table has become empty, so every subroutine you restart this table because every subroutine the scope of the variable are restricted to that subroutine.

So every subroutine I start I will go and restart this symbol table now you also name the current subroutine name as subroutine name that you have seen here and current subroutine type as constructor function or method, if current subroutine is a method add this argument class name zero, right if I am having a method as we explained last time if I am executing a method then before that method is called the argument zero for this method would be the class to which this the object to which this method belongs to right.

So the zeroth argument will always be the object to which this method belongs to this method is assigned to because when I am when I am executing the method I will land up with class variables and field variable and for me to access that I need an access to the object to which I am this particular variable is associated, so this will be the first argument, so this routine which holds the pointer to the current object will be the first argument, so this argument zero class name argument that is a zeroth argument and the class name which will be the type.

So that is something that we put on the current subroutine table and we increment argument count and total count, so these three things are done by the subroutine declaration then it goes and check for symbol after checking for symbol it has to see parameter list it will see parameter list so it will call compiled parameter list after compiled parameter list finishes and the control comes back to this routine it will see slash parameter list after that it will check for end of parentheses then it should see sub routine body, so it will do compel subroutine body and after subroutine body finishes it should see slash subroutine body so this is the overall structure.

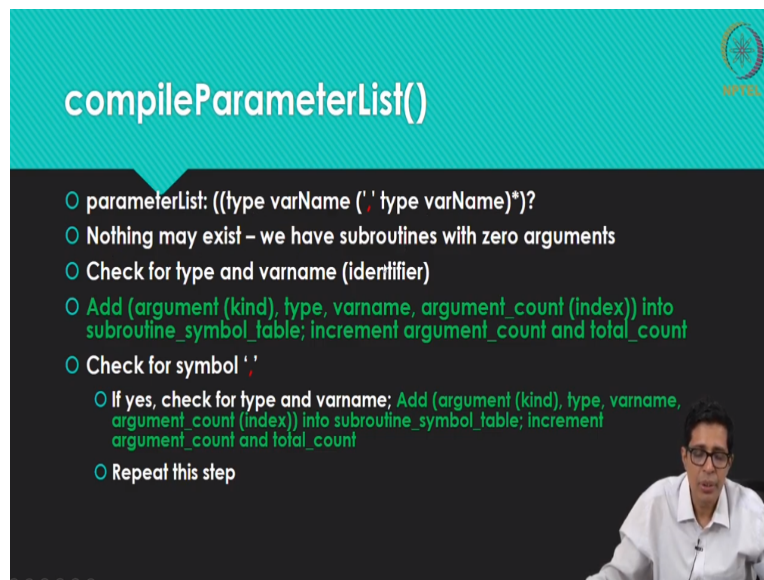
(Refer Slide Time: 17:31)



So let us just go back to this yes after class vardec finished there is a subroutine Dec right, so the subroutine Dec will first check it is function yes and type is wide and name is means and there is none in the parameter list we will just see what it is and then subroutine body right so the subroutine body goes till this and then I come back to subroutine Dec and then there is another subroutine Dec here call subroutine Dec that I finish and then come back and go to class.

So this is so these tokens are processes this is how the XML actually gets the token, so coming back here, so these three are the actions that compiled subroutine Dec has to take with respect to code generation right, now so what will compel subroutine Dec do it will call compile parameter list and then it will call compiled subroutine body.

(Refer Slide Time: 18:54)



So now we will go and see compiled parameter list what is perhaps compile parameter list it will have tape where name comma type where names right so this is the parameter list, so right so whenever I say int I comma into J comma bull K, so I is integer J is integer K is a bull right, so the parameter list is of this, so please see that there is a question mark here the question mark essentially says that I need not have anything in the parameter list.

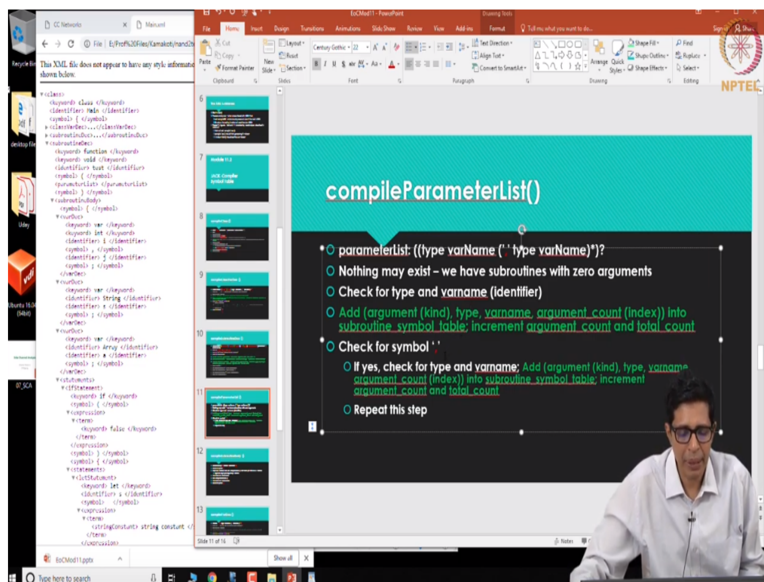
(Refer Slide Time: 19:30)

For example as you see here if you take this particular subroutine declaration right, so if you see here this is of.

(Refer Slide Time: 19:42)

If you see here main has no parameters the small main had no parameter, so the same thing is valid here.

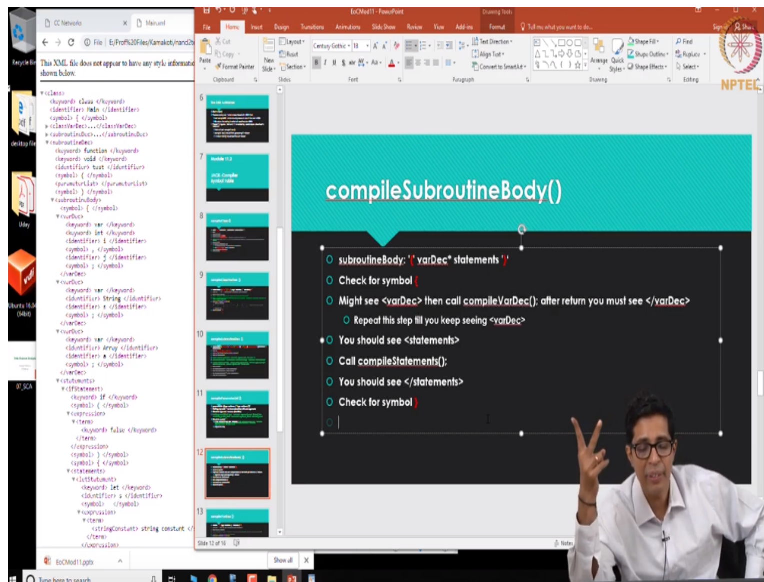
(Refer Slide Time: 19:54)



Same thing is valid here so parameter list and slash parameter if you just look at this there is no parameter at all just parts with parameter list right so what will happen is the first time I see I will see type bar name so now you can see here, so I will have type there name so the type the kind of this since I am in the parameter list the kind is argument the type is whatever type you are said here that will be the type then the Var name then the index is the argument count.

So then this will be into the subroutine symbol table then we will increment argument count and total count this will be repeat at as I will see comma here I will it will be repeated again and again till there is no comma and you end with the parenthesis so we will keep repeating this step, so what will the parameter list to it will populate the subroutine symbol table with all the argument parameters and this is where you should fit the code and this is where your argument parameters get added right and then then we, so this is all about parameter list once the parameter list finishes then you go and call compel subroutine body.

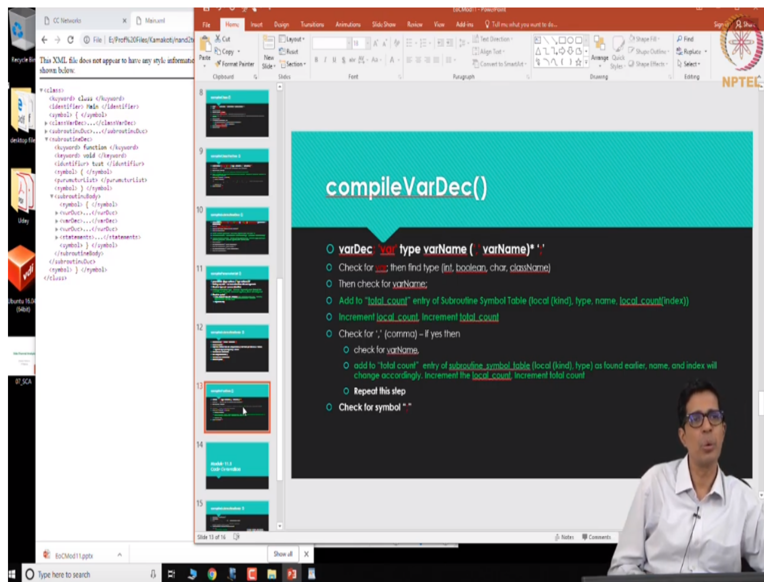
(Refer Slide Time: 21:42)



So what will compel subroutine body to subroutine body actually starts with the semicolon then there could be zero or many Var declaration then it will end up with it will have statements and then it will end up with this parenthesis close braces, so what will compile subroutine body do it will check for symbol this one and it will might see Var Dec it see Var Dec then it will call compile Var Dec after return you must see slash fabric.

So compelled subroutine will see Var Dec then it will call compiled Var Dec after compile Var Dec it should know it must see slash Var Dec and you keep repeating this till you keep on seeing Var Dec because Var Dec star means zero are many times repeating of the Var Dec, so this is how it goes, so I will be calling this compile Var Dec again and again zero or many times and after that finishes I should see statements here and immediately we should call compelled statement and then end you should see slash statements then you go and check for this symbol closing parentheses and that is where subroutine body ends right, so the subroutine body basically calls two routines compiled Var Dec and compile statements.

(Refer Slide Time: 23:28)



So as such you see here after I finish your subroutine declaration first I say function wide parameter lists over after that immediately we call subroutine body, the subroutine body will have multiple Var declaration, so I have one Var declaration and there Var declaration, so I have multiple Var declarations right followed by statements just what we see so your you should it will have so your subroutine body will have Var declaration, Var declaration of activation followed by statements like so this is the entire thing, so let us go and see what will where declaration due in the Var declaration again these are the variable inside the subroutine.

So there is are local variable, so the compiled Var Dec is responsible for populating the subroutine table with local variable, so it will see Var type Var name comma Var name till semicolon, so it will say where int I comma J comma K, so I J and K will become three entries in the symbol table in the subroutine symbol table there kind will be local that type will be int in this case and there index will go from zero one two and that values that count is actually maintained by your local count.

So what will compile Var Dec do it will check for Var it will then check for Var name then add to total count entry of subroutine symbol table local type name and local counter increment local count and increment total count okay right, so this is hot compiled Var Dec two so at the end of this compiled Var Dec as you seen we are now completely populated our class symbol table and the subroutine symbol table this is exactly where you need to fit in your code and populate this,

so in the next module we will know go into the next part of thing which we call as code generation, thank you.