FUNDAMENTALS OF OBJECT ORIENTED PROGRAMMING

Lecture14

Lecture 14: Inheritance - Multiple, Hierarchial, and Hybrid

Member Access in Inheritance

Base Class Access Level	Access Level in Derived Class		
	Public Inheritance	Private Inheritance	Protected Inheritance
Private	Not Inherited	Not Inherited	Not Inherited
Protected	Protected	Private	Protected
Public	Public	Private	Protected

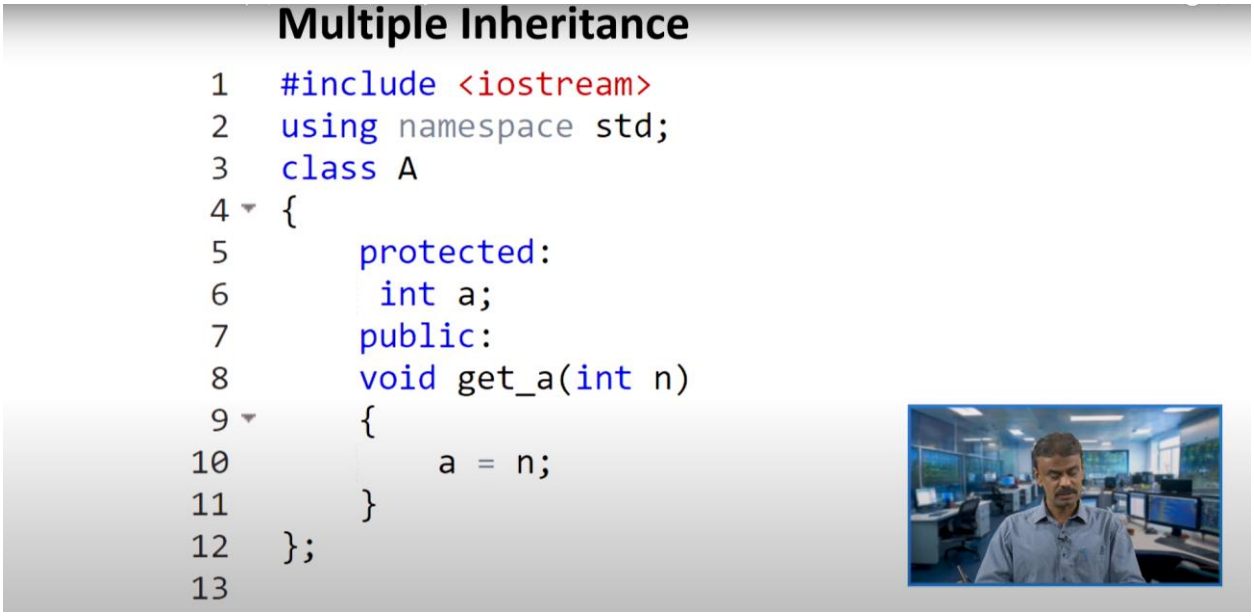
So, welcome to Lecture 14: Inheritance. In the last lecture, when we talked about inheritance, right? So, the visibility where class B may privately inherit class A, or in a protected way, it can inherit class A, or what will happen if class B is publicly inheriting class A. So, in that case, suppose the inheritance is public, all right. So, in the case of private, it will not be inherited. So, you can see the table where, in the private case, whether it is public, private, or protected, the access level in the derived class will not be inherited. Whereas, in the protected case, right, the public will become protected.

The private will remain private, and in the protected case, it will become protected. And in the case of public inheritance, right, if the base class access level is public. So, in the public level, right, in the case of access level in the derived class, the public inheritance

will remain public. In private inheritance, it will become private. And in the case of protected, it will become protected.

So, these are all the cases. So, we can inherit. So, in fact, we had seen the case of what will happen if it is publicly inherited or if it is privately inherited, which we had seen. So, in the case of protected, assume that the base class access level is protected, and the access level in the derived class is protected. In public inheritance, it will become protected.

And in private inheritance, it will become private. And in protected inheritance, it will become protected, right? So, we have already seen simple inheritance and multi-level inheritance. And what do you mean by multiple inheritance? Suppose you have class A, class B, and class C. Assume that class C is inheriting both class A and class B, right?



As you are seeing in this particular figure. Right. So, let us see what will happen. So, let us consider an example from C++, right. So, the multiple inheritance.

So, assume that using namespace std, right. So, because we are going to use cout. So, we use using namespace std. Let us assume that you have class A, right. Assume that you have class A. So, you have one protected member data a and you have a public member function get_a, right.

So, you are passing the parameter; that parameter will be copied into A. So, that is your class A. Now, your class B is defined, and you have one data member b. Similarly, get_b member function. So, n will be copied into b, alright. So, this is a member function in class B, and you have one protected member data b. So, now this is the syntax that you are seeing. When class C inherits both A and C publicly, right, both

```
14  class B
15  {
16      protected:
17      int b;
18      public:
19      void get_b(int n)
20      {
21          b = n;
22      }
23  };
```

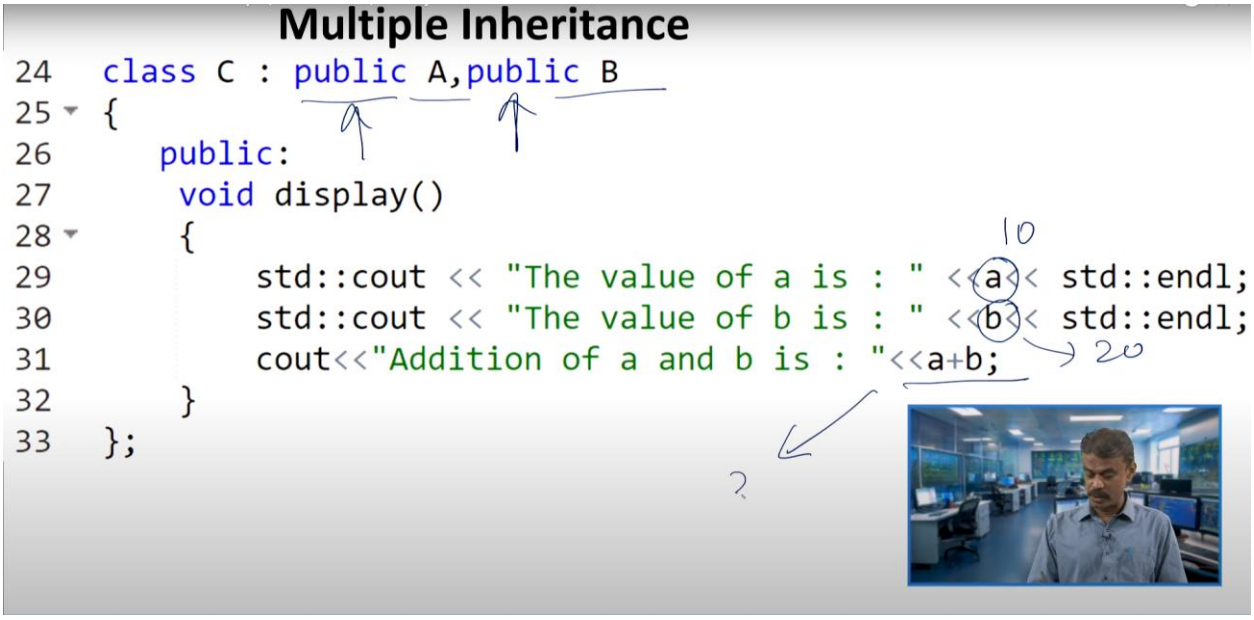
A and C publicly. What is the syntax that you are seeing? Alright. And then you are defining public void display. You have cout.

The value of a is a. Alright. So, it can access a because you are inheriting publicly. And you have a protected member data. So, the protected member data.

Will become. Alright. It can, in fact, be accessed by C. Alright. A and B can access. So.

The value of A, right? So, that will be printed. So, protected. Publicly, you are editing. So, A and B can be accessed.

That is what we had seen in the first slide. And next, cout is B. And another one is A plus B, right? So, this is what class C is doing. Class C has one member function called display. So now, in the main program, let us create a class.



Let us create an object c whose class is capital C. Now this object is invoking get underscore a by passing 10. So get underscore a when you are passing 10, what is happening? Get underscore a when you are passing 10. So n is 10.

So that will be copied into a. So a will become 10, right? And the same object c is invoking get underscore b by passing 20, right? So now in that case, what will happen? n is 20, so b will take the value 20. So what is the a value? 10, and b is taking the value 20, right.

So now the third function it is invoking the third function c dot display, all right. So when it is invoking display, a will be printed. So we know what is a? a is 10, right? And b is 20.

And what about a plus b? a plus b should be 10 plus 20, which is 30. So this is the output we have to get. So when you run the code, you have to get 10, 20, and the addition of 10 and 20, which we know is 30, right? So you can see the output over here, right?

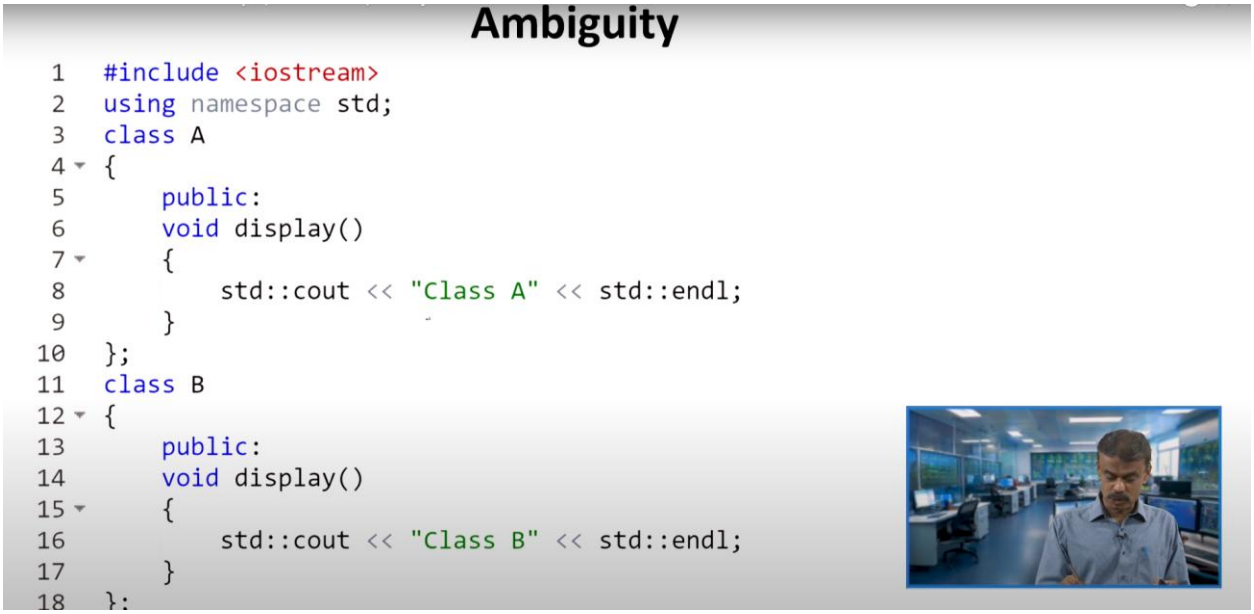
So you can see the output. So the main idea here is in line number 24. Class C Which is inheriting both A and B. That means it is using the properties of A and B publicly. Alright.

So in that case, since it is a protected member data and public, alright. So we know that. Alright.

So, we know that. So, the base class. You have a protected. And here, it is public. Protected will become protected again.

So, the member data will be protected again. Right. So, the base class. Access level. Alright.

So, you have the protected member data. And when you use public inheritance. The data will become protected again. So, that means the data int a, right, will become protected in class C. Similarly, int b will be protected in class C. And line number 24, class C is inheriting A and B, right. So, this is an example of multiple inheritance.



So there are cases. So you have ambiguity, right? Suppose in class A, you have one display. Assume that it is cout in class A. And in class B, again, you have display.

Right. So in class A, you have display. And class B is also having the display method. Right. Display function.

Member function. That is displaying class B. So now class C is publicly inheriting both A and B. Right. So now when class C is having a member function view. And then it is calling display. Then there will be a problem.

So, which display function will be called? Will it call A or B? Alright. So, that is why this particular program will give you an ambiguity. So, you have object c under capital C class C, and this object is invoking, let us say, view.

```
19  class C : public A, public B
20  {
21      void view()
22      {
23          display();
24      }
25  };
26  int main()
27  {
28      C c;
29      c.view();
30      return 0;
31  }
```

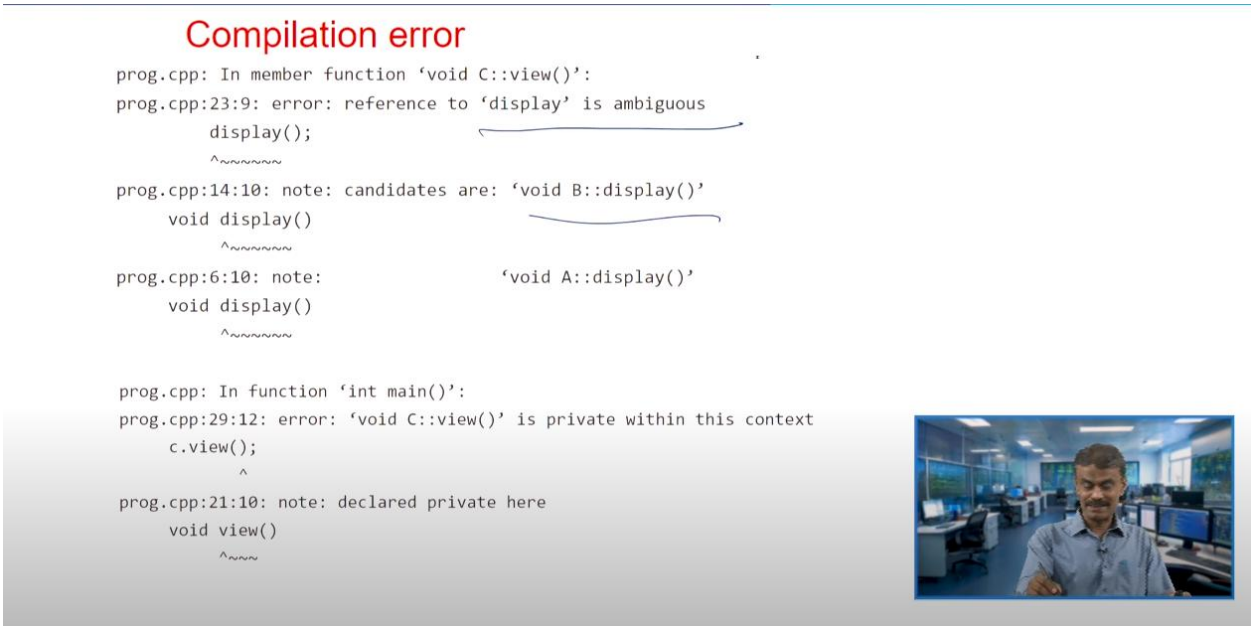
Put 'A' followed by two colons, right? So this is exactly what we are doing. If you look at it, we used to put 'std::cout'. Do you recall? All right.

So, explicitly, I am calling from 'std'. That means 'std' is an inbuilt class. I am doing it like this. So, 'cout'. All right.

So, this is what we have written. In fact, we have written it in several cases. Yes, here itself. Look at line number 8. All right.

So that means, in order to solve this ambiguity, what we are getting in line number 8, Right. So, which function has to be called, which member function has to be called. So this will be solved when you explicitly call the display of class A and explicitly call the display of class B. So that you have to do A double colon. Similarly, B double colon.

So that means, when you write A double colon display, your class A display will be called. So first, class A will be printed. Next, class B will be printed. So this is how, when you run the code, you have to get the output like this. So, class A



class B will be displayed.

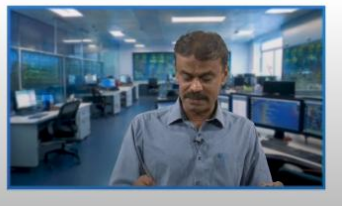
This is how we can resolve it. Now, from Java's point of view, with respect to class, we do not have the concept of multiple inheritance. We do not have the concept of multiple inheritance in Java with respect to class. So for this, We have to introduce the concept called interface in Java.

So, the interface will take care of this multiple inheritance problem, right. So, we will see that when the interfaces concept comes, okay. So, again, we will recall this multiple inheritance concept, right. When we talk about Java, we will see how we are going to take care of it. So, the multiple inheritance concept, you can't find it in Java.

How to resolve

```
19  class C : public A, public B {
20  public:
21      void view() {
22          A::display();  // Explicitly calling display() of class A
23          B::display();  // Explicitly calling display() of class B
24      }
25  };
26
27  int main() {
28      C c;
29      c.view();
30      return 0;
31  }
```

std::cout



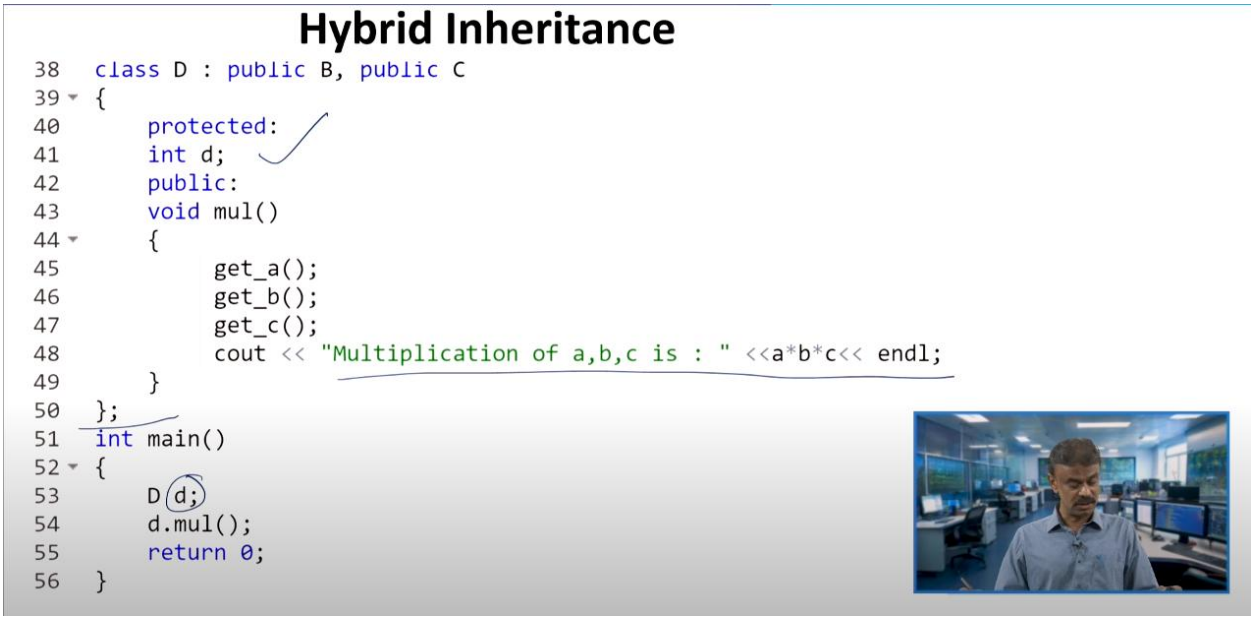
Hybrid Inheritance

```
1   #include <iostream>
2   using namespace std;
3   class A
4   {
5       protected:
6       int a;
7       public:
8       void get_a()
9       {
10          cout << "Enter the value of 'a' : " << endl;
11          cin>>a;
12      }
13  };
14
```

So, obviously, you do not have hybrid inheritance in Java either. Whereas, in C++, we have already seen that it is there. So, in the case of hybrid inheritance, let us see what will happen in the case of hybrid inheritance. So, assume that you have class A. So, class A is almost similar to what we have written, right? You have a protected member data called a, right?

And void get_underscore_a, right? So, you are taking the input from the user: cin a. You are printing: enter the value of a. Get_underscore_a is a member function. Int a get_underscore_a. Enter the value of a. You are taking the input from the user. So, that

is of class A. And class B, which is publicly inheriting class A. So, you have class A and class B, which is publicly inheriting class A. Right.



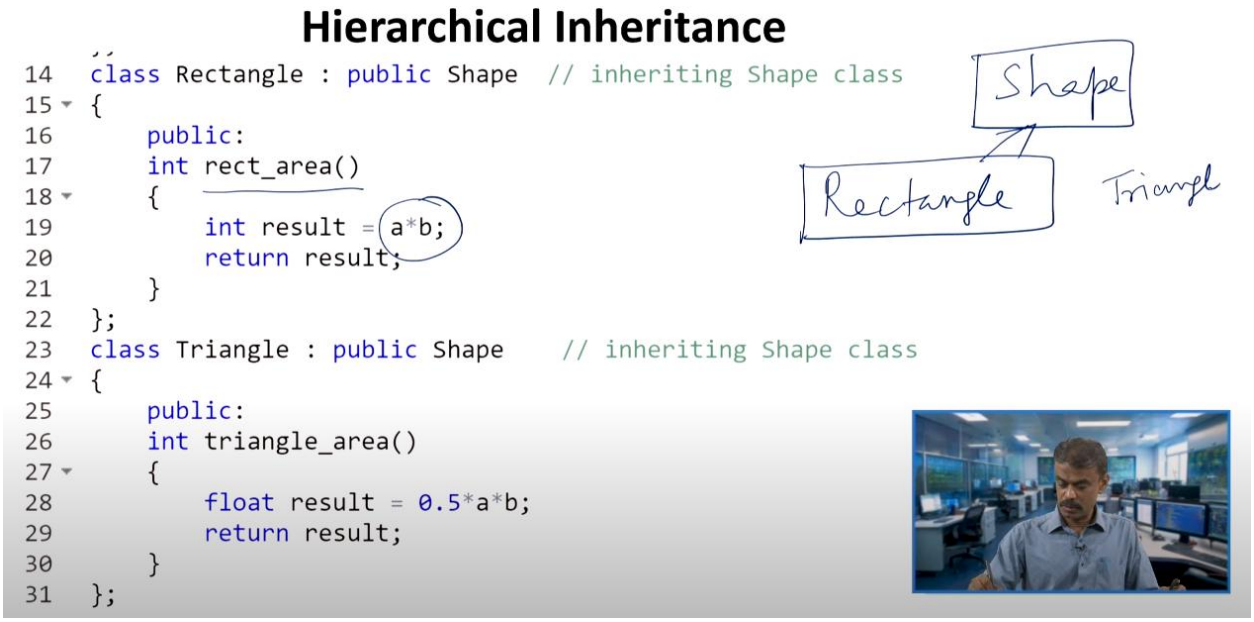
So you have protected member data and then get underscore b. You are taking the input from the user. So that is class B and you have class C. Right. So assume that your class C. So you have a protected member data called C. Right. And then you have get_c, which is taking the input from the user. cin c. Right.

So you have C. I will just simply write C. B is publicly inheriting over here. So which is having a variable c. Right. Member data c. Small c. And then it is taking the input from the user. Now class D, which is publicly inheriting B and C. Right. Class D, which is publicly inheriting class B and C. Publicly inheriting class B and C. Right.

You have protected member data d. And then assume that you have one member function called multiplication. So the multiplication, you have get underscore a, get underscore b, and get underscore c. All right. So you are calling these three functions. Then you are doing multiplication of the three numbers.

So, get_underscore_a, get_underscore_a, right. So, if you look at the diagram, D can access B, and B can access A, right. So, D can access B, and B can access A. So, there is no problem. So, get_underscore_a, you are taking the input from the user, right. So, a, you can take input from the user.

So, this can be accessed, right, the hybrid inheritance, right. And class B, which is having get_underscore_b. Right. You are taking the input from the user. So, A, B, you have taken the input from the user.



Right. And then, class D can access C as well. Right. So, class D can access C as well. So, class C, you have get_underscore_c. So, you are taking the input from the user, cin c. So now, what will happen?

Right. You have A, B, C from line numbers 45, 46, and 47. So the multiplication of A, B, and C. Will give you. Right.

Whatever number you are taking. For example. I am giving 10, 10, 10. So the answer will be. 10 into 10 into 10.

Right. 1000 will be the output. If. The input you are giving. 10, 10, 10.

Right. So now we will see. How. This will happen. Right.

So the class is over. Class D will be over here. Now I am going to the main program. Creating an object D. Small d. Under capital D. Capital D is the class. So now.

This small d. Invokes the multiplication function. Multiplication member function. So, you are getting A, B, C right from the user and multiplication of A, B, C right. So, assume that I am giving 10, 5, and 5 right.

So, 10 into 5 is 50, and 50 into 5 is 250. I have to get the output right. So, let us see what the input I am giving is. Assume that I am giving the input 20, 30, and 40, right. So, the multiplication of these three, 2 into 3 into 4 is 24, followed by 3 zeros.

So, 24000 will be my output, all right. So, ABC, you are giving as an input. And it is calling get_a, get_b, and get_c. So, you are getting the output 24000, right. So, this is how the hybrid inheritance is working, right. So, this is how the hybrid inheritance is working.

So, similarly, like what we had seen in the case of multiple inheritance in Java, you obviously do not have hybrid inheritance, right. You obviously do not have the hybrid inheritance. So, in fact, in this particular problem, we have the hybrid inheritance like this, right. B is inheriting A, and D is inheriting both B and C, right. But whereas, if the same problem we require to solve in Java, we cannot do that because Java does not support this multiple inheritance.

So, here you have this multiple inheritance. D is, right, multiply inheriting B and C. So, this concept we cannot see. So, hybrid inheritance is also not available in Java. So, in Java, you cannot find both multiple and, obviously, hybrid inheritance is not there.

So, hybrid inheritance is nothing but the multiple and the multilevel or the single inheritance, right, the combination. So, when the multiple inheritance is not there, obviously, you do not have this hybrid inheritance also in Java, all right. So, with this, the next concept hierarchical inheritance, right? What do you mean by hierarchical inheritance? So, suppose class B and class C, right? Class B is inheriting class A, and class C is inheriting class A, alright? So, is it possible? Yes, in C++, also in Java, it is possible. So, now we are going to see how this can happen. So, suppose I have, let us assume that class Shape, right? Assume that I have class Shape. So, you have two member data, a and b, right.

You have two member data, a and b, and you have get underscore data member function. So, that means whatever you are passing, n and m, will be stored in a and b, right. Whatever you are passing here, you are seeing the arguments n and m. So, that will be stored in a and b, right. So, now let us assume that I have a class Rectangle, Another class Rectangle.

So here you have class Shape. So Shape is the base class. Right. So here you go. I have Shape, where the class Rectangle is inheriting Shape.

Class Rectangle is inheriting Shape. So here you have one member function. Right. So that member function is finding out the area of the rectangle. Right.

Rect underscore area. So the result will be a star b. So, since it is inheriting Shape, in Shape you have this a and b. Whatever you are passing to get data, right? So, that you are setting over here. So, a and b values can be accessed by Rectangle.

If you look at the figure, they are publicly inheriting, right? So, a and b will be multiplied, and that will be stored in the result. And in line number 20, you can see it will be returned, right? So, that is your rectangle inheriting. Member function, so now I have another member function called Triangle, right? I have another member function called Triangle, right? So here,

So let us see how it is going. You have a member function Triangle, which is again inheriting Shape, all right? So that is why it's called hierarchical. Rectangle also is inheriting Shape. Triangle is also inheriting Shape, which is possible in C++. Also, we are going to see this is possible in Java as well, all right? So now your Triangle, Which is inheriting Shape. Alright. So, Triangle which is inheriting Shape.

Alright. So, here you go. So, you have triangle_area. Alright. Your member function is called triangle_area.

So, the float result you have 0.5 into a into b. Right. Of BH. Alright. So, whatever you are passing the base and height. So, off into b.

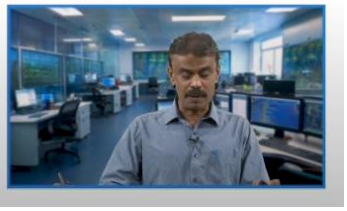
a into b passing as the number get data number. Right. The numbers. So a and b. So a and b can be accessed by Triangle as well because Triangle is inheriting Shape. Right.

So off into a into b. So that will be stored in the result. And then return result. Right. So these three classes. So Shape is the base class.

Rectangle is a derived class of Shape. Similarly, Triangle is also a derived class of Shape. So, if you consider this scenario, you call this hierarchical inheritance, right? You call this scenario hierarchical inheritance. So, now let us consider the main program.

Hierarchical Inheritance

```
32  int main()
33  {
34      Rectangle r;
35      Triangle t;
36      int length,breadth,base,height;
37      std::cout << "Enter the length and breadth of a rectangle: " << std::endl;
38      cin>>length>>breadth;
39      r.get_data(length,breadth);
40      int m = r.rect_area();
41      std::cout << "Area of the rectangle is : " <<m<< std::endl;
42      std::cout << "Enter the base and height of the triangle: " << std::endl;
43      cin>>base>>height;
44      t.get_data(base,height);
45      float n = t.triangle_area();
46      std::cout <<"Area of the triangle is : "  << n<<std::endl;
47      return 0;
48  }
```



So, in the main program, assume that you have an object r, right? Assume that you have an object r, right? So, whose class is Rectangle, and then you have another object called t, right? Whose object is, right? Whose class is?

Triangle. Object is t, whose class is Triangle. So here, you have four variables: length, breadth, base, and height. Alright. So, I am going to give us some data.

So, to get data. So, enter the length and breadth of a rectangle. Right. Enter the length and the breadth of a rectangle. So, when you do this, you are taking the input length and breadth from the user, line number 38.

Right. And then, The object 'r' that you have created under Rectangle. Right. So here, you have object 'r' that you have created under Rectangle.

Correct. So, that will be invoking get data. You are passing length and breadth. Correct. So, you are passing length and breadth.

So assume that I am passing, let us say, 10 and 20. Right. Without loss of generality. 10 and 20. Length is 10.

Breadth is 20. Correct. So what will happen? So your rectangle function, right? So you are passing, in fact, so you call it as r is invoking get_data.

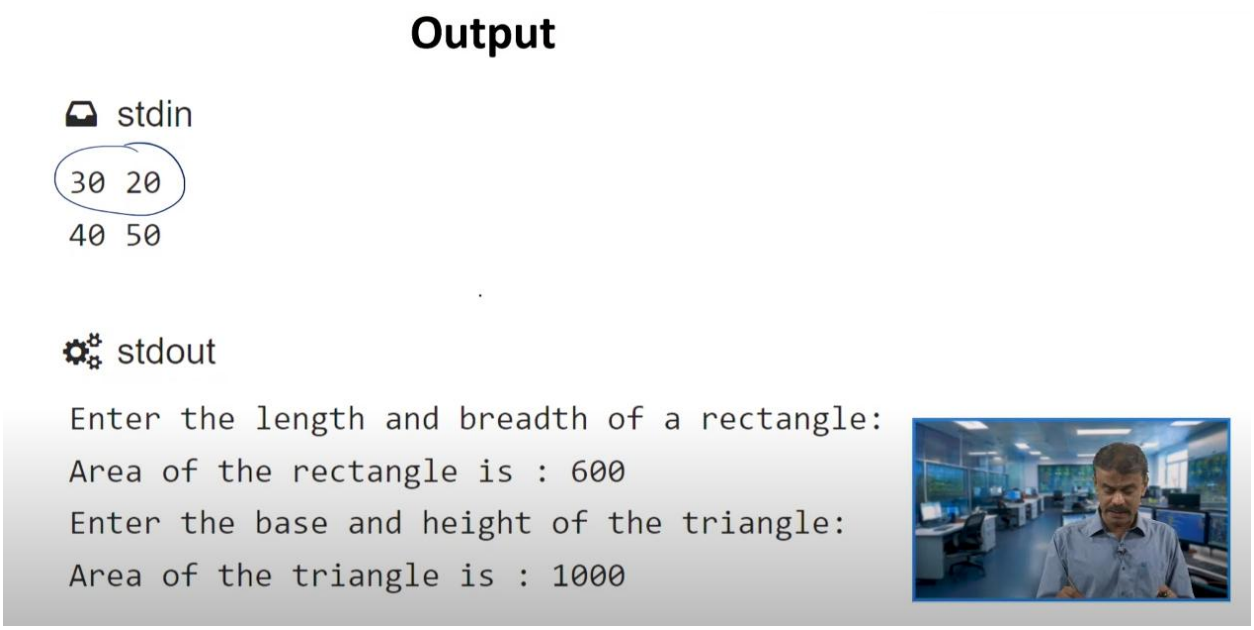
So r is under Rectangle, right? So it's publicly inheriting Shape. So it's going to Shape, right? So get_data, assume that I'm passing 10 and 20. n is 10 and m is 20.

So that means a will get the value 10, right? And b will get the value 20. Right. a will get the value 10. And b will get the value 20.

Right. So now what is happening? So you are getting data. That is what you are doing. Right.

a is getting the value 10. And b is getting the value 20. So now what is happening? You have r dot rect underscore area. Right?

r is invoking rect_underscore_area. So, what is happening with rect_underscore_area, right? So, here you go, rect_underscore_area, yes. So, here you have result is equal to a * b, right? So, I pass, as I said, I pass 10 and 20.



So, 10 and 20 will be multiplied. So, you will get 200 if you are passing 10 and 20, all right? So, It will be stored in result and it is returning result. Right.

So when it is returning result, it will be stored in m. Right. It will be stored in m. Right. r is invoking rect_underscore_area. Already you have 10 and 20 value. So these 10 and 20 will be multiplied and that is being stored in result.

Right. So, the next line you are seeing is return result. So, 10 and 20 will be multiplied, and then 200. So, the result is 200. So, that value will be returned.

Okay. So, that is exactly what is happening over here. So, whatever value it is returning will be stored in m, and in the next line, you are displaying the area of the rectangle, and whatever the m value is will be displayed. Right. Similarly, so let us give base and height.

Another data you are providing. Right. Base and height. Right. So, endl it will be.

Right. Next line. So, you are providing base and height. The user is providing base and height. So, when you are providing base and height, here you go.

The t is invoking get_data, right? So, t is invoking. Get_data. You are passing base and height. All right.

So, t is invoking get_data by passing base and height. So, you can see that. All right. So, you can see that the get_data a and b. Assume that I am passing, let us say, 50 and 100. All right.

Now, n is 50. Here, you are passing 50, and this is 100. n is 50. m is 100. So that means a takes the value 50, and b takes the value 100, right?

So that means you are taking the base and the height, right, from the user, and you are passing. Let us assume 50 and 100. So get_data a and b will have will be now 50 and 100. So now again, line number 45, t. So t under Triangle, right. So Triangle is accessing Shape, right, is inheriting Shape.

So t under Triangle will be invoking the function triangle_underscore_area. Right. So when it is invoking triangle_underscore_area. So here you go. Right.

So this is the member function. 0.5 will be multiplied by a and b. So that is the base and height. So 0.5 will be multiplied by a and b. Right. Assume that I am giving 50 and 100. So, 0.5 times 50 and 100, so whatever the result will be stored in result and you are returning result, right.

So, this will be assigned to n, whatever you are getting as the output in the result by invoking triangle underscore area, that means the object t is invoking triangle underscore area, the result will be put into n. So the area of the triangle you are getting is n. Earlier, the area of the rectangle we got was m. You are passing length and breadth which will be multiplied when r is invoking this function. And similarly, when I am passing base and height when t is invoking triangle underscore area. So the

resultant value will be stored in n. And then you are displaying n. So now we will see what the output is. I am passing, initially I am passing 30 and 20.

So I am passing 30 and 20 here. Your length is 30, breadth is 20. And r is invoking rectangle underscore area. r is invoking rectangle underscore area. So what is happening?

When it is invoking rectangle_underscore_area, 30 and 20 are being passed. So, a will get 30. So, 30 and 20 are being passed. Alright, 30 and 20, right? Yes.

So, a will become 30, and b will become 20. So, in that case, what will happen when it is calling, let us say, r_rectangle_underscore_area, right? Rect_underscore_area. So, these two will be multiplied, 30 and 20. So, 600 will be the first output, alright?

So, for this 600, we are getting the first output, correct? Correct. And in the second case, when I am passing 40 and 50, so get_data is, get_underscore_data base and height, line number 44, right. So, this will become, there is a change, 40 and 50, right. So, there is a change here, change in the sense, yes, it is in a different case.

So, here you have, this will be 40, and this will be 50. So, the meaning is a is 40 and b is 50. When t is calling the function triangle_area, it is 0.5 into a into b: 0.5 into 40 into 50, 2000 by 2, which is equal to 1000. So, you have to get the output 1000, which we are getting. So, this is how exactly your hierarchical inheritance is working. So, if you look at the diagram, your hierarchical inheritance is

The Rectangle is a derived class, whereas the base class is Shape. Similarly, Triangle is a derived class, and the base class is Shape, all right. So, hierarchical inheritance in C++ we have seen. And in the next class, we will see hierarchical inheritance using Java, all right. So, we will see the concept in Java, all right.

So, with this, I am concluding this lecture. Thank you.