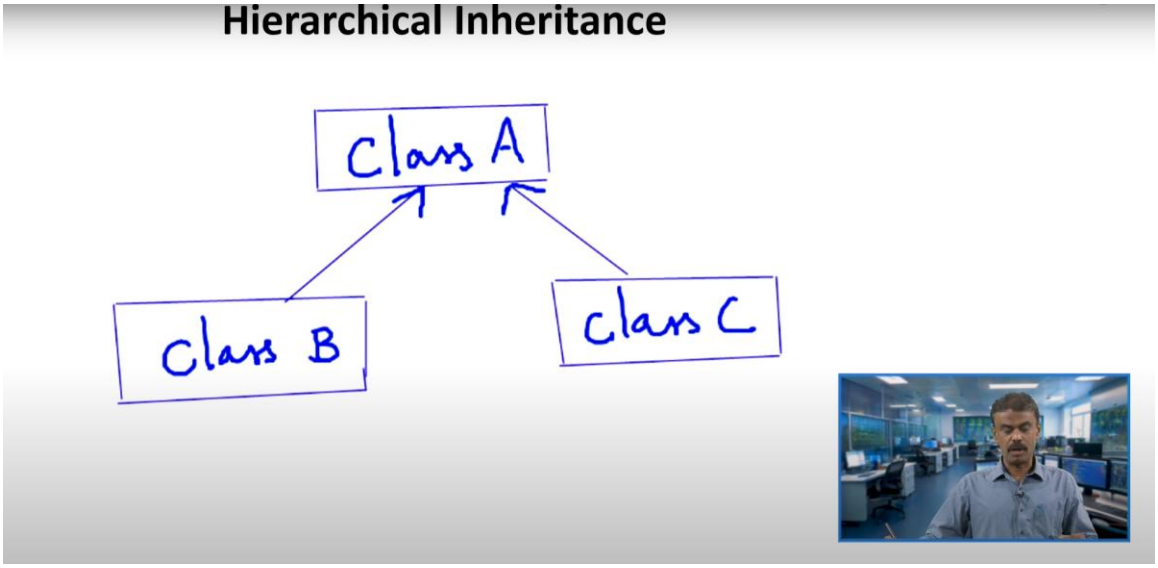
FUNDAMENTALS OF OBJECT ORIENTED PROGRAMMING

Lecture15

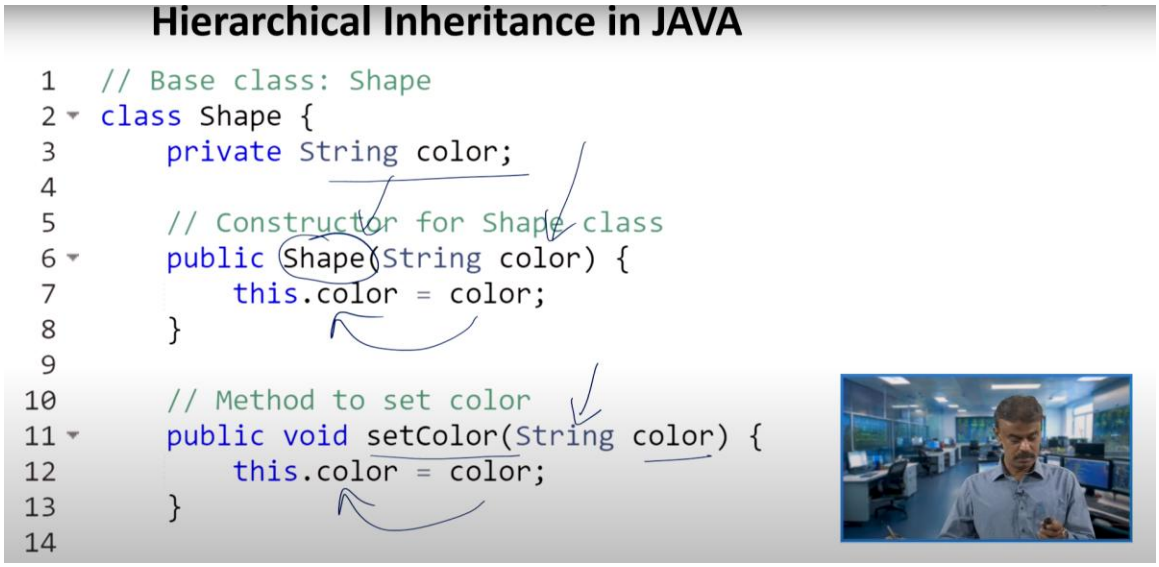
Lecture 15: Inheritance and Introduction to Friend Function



So, welcome to Lecture 15 on inheritance. In the last class, we discussed hierarchical inheritance, where we saw an example in C++, right? We took an example where class A was considered the base class, class B was one derived class, and class C was another derived class, where classes B and C could utilize the properties of class A, right? That means they can use the instances, meaning the member data and member functions.

So, with respect to C++, we saw one example, right. Continuing, what we can do here, we will take an example in Java. In Java, hierarchical inheritance is possible. However, we saw that multiple inheritance is not possible.

Right. So, when multiple inheritance is not possible, hybrid inheritance is also not possible. Correct. However, hierarchical inheritance is possible in Java. We will see one example of hierarchical inheritance.



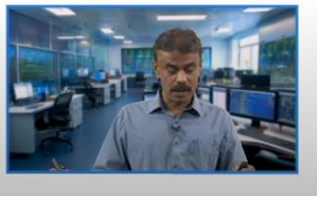
So let us consider class Shape. Right. So you have the private member data. So let us say you have string color as private member data. Right.

So class Shape is a base class. And then I have one constructor, a one-argument constructor, with string color as the argument. So the name of the class and the special member function are the same. So here you have public, which you call a constructor. So the color, this pointer we have already seen.

So color, you will get the reference. This dot color, so it is nothing but the reference. This is a pointer, so that is getting the reference. And similarly, we have one setColor method. All right.

So that is a constructor. Lines number 5 to 7 is a constructor. 5 to 8. And line number 11. So let us assume that you have one method, which is setColor.

```
26  // Derived class: Circle
27  class Circle extends Shape {
28      private double radius;
29
30      // Constructor for Circle class
31      public Circle(String color, double radius) {
32          super(color); // Call to the base class constructor
33          this.radius = radius;
34      }
35
36      // Method to calculate area of the circle
37      @Override
38      public double calculateArea() {
39          return Math.PI * radius * radius;
40      }
41  }
42
```



All right. So you have an argument color whose data type is string. And color, right, it is going into this.color. It is getting a reference. Right.

So this is one of the methods, setColor. You have another method, getColor. Right. So get color. You can see the return type is string.

Right. It is returning color. So now you have another method. Right. So which is called calculateArea.

Right. So the calculateArea is returning 0. Right. Return 0. I have not defined anything which is returning 0.

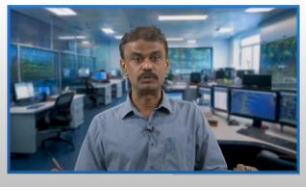
So your class is getting over here. Right. Your class is in line number 24. Class is over. Right.

So the name of the class is Shape. So I'll just draw somewhere maybe Shape. Right. It is the base class. You call it a superclass.

In Java, it is a superclass. And then, assume that I have a derived class. Right. So here you go. You have the derived class.

The derived class is Circle. Right. So let us have the derived class called Circle. Which is extending Shape. Class Circle extends Shape.

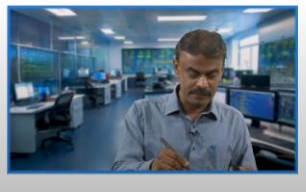
```java
// Derived class: Circle
class Circle extends Shape {
    private double radius;

    // Constructor for Circle class
    public Circle(String color, double radius) {
        super(color); // Call to the base class constructor
        this.radius = radius;
    }

    // Method to calculate area of the circle
    @Override
    public double calculateArea() {
        return Math.PI * radius * radius;
    }
}
```



So, this is the syntax in Java we have already seen. So, that means here you have the class called Circle. Circle is utilizing the properties of Shape, which means Circle is inheriting Shape, right? So, in Circle, you have the private member data called radius, whose data type is double, and then you have a member function, right? So, you have a member function called Circle.

So, in fact, it is a constructor, a special member function. Nothing wrong if I say that, right? In fact, it is a special method, right? You have a constructor, a two-argument constructor, where color and radius are of data types string and double, respectively. So now, yeah, we have in the last class we had seen super, right? The keyword super, so super of color, right? So that means it is calling the base class constructor. Super of color is nothing but calling the base class constructor. What is the base class constructor? So here you go, you are passing this.

```
61
62  // Main class to test the hierarchical inheritance
63  public class Main {
64      public static void main(String[] args) {
65          // Creating a Circle object
66          Circle circle = new Circle("Red", 5.0);
67          System.out.println("Circle Color: " + circle.getColor());
68          System.out.println("Circle Area: " + circle.calculateArea());
69
70          // Creating a Rectangle object
71          Rectangle rectangle = new Rectangle("Blue", 4.0, 6.0);
72          System.out.println("\nRectangle Color: " + rectangle.getColor());
73          System.out.println("Rectangle Area: " + rectangle.calculateArea());
74      }
75  }
76
```



So, the base class constructor, right? So, in Shape, you are passing the color. So, color will be assigned to this dot color. So that is what exactly happening over here, line number 32, and here in this particular constructor, radius is assigned to this dot radius. This is the pointer, so that means it is getting the reference.

So now I have another member function of the same name, right? Method, right? I have the method calculateArea, which is in Circle. Slowly, we are getting the overwrite concept, right? When I write overwrite In Java, which is possible, that means overriding is happening over here, right? Because if you look, I have calculateArea in line number 21, which is available in the Shape class, right? So this function or this method is available in both classes, calculateArea, right? Shape returns 0, and in the case of a circle, it is calculating pi r squared, Math.PI, right? I have included Right. Assume that I have included the math.

Right. Math library. And then so Math.PI. And then you have radius multiplied by radius. Pi r squared.

Basically, pi r squared. Return pi r squared. Pi r squared, we know that it is the area of the circle. Correct. So here, in the case of a circle, we have the area.

Calculate area. Right. Overriding will happen. We will see which one will be called. Correct.

So, anyway, we will extensively study the overriding concept, right? So, that is one class, right? What about another class? Another class is Rectangle. Rectangle is also extending Shape, right?

The Rectangle is also extending Shape. So, let us see. I will put the diagram. So, Rectangle is also extending the Shape, right? So, that means Rectangle So, this is the perfect example of hierarchical inheritance, meaning the class Circle is also inheriting class Shape, and Rectangle is also inheriting class Shape, right?

So, this is exactly what is happening. So, let us see what all the. Member data and methods are being used in Rectangle. Here you go. So you have length and width.

Both are private member data. The data type is double. Right. And then you have a constructor Rectangle. So you are passing color, length, and width.

Three argument constructor. So super of color. That means it has to call the base class. Exactly like what we had seen in the case of Circles. So, in Rectangle also you have super of color, line number 32, like line number 32 you have line number 50 as well, super of color, right.

So, that means the base class constructor will be called, will be invoked, and this dot length is length, this dot width is width. That means both length and width are getting the reference right for this particular object. Again, here you have an overriding function, an overriding method. So here you go, calculateArea, so that calculateArea will return length into width. calculateArea here, in this case, will return length into width. So that means all these classes are having this calculateArea method right? Shape is also having it, which is returning 0, line number 22. 23, and here also you have line number 36 to 41, right? 36 to 40, you have calculateArea for the case of Circle, and for the case of Rectangle, you have 55 to 58, right? 59, right.

So, same function, same method. So now we will see in the case of the main program how they are being called, right. So in the main, how they are being invoked, right? So which will be challenging. So, you have void main. So, this is a driver class, class Main, right.

So, you have the hierarchical inheritance. So, hierarchical inheritance, in the sense that both Circle and Rectangle are being inherited by Shape, right. They

are inheriting Shape. Now, in the case of main, assume that I have one circle object, small c. So here, I have circle small c, I have one circle object, right.

So, circle object you are passing what? Circle object you are passing red and 5.0, right? So, circle, you are creating an object under the capital C class, right? The capital Circle class, you are calling this inheritance, sorry, you are calling this constructor, right. Dynamically, you are allocating memory with the new operator, and you are calling this. Circle constructor by passing two arguments, right? Parameters are constructor, you are passing red and 5.0.

So, when you are passing red and 5.0, let us see what is happening. Circle, go to Circle. So, here you are passing red and 5.0, color is red, and radius is 5.0, right. So, here I write. And 5.0. Right.

So now you have here the super color. That means the base class constructor will be invoked. When the base class constructor is invoked. You are passing red. Right.

The color is red. Right. So, the base class constructor. What is the base class constructor? So, you have Shape.

Right. With the help of super, you are doing. Right. With the help of super, you are doing. By passing the color.

Right. So, that means the base class constructor will be called. So, that means line number 6 will be invoked by passing the color. The color is red. So, that means this dot color is red.

So the color will become red now. Right. So the color will become red now. This dot color is red. And this dot radius is the radius.

5.0 you are passing. So this dot radius is 5.0. Right. This dot radius is 5.0. So this dot color is red.

This dot color is red. And this dot radius is red. This dot radius is 5.0, right? So, when you are calling, when the circle is calling getColor, right? When the circle is calling the getColor method, what is happening?

So, circle is calling getColor. getColor is returning the color, right? Right? We are not setting any color. Yes, getColor, right?

We will return the color. Circle dot getColor. So that means it is returning the color. So what is the color? So you are already saved.

This dot color is the color, right? So here you go, return the color. So the color is red. So for this particular output, I should get red. For this particular output, I should get red.

And the next one, circle dot calculateArea, right? Circle dot calculateArea, right? So, circle, you have under circle class, right? Right. Circle, under you have circle class.

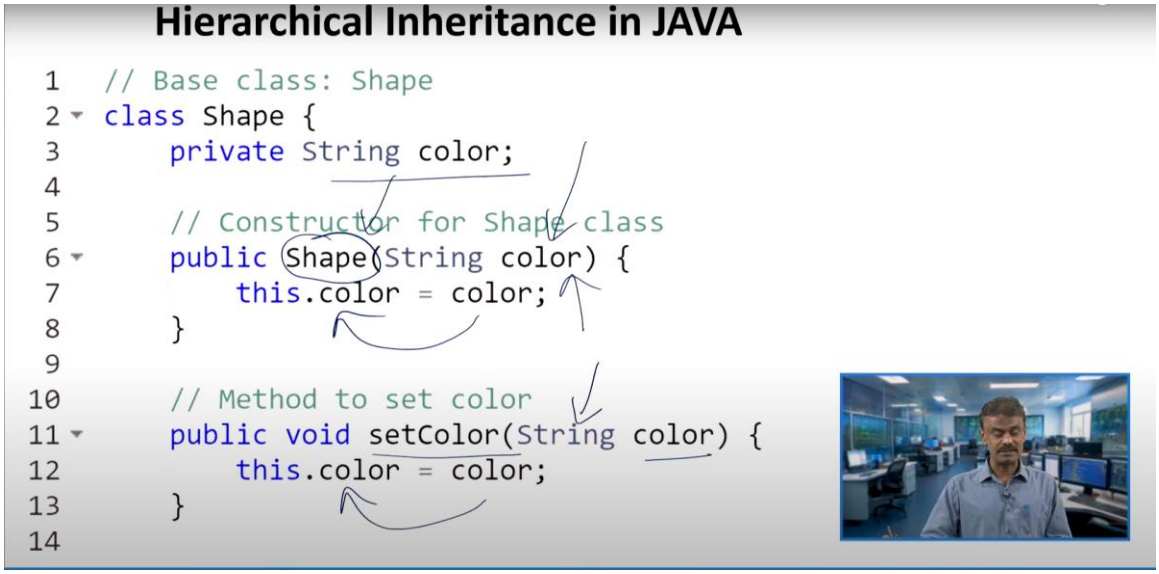
Overriding. Right. So, pi into R into R. So here it is overridden. So it will not call this. So here you have.

So this will be called. So, pi into R star R. So R you are passing 5 means pi into 5 into 5. Right. Pi value into 5 into 5. So that will be displayed.

Right. When I run the code. So this will be pi times R. 5.0 the whole square. Calculate and keep it ready.

So 3.14 times 25. So you see what is the result. Similarly, when you are creating an object rectangle under capital R Rectangle class, you are passing blue, 4.0, and 6.0. So three we are passing. Three arguments we are passing.

So three parameters. Color and then Hopefully, length and width. Color, length, and width. Right.



So when you are passing this, it is calling the base class super color. So that means color is being set. Like the last derived class, what happened is the same thing. Same thing. This dot color is color.

Right. So what is the color you passed? Blue. Right. So we have passed blue.

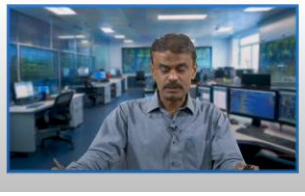
So now this will be blue. Right. And then when it is calling rectangle dot get color, color will be returned. So here I will get the output blue. Here I will get the output blue.

So, these are all the outputs. I will just tick it. We will see. Right. You will get the output in blue.

And rectangle dot calculateArea. CalculateArea. It will be overridden, and it goes here. CalculateArea is nothing but length multiplied by width. So, what is the length you are passing?

```
61
62  // Main class to test the hierarchical inheritance
63  public class Main {
64      public static void main(String[] args) {
65          // Creating a Circle object
66          Circle circle = new Circle("Red", 5.0);
67          System.out.println("Circle Color: " + circle.getColor());
68          System.out.println("Circle Area: " + circle.calculateArea());
69
70          // Creating a Rectangle object
71          Rectangle rectangle = new Rectangle("Blue", 4.0, 6.0);
72          System.out.println("\nRectangle Color: " + rectangle.getColor());
73          System.out.println("Rectangle Area: " + rectangle.calculateArea());
74      }
75  }
76
```

this.color = 'Red';
this.radius = 5.0
Red



4 and 6. 4 multiplied by 6, I will get 24. This calculation is slightly easier. So, 24.0, I have to get 24.0. So, red, pi will be multiplied by 5 into 5.

Blue, 24.0 should be my output. So, when I run the code, So, I will get the output of this, right. So, you can try. So, what can you do?

You try and get the output of this. So, you should get one output as red and another output as pi into 5.0 the whole square, right. So, the third output for rectangle.getColor, you have to get blue as output, and at the last, 24.04 into 6 will be the output. So, the fourth output you will get. Right.

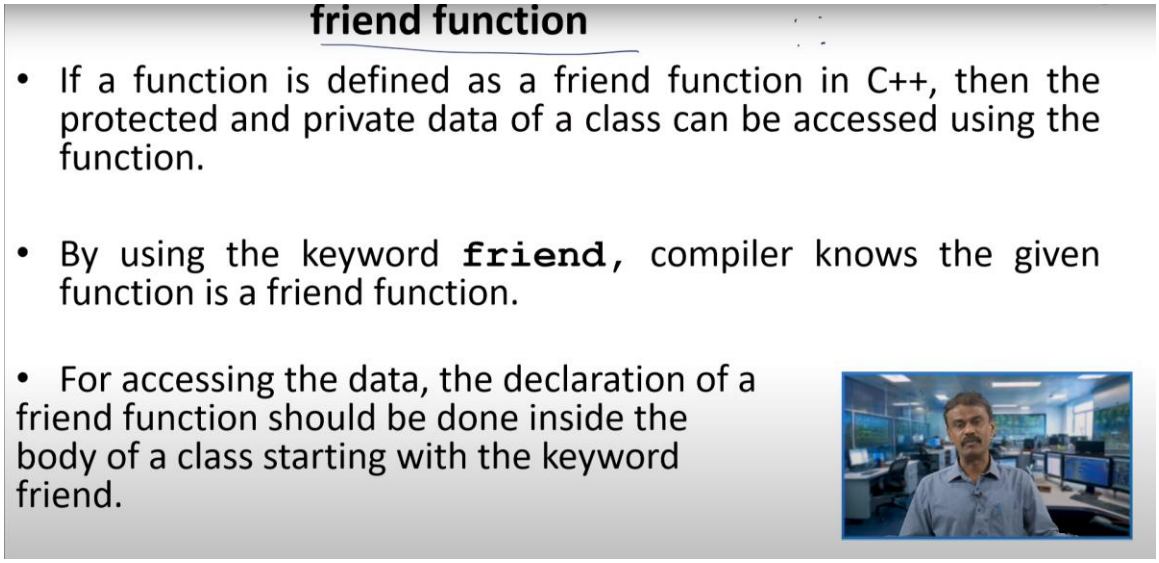
So, this is how hierarchical inheritance works, right. So, here, when you look at the scenario, your Shape is the base class, right. Your Shape is the base class, and Circle and Rectangle both are inheriting Shape, right. So, this is exactly how it happens, right. Circle is the derived class where the base class is Shape, and Rectangle is the derived class where the base class is Shape, right.

And then you are having. The set color, get color, calculateArea. In fact, calculateArea is being overridden, right? So, when we talk about method overriding, right? So, we will take the same example, right?

We will take the same example and work it out, right? So, now the task is, what I am asking you is to run the code and get the output. So, you have to get the output. Anyway, we are going to see this program again. Right, in the case of overriding, at that time we will verify what outputs we are going to get.

So, one is red, another one is pi will be multiplied by 5.0 into 5.0, third output you have to get blue, and fourth output you have to get 24.0. So, note this down. So, when we talk about overriding methods, we will take the same example, okay? So, the next concept is the friend function, right? Suppose, right, if a function is defined as a friend function in programming like C++, then suppose you want to access the protected and private member data, right, of a particular class.

So, this can be accessed using the friend function. So, it is exactly the meaning of a friend, right? So, I mean you can take any help from a friend, the usual meaning, the physical meaning. Right, so in a similar way, if you recall, we had some problems with using this protected member data and private member data, right? So, how can we access it? We had a lot of restrictions when you use inheritance. We had a lot of restrictions, but suppose you are using a friend function. These protected and private member data of a class can be accessed, right?



So, that you are defining as a friend function. Here, we use the keyword called 'friend,' right? So, we use the keyword called 'friend.' So, when you use the keyword called 'friend,' the compiler knows the given function is a friend function. All right. So, the compiler will come to know the given function is a friend function.

So now, suppose I want to access the data. Right. So, when you have the declaration of a friend function. Right. So, this should be done inside the body of a class.

Right. So, which is starting with the keyword 'friend'. We'll see an example. All right. So, I will write inside the class.

So, I'll put 'friend' and then whatever the name of the function is. So, we'll see with the help of syntax. So here, you have the class name of the class. So, inside. So, let us assume one of the statements.

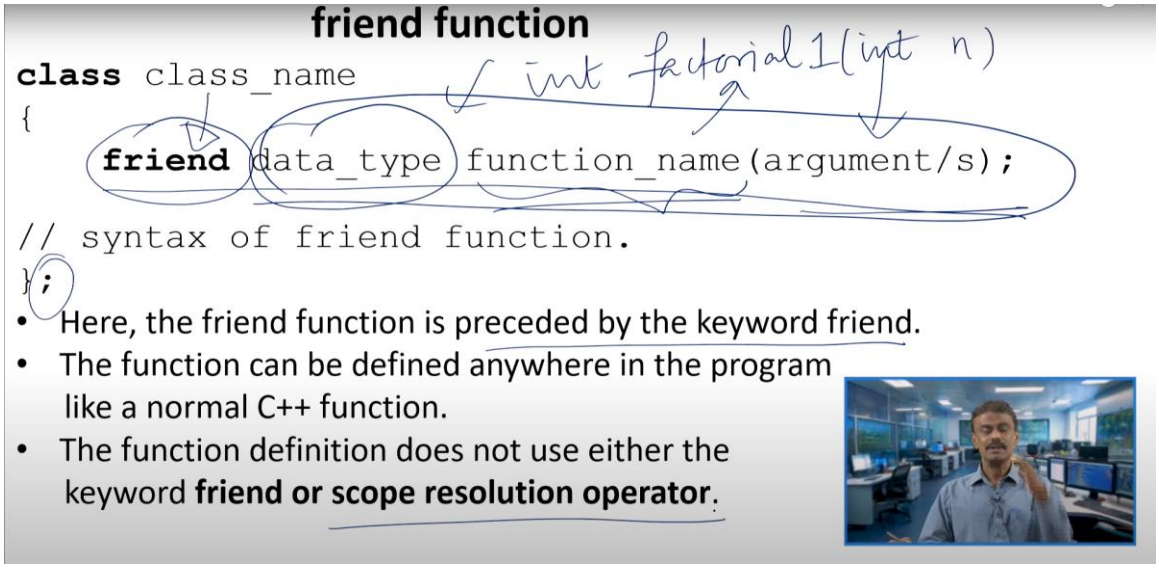
All right. I am writing like this. So, I use the keyword called 'friend'. Right. I use the keyword called 'friend'.

And then you have a data type. Right. So you have a data type. And then here you have the name of the function. Right.

Name of the function. As usual, you will have an argument. The usual function name. With the argument. Right.

So this is exactly the syntax. And then your class is over. So inside the class, you can have. Right. Anything like a member data, other member functions.

So, once you put the word friend, the compiler knows that the given function is a friend function. Right. So that means when I want to access any data, I am going to do it. Right. Suppose I want to declare this function outside the class, so I can do it.



Right. So that we are going to see. So, suppose I want to write define over here inside the class and I want to define outside. Yes, it is being done. So that means that function I am making as a friend.

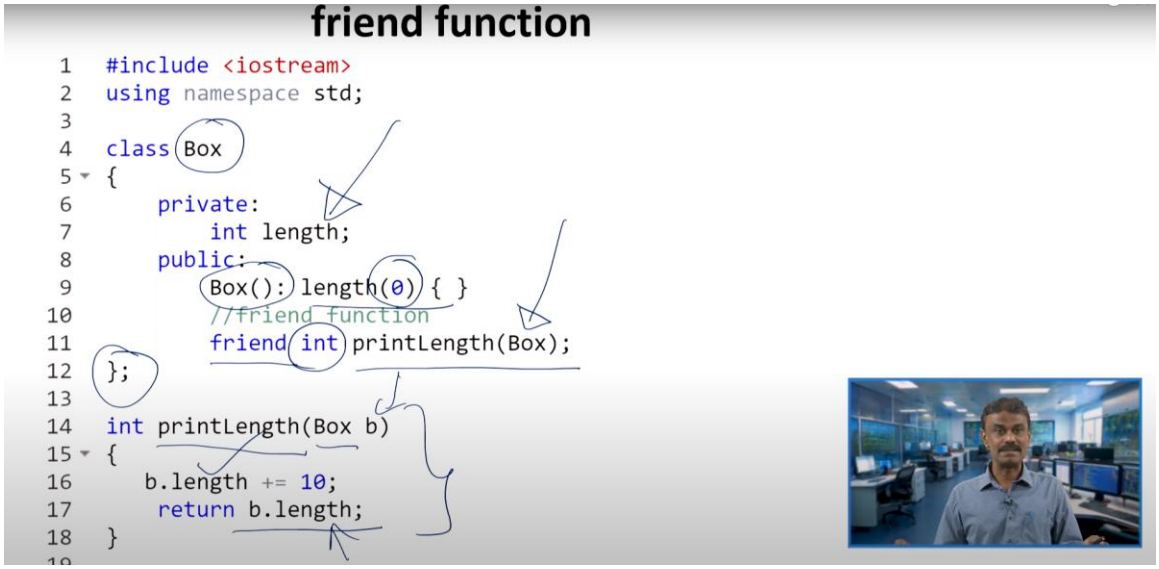
So, with the keyword friend, I can use this particular concept. So that means the friend function is preceded by the keyword friend. So here you go. I am using the keyword friend, right? And then a usual function, return type, you have data type and the name of the function, and you will have the arguments, right?

And the beauty is So here the function you can define anywhere in the program, like the procedural-oriented program you have seen. Suppose I want to write a factorial function. So I can write it anywhere, and then you are calling the function. In a similar way, you can define a function and inside the class you make this as a friend.

All right. So, that is what if you look at the second point. So, the function can be defined anywhere in the program. Right. Like what you have done in the case of C++.

All right. And the last point is the function definition does not use either the keyword friend or scope resolution operator. So, when I want to define a function, you can define it as a normal function. Right. For example, here you have friend int factorial.

Right. Int, let us say factorial 1. int n. So this function assumes that instead of this, I am writing this. So this function I can write anywhere, right? I am not going to use any friend keyword or I am not going to use any scope resolution operator.



So, like a normal function, I am going to define it outside the class, right? Anyway, we will see an example of how this friend function is working, right? So here you go. So you have class Box. Under class Box, you have a private member data called length, right?

So, here you have a constructor, length is equal to 0, right? Here you have a constructor, length equal to 0. So, now I have one friend function. Friend, whose function is printLength, you are passing the object Box, right? You have a class Box, and the return type is integer, right?

You have a parameter argument, whose class is Box, right? And the name of the function, if you look, it is print length. Your class is over here. So now, with the usual function definition, I am doing this. With the usual function definition, I am doing this.

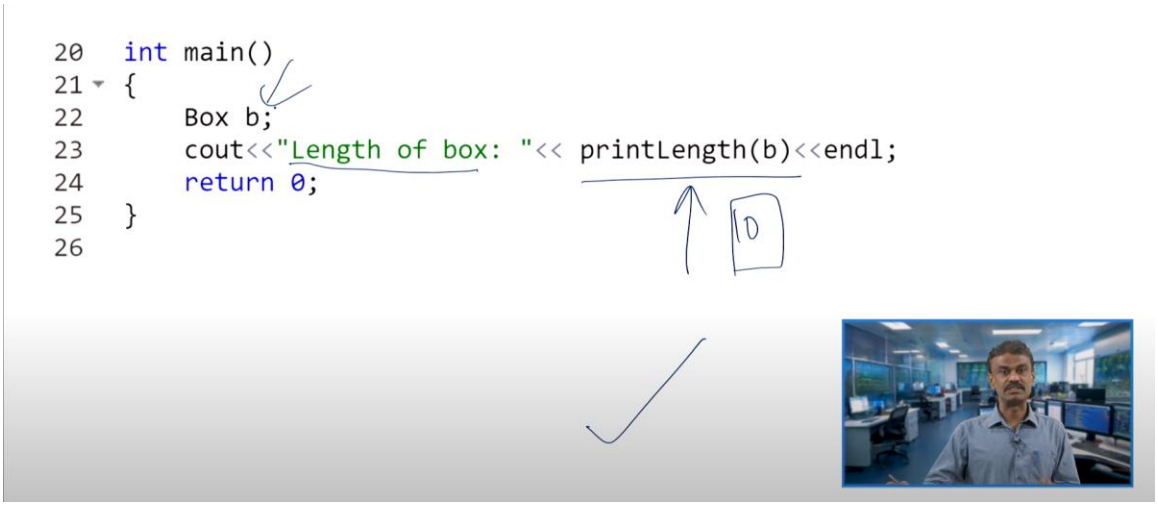
I have defined an object b. Under class Box. The name of the function is print length. Which is nothing but friend. So this function is friend to class Box.

That is the meaning. Alright. So this function is friend to class Box. So that means this function can access. Alright.

Any member data or member function. For example. Here I have b dot length. Right. Small b I created an object.

Under Box. So b dot length. If you look, I am using this outside the class. So this can be very well done, and moreover it is a private member data, right? Int length is a private member data, that is the beauty over here.

So now since this is a front function, I can very well use it; you do not get an error. So b.length is equal to b.length plus 10, that is the meaning. And then whatever you are getting the output, return b.length, alright? So that means b.length is being returned when it is being called. So, now we will see how we are going to call this, right?



So, here you have int main. I created an object b. Under Box cout, I call this function print length of b, passing b, right? So when you're calling this function, right, this object is being passed b dot length, right? So b dot length is initially 0 by default. It will call the constructor, right? When you're creating this object Box b, it will call the constructor, and in the constructor, what is the value? The value is 0, right? So this value Will be added to 10. 0 plus 10.

Right. So that means b dot length is 10. b dot length will be returned. So here you will get the output 10. Right. Here you will get the output 10.

So return 0. So, is it clear? So the key point here is line number 11, the friend function. So you have a friend function, and the friend function is being defined in

line number 14 to 18. In fact, you are defining it outside the class, and outside the class, the beauty is you are using length.

Which is the private member data for box, right? And when you call this, you get the output 10, right? So, when I run the code, I will get the length of the box, right? So, this is the output, and what is the value? 10 I am getting, right?

So, initially, when you have the constructor, right? When you instantiate this object b, the constructor will be called, length is 0, right? That is what you are doing. Length is 0. Right.

And then, when you call this function printLength. Right. So here you go, you have printLength. So b.length. Right.

You are passing b. So, what is b.length? It was 0 previously. The 0 will be added to 10. That is what the statement here says. So, 0 plus 10 is 10.

Therefore, you are getting the output 10. When you are calling this function, because it is returning b.length. So, you are getting The output 10. And when you are running the code, here you go.

You will get the output 10. I hope it is clear for everyone. This is how the friend function works. So, in fact, we have seen five different types of inheritance. And then, the friend function, when I want to access outside the class, I can even access the private member data and protected member data without any problem, particularly the private member data.

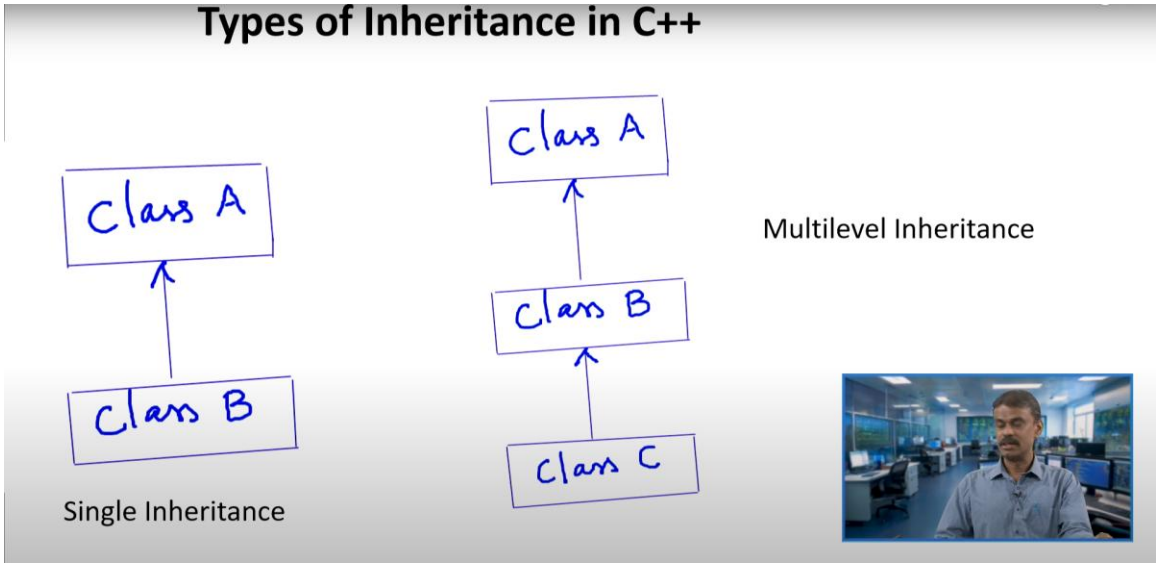
Here you go. Right. So, particularly when I want to access the private member data in this particular program, not a problem. Line number 16, you can see b.length equal to b.length plus 10, right. 0 plus 10, you are getting the value 10, and that is being returned, and here you have the value 10, right.

So, this is how the friend function works, right. You are getting the output 10. So, in the last few lectures, we had seen different types of inheritance in C++. Right. So, we call this as single inheritance, where class A is the base class and class B is the derived class, right.

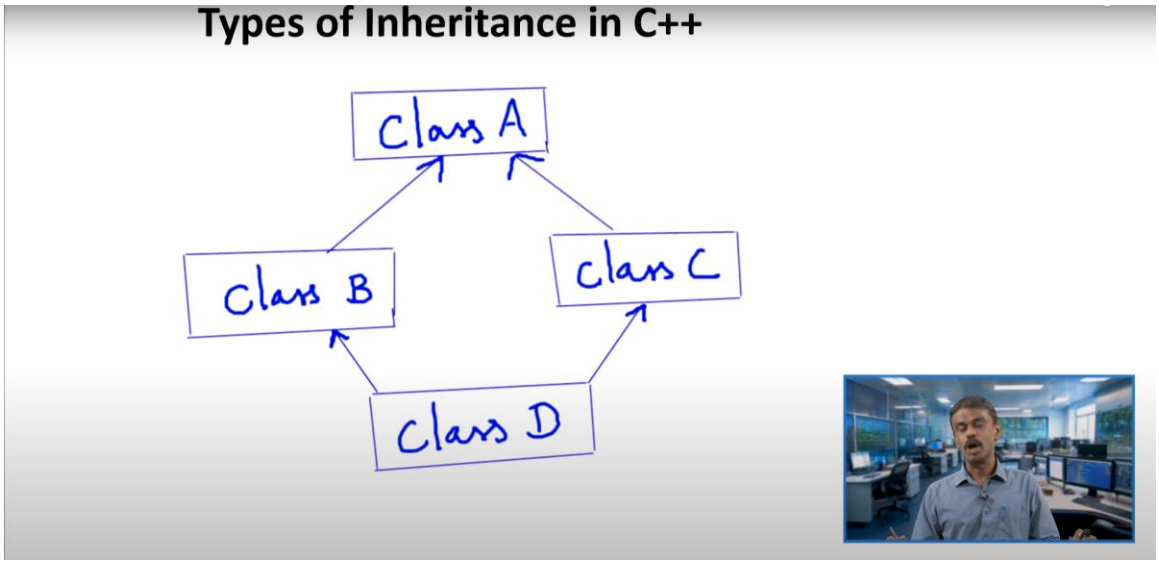
So, you call this as single inheritance, whereas all the properties of class A can be utilized by class B. Similarly, you have multilevel inheritance, class A is the base class, right, class B is inherited, right, it is inheriting The properties of class

A, and class A is inheriting the properties of class B. So, when you have this hierarchy, you call this as multi-level inheritance. So, here class A is the base class, class B is the derived class of class A, and class C is the derived class of class B. We call this as multi-level inheritance. And the third one is multiple inheritance, right? So, here class C is a derived class, but the base classes are class A and class B. So, which is possible in C++, right?

Whereas, this concept is not available in Java, right? Multiple inheritance is not available with respect to classes, right? So, later, sooner or later, we are going to study about interfaces. The interface part in Java will take care of this multiple inheritance, right? And the fourth one is hierarchical inheritance.



We have seen the example, right, in both C++ and Java where class B and class C, right, they are the derived classes, whereas for both these B and C, class A is the base class. So, you call this hierarchical inheritance. Now, the combination of multiple and hierarchical. So, that means this is possible in C++. The combination of multiple and hierarchical is called hybrid.



So, class B and class C are derived from class A, and class D is derived from class B and class C, which is possible, right? So, these are all the types of inheritance we have seen in C++: all the examples, right, all the properties, along with the friend function, right. So, we have concluded that all these five types of inheritance are available in C++, whereas multiple inheritance is not available in Java. So, obviously, hybrid inheritance is also not available in Java. We have seen several examples and case studies. With this, we are concluding the inheritance concept. Thank you.