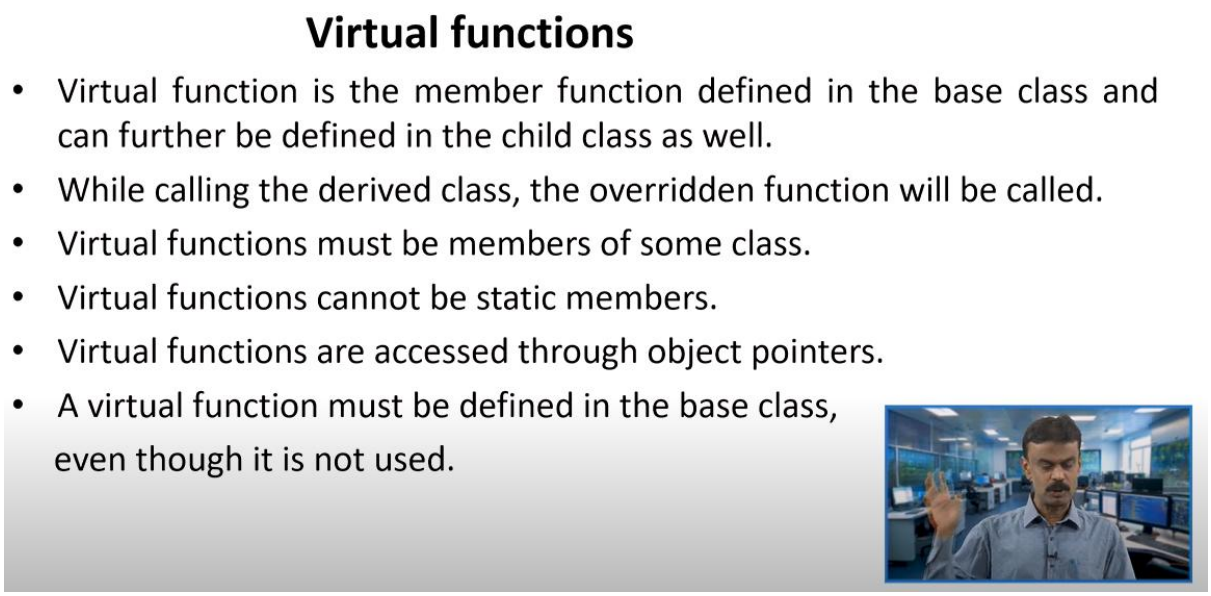
FUNDAMENTALS OF OBJECT ORIENTED PROGRAMMING

Lecture24
Lecture 24: Virtual Functions in C++ and Abstract Class



Welcome to Lecture 4: Encapsulation and Abstraction. So, in today's lecture, we are going to see virtual functions in C++. So, the virtual function is nothing but a member function in the class. In fact, it is defined in the base class and can also be further defined in the child class. So, in the case where you are defining this virtual function,

You have to define it in the derived class, right? That means the function you have written in the base class should be overridden, right? And in the derived class, that particular function will be called. And a virtual function should be a member of some class and cannot be a static member, right? And this can be accessed through object pointers.

Okay. So, when you are defining a virtual function in the base class, right? That means even though we are not defining it inside the base class, right? Even though you are not giving any proper explanation, or defining it, we still have to declare it in the base class. So, the prototypes of virtual functions in the base class and all its derived classes should be identical.

So, I mean, you may ask the question: suppose the function name is the same, but the parameters are different, right? Or the number of arguments is different, or the return type is different. Then it will be called an overloaded function. Whereas here, we are talking about the overridden. And also, in the case of the constructor point of

view, there is no concept called a virtual constructor. Whereas we have a virtual destructor. So now we will see an example.

Virtual functions

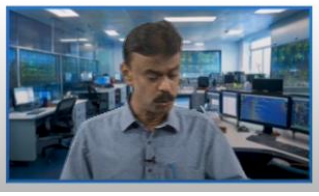
- The prototypes of a virtual function of the base class and all the derived classes must be identical.
- If the two functions with the same name but different prototypes, C++ will consider them as the overloaded functions.
- We cannot have a virtual constructor.

What do you mean by a virtual function? Let's see an example. Consider this particular class. So you have one member data, x equal to 5. And you have one member function called display. So this is displaying the value of x, which is what?

virtual functions

- Consider the situation when we don't use the virtual keyword.

```
1   #include <iostream>
2   using namespace std;
3   class A
4   {
5      int x=5;
6       public:
7       void display()
8       {
9           std::cout << "Value of x is : " << x<<std::endl;
10      }
11  };
12
```



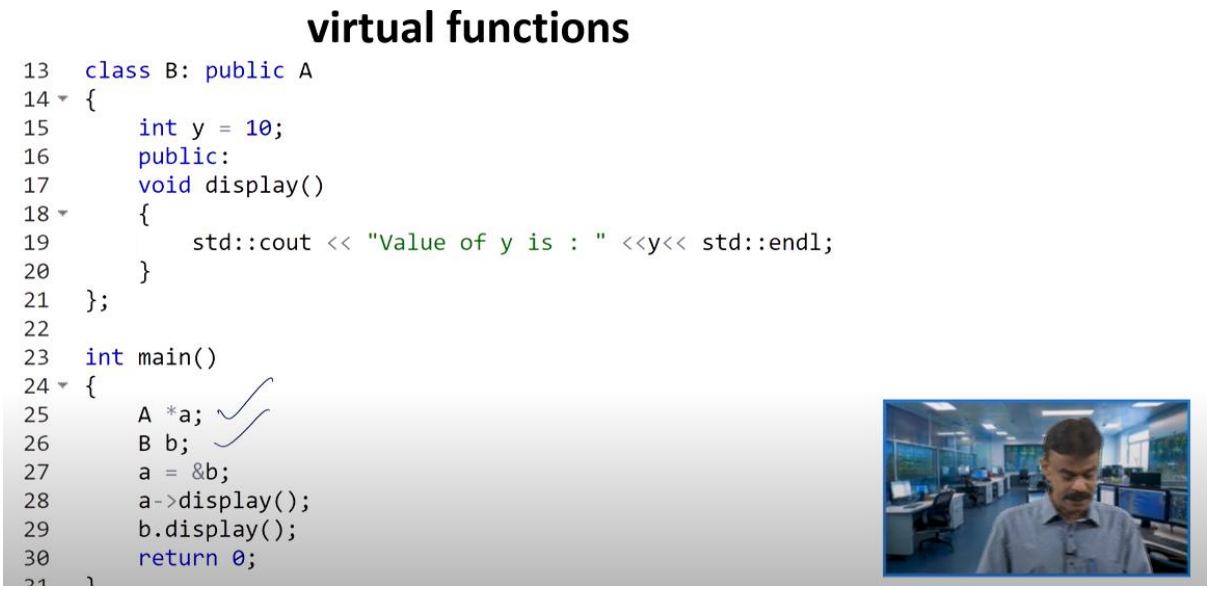
x. Alright? And let us say this is a derived class. Class B, right? Which is publicly inheriting class A. So, here y is equal to 10, and here you have a member function.

Alright? So, in the member function, let us say the value of y. Alright? The value of y is we are displaying y. Right? So, the same display function exists in both the base class and the derived class. Right?

So, now let us consider the pointer object A, right? So, you have a pointer object small a under class A, right? And then the usual object b under class B. So now this a is pointing to the address of b, right? The a is pointing to the address of b. So now you do a display.

So, a display is nothing but, so this will call the base class, right? So, this will call the base class and it will print 5. All right. So, this will call the base class A. So, you have an address. Assume that you have a pointer a, a pointer variable a. Right.

So pointer variable a and you have object b. Right. So let us say address b is at 4002. So now a is also pointing to 4002 according to this line. OK. So a arrow display.



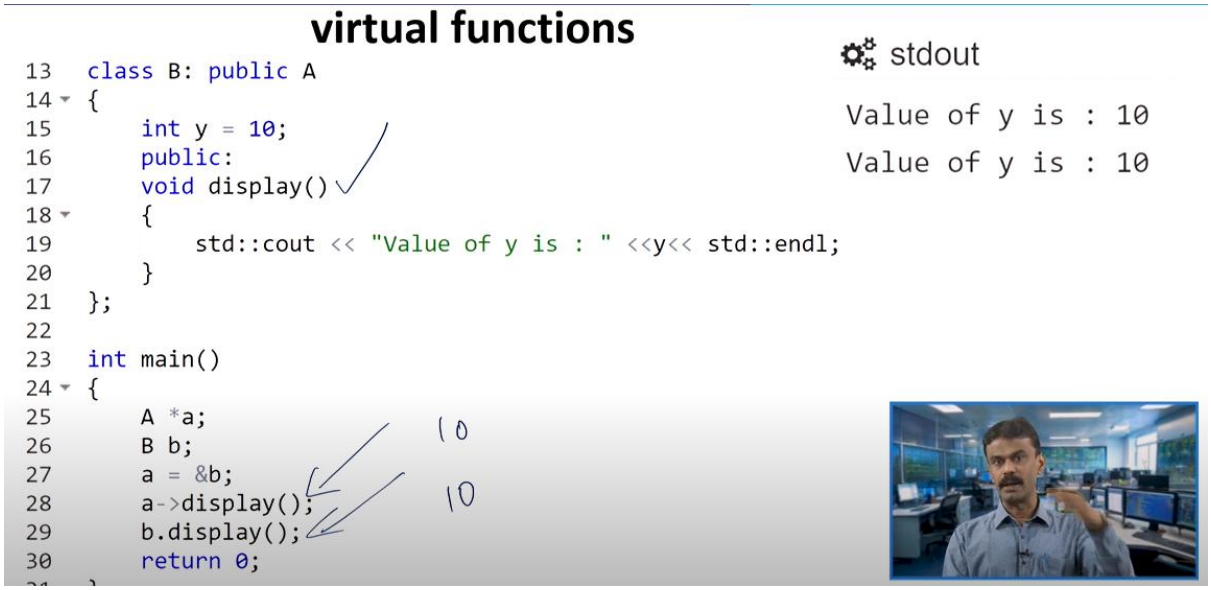
So a arrow display. Right. So which will print A. In fact, it will call the base class of the member function. So value of X is what?

5. So 5 will be printed. So the first output we will get 5. So B dot display. So b is an object.

Small b is an object under capital B. So you have a display function value of y. y is 10. So you can see, as expected. So B dot display will print 10. We already know 5. And then 10, so these two are the outputs we are getting, right?

So now, if I use the keyword virtual, right? So here, line number 7, assume that I am using the keyword virtual. So when I use the keyword virtual, the same program, right? So now, that is the usage of virtual, the keyword virtual. So virtual void display.

The same function, all are ditto, right? So whatever we have seen in the previous program, it is exactly the same, right? So now, if you look, a display, it will be overridden because of the virtual, correct? So when it is overridden, what is the output expected? So you will expect this display function will be called for this call, right?



And similarly, B dot display, this one will be called. So, I should get 10 and 10. In this case, I should get 10 and 10. So, this is the usage of the virtual function. So, when, I mean, we have studied overriding, right, overriding.

So, here in this case, when you have the virtual function, right, so even though you have defined, you can see the difference between the previous program and this particular program. So, now, what do you mean by a pure virtual function? All right. So, as we know, the virtual function will not perform any task on its own. All right.

Pure Virtual Function

- A virtual function does not perform any task on its own; it acts as a placeholder.
- When a function has no definition, it is referred to as a "do-nothing" function.
- Such a function is known as a pure virtual function, which is declared in the base class but has no definition for that class.
- A class containing a pure virtual function cannot be instantiated, and such classes are called abstract base classes.
- The primary purpose of an abstract base class is to provide characteristics for derived classes and to enable the creation of base pointers, which facilitate runtime polymorphism.

int fact(int n)
{ }

So, it is acting like a placeholder. So, assume that in the case of the virtual function. So, let us assume I am not defining anything. Right. It is like a do-nothing function.

Right. So, suppose I have int fact. Suppose I have int fact and then int n, and assume that I am not defining anything. So, that is in the base class, right? So, you call this a do-nothing function.

So, when you have such a function, you call this a pure virtual function, right? So, which is declared in the base class, but that has no definition. That is the example that I had given. Right. So, this has no definition for that class.

When you have a pure virtual function, you cannot instantiate any object. So, you cannot define any object. Right. Because you have not defined anything inside the pure virtual functions. Right.

Suppose we assume that a class has a pure virtual function. Right? Since you have not defined anything, I cannot create any object. Such a class is called an abstract-based class. This is what we are going to see.

```
 9  class Derived : public Base
10  {
11      public:
12      void show()
13      {
14          std::cout << "Derived class is derived from the base class." << std::endl;
15      }
16  };
17
```

In fact, in the case of Java, we are going to see what an abstract class means. Right? So this is called an abstract class. I already defined it. Right?

What do you mean by abstract? Right? When you are writing a research paper, or those who are reading a research paper, or those who have done some project might have referred to it. Or you might have written it in the research paper. So after the title, you write the abstract.

So the abstract contains only what you are going to do. It won't have any definition, right? Any theorem proof or any simulation work that you are going to put there. So you are going to say that, right? So in a similar way, when you are defining, right?

When you are, let us say, declaring a function prototype, assume that you are not defining anything, right? So such a class is called, suppose it has a function, right? A pure virtual function, a class which contains a pure virtual function, such a class is called an abstract class. But when you have the abstract base class, right? So assume that I have this factorial program, right?

So this should be defined in the derived class, right? So this fact function, the fact member function, which is in the abstract class, which is abstract. In fact, it is a pure virtual function. So this should be defined in the derived class, right? We know that, right? So what we have to do is we have to define a base pointer, right? You have to create a pointer in C++, and during runtime, that will automatically go to the derived class, which we had already studied, right?

So, in fact, when we know that it is a do-nothing function, then during runtime So, it will go to the derived class, and then in the derived class, since you have defined the function, that will be executed. So, this is also called runtime polymorphism. So, the

pure virtual function, which is nothing but the base of runtime polymorphism. So, we will see how do you define the syntax?

How do you define the pure virtual function? Right. So, in the case of a pure virtual function, what you have to do is use the keyword virtual. Right. Void show.

Usually, you write like this. Right. The function you write like this. So, now you are also putting equal to zero. Right.

You also set it equal to zero. So virtual equals zero. So when I write it like this. So this is called a pure virtual function. You may ask, what is the use of this pure virtual function?

So, for example, I want to define, let us say, hundreds of functions. You are writing a million lines of code, and you want to have hundreds of functions. So this will give an idea of all the functions I am going to use. And at a later stage, in the derived class, I will define all these functions. So now let us consider the derived class.

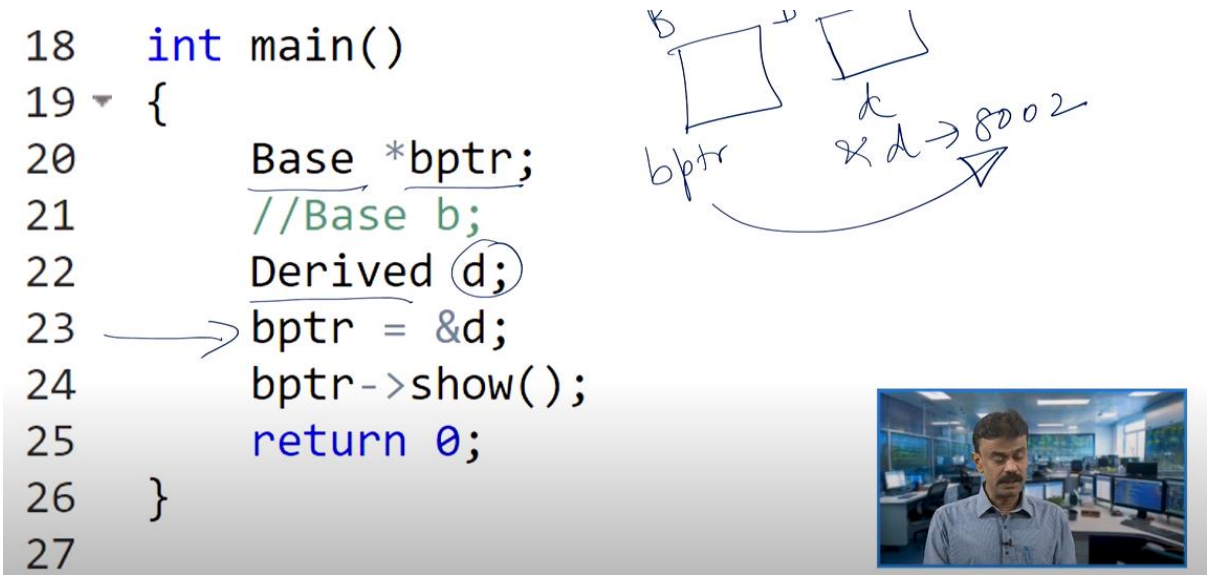
```
18  int main()
19  {
20      Base *bptr;
21      //Base b;
22      Derived d;
23      bptr = &d;
24      bptr->show();
25      return 0;
26  }
27
```

So class derived, your public is base. So I mean your base class is publicly inherited. And then here you have This virtual function, in fact, a pure virtual function, you have show. So this is being defined over here.

Just recall your definition of abstract, right, in the paper. So you have just said, I am going to prove this theorem, right? So later, in the explanation part, right, you are going to prove the theorem. So similarly, you are seeing the show function here. So, in the show function you are writing, see how the derived class is derived from the base class, right?

So, this is what you are writing. So, how do I call this, right? So, let us assume that you have a base pointer, right? So, I cannot put it like this. So, I have a base pointer.

So, I cannot directly instantiate base B because it contains the abstract, right? So, right now there is no, okay, fine. So, here in this case, let us consider the base pointer, right? bptr is the base pointer and you have the derived class d, alright. You have an object under the derived class.



You have a base pointer under Base, right. So now you have bptr, it is a base pointer, and in the derived you have, so this is base and this is derived, you have d, alright. So now, what is address d? Let us assume address d is 8002. Let us assume the address is 8002.

And bptr is pointing to this address. That is what line number 23 is. bptr is pointing to that address d. Right. So now, bptr arrow show. Right.

So this will call. Right. This will call the Derived class. Obviously, it is a pure virtual function. So this will call this particular function, and 'Derived class is derived from the base class' will be printed when you are calling this function when bptr is invoking.

So, you can see the syntax. So, I have base star bptr. So, bptr arrow show. So, this bptr will invoke show because now bptr is pointing to the address of the derived class object. Right.

Abstract Class in JAVA

- A class that is declared with abstract keyword, is known as abstract class in java.
- It can have abstract and non-abstract methods (method with body).
- Before learning java abstract class, let us understand the abstraction in java first.
- **Abstraction** is a process of hiding the implementation details and showing only functionality to the user.

So, therefore, the derived class function will be called. So, the derived class is derived from the base class will be printed. Right. So, this is what exactly is happening. So, you can see the output.

Right. So, you can see the output derived class is derived from the base class. I hope it is clear. Right. So, now the same concept, the pure virtual function concept with the help of the pure virtual function concept in C++.

Now, we will see what you mean by an abstract class in Java. Right? So, assume that you are declaring a class. Right? With the abstract keyword.

Right? So, when you have the abstract keyword and then you define a class, you call this an abstract class. Right? So, in the case of an abstract class, you can have both abstract and non-abstract methods. Right?

So, inside the abstract class, you can have methods, which are nothing but functions. Right? Member functions in C++. So here, in the case of an abstract class, you can have both abstract and non-abstract methods, right? So let us understand, right?

Ways to achieve Abstraction

- There are two ways to achieve abstraction in java
  - Abstract class (0 to 100%)
  - Interface (100%)

So let us understand. In fact, we partially know what you mean by an abstract class in the case of a pure virtual function. So in the case of Java, right? So it's almost the same or exactly the same. So let us understand abstraction again in the case of Java.

So, what are we doing? We are going to hide the implementation details. And then we show only the functionality to the user. So, you call this abstraction. So, in the abstract class, I am going to write only the functionality.

Implementation details, I can hide. For example, if you put an abstract function. Right, an abstract method, so that means I am not going to define it. It is exactly like a pure virtual function in C++. That is why I am relating both, right? So whenever you are studying OOP, I mean, it is good to relate both C++ and Java. So, an abstract method, when you are writing it, is exactly like a pure virtual function in C++. So that means we are hiding the implementation details and we are going to show only the functionality to the user. So what are the ways you can achieve abstraction? We can achieve it, right? In the case of an abstract class, we can achieve 0 to 100 percent, right? The reason is, it may contain both abstract methods and non-abstract methods. So inside the abstract class, I can have a non-abstract method.

So, for example, assume that you are using two abstract methods and two non-abstract methods. So, that means you have achieved 50% abstraction. Whereas, we are going to study one more interesting concept: interface. So, in an interface, it is 100% exactly like the pure virtual function that you are defining. So, in the interface, all are abstract concepts.

Abstract class in Java

- A class that is declared as abstract is known as **abstract class**.
- It needs to be extended and its method implemented.
- It cannot be instantiated.

- Abstract class Syntax:

```
abstract class A{ }
```

Methods only, you have to define only abstract methods inside the interface, right? So, therefore, we had a problem with multiple inheritance, right? The multiple inheritance concept we don't have in terms of class, so inheritance will take care of us. So, abstraction, see, for example, Assume that you are giving your clothes to the laundry, right? You are giving them to a particular person, right, for the laundry, right? Let us assume, I mean, you are giving them at 9 a.m. and then you are expecting them back at 12 noon, right?

So, the person will deliver them at 12 noon. So, now the question is, who has done the work? And, in fact, we are not bothering about who has done the work, right? So, between 9 to 12, what is happening? So, you are giving the clothes and you are getting the, right, ironed clothes back.

And in between, what is happening. We don't care. Or. Right. That is not our problem.

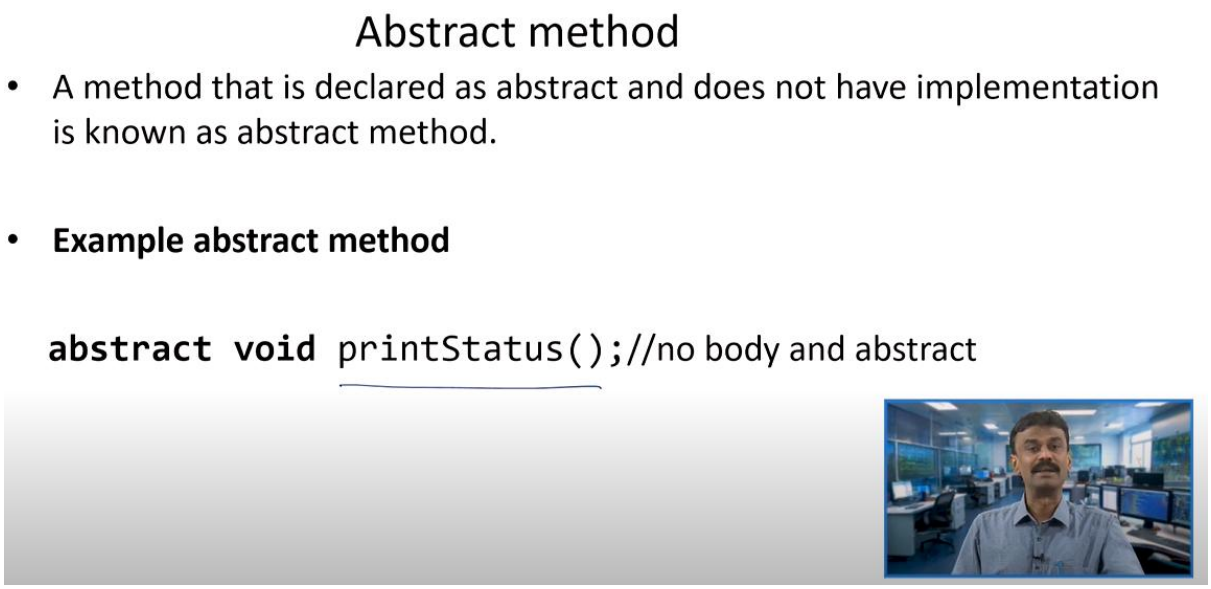
So you call that abstraction. Right. So. No need to explain. You are not asking for any explanation.

Right. Who has done the work. Right. So this is exactly happening. So let us say in the abstract class.

So when I am defining the abstract class. I use the keyword called abstract. Right. And then. So inside the abstract class.

I can have the abstract method and non-abstract method. Right? So what you have to do is, suppose you have the methods. The methods may be abstract or non-abstract. It has to be extended.

Right? That means you should have a derived class. Right? And those methods have to be implemented. And you cannot, when you have an abstract class, instantiate any object.



Right, you cannot instantiate any object, so the syntax goes like this: abstract, and then you have a class A. You are defining a class. So, let us have the definition of an abstract method, right? So, suppose you have a method, right? For example, I have a method, print status, so this I want to have as an abstract method. So, what you have to do is you use the keyword abstract, right? So, you use the keyword abstract, and then it should not have any implementation, right?

It should not have any definition, right? So, such a method is called an abstract method. So, it is also called no body and abstract, right? So, it will be like this. So, you do not have any definition.

You are just declaring it as a function, right? Declaring it as a method. So, print status, which is a method. Return type is void, and which is nothing but the abstract keyword that you are using. So, this is an abstract method, right?

Example of abstract class that has abstract method

```
1  abstract class Bike{
2    abstract void run();
3  }
4
5  class Honda4 extends Bike{
6  void run(){System.out.println("running safely..");}
7
8  public static void main(String args[]){
9   Bike obj = new Honda4();
10  obj.run();
11 }
12 }
```

So, this is the syntax for an abstract method. So, we will see an example. So, let us consider the abstract class Bike, right? So, you have an abstract class Bike and one function, right? So, assume that you have one function, right?

In fact, the method, an abstract method, right? So, you have an abstract method. So, now let us have your derived class, or you call it a subclass in Java, Honda4, which is extending the base class Bike or the superclass Bike, right? The superclass is nothing but the abstract class, right? In the abstract class, you have the same run method.

So, which is giving System.out.println running safely, right? System.out.println running safely, right? So, now under Honda4, all right, under Honda4, you have the method, right, the main method. So, therefore, you call Honda4 as the, what is Honda4? Honda4 is nothing but a driver class, right, so which is inheriting the base class Bike, which is nothing but the abstract class.

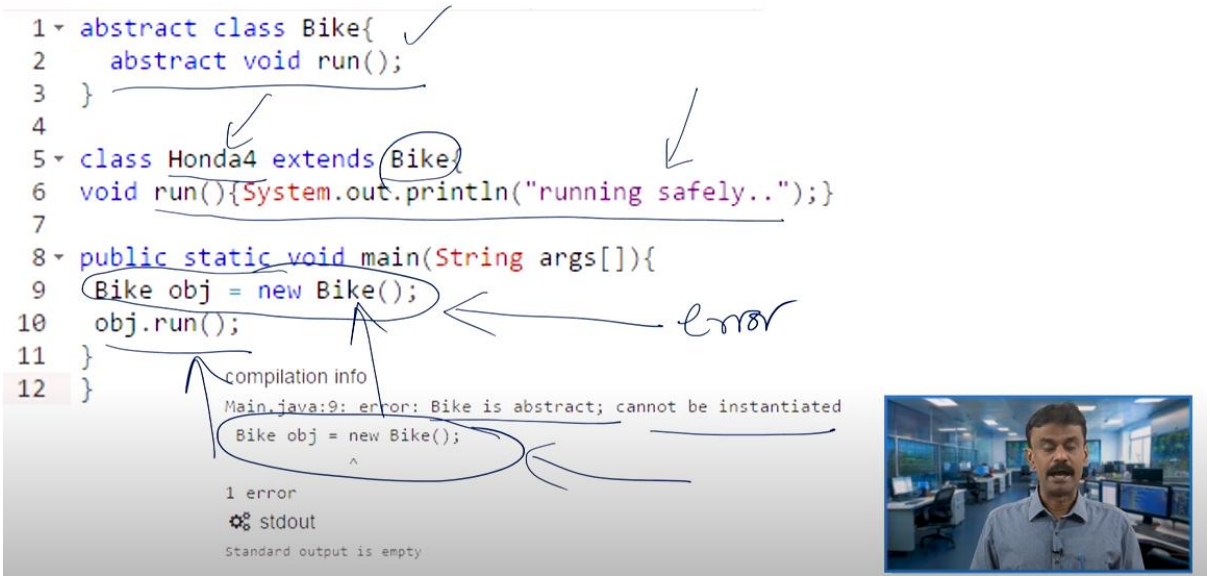
So now you are creating an object obj under Bike, right? You are creating an object obj under Bike. So you are allocating memory and your constructor is Honda4. So no problem, right? Your constructor is Honda4, right? Now your object obj, right?

```
1  abstract class Bike{
2    abstract void run();
3  }
4
5  class Honda4 extends Bike{
6  void run(){System.out.println("running safely..");}
7
8  public static void main(String args[]){
9   Bike obj = new Bike();
10   obj.run();
11  }
12  }
```

Your object obj is invoking the run function, right? So obviously this will be overridden. Right, obviously this will be overridden and this function will be executed, right? The run function will be executed and we can see that you are trying to print 'running safely,' so 'running safely' will be printed, right. So you can try this code, right? 'Running safely,' whatever we are expecting, 'running safely' will be printed. I hope it is clear, right? So this is one of the examples of an abstract class. And then you also have the abstract method, right?

So you also have the abstract method. So now let us consider a similar program, right? All are the same. Here, right, in line number 9, right, all are the same. You have an abstract class Bike, right?

And then you have an abstract method run, right? And then Honda4 is extending Bike, correct? Honda4 is the subclass, right? Which is extending Bike, and then you have this run method, which is printing 'run safely.' All are the same, right? You have Honda4 as a derived class, and in fact, it is a driver class. So now, if you look at this, you are instantiating the obj object under the Bike class, right?

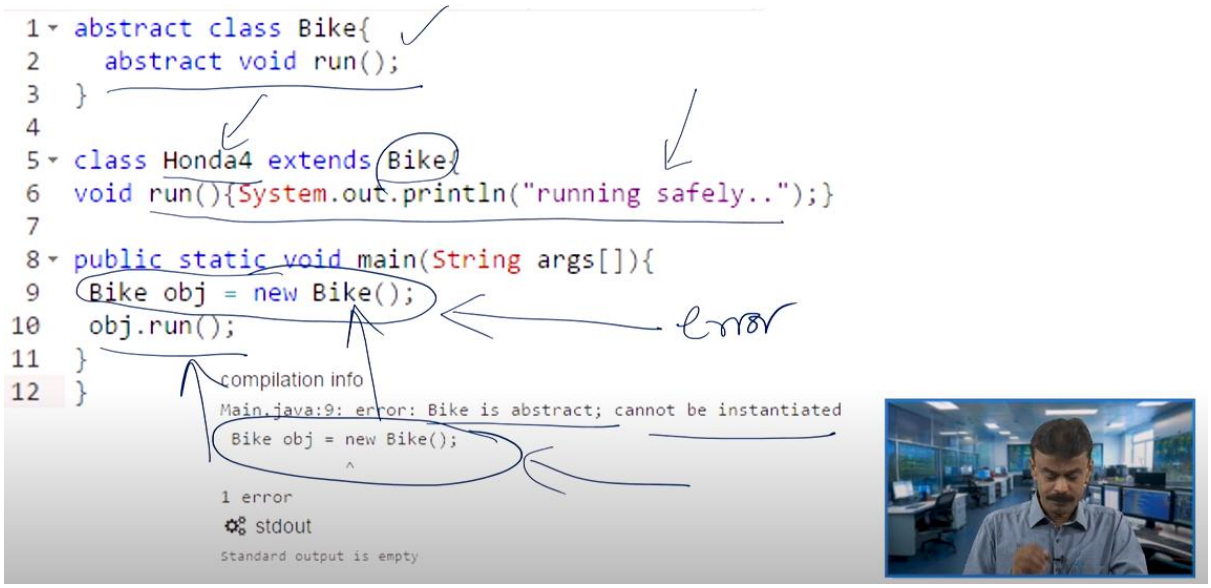


And then your constructor is Bike, new Bike, right? So, what will happen? Yes. So, I already said that when you are creating an object, right? When you are instantiating an object of an abstract class, which is not possible, right?

So, this is not possible. Which will give an error. In fact, line number 10, we are also trying to do, right? We are also trying to, in fact, the object obj, which is invoking run, right? So, line number 9 itself will throw an error, right?

So, it will give an error. The reason is, you cannot instantiate an object under an abstract class, right? So, when you try to do this, you will get an error, right? So, you can see the error message. Bike is abstract.

What is the error message you are getting? Bike is abstract. And it cannot be instantiated. So, this is your error. Exactly as I said.



And then the compiler is also telling. This is an error. So, this is an example where you cannot instantiate an error. An object under an abstract class. Right.

I hope it is very clear. Right. So, how do you? I mean, take care. In the previous program, it is working.

Correct. How is this happening? So, you can see, I mean here, when you are creating an object obj under the class Bike, so you have the constructor Honda4, right? So, the reference and the object you are changing, right? So, when your constructor is Honda4, right?

So, it is creating an object obj, right? You are instantiating an object obj, the reference is. Class Honda4. Right. The reference is class Honda4.

So when it has the reference, it is class Honda4. So the next line, when obj invokes the method run. I mean, it is working fine. Right. So that is why you are getting the output.

Whereas I am trying to change here. Right. Honda4, the constructor I use, is Bike itself. So then, in that case. So you cannot have an object.

And the reference is also Bike. Right. So, then since Bike is the abstract class. So, we cannot do it. In fact, this will throw an error.

We cannot do it in the sense that obj cannot invoke the function run. Right. So, in fact, line number 9 itself is an error. Right. So, from this program, it is suggesting that.

So, you cannot instantiate an object under the abstract class. Right. So, therefore, you are getting an error. So, In this lecture, we had seen virtual functions and pure virtual functions in C++.

So, you understood. So, when you have the virtual function, you will have it in the base class. But the same function should be defined in the derived class also. Whereas in the pure virtual function, you are putting equal to 0. You are not giving any definition in the base class.

But it has to be defined. Defined in the derived class, right? So, which you call it as a pure virtual function. And then we talked about the abstract method and abstract class in Java, right? So, the same concept, or I can say the similar concept, whatever you have studied in C++. So here, you have studied it as the abstract class, right? In Java. So, the abstract class, right now, we achieved 0 to 100% abstraction because we can use the usual method also. Suppose, assume that in this code, I do not put abstract. Assume that it is only void run.

So, this is the usual function, right? Usual method, right? So, the abstract class, we can use the usual method. But when you are going to achieve 100% abstraction, In the next class, we are going to see the interface, right?

So, using the interface, we can achieve 100% abstraction. Also, that will give you the solution for your multiple inheritance. Thank you.