

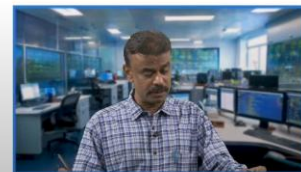
# FUNDAMENTALS OF OBJECT ORIENTED PROGRAMMING

## Lecture12

### Lecture 12: Inheritance - Single Inheritance

```
1  #include <iostream>
2  using namespace std;
3
4  // Base class
5  class Animal {
6  public:
7      void eat() {
8          cout << "Animal is eating." << endl;
9      }
10 };
11
```

#### Single Inheritance, Example 2



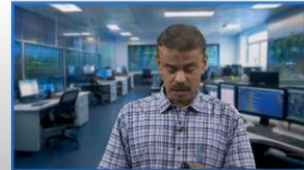
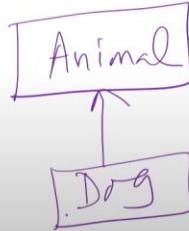
Welcome to Lecture 12. Today we are going to see the further part of inheritance. So in the last class, we saw an example of single or simple inheritance. So in continuation, in today's lecture, we are going to see the second example. So let us consider class Animal, right? Let us consider class Animal.

So under this, you have one member function called void eat, right? So this is one member function. So you have void eat. So void eat You have one cout statement.

```

12 // Derived class inheriting from the base class
13 class Dog : public Animal {
14 public:
15     void bark() {
16         cout << "Dog is barking." << endl;
17     }
18 };
19

```



The cout statement is Animal is eating. Fine. So, now you have class Dog. Right. Which is publicly inheriting Animal.

Right. So, Animal is a base class. Right. Animal is a base class. And you have class Dog, which is publicly inheriting.

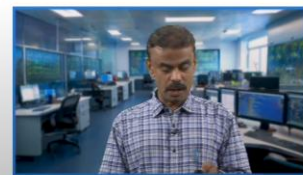
Right. So, that means a Dog can use the properties of the Animal. Right. So you have one member function.

So that can be utilized by the class Dog. So under Dog, you have one member function called bark. So here you have Dog is barking.

```

20 int main() {
21     // Creating an object of the derived class
22     Dog myDog;
23
24     // Accessing base class method using the object of the derived class
25     myDog.eat(); // Accessing the eat method from the Animal base class
26     myDog.bark(); // Accessing the method from the Dog derived class
27
28     return 0;
29 }
30

```



So now we can see. Right. So we are creating an object named myDog. Under the class Dog. Correct.

So now myDog is invoking. The function. Eat, the member function eat. So, what is happening? Eat, right?

So, which is available? So, here you go, here you have the eat member function. So, here you have the bark member, right? So, we have created an object named myDog, under Dog, right? So, the name of the object is myDog.

So, this object is invoking eat. Yes, it is possible, right? Because Dog is publicly inheriting Animal. Right. So, you can see.

So, myDog is an object under the class Dog. Right. And myDog is invoking eat. So, when it is invoking eat. So, here you go.

Animal is eating will be first getting printed. Right. Animal is eating will get printed. Now, myDog is invoking bark. Right.

```
12 // Derived class inheriting from the base class
13 class Dog : public Animal {
14 public:
15     void bark() {
16         cout << "Dog is barking." << endl;
17     }
18 };
19
```

My dog is invoking bark. So here you go. So bark, you have a dog barking. Alright. So here you go, the dog is barking.

So this member function will be invoked. And so first, 'Animal is eating.' will be printed. And in the second case, 'Dog is barking.' will be printed. So if I run the code, you will get the output right. This is what we are expecting.

So this output will get first, you will get 'Animal is eating,' and second, 'Dog is barking.' So this is another example of single or simple inheritance.

### One more example on Single Inheritance

```
1  #include <iostream>
2  using namespace std;
3  class A
4  {
5      int a = 4;
6      int b = 5;
7      public:
8      int mul()
9      {
10         int c = a*b;
11         return c;
12     }
13 };
14
```

A



Now you have one more example, right. So we will see the last example of single inheritance. Suppose assume that I have a class A, right.



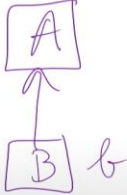
Class A, which you call the base class. Class A contains two data members. One is a equals 4, and the other is b equals 5. It also has one member function. So, the member function you are doing here.

First, we are seeing the member function. So, this is the multiplication. c equals a multiplied by b, and then it returns. So, the multiplication function. Right?

You have only one function. Now, if you look, class B is privately inheriting A. This is the first time you are seeing this. Right? Class B is privately inheriting A. So, A is the base class. You have private inheritance.

B is the derived class. Right? So, here you have class B displayed. So, in display, you are calling the multiplication function, right? So, in display, you are calling the multiplication function, and the multiplication is public, right?

```
15 class B : private A
16 {
17     public:
18     void display()
19     {
20         int result = mul();
21         std::cout << "Multiplication of a and b is : "<< result << std::endl;
22     }
23 };
24 int main()
25 {
26     B b;
27     b.display();
28     return 0;
29 }
30 }
```



Multiplication is publicly available. So, class A multiplication is public, even though it is privately inherited. So, we are going to see how this is going to be accessed, right? So, result multiplication and cout multiplication of a and b. Right.

cout is multiplication of a and b. So, now in the main program. So, you have. Right. So, multiplication will be called over here. That will be stored in result.

Right. Multiplication will be called over here. And that will be stored in the result. So now. You have an object, small b. Right.

So you have an object, small b, whose class is capital B. So under B, I have an object b. So now b is invoking display. Yes, it can because display is the function, a member function, and inside this function, you have multiplication. So it is privately inheriting, which means the public will become private. So the public member function, int multiplication, is a public member function.

So that will be private to class B. All right. So that will be private to class B. So now here you have multiplication. So you are calling this function. So it can be called.

All right. Because class B is inheriting class A privately. And you can see that the multiplication function c is equal to a star b. Right. Your a is 4. b is 5.

So I can get the output 20. 4 into 5 is 20. Right. So that will be stored in the result. So, now the multiplication of a and b is, you can see the result, right.

So, the multiplication of a and b, you can see the result. So, the result is 20. So, when b is invoking display, you will get the output: the multiplication of a and b is 20, right. So, privately, when it is inheriting, the public is becoming, right? It can be accessed. So, it is becoming private.

So now, in this example, the previous example that we had seen, class A is privately inherited. All right. So class A is privately inherited. So when it is privately inherited, if there is any, for example, A and B, we cannot access. All right.

So that means, suppose I put, let us say b dot, let us say result or something. All right. Or anything outside the main function when you are trying to access. So it cannot be accessed. So for example, here the multiplication function you have.

All right. So this class A cannot be accessed by the object of class B. All right. And the multiplication function, suppose you want to access. All right. So assume that I have an object b. All right.

I have an object b. So you may ask the question, why don't you put b dot multiplication? All right. So, b dot you can try, you will get an error, all right. So, suppose I am writing here b dot multiplication, right. So, this will give an error, all right.

So, because this is becoming private, all right. So, you are trying to access outside the class, then you are getting an error, all right. So, b dot multiplication, suppose I put b dot multiplication function Of class A. So cannot be accessed by the object of class B. So b dot multiplication if I put. So, this will give an error.

So, therefore, we can conclude that it can only be accessed by the member function of class B, right? What is the member function of class B? So, b dot display. So, that is a member function of class B. And then inside your class B, you are calling this multiplication. Yes, it is now perfectly valid.

And you can see the output; we are getting the multiplication of a and b as 20. I hope it is clear to everyone. Now, similarly, In Java, we have a syntax like this, right? So, what we have done in the case of C++ is: class B colon private A, right?

## Syntax of Java Inheritance

```
class Subclass-name extends Superclass-name
{
    //methods and fields
}
```

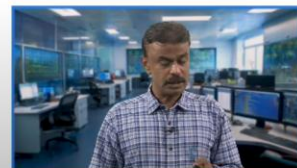


So, class B is the class, and private A is the class. So, here in Java, we have to use the keyword extends, right? So, class, name of the subclass, right? Extends a superclass. So, class B extends class A.

All right. So, class B and class A are names of the classes. And then you have methods and fields. All right. So, you can use the keyword extends.

So, in fact, we have seen an example. All right. Maybe one more example we'll see. So let us assume we have a class Calculation. All right.

```
1 class Calculation{
2     int z;
3     public void addition(int x, int y){
4         z=x+y;
5         System.out.println("The sum of the given numbers:"+z);
6     }
7     public void Substraction(int x,int y){
8         z=x-y;
9         System.out.println("The difference between the given numbers:"+z);
10    }
11 }
12 }
13 }
```





So we have a class Calculation. You have a member data or data member, and you have a method addition. So the addition is adding two numbers. You're passing x and y. So, these are the arguments here.

So, z is equal to x plus y. So, now you have System.out.println, 'The addition of two numbers is nothing but z.' And similarly, subtraction is another function called subtraction, and you have two arguments: z is equal to x minus y. So, System.out.println. So, the difference between the given numbers So, here it is z, right? The same z variable we are using. It is local to subtraction and local to addition, right?


The class is over here. So, now how do you inherit? I have another class called My\_Calculation. So, My\_Calculation extends Calculation, right? So, Calculation is a base class and you can call it a superclass, and My\_Calculation is a subclass.

Right? In Java. Whereas we call a base class in C++ and a derived class. So here you have a superclass and a subclass. Superclass, child class, or parent class, child class.

```
14 public class My_Calculation extends Calculation{
15
16     public void multiplication(int x, int y){
17         z=x*y;
18         System.out.println("The product of the given numbers:"+z);
19     }
20     public static void main(String args[]){
21         int a=20, b=10;
22         My_Calculation demo = new My_Calculation();
23         demo.addition(a, b);
24         demo.Substraction(a, b);
25         demo.multiplication(a, b);
26     }
27 }
28
29 }
```

Terminal

```
sh-4.3$ javac My_Calculation.java
sh-4.3$ java My_Calculation
The sum of the given numbers:30
The difference between the given numbers:10
The product of the given numbers:200
sh-4.3$
```



Okay? So now you have under My\_Calculation, you have multiplication. Right? So here I am defining multiplication. So z is equal to x star y and System.out.println.



So, you will have the product of given numbers as plus z, right? The product of. So, z is equal to x star y. So, a is 20, b is equal to 20. So, now I have the object demo, right? Your class is my underscore calculation, which is a subclass, and here you have a new operator and this is your constructor, the default constructor. Correct. So, the default constructor here has no role.

Fine. So, then your demo, the object demo, is invoking addition. So, a comma b means you are passing 20 and 10. So, in passing 20 and 10, you will get 20 plus 10, which is 30. All right.

So, this is calling this function, addition. Z is equal to x plus y. Right. So, 20 and 10, 30. So, that will be added, and here you have to get 30 as the output. And 20 minus 10, when you are calling subtraction, subtraction is 20 minus 10, I should get 10 as the output.

And the last one is multiplication, which is available. So, the first two member functions, or you can call them methods, 23 and 24, are available in the superclass. And multiplication is available in the subclass. So, in the subclass, 20 multiplied by 10 should give you 200. All right.

So, when I run the code, I should get this output. All right. The sum of the given numbers is 30. All right. The difference is 10.

And you can see the multiplication is 200. And here we are getting 200. So, in the terminal, when you run this code, you have `javac My_Calculation.java`, and for running the code, the next line will give you the sum of the given numbers, the subtraction of the given numbers, and the multiplication of the given numbers. So, this is single inheritance. So, under single inheritance in Java, you have the super keyword.

## super keyword in JAVA

```
1. class Vehicle{
2.     int speed=50;
3. }
4. class Bike3 extends Vehicle{
5.     int speed=100;
6.     void display(){
7.         System.out.println(speed); //will print speed of Bike
8.     }
9.     public static void main(String args[]){
10.        Bike3 b=new Bike3();
11.        b.display();
12.    }
13. }
```

stdout

100



So, we talked about superclass and subclass. Correct? Correct. So, let us consider this particular example, right? I am not using the super keyword here, but suppose I have this, right?

So, class Vehicle speed is equal to 50, right? Class Vehicle speed is equal to 50, and Bike3 extends Vehicle. So, Bike3 is a subclass that is extending Vehicle. So, assume that I am using the same data, right? Same data member, same name.

Speed is equal to 100 here. Right. So, slowly we are going to polymorphism. Right. So now you have one member function called void display.

So, void display what it is doing. So, it is trying to print speed. Right. It is trying to print speed. So, you are creating an object b. In the main, you are creating an object b whose class is Bike3.


So, Bike3 is equal to you have a new operator and then you have a constructor, correct? So, this object b is invoking display, right? So, can you think what will be the output? So, right, when it is calling the function display, right, it is trying to print System.out.println speed, right? So, speed which is taking in the Bike3 class.

Right. In the Bike3 class, it is taking the speed and it is being printed over here. Right. So now you may ask the question, you want to print, let us say, speed, which is equal to 50. Right.

So, what will you do? In that case, we are going to use the keyword called super. Right. So, here you go. The same program.

```
1. class Vehicle{
2.     int speed=50;
3. }
4.
5. class Bike4 extends Vehicle{
6.     int speed=100;
7.
8.     void display(){
9.         System.out.println(super.speed); //will print speed of Vehicle now
10.    }
11.    public static void main(String args[]){
12.        Bike4 b=new Bike4();
13.        b.display();
14.    }
15. }
16. }
```

stdout  
50



Right. So, I just change the name to Bike4. Vehicle is the superclass. Bike4 is a subclass extending Vehicle. Speed is equal to 100.

Right. You have void display. First time, you are seeing super dot speed. Right. Super dot speed.

So public static void main. So main is over here. Object b. Right. Whose class is Bike4? Dynamically allocating memory with a constructor.

Default constructor Bike4. Now b is invoking display. Now b is invoking display. So when it is invoking display, System.out.println super.speed. So when you have super.speed in the previous program, what have we done?

We have simply printed speed. So speed equal to 100 was printing. So now I want to print this. So in that case, use super.speed so you can see the output. So you will get 50 as the output.

So that is the advantage of using super. That means if you want to access the same member data and if you want to access the super class right in Java, you have the keyword called super, right? Super and then the dot operator. So you have speed, right? Super dot speed, which is printing 50. That means in the base

class, or you can call it the super class, if you want to print. The member data. A data member. So this is the way to print. So now.

## super keyword in java

- The **super** keyword in java is a reference variable that is used to refer immediate parent class object.
- Whenever you create the instance of subclass, an instance of parent class is created implicitly i.e. referred by super reference variable.



First time we are seeing the word super. Right. So this keyword. Which is nothing but a reference variable. Right.

So in Java. This is a reference variable. So which is being used to refer. Immediate parent class of the object. Immediate parent class.

Right. So sooner or later, we are going to talk about the multi-level. And then we will see several examples. So which one will be called? Right?

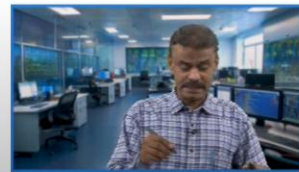
So when I use the super keyword, the super keyword is nothing but a reference variable. Right? So that is used to refer to the immediate parent class object. Right? Super.speed.

Right? So whenever, suppose we are creating an instance of a subclass. Right? So here we have created an instance of subclass b. Object b under Bike4. So that is nothing but an instance of a subclass.

Right. So, now an instance of a parent class will be created implicitly. So, which will be referred to by the super reference variable, right. So, which will be referred to by the super reference variable. So, now we can use super as an instance variable; we use super dot right speed.

## Usage of JAVA super Keyword


- super is used to refer immediate parent class instance variable.
- super() is used to invoke immediate parent class constructor.
- super is used to invoke immediate parent class method.



So, We know that the immediate parent class instance variable right that is used to refer to the immediate parent class instance variable. And this is nothing but the constructor right; the immediate parent class constructor will be invoked. So, whenever you are using super or this, we know that in the case of the constructor, we use the same name. So now, when I use this, this is used to invoke the immediate parent class constructor, and similarly, the method right. So when I want to invoke the immediate parent class method, we can use the super keyword. So now we will see this example: so class Vehicle. So under class Vehicle, I have one member function.

```
1. class Vehicle{
2.     Vehicle(){System.out.println("Vehicle is created");}
3. }
4.
5. class Bike5 extends Vehicle{
6.     Bike5(){
7.         super();//will invoke parent class constructor
8.         System.out.println("Bike is created");
9.     }
10.    public static void main(String args[]){
11.        Bike5 b=new Bike5();
12.    }
13. }
14. }
```

stdout  
Vehicle is created  
Bike is created



Call. Right. In fact, that is a special member function, which is a constructor. Default constructor. System.out.println vehicle is created.

And then I have class Bike5. Right. Which is extending Vehicle. So, here you have the constructor. Right.

So, here you have the constructor for Bike5. That means a subclass. Correct. And then you use the keyword called super. So, the keyword super will invoke the parent class constructor.

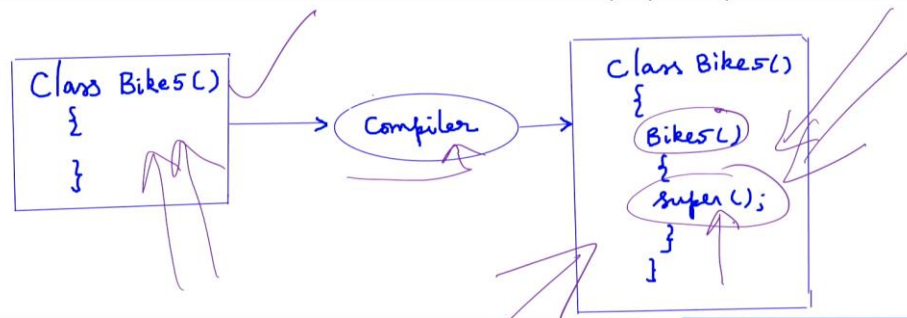
Right. So now we have one print statement for Bike5 under the Bike5 constructor. So, which is 'Bike is created.' Right. So now you have void main.

B is an object. Small b is an object under Bike5. You are dynamically creating a space for the small b object, and Bike5 is a constructor, right. So, now what will happen when you are creating an object b, right, under Bike5.

So, this will go to line number 6, right. This constructor will be invoked. So, here you have super. The keyword super will invoke the base class constructor or the superclass constructor. So, the first print you are getting is 'Vehicle is created'.

The first print statement you will get is 'Vehicle is created'. So, next, it is going to line number 8. So, on line number 8, you have 'Bike is created'. So, these two will be printed. These two will be printed.

- `super()` is added in each class constructor automatically by compiler.



So when I run the code, I can see that 'vehicle is created' is being printed, and the second one is 'bike is created'. So when I use the keyword 'super', we can see in the case of instance or the member data, and here you go: when I use line number 7, the base class constructor—I mean to say, the immediate parent class constructor—will be invoked. So therefore you are getting 'vehicle is created' first, so this is giving the first input, that is what the first input is over here. And automatically, the second input. Right.

`System.out.println` is there. Since you are creating under `Bike5`. So this will be printed. And here you are getting 'bike is created'. Right.

I hope it is clear. So now we have used 'super' for constructor as well. So now, according to the compiler, even if you leave it like this. So let us assume class `Bike5`. The previous example.

Right? According to the compiler, it will be like this. Right? `Bike5`, you have `Bike5`, which is the constructor. Right?

So, under the constructor, it is assumed that you have a `super` keyword. It is assumed. Right? So, this is what exactly is happening. So, the `super` is added in every class constructor automatically by the compiler.

So, this is how exactly, even though it is, if you are writing like this, it is equivalent to this. According to the compiler, it is equivalent to this. That is the meaning. Okay.



So, even if I am writing on the left-hand side, as far as the compiler is concerned, it will be like this. So, that means class Bike5, which is a class. Under this, you have the Bike5 constructor. So, inside the constructor, the first line is super. So, the super keyword means the parent class will be called.

We will see an example. All right. So, here you go. All right. Line number 7, exactly like the last program.

Line number 7, I commented. Line number 7, I commented. All right. So, class Vehicle, system.out.println vehicle is created. All right.

So, Bike5 is extending Vehicle. So, here you have the constructor Bike5. So, the superclass I commented, and then the bike is created. Alright. So, now you have an object b under Bike5.

So, when you are creating an object Bike5 b, this will call this. Alright. So, we will go back to the previous diagram. So, it is like Bike5. It has gone to Bike5.

So, when it is calling the default constructor, what is the first line? The first line is like super. Super will be there. Alright. So, when super is being invoked, alright, it will call the parent class constructor.

Alright. So it is calling the parent class constructor. So when it is calling the parent class constructor, 'Vehicle is created' will be printed. So this one will be printed automatically. And then 'Bike is created' will be printed.

So that is the meaning of this. Right. So that is the meaning of the previous diagram. You can see the output. You can try this code.

Very interesting code. All right. I have not given super, but as far as the compiler is concerned, you have this super keyword. Even though you have not written it, it is as if you have the super keyword. So let us consider another example.

## Another example

```
1. class Vehicle{
2.     Vehicle(){System.out.println("Vehicle is created");}
3. }
4.
5. class Bike6 extends Vehicle{
6.     int speed;
7.     Bike6(int speed){
8.         this.speed=speed;
9.         System.out.println(speed);
10.    }
11.    public static void main(String args[]){
12.        Bike6 b=new Bike6(10);
13.    }
14. }
```



So you have a base class or super class Vehicle. So you have a base class or super class Vehicle. Under this, you have a constructor: `System.out.println` 'Vehicle is created.' So now I have another class, Bike6. So which is extending Vehicle?

So that means Vehicle is a superclass and Bike6 is a subclass. Vehicle is a superclass and Bike6 is a subclass. So I have `int speed`. Under this, I have a one-argument constructor, a parameterized constructor, Bike6. You are passing speed. So now you have, for the first time, we are seeing this pointer: `this.speed` equal to speed.

So `this.speed` equal to speed, and then I am printing speed. So look at the main program. Look at the main program. So `public static void main`. You have object b. You have object b, whose class is Bike6.

You have an object b whose class is Bike6. So here you have a constructor, and then you are passing the value 10. Alright. So here you have 10 and speed, right? It is a reference variable. So that means your speed will be 10.

## super can be used to invoke parent class method

```
3. class Person{
4. void message(){System.out.println("welcome");}
5. }
6.
7. class Student16 extends Person{
8. void message(){System.out.println("welcome to java");}
9.
10. void display(){
11. message();//will invoke current class message() method
12. super.message();//will invoke parent class message() method
13. }
14.
15. public static void main(String args[]){
16. Student16 s=new Student16();
17. s.display();
18. }
19. }
```




Speed will be initialized to 10. Alright. You can check this. Remove line number 8 and simply try to print `System.out.println(speed)`. You will get 0.

Right. Whereas, if you use this, right, with a reference, this is a pointer. So when you are using this pointer, write dot the variable name that you are using, the member data that you are using inside the class. So then whatever you are passing, 10 will be printed. So 10 will be printed.

So, when I run the code, you will get the output 'vehicle is created.' Yes, of course, when you are creating an object 'b,' it is assumed that you have the super constructor, and the super constructor will be calling this, right. So, therefore, 'vehicle is created' will be printed, and since you are passing 10, so 10 will be printed. So, now, 'super' can be used to invoke the parent class method.

```
3. class Person{
4.     void message(){System.out.println("welcome");}
5. }
6.
7. class Student16 extends Person{
8.     void message(){System.out.println("welcome to java");}
9.
10.    void display(){
11.        message(); //will invoke current class message() method
12.        super.message(); //will invoke parent class message() method
13.    }
14.
15.    public static void main(String args[]){
16.        Student16 s=new Student16();
17.        s.display();
18.    }
19. }
```

stdout  
welcome to java  
welcome



Right. So here you have, let us say, class 'Person.' All right. So you have class 'Person.' All right.

So 'Person' is the class. So you have void 'message.' So I am using 'super' for a method, a parent class method. All right. So you have void 'message' under this void 'message.'

You are printing System.out.println. Welcome. Right. System.out.println. Welcome.

All right. So class Student16. So you have another subclass. All right. So which is extending Person.

All right. So class Student16, which is extending Person. All right. So here you have. void message.

Correct. So. Yeah. One more message statement. Correct?

So, one more message statement. I mean to say one more message function. So, System.out.println, welcome to Java. So, the superclass also has the same name. The subclass also has the same name.

Alright? Now, slowly, we are going into polymorphism. Slowly. Not exactly, but slowly. You have void display.

Void display, here you are calling message. So, let us say which message will be called. On the next line, number 12, you have super dot message. Right? So line number 12, you have super dot message.

So message, when it is calling, let us say, when you are going to the main program, you have an object, yes. Right? You have an object, yes. And Student16 is the class. Right?

So this object is invoking display. Right. So this object is invoking display. So when it is invoking display, you have line number 11 message and line number 12, you have super dot message. Yeah, by the keyword super, it is very clear to you.

So, when the message is invoking, it is invoking the subclass. Right. So that means first, 'Welcome to Java' will be printed. 'Welcome to Java' will be printed first, and line number 12 is when it is invoking super.message. All right.

It is invoking the superclass. So, first this will be the output, and second this will be the output. All right. So, when I run the code, I have to get the output like this. So, 'Welcome to Java.'

Yes, as we expected, and 'Welcome' is the second output. So, here you go. You are getting this output. 'Welcome to Java.' That is what we are expecting: the first output and the second output.

Welcome. Okay. So, super can be used for the member data, and super can also be used for the constructor. And super, in this particular example, we have used for the parent class method, all right. So, there are cases where super may not be required, right. So, that we will see in the next class.

Thank you.