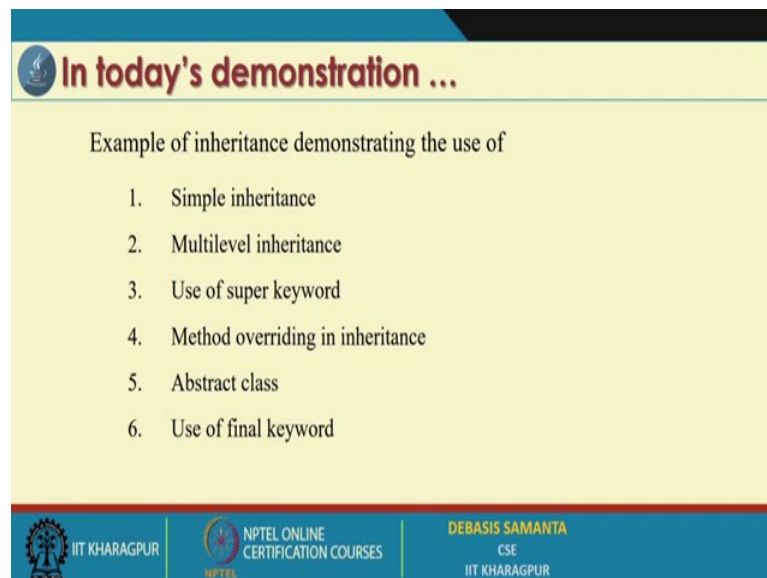


**Programming in Java**  
**Prof. Debasis Samanta**  
**Department of Computer Science Engineering**  
**Indian Institute of Technology, Kharagpur**

**Lecture – 14**  
**Demonstration – VI**

Let us have a quick demonstration on the topics that we have learned in the last module. Our last module was on was based on the inheritance in Java. So, today we are going to have a quick demo on the topic that we have learned in the last class.

(Refer Slide Time: 00:37)



**In today's demonstration ...**

Example of inheritance demonstrating the use of

1. Simple inheritance
2. Multilevel inheritance
3. Use of super keyword
4. Method overriding in inheritance
5. Abstract class
6. Use of final keyword

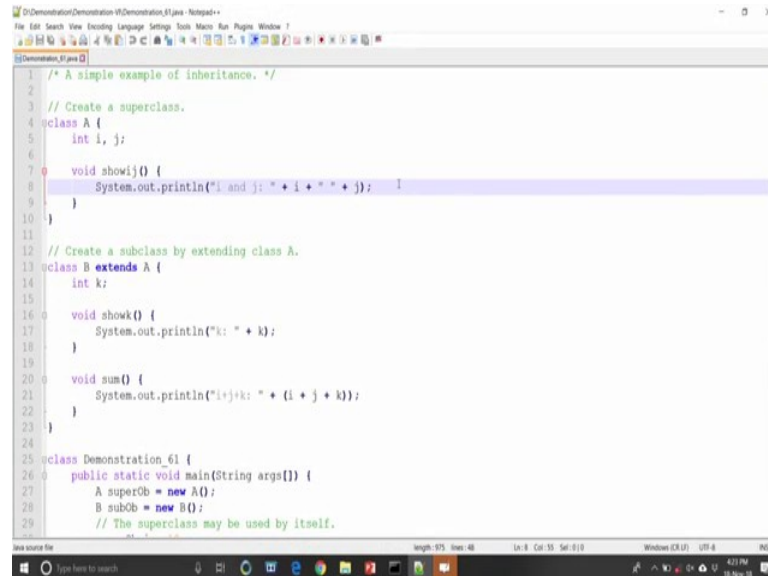
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | DEBASIS SAMANTA CSE IIT KHARAGPUR

So, to in today's demo we are going to cover the basic concept of inheritance, namely the simple inheritance and then we will discuss about multi-level inheritance, we have discussed about the super keyword which used to avoid the names space collision, the super keyword use of the super keyword to invoke the superclass constructor and then using the super keyword we can reference to some variable of the superclass. And then overriding is an important concept in any inheritance, so method overriding will be discussed.

There are two more keywords namely; abstract and then, the final keyword that is come on the way of inheritance. So, we will discuss this one. So, these are the topic that we are going to discuss in this demonstration lecture. Now let us have the first demonstration on

simple inheritance. So, how is a class can be needed from another class we can see it let us have the demo.

(Refer Slide Time: 01:45)

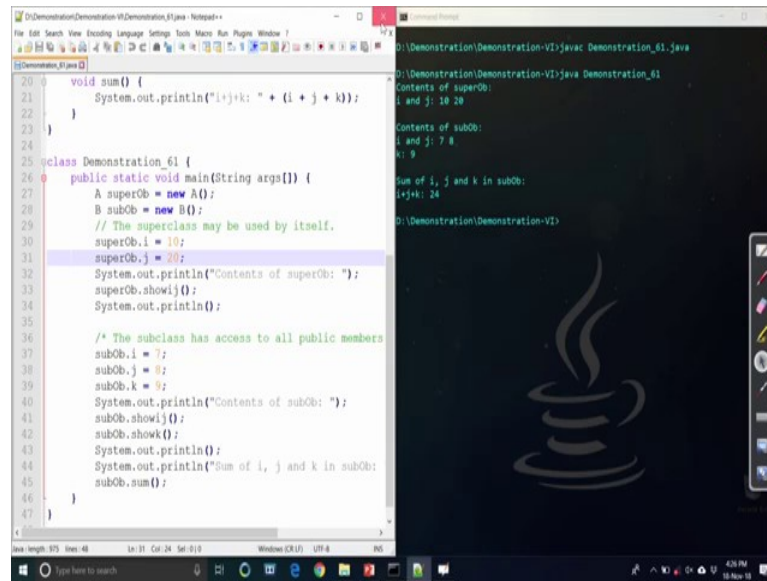


```
1  /* A simple example of inheritance. */
2
3  // Create a superclass.
4  class A {
5      int i, j;
6
7      void showij() {
8          System.out.println("i and j: " + i + " " + j);
9      }
10 }
11
12 // Create a subclass by extending class A.
13 class B extends A {
14     int k;
15
16     void showk() {
17         System.out.println("k: " + k);
18     }
19
20     void sum() {
21         System.out.println("i+j+k: " + (i + j + k));
22     }
23 }
24
25 class Demonstration_61 {
26     public static void main(String args[]) {
27         A superObj = new A();
28         B subObj = new B();
29         // The superclass may be used by itself.
```

Now, let us watch this program here in this program we see a class namely class A is declared. This is the superclass in this case, this class has 2 members i and j and it has 2 one method called the show i j printing the values of i j i and j in this class.

Now, in the next class the class B which basically inherits the class A. So, class B extends A and we can see that in class B we declare one variable called the k of type integer and then class B has its own methods. So, k printing the variable k in this class and also it has another method sum, it will print the value of i j, which are inherited from the superclass and the value of k, which is in its own variable.

(Refer Slide Time: 02:47)



```
20 void sum() {
21     System.out.println("i+j+k: " + (i + j + k));
22 }
23 }
24
25 class Demonstration_61 {
26     public static void main(String args[]) {
27         A superOb = new A();
28         B subOb = new B();
29         // The superclass may be used by itself.
30         superOb.i = 10;
31         superOb.j = 20;
32         System.out.println("Contents of superOb:");
33         superOb.showi();
34         System.out.println();
35
36         /* The subclass has access to all public members
37         subOb.i = 7;
38         subOb.j = 8;
39         subOb.k = 9;
40         System.out.println("Contents of subOb:");
41         subOb.showi();
42         subOb.showj();
43         System.out.println();
44         System.out.println("Sum of i, j and k in subOb:");
45         subOb.sum();
46     }
47 }
```

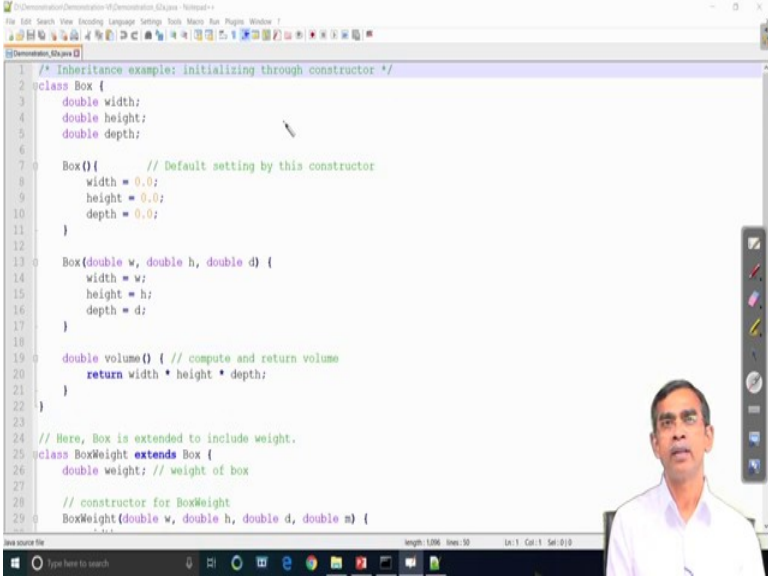
So, these are this is the inherited class B and as you know class B therefore, by virtue of inheritance have the access of both i and j which are declared in class A as a default access specifier as there in the same file they are readily accessible by virtue of inheritance as well as default access specification. Now let us have that main class the main class here we name as demonstration\_61 and in this class, we create 2 objects of type class A and class B namely super Ob and sub Ob.

So, 2 objects of 2 different classes are created and in the next statement we initialize the values of i and j in the superclass objects and then also print, I call the method of so i and j or the superclass object. And in the next statement we see we initialize, the subclass object i and j and k as 7 8 9 and then we display the values of show i, show j as it is accessible to sub Ob by means of the inheritance and then also we call the show k method of thus subclass objects sub Ob and then we print it.

Now, let us have a quick demo. So, you can see that this is all legitimate access, we can use this thing and it will give that results that we have initialized and finally, it will also display the value accordingly. So, we run this program as using javac and then execution, so fine. So, it will work because there is no error in the program and it will print accordingly. So, this is the value that this class will print for us ok. So, this is the first example showing how the simple inheritance in Java can be done.

Now, our next demonstration is based on the initialization of the subclass object by the constructors that are there in the superclass.

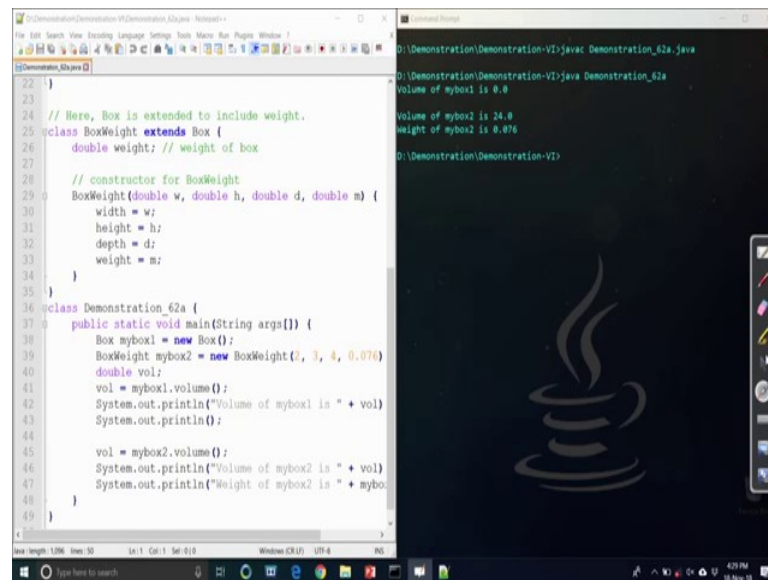
(Refer Slide Time: 05:01)

A screenshot of a Java IDE window titled "Demonstration\Demonstration\Box.java - NetBeans". The code defines a superclass `Box` and a subclass `BoxWeight`. The `Box` class has three `double` attributes: `width`, `height`, and `depth`. It has two constructors: a default constructor `Box()` that initializes all three attributes to `0.0`, and a parameterized constructor `Box(double w, double h, double d)` that initializes them with the provided values. It also has a `volume()` method that returns the product of width, height, and depth. The `BoxWeight` class extends `Box` and adds a `double weight` attribute. It has a constructor `BoxWeight(double w, double h, double d, double m)` that calls the `Box` constructor to initialize the superclass attributes and then sets the `weight` attribute. A comment above the `BoxWeight` class states: "// Here, Box is extended to include weight." The IDE interface includes a menu bar, a toolbar, and a status bar at the bottom showing "Line 1, Col 1, Sel 0/0".

```
1  /* Inheritance example: initializing through constructor */
2  class Box {
3      double width;
4      double height;
5      double depth;
6
7      Box() { // Default setting by this constructor
8          width = 0.0;
9          height = 0.0;
10         depth = 0.0;
11     }
12
13     Box(double w, double h, double d) {
14         width = w;
15         height = h;
16         depth = d;
17     }
18
19     double volume() { // compute and return volume
20         return width * height * depth;
21     }
22 }
23
24 // Here, Box is extended to include weight.
25 class BoxWeight extends Box {
26     double weight; // weight of box
27
28     // constructor for BoxWeight
29     BoxWeight(double w, double h, double d, double m) {
```

So, this demonstration will tell will go show us how we can initialize a subclass object using the constructor which is defined there in the superclass. Now let us have the program where we can see the class `Box` is a superclass class having 3 data width height and depth they are declared as `double`. And, the box is the constructor it is a superclass constructor default constructor and in addition to this default constructor there is one more constructors is initializing the different values in the class objects and also it has one method `volume` it is a simple multiplication of the values of the objects.

(Refer Slide Time: 05:49)



```
22 }
23
24 // Here, Box is extended to include weight.
25 class BoxWeight extends Box {
26     double weight; // weight of box
27
28     // constructor for BoxWeight
29     BoxWeight(double w, double h, double d, double m) {
30         width = w;
31         height = h;
32         depth = d;
33         weight = m;
34     }
35
36 class Demonstration_62a {
37     public static void main(String args[]) {
38         Box mybox1 = new Box();
39         BoxWeight mybox2 = new BoxWeight(2, 3, 4, 0.076);
40         double vol;
41         vol = mybox1.volume();
42         System.out.println("Volume of mybox1 is " + vol);
43         System.out.println();
44
45         vol = mybox2.volume();
46         System.out.println("Volume of mybox2 is " + vol);
47         System.out.println("Weight of mybox2 is " + mybox2.weight);
48     }
49 }
```

The screenshot shows an IDE with two windows. The left window displays the Java source code for a `BoxWeight` class that extends `Box`, and a `Demonstration_62a` class with a `main` method. The right window shows the output of the program, which prints the volume and weight of two objects: `mybox1` (a `Box` object) and `mybox2` (a `BoxWeight` object).

So, this is a class superclass. Now we derive one class give the name as box weight it is an inheritance of the superclass. Box here in this case and in addition to w h and d which are there in the box class we define another data weight of type w see this is the data of its own and then we declare a constructor in this class box weight which we pass the value there and then initialization if this one.

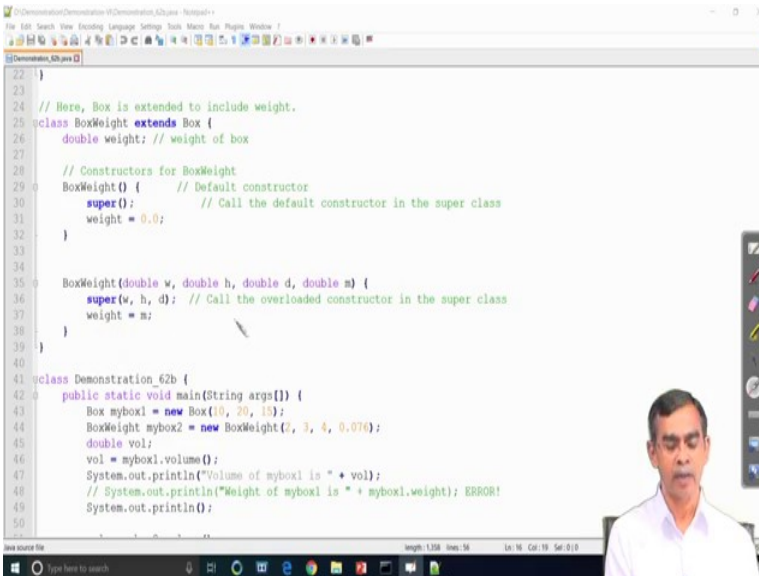
So, we can create an object of superclass as well as subclass here in the means class we can see. The main class is demonstration\_62 a. So, we create an object my box 1 of the class box and then also we create another object my box 2 of the type box weight which is the subclass in this case passing the values 2 3 4 and 0.076 as an argument we initialize the object. And then finally, we print the volume for the object.

So, there we can see we have created a subclass object, we have created a superclass object; although no problem. For the superclass object, there is 2 constructor. So, in this example, the default constructor will be called whereas, for the initialization of the subclass object the constructor only constructor here that will be called. So, this program will be executed and let us see the execution of this program. So, this program (Refer Time: 07:22) you successful compilation and execution. It will print the volume of both objects my box 1 and my box 2 which we have created yeah.

So, we can see that 2 objects are successfully created one object is that a subclass object another is the superclass object. Now, we have shown in this the example that how the

subclass object can be initialized. Now we are going to have another illustration where a subclass object can be initialized with the help of superclass constructor. So, this is the one program in this direction it is simple as usual earlier the superclass remains the same wherever we just redefine the sub superclass subclass object in the following let's go down yeah.

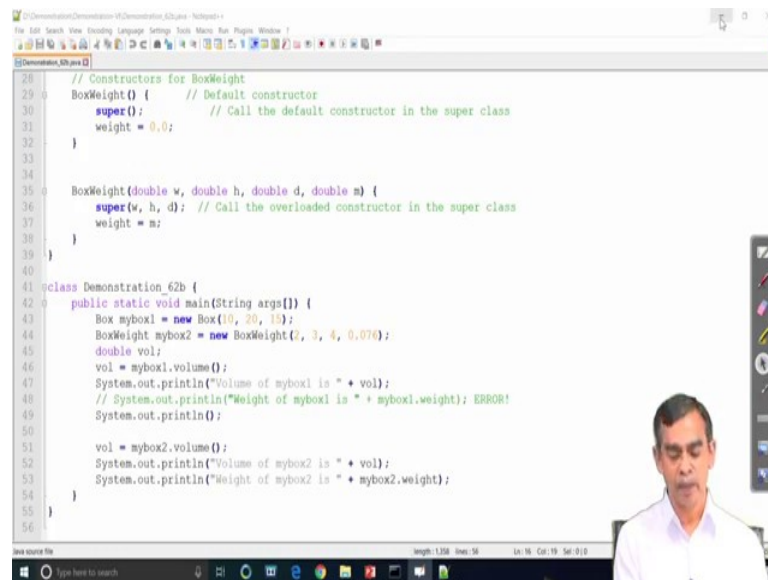
(Refer Slide Time: 08:33)

A screenshot of a Java IDE window titled "Demonstration\_62b.java". The code defines a class `BoxWeight` that extends `Box`. It includes a `weight` attribute and two constructors: a default constructor that calls `super()` and sets `weight = 0.0`, and an overloaded constructor `BoxWeight(double w, double h, double d, double m)` that calls `super(w, h, d)` and sets `weight = m`. Below this, a `Demonstration_62b` class contains a `main` method that creates a `Box` object `mybox1` and a `BoxWeight` object `mybox2`, calculates the volume of `mybox1`, and attempts to print the weight of `mybox1`, which results in an error because `mybox1` is not a `BoxWeight` object.

```
22 }
23
24 // Here, Box is extended to include weight.
25 class BoxWeight extends Box {
26     double weight; // weight of box
27
28     // Constructors for BoxWeight
29     BoxWeight() { // Default constructor
30         super(); // Call the default constructor in the super class
31         weight = 0.0;
32     }
33
34
35     BoxWeight(double w, double h, double d, double m) {
36         super(w, h, d); // Call the overloaded constructor in the super class
37         weight = m;
38     }
39 }
40
41 class Demonstration_62b {
42     public static void main(String args[]) {
43         Box mybox1 = new Box(10, 20, 10);
44         BoxWeight mybox2 = new BoxWeight(2, 3, 4, 0.076);
45         double vol;
46         vol = mybox1.volume();
47         System.out.println("Volume of mybox1 is " + vol);
48         // System.out.println("Weight of mybox1 is " + mybox1.weight); ERROR!
49         System.out.println();
50     }
51 }
```

So, here you can see we just have the 2 constructors, one is the default constructor in the subclass object subclass the box weight the default and another is the width some values. Now in case of default constructor we see we call super within this one, this basically called the superclass constructor in the superclass namely box ok. So, it will call this one, so it will initialize with the 0 0 0 values to the members.

(Refer Slide Time: 09:03)



```
28 // Constructors for BoxWeight
29 BoxWeight() { // Default constructor
30     super(); // Call the default constructor in the super class
31     weight = 0.0;
32 }
33
34 BoxWeight(double w, double h, double d, double m) {
35     super(w, h, d); // Call the overloaded constructor in the super class
36     weight = m;
37 }
38
39 }
40
41 class Demonstration_62b {
42     public static void main(String args[]) {
43         Box mybox1 = new Box(10, 20, 10);
44         BoxWeight mybox2 = new BoxWeight(2, 3, 4, 0.070);
45         double vol;
46         vol = mybox1.volume();
47         System.out.println("Volume of mybox1 is " + vol);
48         // System.out.println("Weight of mybox1 is " + mybox1.weight); ERROR!
49         System.out.println();
50
51         vol = mybox2.volume();
52         System.out.println("Volume of mybox2 is " + vol);
53         System.out.println("Weight of mybox2 is " + mybox2.weight);
54     }
55 }
56
```

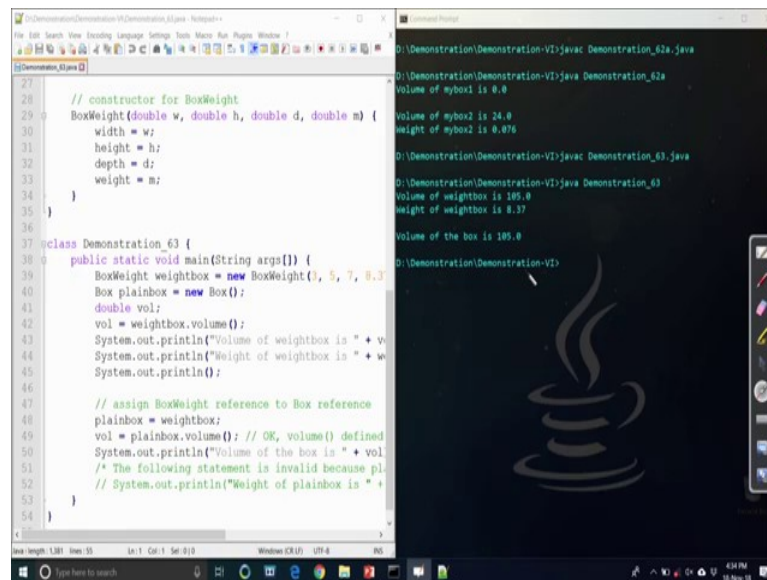
Now, again the super w h here basically it is the superclass constructor, which has the 3 arguments that is required and then we call this superclass constructor to initialize this using the box weight constructor here.

Now, let us have the same demo here it is the same thing we create 2 objects, my box on for the superclass object my box 2 for the inherited class objects and then it is the same program as earlier, only the thing that we have initialized with the help of superclass constructor. Now, so in this demo, the superclass constructor is basically, super with the certain argument. The argument which will fit with the superclass constructor will be used herein the subclass constructor.

So, let us have another demo. It basically super use of the super keyword that basically how we can refer a subclass object with the help of superclass variable. Now let us have the demo 6.3. Now, in this case, we will see how the superclass variable can be referent to the subclass variable like this one. We can better explain this now this is the subclass definition class box, it is the same as earlier it has 2 constructors, the if the class definition is same already that we have discussed here. Now, let us come to the superclass object. It is also same, it basically has the weights and then the only constructor here in this case.



(Refer Slide Time: 10:43)

The screenshot shows an IDE with two windows. The left window displays the source code for a Java program. It includes a constructor for a subclass named BoxWeight, which inherits from a superclass named Box. The constructor takes four parameters: width, height, depth, and weight. The main method in the Demonstration\_63 class creates two objects: a BoxWeight object named weightbox and a Box object named plainbox. It then calls the volume() method on weightbox, prints the result, and assigns weightbox to plainbox. Finally, it calls the volume() method on plainbox and prints the result. The right window shows the output of the program, which displays the volume of mybox1 as 0.0, the volume of mybox2 as 24.0, the weight of mybox2 as 0.078, the volume of weightbox as 189.0, the weight of weightbox as 0.37, and the volume of the box as 189.0. The IDE's status bar at the bottom indicates the file length is 1,381 bytes and the cursor is at line 1, column 1.

Now, let us see the main class method here. This needs to be checked very carefully. Here we define one object of subclass box weight, the namely weight box is the object passing the parameter for it. Now in addition to this also we declare here another object plain box of class superclass box here and volume is the volume to hold the volume of this one.

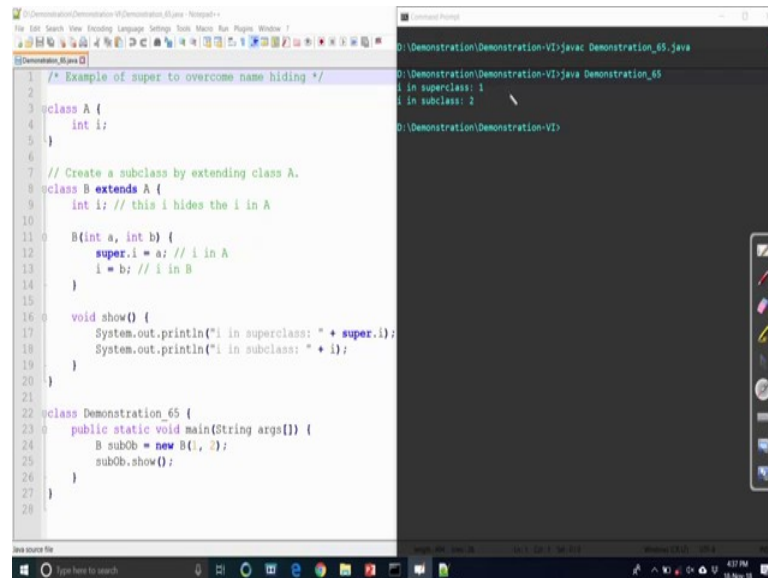
Now, if we call the volume method for the weight box. So, it will call the volume method in which is defined in the subclass and accordingly the volume will be calculated now. So, it will be printed now so, that is all. Now let us come to the next one. Now here we can see the plain box is equal to weight box. So, it is possible, here basically we are referencing a subclass with the help of superclass. The plain box is a superclass object and weight of box is a subclass object. This kind of assignment is quite legitimate; that means, we can reference a subclass object name with the help of superclass object name and next statement is also quite valid volume will be obtained for the plain box there.

Now, again you can see which method will be called here. It is basically the volume of the weight box method will be called here. Now let us run this program 6.3. So, we can see the volume that we can print here is the volume of the subclass object, but it is a reference to the superclass object ok. So, this is the 1 demo. Let us have another demo, this demo is basically planned to explain using the usefulness of super to avoid the name namespace collision. So, basically, we can write overcome the name hiding using the



super construct. Let us 6.5 the demo yes, so this is the one simple program that we can check it.

(Refer Slide Time: 13:09)



```
1  /* Example of super to overcome name hiding */
2
3  class A {
4      int i;
5  }
6
7  // Create a subclass by extending class A.
8  class B extends A {
9      int i; // this i hides the i in A
10
11     B(int a, int b) {
12         super.i = a; // i in A
13         i = b; // i in B
14     }
15
16     void show() {
17         System.out.println("i in superclass: " + super.i);
18         System.out.println("i in subclass: " + i);
19     }
20 }
21
22 class Demonstration_65 {
23     public static void main(String args[]) {
24         B subOb = new B(1, 2);
25         subOb.show();
26     }
27 }
28
```

```
D:\Demonstration\Demonstration-VI>javac Demonstration_65.java
D:\Demonstration\Demonstration-VI>java Demonstration_65
i in superclass: 1
i in subclass: 2
D:\Demonstration\Demonstration-VI>
```

Now, here let us look at the program class A is a class declared here having an integer as a variable in it and class B is inherited class from A and also see i integer is declared of its own. Now here whenever by means of inheritance, the value the variable i both that is there in the superclass is also accessible to the subclass, then it becomes a problem it is called the collision, collision means both i is there ok.

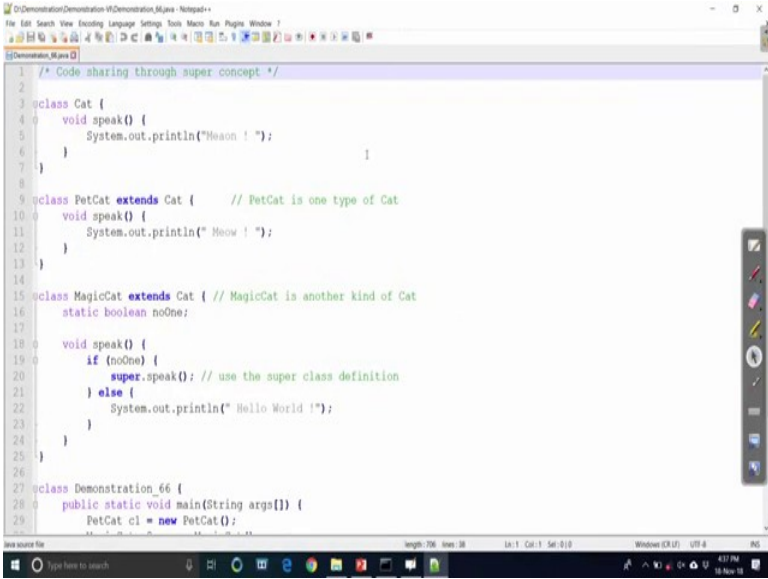
Now, of course, according to the inheritance, it basically overwrites that mean the scope according to the see this I, which is declared in class b is basically i of this subclass objects not that one. So, this i, which is declared in class B in fact, heights the i which is already there in A; however, we can refer both the variable and this reference is possible using the super keyword.

Now let us see the constructor which we have defined for the subclass object B is like passing A and B as the arguments. Now if I mention super.i this refer to the variable I, which is there in the superclass object and similarly I, if we do not mention anything it basically refers to the variable i in the same class itself that is here in the B. So, this way we can refer to some superclass variable as well as the subclass variable this way.

So, super can be used to resolve the collision that is what happens in this case. So, the rest of the program is very simple. So, this method we will print all the values those are there in the subclass as well as the superclass to print statement is used for that and these is the main() method, a sub-object subclass object is created and then we call the so method, it will print the 2 values there ok.

So, for example, 1 and 2 will be painted here, 1 will go to the eye, the superclass values and then 2 will go to the value to the subclass ok. Let us run this program quickly so that we can see exactly whether it is running or not and then we can have the understanding then that superclass can be used to resolve the name collision. So we can see that ok, so it is it basically is successful so far execution is concerned. So, it works. Now, our next demonstration basically to see how the coat shearing is possible, it is also a very good example of the dynamic binding concept that is there.

(Refer Slide Time: 16:07)



```
1  /* Code sharing through super concept */
2
3  class Cat {
4      void speak() {
5          System.out.println("Meow ! ");
6      }
7  }
8
9  class PetCat extends Cat { // PetCat is one type of Cat
10     void speak() {
11         System.out.println(" Meow ! ");
12     }
13 }
14
15 class MagicCat extends Cat { // MagicCat is another kind of Cat
16     static boolean noOne;
17
18     void speak() {
19         if (noOne) {
20             super.speak(); // use the super class definition
21         } else {
22             System.out.println(" Hello World !");
23         }
24     }
25 }
26
27 class Demonstration_66 {
28     public static void main(String args[]) {
29         PetCat c1 = new PetCat();
```

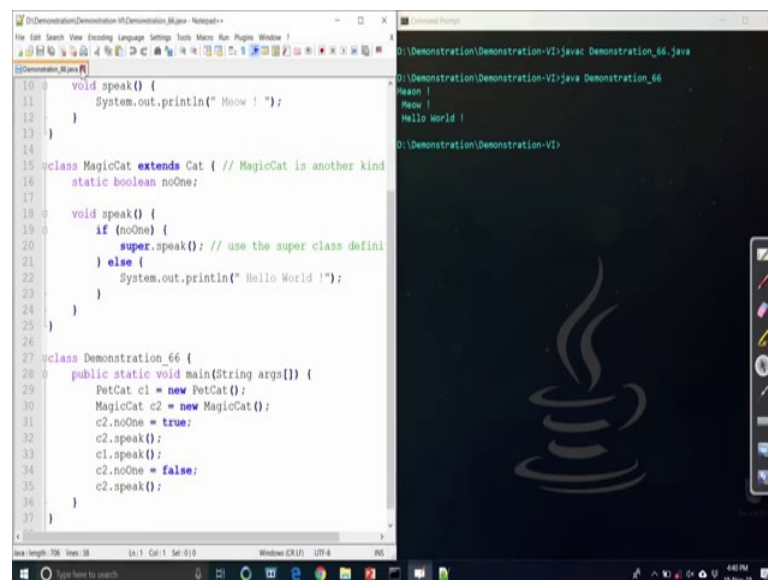
Actually, it is a runtime polymorphism concept, it is they are during runtime it will resolve which method is basically called here. Now here we can see first we declare one class the name of the class is a cat and it has one method speak and then it basically print this meow statement here.

Now, another class which is basically inherited from that class cat is a subclass pet cat of superclass cat. It has also the method speak and this method has this statement meow. Now here you can see the 2 methods are defined, but it is a method overriding. Thus the

speak method in pet care overridden then, then the method that is there in the subclass method cat. Now we declare another one class also an extension of cat it is basically multi-level multi multiple it is we can say that 2 inheritances, 2 multiple single intents we can say here because we another inherit another class magic cat from the class cat and we can define one variable is a Boolean type no one.

Now, void speaks if no one, if it is true then it will call the super speak. Super speak mean in this case, it will call the cat class to speak that is there in declare method meaning in discussing it will spin meow and if it is false then it will call this simple message. Now let us see how dynamically we can bind to this.

(Refer Slide Time: 17:47)



```
10 void speak() {
11     System.out.println(" Meow ! ");
12 }
13 }
14
15 class MagicCat extends Cat { // MagicCat is another kind
16     static boolean noOne;
17
18     void speak() {
19         if (noOne)
20             super.speak(); // use the super class defini
21         else {
22             System.out.println(" Hello World !");
23         }
24     }
25 }
26
27 class Demonstration_66 {
28     public static void main(String args[]) {
29         PetCat c1 = new PetCat();
30         MagicCat c2 = new MagicCat();
31         c2.noOne = true;
32         c2.speak();
33         c1.speak();
34         c2.noOne = false;
35         c2.speak();
36     }
37 }
```

```
D:\Demonstration\Demonstration-VI\java Demonstration_66
Meow !
Meow !
Hello world !
D:\Demonstration\Demonstration-VI
```

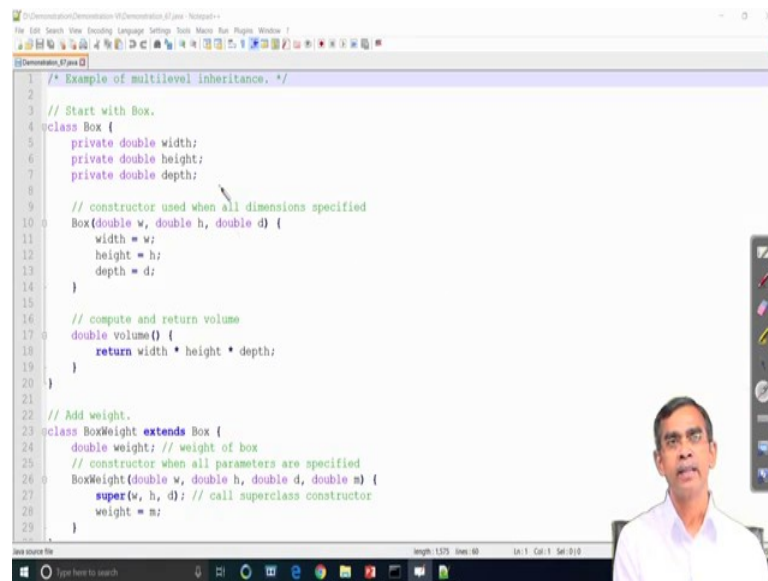
Let us have this program, this is a little bit tricky. You can see how these statements are here. So, demonstration\_66 is basically giving the idea about runtime polymorphism in this case, but we will resolve it using the super concept here. So, here we create an object of a subclass pet cat c 1.

So, that is very simple. Also, you create another object c 2, the magiccat and so C2 no one we mention true; that means, if it is no one, it will spin the superclass method for this c 2. Now, again c 2 speak. So, it will call the method here. Now c 1 speak if we call that it will call another subclass objects that is the B met one. Now we can mets c 2 noOne as false. So, it is now false and if we call again C 2 speak, then it will call another method.

So, it accordingly it will print meow meow and then hello cat. Now let us see the run the program we will see exactly how it will work yeah ok.

Now, you can see this basically print according to the different states depending on the concept it is there. So, this is one example here basically we can see how that 2 or more classes can be inherited from one superclass. This also example signify this fact. Now let us have another instance of multi-level inheritance.

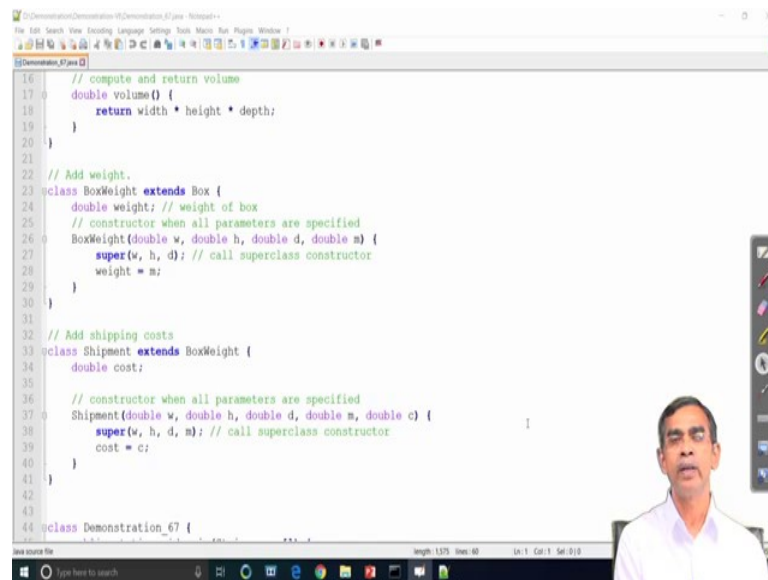
(Refer Slide Time: 19:33)



```
1  /* Example of multilevel inheritance. */
2
3  // Start with Box.
4  class Box {
5      private double width;
6      private double height;
7      private double depth;
8
9      // constructor used when all dimensions specified
10     Box(double w, double h, double d) {
11         width = w;
12         height = h;
13         depth = d;
14     }
15
16     // compute and return volume
17     double volume() {
18         return width * height * depth;
19     }
20 }
21
22 // Add weight.
23 class BoxWeight extends Box {
24     double weight; // weight of box
25     // constructor when all parameters are specified
26     BoxWeight(double w, double h, double d, double m) {
27         super(w, h, d); // call superclass constructor
28         weight = m;
29     }
29 }
```

Multilevel inheritance means if we can derive from one class subclass sub from subclass we can derive another subclass. So, like this one the example of multi-level inheritance, now, here let us see the class box which is already the same as we have discussed in the earlier demonstration and also we used the box weight another subclass derived from the class box. So, it is more or less the same as we have already discussed, now, here the simple inheritance of 2 level.

(Refer Slide Time: 20:01)

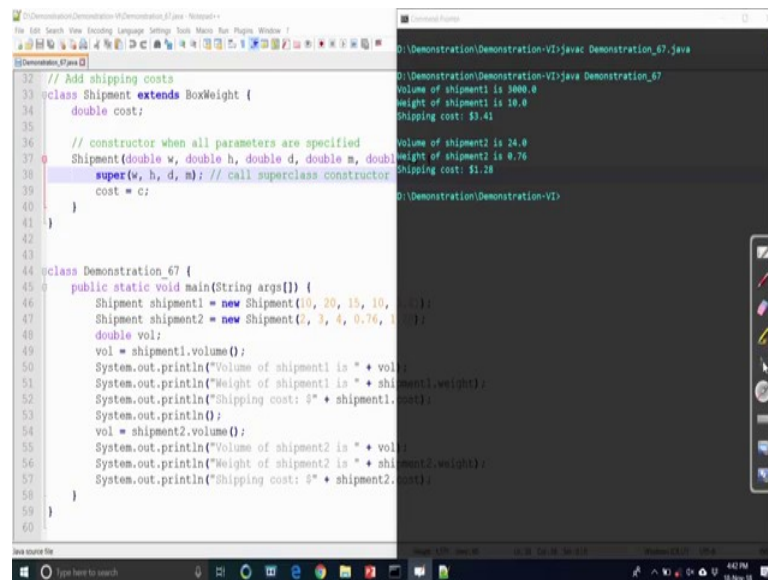


```
16 // compute and return volume
17 double volume() {
18     return width * height * depth;
19 }
20 }
21
22 // Add weight.
23 class BoxWeight extends Box {
24     double weight; // weight of box
25     // constructor when all parameters are specified
26     BoxWeight(double w, double h, double d, double m) {
27         super(w, h, d); // call superclass constructor
28         weight = m;
29     }
30 }
31
32 // Add shipping costs
33 class Shipment extends BoxWeight {
34     double cost;
35
36     // constructor when all parameters are specified
37     Shipment(double w, double h, double d, double m, double c) {
38         super(w, h, d, m); // call superclass constructor
39         cost = c;
40     }
41 }
42
43
44 class Demonstration_67 {
```

Now, in the next level we inherit another, so we define another class shipment it is basically subclass of the class box weight; that means, box weight is a derived class from the class box and shipment is another derived class from the box weight. So, this is the shipment is an example of multi-level inheritance. And again for the same concept, it is also applicable here, the multi-level inheritance can be initialized by calling its superclass constructor, in this case, box weight constructor.

So, super wh dm basically called the constructor that is defined there box weight, it is like this way and it is initialization. Now let us come to the creation of objects.

(Refer Slide Time: 20:47)



```
// Add shipping costs
class Shipment extends BoxWeight {
    double cost;

    // constructor when all parameters are specified
    Shipment(double w, double h, double d, double m, double c) {
        super(w, h, d, m); // call superclass constructor
        cost = c;
    }
}

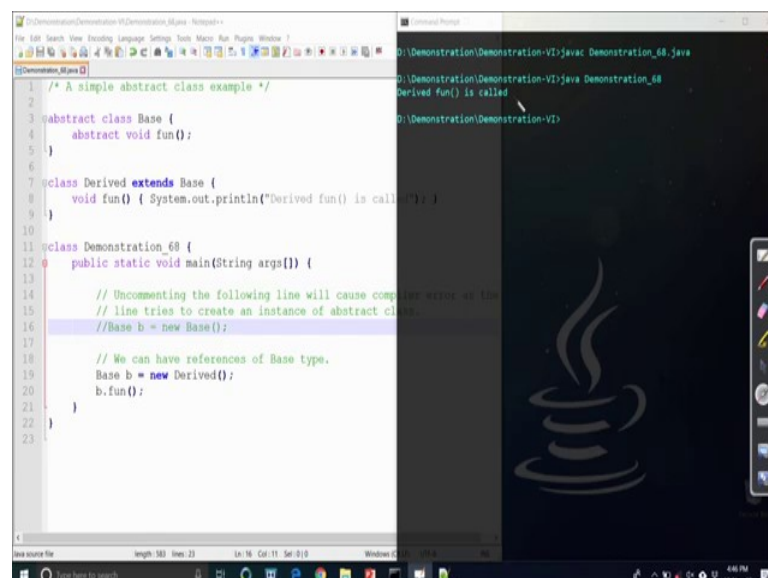
class Demonstration_67 {
    public static void main(String args[]) {
        Shipment shipment1 = new Shipment(10, 20, 15, 10, 1.28);
        Shipment shipment2 = new Shipment(2, 3, 4, 0.76, 1.41);
        double vol;
        vol = shipment1.volume();
        System.out.println("Volume of shipment1 is " + vol);
        System.out.println("Weight of shipment1 is " + shipment1.weight());
        System.out.println("Shipping cost: $" + shipment1.cost());
        vol = shipment2.volume();
        System.out.println("Volume of shipment2 is " + vol);
        System.out.println("Weight of shipment2 is " + shipment2.weight());
        System.out.println("Shipping cost: $" + shipment2.cost());
    }
}
```

```
D:\Demonstration\Demonstration-VI\java Demonstration_67.java
Volume of shipment1 is 3000.0
Weight of shipment1 is 10.0
Shipping cost: $3.41
Volume of shipment2 is 24.0
Weight of shipment2 is 0.76
Shipping cost: $1.28
D:\Demonstration\Demonstration-VI>
```

So, demonstration 6 7 there is a program, here we can create 2 objects shipment 1 and shipment 2 and then we can call this method it works and then, so let us run this program. So, that you can see them in the different for the 2 different objects, which are derived in a multilevel way can be used to create objects and then the different methods in those objects can be accessed by a Java program ok. So, this is an example that we can verify with the court, so that it is working correctly.

So, this is an example of multilevel inheritance. Now let us discuss the abstract class.

(Refer Slide Time: 21:31)



```
/* A simple abstract class example */
abstract class Base {
    abstract void fun();
}

class Derived extends Base {
    void fun() { System.out.println("Derived fun() is called"); }
}

class Demonstration_68 {
    public static void main(String args[]) {
        // Uncommenting the following line will cause compiler error as line
        // line tries to create an instance of abstract class.
        //Base b = new Base();

        // We can have references of Base type.
        Base b = new Derived();
        b.fun();
    }
}
```

```
D:\Demonstration\Demonstration-VI\java Demonstration_68.java
Derived fun() is called
D:\Demonstration\Demonstration-VI>
```

A class is defined as an abstract class, all the classes that we have discussed earlier superclass they are they are with the access specification the defaults So, they is a default, there is no other access specifier it is used here; otherwise, we can use some other access specifier depending on its application. Now here we use one keyword called abstract. If we specify an abstract keyword before a class then that class is called abstract class. So, in this case, the base class is declared as an abstract and also one method if a method is specified by a specifier called abstract then the method is called abstract.

So, in this case, a class is an abstract and in this class, one method is declared which is also abstract. Even we can also declare a method without any abstract also is called a non-abstract method, but in this case, let us have the method is now an abstract class. So, what is the meaning of this abstract class? As we have already know learn about that, if we declare a class as an abstract class this means that no object can be created for this, but this class can be used to inherit some other class; means that an abstract class can be used for superclass, but no object can be created for this kind of class abstract class.

Now, let us have the one example here, we can see class derived is a subclass of the superclass base, so it is quite and if there are any abstract method, then in the subclass the method should be declared and defined properly. So, the method if you see abstract, whenever you declared abstract, no code need to be mention there, so there is no code, so it is a blank. Now here you see in the subclass declaration we fully declared the method fun and this basically includes on system pin statement it will basically pin this one.

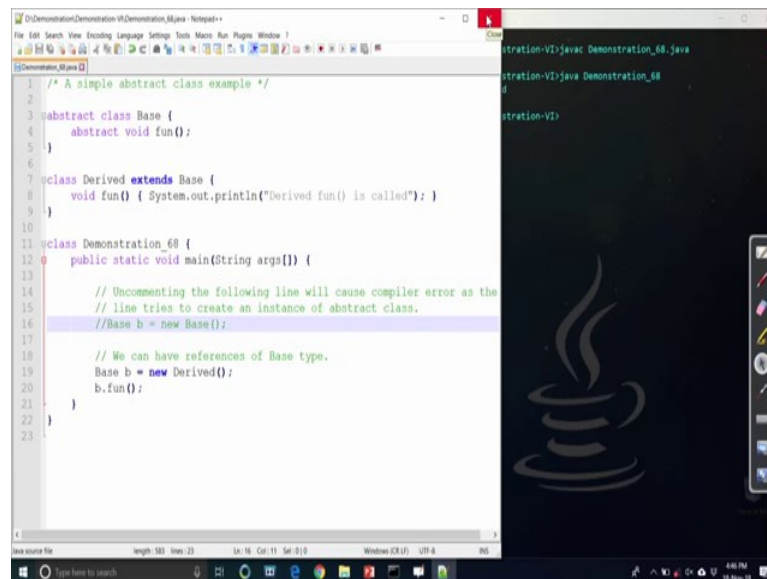
Now, let us come to the main method here demonstration 6 (Refer Time: 23:40) now 6 8, so here we can see, so here we can see, if we uncomment the statement like base b, new b then, let us uncomment the statement and then try to run it and we will see what is the consequence. So, this will give an error, because the base class here is an abstract class and we are not privileged to create an object. As we can see the state the error during the compilation base b, newly derived it is basically saying variable b is already defined in a method mean like this one.

So, basically, abstract class cannot be instantiated you can see an abstract class cannot be instantiated because it is like this one. So, let us uncomment comment it again now have the next one base b, newly derived this is quite yeah. So, now, see we can have the reference of base type by means of this kind of upcasting is quite possible there. So, now,



we create an object of type derived class but reference it through a base class object this is quite possible and then we call the b.fun as it is there. So, this fun is basically the fun method, which is declaring the subclass method. So, this program if it is run, then it will give the output, so this is working correctly. So, this is a concept it is there, so far the abstract class is concerned. Now again here that whether abstract class we have understood that no object can be created.

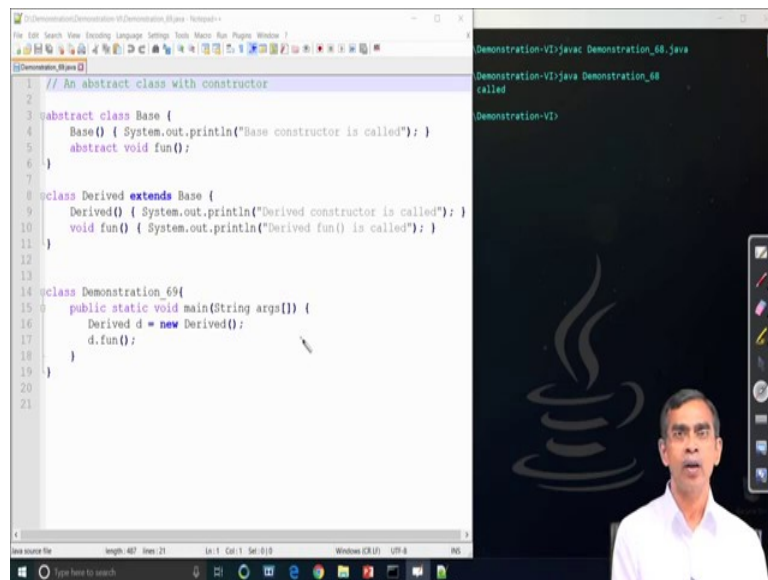
(Refer Slide Time: 25:11)



```
1  /* A simple abstract class example */
2
3  abstract class Base {
4      abstract void fun();
5  }
6
7  class Derived extends Base {
8      void fun() { System.out.println("Derived fun() is called"); }
9  }
10
11 class Demonstration_68 {
12     public static void main(String args[]) {
13
14         // Uncommenting the following line will cause compiler error as the
15         // line tries to create an instance of abstract class.
16         //Base b = new Base();
17
18         // We can have references of Base type.
19         Base b = new Derived();
20         b.fun();
21     }
22 }
23
```

Now, whether abstract class, can have any constructor or not. So, our next example showing this thing that yes an abstract class may have its own constructor.

(Refer Slide Time: 25:25)



```
1 // An abstract class with constructor
2
3 abstract class Base {
4     Base() { System.out.println("Base constructor is called"); }
5     abstract void fun();
6 }
7
8 class Derived extends Base {
9     Derived() { System.out.println("Derived constructor is called"); }
10    void fun() { System.out.println("Derived fun() is called"); }
11 }
12
13
14 class Demonstation_69 {
15     public static void main(String args[]) {
16         Derived d = new Derived();
17         d.fun();
18     }
19 }
20
21
```

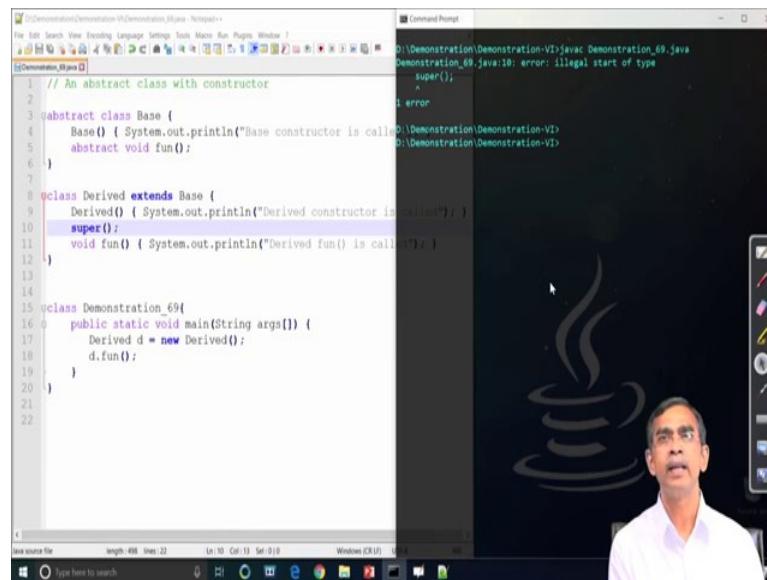
Terminal output:

```
Demonstration-VI: javac Demonstation_68.java
Demonstration-VI: java Demonstation_68
called
Demonstration-VI:
```

That means abstractor can be used to initialize the member elements if it is there if it is not an object creation event. Actually, this constructor will be useful to initialize the object of the subclass of this class because for an abstract class subclass can be created. Now, here is an example where you can see how an abstract class can have its own constructor and how the same constructor can be useful to initialize the subclass objects. So, here we can see the base is an abstract class constructor here it basically prints the same statement and the next is basically an abstract method namely the fun here. And the see derived class is a subclass of the class base here and derived is a constructor of its own it derives is there and void fun is the method which is basically an implementation of the abstract method that is there in the base class.

So, here derived is there, although this constructor is not called; that means, superclass constructor is not called here. I will tell you how this can be called here anyway. Now let us have the demo about it so; that means, the constructor it is there we can call it ok. We can just little change this program whether base class constructor can be called here in this method in the derived class. So, you can do that yeah this program is running fine yeah, so it is running.

(Refer Slide Time: 26:51)



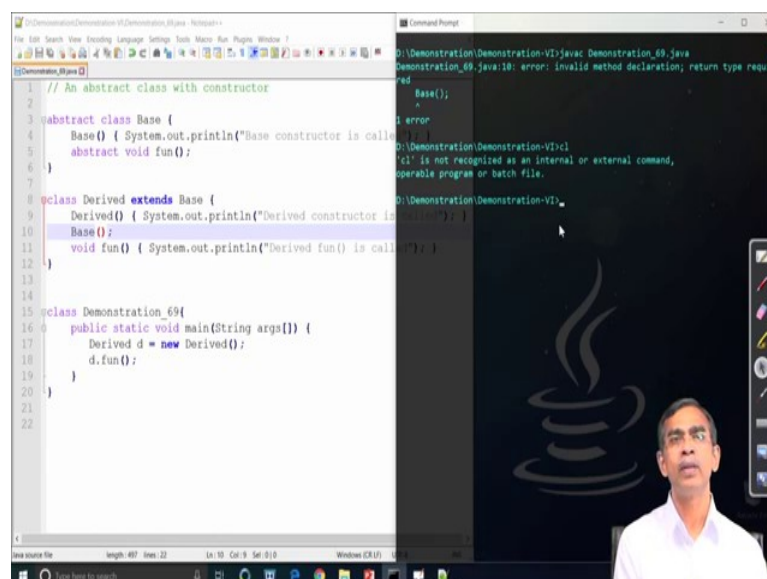
The screenshot shows an IDE with a Java file named `Demonstration_69.java`. The code defines an abstract class `Base` with a constructor `Base()` and an abstract method `fun()`. A class `Derived` extends `Base` and implements `fun()`. The `main` method in `Demonstration_69` creates a `Derived` object and calls `fun()`. The terminal window shows the command `javac Demonstration_69.java` and the error message: `error: illegal start of type`.

```
1 // An abstract class with constructor
2
3 abstract class Base {
4     Base() { System.out.println("Base constructor is called"); }
5     abstract void fun();
6 }
7
8 class Derived extends Base {
9     Derived() { System.out.println("Derived constructor is called"); }
10    super();
11    void fun() { System.out.println("Derived fun() is called"); }
12 }
13
14
15 class Demonstration_69 {
16     public static void main(String args[]) {
17         Derived d = new Derived();
18         d.fun();
19     }
20 }
21
22
```

```
D:\Demonstration\Demonstration-VI\javac Demonstration_69.java
Demonstration_69.java:10: error: illegal start of type
    super();
    ^
1 error
```

Now, let us come to the code again switch to the code. Now we can call the base class constructor here. So, in the derived class got not yeah yes kind. So, right next statement we can add here before right yes. So, you can write `super` then within this one. So, basically superclass constructor namely, the base constructor will be called here right ok, then save it compile it. So, here you can see both the derived class constructor, as well as superclass constructor will be called here, illegal start of type ok.

(Refer Slide Time: 27:39)



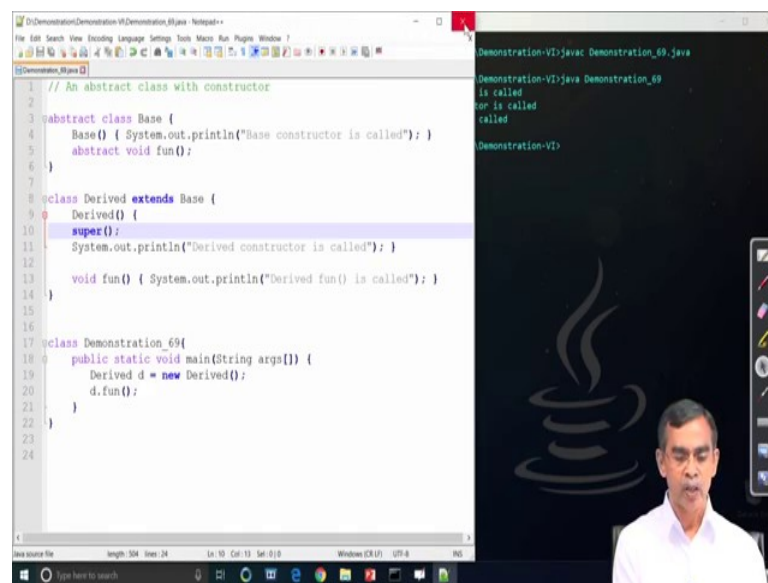
The screenshot shows the same IDE as before, but the code in `Derived` now includes `Base()` before `super()`. The terminal window shows the command `javac Demonstration_69.java` and the error message: `error: invalid method declaration; return type required`.

```
1 // An abstract class with constructor
2
3 abstract class Base {
4     Base() { System.out.println("Base constructor is called"); }
5     abstract void fun();
6 }
7
8 class Derived extends Base {
9     Derived() { System.out.println("Derived constructor is called"); }
10    Base();
11    super();
12    void fun() { System.out.println("Derived fun() is called"); }
13 }
14
15
16 class Demonstration_69 {
17     public static void main(String args[]) {
18         Derived d = new Derived();
19         d.fun();
20     }
21 }
22
```

```
D:\Demonstration\Demonstration-VI\javac Demonstration_69.java
Demonstration_69.java:10: error: invalid method declaration; return type required
    Base();
    ^
1 error
```

So, can we write base here, simple base. Let us try whether the base call concern can be called in this method or not. In this case, super it does not work here, anyway so, the superclass constructor cannot be used here, but we can use in the derived class constructor base class constructor right system. yeah, you can just comment it yeah fine. Then within this derived class method constructor go there right here right, then go to the 2 statement right one is that super yeah, super within write yeah construct not 0, right oh yeah they're fine.

(Refer Slide Time: 28:09)



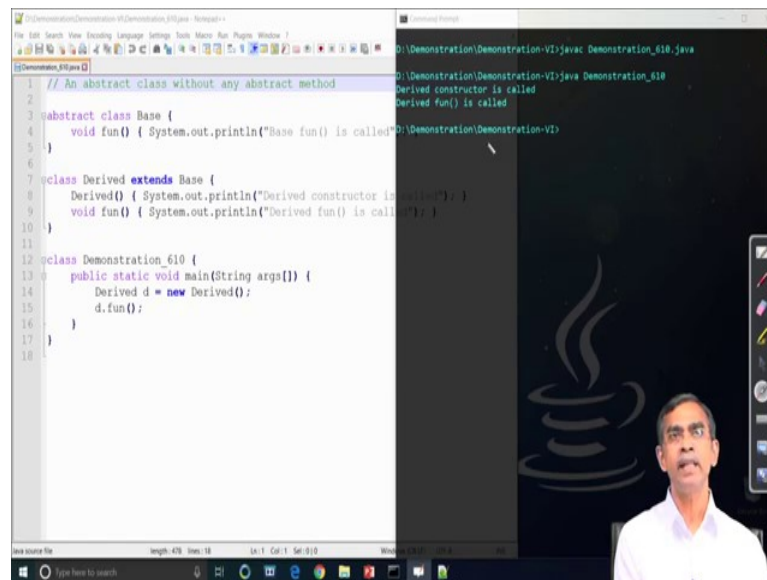
```
// An abstract class with constructor
1
2
3 abstract class Base {
4     Base() { System.out.println("Base constructor is called"); }
5     abstract void fun();
6 }
7
8 class Derived extends Base {
9     Derived() {
10         super();
11         System.out.println("Derived constructor is called"); }
12     void fun() { System.out.println("Derived fun() is called"); }
13 }
14
15
16 class Demonstration_69 {
17     public static void main(String args[]) {
18         Derived d = new Derived();
19         d.fun();
20     }
21 }
22
23
24
```

```
Demonstration-VI: javac Demonstration_69.java
Demonstration-VI: java Demonstration_69
Base constructor is called
Derived constructor is called
Derived fun() is called
Demonstration-VI:
```

Now, we call this constructor a through the derived class constructor. Let us see whether it works for us or not yeah. So, in this case, it works that mean a constructor can be called by means of the subclass constructor only now let us run this program. It will call the superclass constructor as the base class constructor ok. So, this is basically we have understood that a constructor can be declared in an abstract class and that same constructor can be used in the derived class objects. So, this is one example.

Now, let us have another example, as in the last example we have discussed that an abstract class with an abstract method, but an abstract class may have the non-abstract that means without any.

(Refer Slide Time: 29:03)



```
// An abstract class without any abstract method
abstract class Base {
    void fun() { System.out.println("Base fun() is called"); }
}

class Derived extends Base {
    Derived() { System.out.println("Derived constructor is called"); }
    void fun() { System.out.println("Derived fun() is called"); }
}

class Demonstration_610 {
    public static void main(String args[]) {
        Derived d = new Derived();
        d.fun();
    }
}
```

Terminal Output:

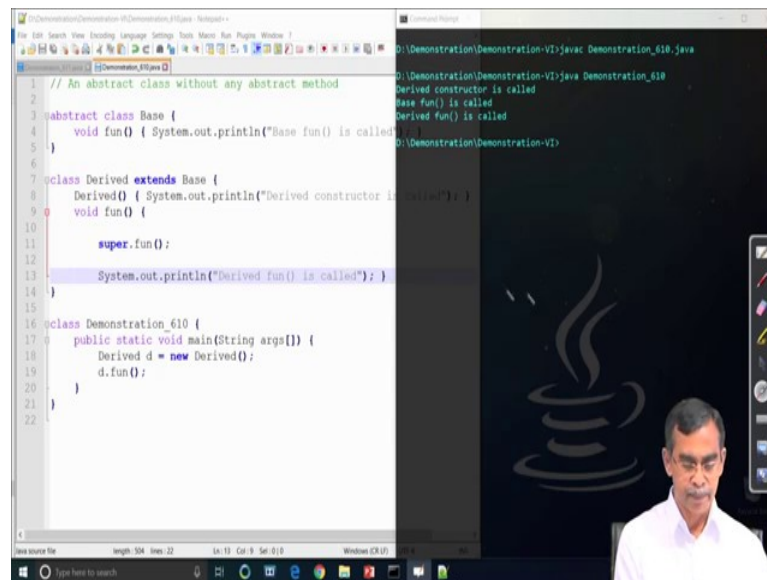
```
D:\Demonstration\Demonstration-VI\javac Demonstration_610.java
D:\Demonstration\Demonstration-VI>java Demonstration_610
Derived constructor is called
Derived fun() is called
```

So, that method also can exist, but this method can be accessed through the subclass object. So, this is the one example that we are going to give a demo, here the class base is the abstract class defined as an abstract keyword and then it has the method fun which is basically non-abstract method.

Now, we can call we can create a subclass object derived here inherited from the base class and then the fun method is here it is basically overridden method here because we have overridden the fun method there it is like this one fine. Now let us come to the main class here demonstration\_610 main class, we basically create an object for the derived class derived the new derived. So, in this case, you can understand d.fun we will call the fun method there ok.

So, let us run this program we can understand how it works for us yeah. So, it is derived is called here ok. So, derive constructor is called and derive phone is called, so it is like this. Now let us see how we can access the fun method which is defined they are in the base class method lets come to the object here no yeah here.

(Refer Slide Time: 30:41)



```
// An abstract class without any abstract method
abstract class Base {
    void fun() { System.out.println("Base fun() is called"); }
}

class Derived extends Base {
    Derived() { System.out.println("Derived constructor is called"); }
    void fun() {
        super.fun();
        System.out.println("derived fun() is called"); }
}

class Demonstration_610 {
    public static void main(String args[]) {
        Derived d = new Derived();
        d.fun();
    }
}
```

Terminal Output:

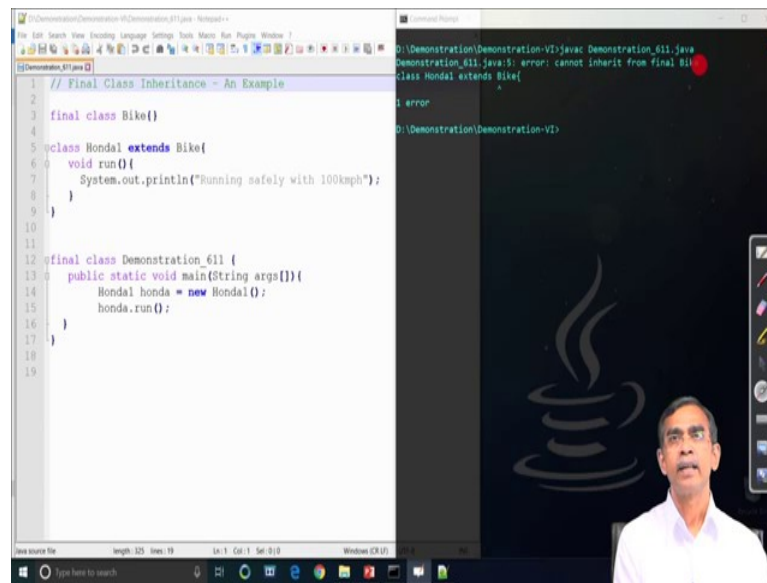
```
D:\Demonstration\Demonstration-VI\javac Demonstration_610.java
D:\Demonstration\Demonstration-VI>java Demonstration_610
Derived constructor is called
Base fun() is called
Derived fun() is called
D:\Demonstration\Demonstration-VI>
```

So, now not here these as the previous program you have switched to the next one yeah. So, this fun method which is defined they are in the base method is basically it is allowed that abstract non-abstract method.

Now, my question is whether we can call this non-abstract method here in the derived class or not. We can use it here we can use as a super right, you can use the super keyword. For example, in the fun method or somewhere right we can write super. fun, super.fun right and then this one. Now we can understand that we used the fun method in the right. So, by super keyword we can refer to the member which is there in the base class; although the object is not created, it will be accessed yeah. So, it is right yeah we can understand this one.

So, now non-abstract method may be there in the abstract class, like non-obstructive data may be there in the abstract class they can be accessed using the super keyword that is there in any subclass objects.

(Refer Slide Time: 32:07)



```
// Final Class Inheritance - An Example
1
2
3 final class Bike{
4
5     class Honda1 extends Bike{
6         void run(){
7             System.out.println("Running safely with 100kmph");
8         }
9     }
10
11
12 final class Demonstration_611 {
13     public static void main(String args[]){
14         Honda1 honda = new Honda1();
15         honda.run();
16     }
17 }
18
19
```

D:\Demonstration\Demonstration-VI\java: Demonstration\_611.java  
Demonstration\_611.java:5: error: cannot inherit from final Bike  
class Honda1 extends Bike{  
^  
1 error  
D:\Demonstration\Demonstration-VI>

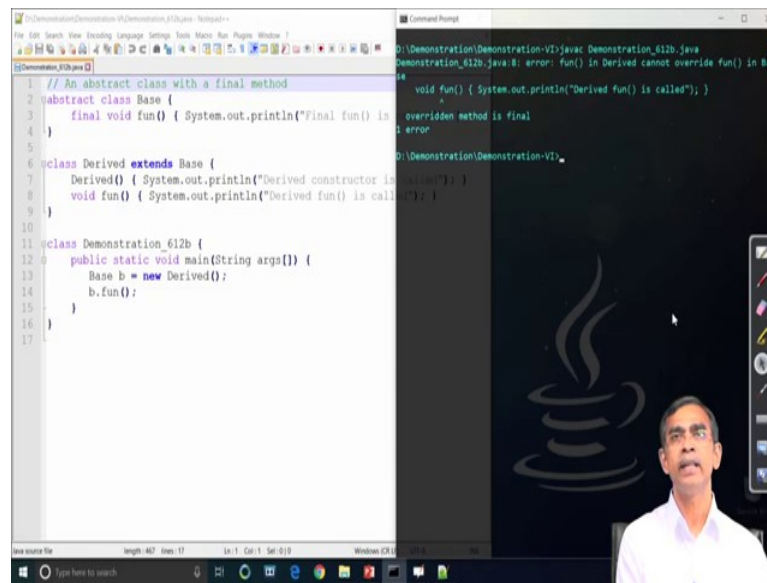
Our next example: basically demonstrating the final keyword that is there. Final keyword is a very strict keyword if you declare a class as final; that means, this class cannot be inherited in any other class, that means no subclass can be created from this one.

So, this is the one example where the bike is the one class we have declared as a final. So, final means, no inheritance is possible. Now, this code definitely it is not a valid code because, we are attempting to create a subclass called Honda 1, extending bike class, bike class and then definitely if it is not possible. So, the next statement main class is also not a valid one, now let us run this program, see whether this program gives a compilation error or it works ok. So now, we can see it gives an error that that cannot inherit from the final bike. So, this means that we cannot do this one ok. So, this is the one.

Now, this question that arises that then what is the use of the final? Sometimes we can have a stick restriction that this class is a stick class that no on class can be derived because derivation means is an accessing some member in the superclass. So, if we want to protect it, so we can fix the final keyword. Now a class can be made as a final like a method also can be made as a final, a variable also can be made as a final. Now, here in the next example that, we are going to give a demo 6.12 a, showing that, how a method can be declared as a final. If we declare a method as a final that means, this method cannot be overridden in any way.



(Refer Slide Time: 33:55)



```
// An abstract class with a final method
abstract class Base {
    final void fun() { System.out.println("Final fun() is called"); }
}

class Derived extends Base {
    Derived() { System.out.println("Derived constructor is called"); }
    void fun() { System.out.println("Derived fun() is called"); }
}

class Demonstration_612b {
    public static void main(String args[]) {
        Base b = new Derived();
        b.fun();
    }
}
```

D:\Demonstration\Demonstration-VI\javac Demonstration\_612b.java  
Demonstration\_612b.java:8: error: fun() in Derived cannot override fun() in Base; overridden method is final  
 void fun() { System.out.println("Derived fun() is called"); }  
 ^  
error  
1 error  
D:\Demonstration\Demonstration-VI>

So, this is one example that we can see. The class base is the abstract class, that's fine and there is a method fun which is declared as a final. This means that no overheating is possible. Now here derived is the derived class, extent base class it does not have any other method or members is ok. Now we can create an object of the derived class, but referencing to the base class and then we call the b.fun. So, it will basically call the apps, the final method which is the fun method derive declare in defined in the base class. So, this fun method is basically system.out.println final fun is called.

Now, let us run this program, we can have the quick demo so that we can see about it yeah. So, fine, so this is running. Now let us have to see whether we can override it or not? This is an attempt to override a method, this is the next demonstration, please. So, we can see, we are trying to override it at one method which is declared as a final method in a base class and derived class we are going to overwrite it in.

Now let us have the quick look at the program here. So, here class based abstract method and then fun is also final. In derived we have the method derived is the constructor no issue. Now, void pan here is basically our attempt to override the method which is there in the base class. Now let us compile this program. If it compiles then means that over it is in successful. Now let us run this program, compile this program it is 62b yeah fine right. Now see it gives compilation that derives cannot overwrite fun in this. So, we have understood that a method if it is declared as a final, in super class then, it cannot be

overridden; however, it can be accessed in the subclass object by referring to this either super or this course ok.

So, this is the demonstration about the inheritance and the many features in inheritance in Java program. And we have discussed, so many things are there. It is advised that you should practice all the program that we have this used in this demonstration, so that, you can understand more. And if you have any doubt, any confusion you are you can feel free to approach at posting your doubts in the forum so that, we can answer to your question.

Thank you, thanks for your attention.