

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

**Lecture - 22**  
**Demonstration – IX**

So, we have learned about the interface in last two modules and this is a plan for this module plans for an I mean quick a demonstration of the different concepts that we have learned in our lecture session.

(Refer Slide Time: 00:36)

**In today's demonstration**

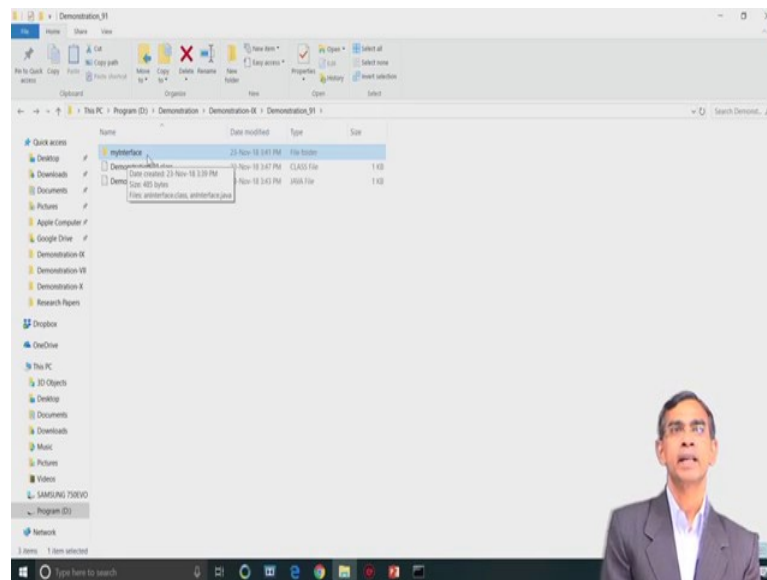
1. Creating an interface.
2. Some properties of interface.
3. Interface and single inheritance.
4. Interface and multiple inheritance.
5. Runtime polymorphism with interface
6. Interface versus abstract class

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

So, in today's Demonstrations mainly we have planned that how a user can create an interface and then maintain it in their package. An interface as we have discussed that it follows certain specific properties so if all those properties are not satisfied then whether the interface creation will be successful or not that we will learn it. And then interface is mainly used for inheritance purpose more specifically it is a multiple inheritance. So, single inheritance as a multiple inheritance using interface will be discussed. And another great application of the interface is to provide a dynamic binding in the form of runtime polymorphism so, that will be discussed.

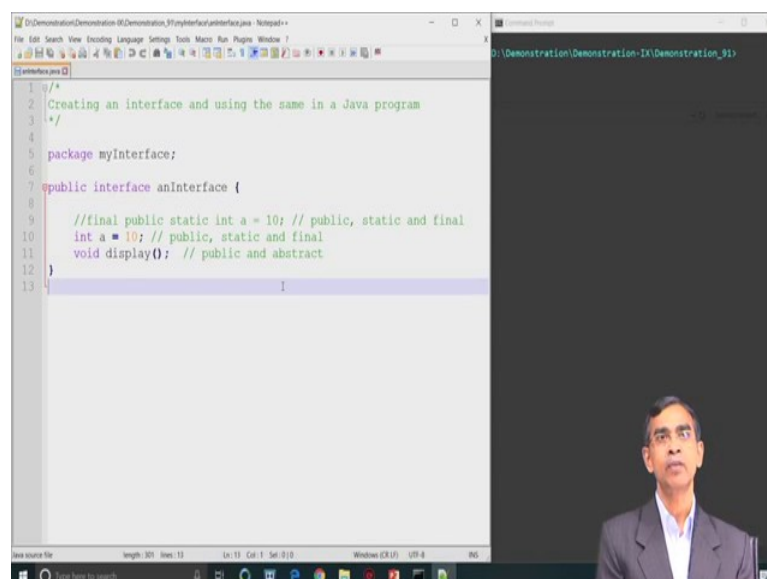
And then we have also mentioned that the abstract class and the interface has many similarities between the two. So, what is the difference between the two so, in our last demo that we will try to clarify further.

(Refer Slide Time: 02:04)



So, let us have the demonstration fast this is regarding how we can create an interface. So, these are the first program that you can think about how an interface can be created. So, we have to first create a package where we have to store the interface. let us this is the name of the package, my interface. We have created a subdirectory named my interface whereas the interface will be stored there. So, let us go to the subdirectory of my interface and as we see there is one program then you can just open the program.

(Refer Slide Time: 02:22)

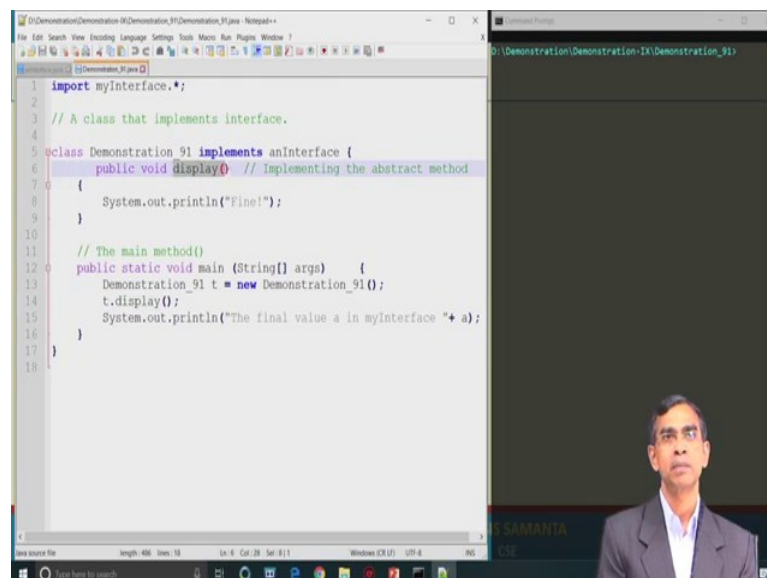


So, this a program, yes. So, this is the one program that we have developed just this program as you see the package statement includes that this my interface is the in the package named my interface and then the name of the interface as we have mentioned here with the keyword interface the name of the interface is an interface. And in this interface, we include two elements an as a member type integer and then the method display or we can type void.

And as we see the member elements that we have discussed here we do not have mentioned explicitly, but as you know in case of the interface the member element should be public, static, as well as final anyway. So, these are the default explicitly you do not have to mention implicitly also if you do not mention anything in an interface if it is declared they will be created as a public, final, static and the method as we have mentioned here the method should be public and abstract. So, we have we do not have to mention it explicitly again. If you do it very it is good if you do not do also the java compile time compiler will take care of it.

So, this is the interface that we have created and we stored this interface in the package my interface. Now, let us see the application of this interface in our program.

(Refer Slide Time: 03:56)

A screenshot of a Java IDE window. The main editor shows a Java class named 'Demonstration\_91' that implements an interface. The code includes an import statement for 'myInterface.\*', a comment '// A class that implements interface.', and a class declaration 'class Demonstration\_91 implements aninterface {'. Inside the class, there is a 'display()' method that prints 'Fine!' and a 'main' method that creates an instance of 'Demonstration\_91', calls 'display()', and prints a message about the final value 'a' in 'myInterface'. The IDE's output window on the right shows the execution path. A small video inset in the bottom right corner shows a man speaking.

```
1 import myInterface.*;
2
3 // A class that implements interface.
4
5 class Demonstration_91 implements aninterface {
6     public void display() // Implementing the abstract method
7     {
8         System.out.println("Fine!");
9     }
10
11 // The main method()
12 public static void main (String[] args) {
13     Demonstration_91 t = new Demonstration_91();
14     t.display();
15     System.out.println("The final value a in myInterface "+ a);
16 }
17
18 }
```

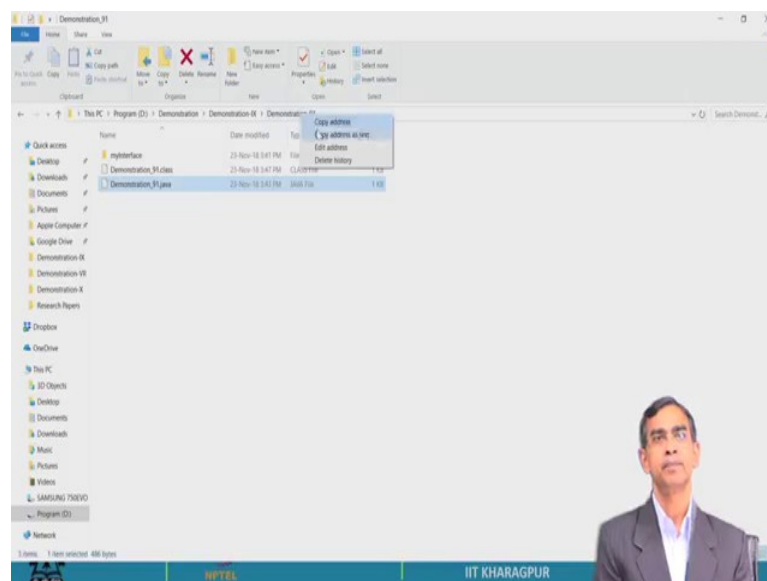
So, for this we can write one program here as we know an in interface can be used if you want to use this interface then a class should implement it is. So, this is a first the class declaration the name of the class is Demonstration underscore 91 implements the an

interface which is in the package my interface we have to input this of course. So, this is the input statement as we see in the code at the top.

Now, so this method the method in this class demonstration as the implementation of display because in the interface the method display is there. So, we just implement the code the display here. So, the code includes a simple print statement, fine and as you do not have to do anything the member element so no issue it is here and in this method the main method as we see we create an object of the class Demonstration underscore 91 and then we call the method display is basically interface method it is here.

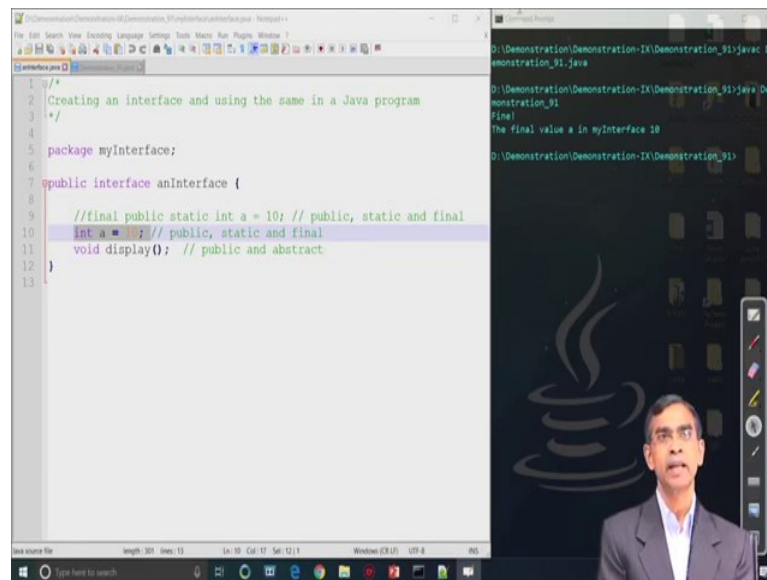
And also in the last statement, you can note it is basically the print state statement in the main method, but it can access the member elements which is already there in the interface method a. So, this is the way this program compares with use and interface and let us run this program.

(Refer Slide Time: 05:20)



Here, in this case, the main program is Demonstration underscore 91. So, if you run it, so, if you run it we will see exactly how it works ok.

(Refer Slide Time: 05:42)

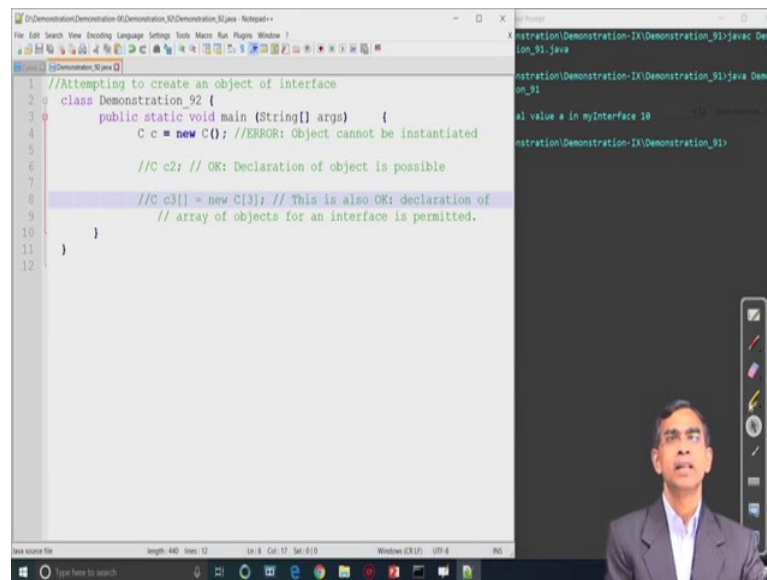


So, the program is compiled successfully. So, this program runs it, right. One thing you can see whenever you create an interface a class file also should be stored there as you see in my interface both the interface in the form of .java file as well as the .class file is also stored there. So, it is the customers as we know the process of package creation both .java file, as well as the class file, should be there.

Now, so in the interface, as you see that thus keywords public then final static void a method is not necessary. So, if you put it that will work fine no problem, but if you do not put it also it will work of it is own no problem. Now, our next demo let us have the next demonstration after knowing how an interface can be created and then how the same interface can be used in the program.

So, our next to the illustration that the whether we can create any object of the interface type? Now, so, in this illustration will give you an idea about. Let us consider the interface.

(Refer Slide Time: 07:03)



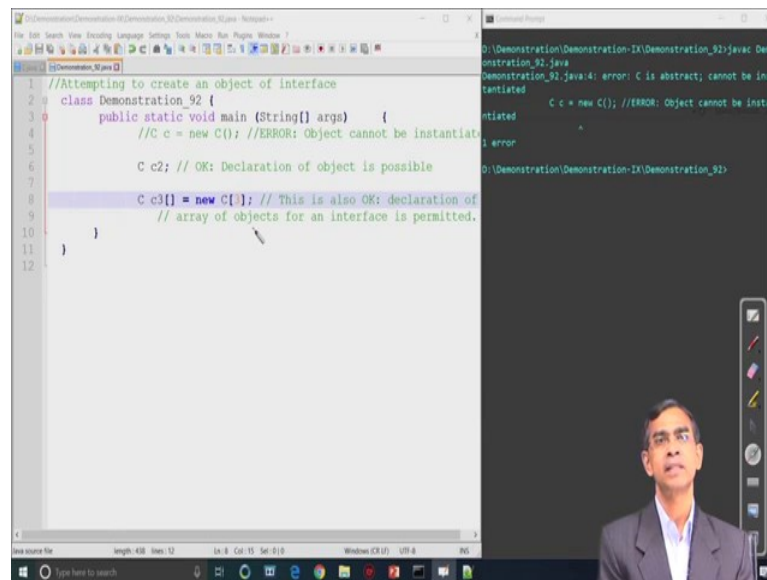
```
1 //Attempting to create an object of interface
2 class Demonstration_92 {
3     public static void main (String[] args) {
4         C c = new C(); //ERROR: Object cannot be instantiated
5
6         //C c2; // OK: Declaration of object is possible
7
8         //C c3[] = new C[3]; // This is also OK: declaration of
9         // array of objects for an interface is permitted.
10    }
11 }
12
```

Let us consider an interface C here. So, this is the interface ok, built it here. It can be stored in any subdirectory, in the same directory so, this is the interface. Now, our objective is to if we attempt to create an object of this interface so, what will happen? Now, let us see the program that we can write it a class program maybe and then we want to create an object of this.

So, let us have the class program here. So, this is a class program the class file name of the file is Demonstration underscore 92 and here you see the main method. The main method is basically in the main method we try to create objects of the type interface there are many ways we can create. So, in the first statement as we see the C is the interface type and small c is an object name-new C. Here we want to create an object of the interface C. As we told you as interfaces is an abstract so, no object can be created. So, definitely, this will leads to an error.

And, then so, the last two statements you can make it comment ok. So, we will discuss about the last one. Now, let us see if we do it whether we can do it or not.

(Refer Slide Time: 08:42)



The screenshot shows an IDE with two windows. The left window displays the source code for a class named `Demonstration_92`. The code includes a comment about attempting to create an object of interface `C`, followed by a `public static void main` method. Inside the method, there are three lines of code: a commented-out `C c = new C();`, a declaration `C c2;`, and an array declaration `C c3[] = new C[1];`. The right window shows the compilation output, which includes error messages for the commented-out line and the array declaration, stating that `C` is abstract and cannot be instantiated.

```
//Attempting to create an object of interface
class Demonstration_92 {
    public static void main (String[] args) {
        //C c = new C(); //ERROR: Object cannot be instantiated
        C c2; // OK: Declaration of object is possible
        C c3[] = new C[1]; // This is also OK: declaration of
        // array of objects for an interface is permitted.
    }
}
```

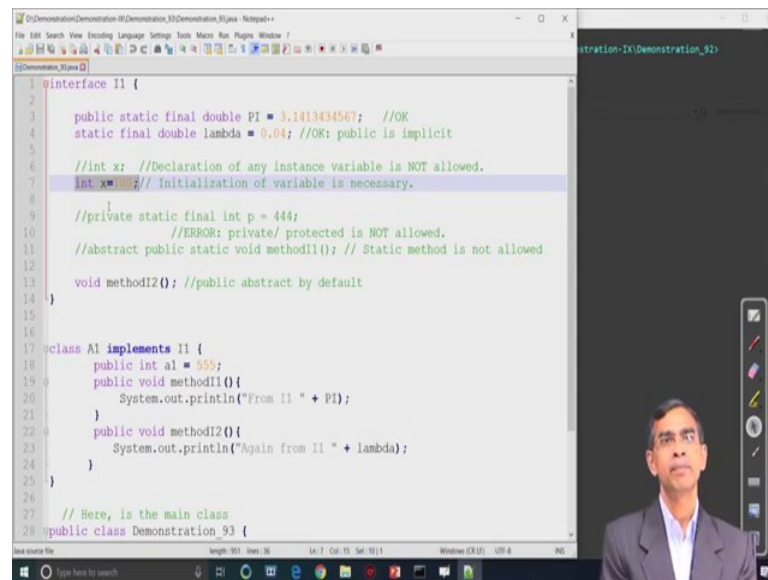
```
D:\Demonstration\Demonstration-IX\Demonstration_92\javac Demonstration_92.java
Demonstration_92.java:4: error: C is abstract; cannot be instantiated
        C c = new C(); //ERROR: Object cannot be instantiated
        ^
1 error
D:\Demonstration\Demonstration-IX\Demonstration_92>
```

Now, as you see here is a compilation error in the statement where we have attempted to create an object of interface `C` so, it is not possible. So, now, let us again revise the program so that we can comment on this statement and then now the other two statement let us see. Now, the other two uncomment it so, other two statements uncomment, nowhere again. So, here `C` is an interface type and it create an object `c 2`, this is quite ok.

Here basically we so, there is two difference between the first statement and the this second statement. The second statement basically creates an object and instantiation by calling the constructor, but here we just simply declare an object `c 2` of `C` type. So, this is quite possible we are declaring only, but not creating a new object, no instantiation is involved. In our second case also we can declare an array of types `C` through an array of the interface object, but not any instantiation. So, what we can say is that instantiation of an interface type is not possible however, declaration of interface object is possible.

So, in that sense the last two statement is correct. Now, let us see if you run it. So, this program run successfully this means that we can create it. Now, we will see the usage of this kind of declaration in our next example we will discuss it, not now ok in due time.

(Refer Slide Time: 10:30)



```
1 interface I1 {
2
3     public static final double PI = 3.1413434567; //OK
4     static final double lambda = 0.04; //OK: public is implicit
5
6     //int x; //Declaration of any instance variable is NOT allowed.
7     int x=100; // Initialization of variable is necessary.
8
9     //private static final int p = 444;
10    //ERROR: private/ protected is NOT allowed.
11    //abstract public static void methodI1(); // Static method is not allowed
12
13    void methodI2(); //public abstract by default
14 }
15
16
17 class A1 implements I1 {
18     public int a1 = 555;
19     public void methodI1(){
20         System.out.println("From I1 " + PI);
21     }
22     public void methodI2(){
23         System.out.println("Again from I1 " + lambda);
24     }
25 }
26
27 // Here, is the main class
28 public class Demonstration_93 {
```

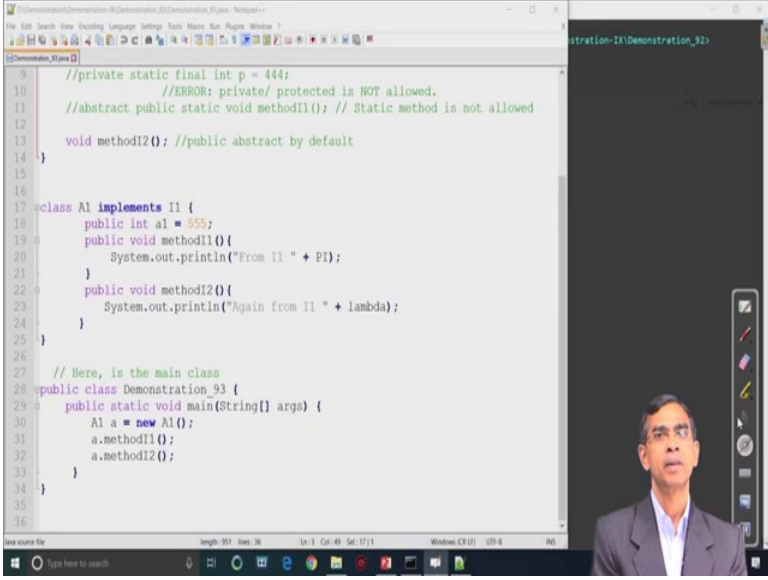
So, now we have discussed about the interface; how a interface can be created and then how the same interface can be used in our class program. Now, as you have told that an interface should be implemented and in the process of implementing all the methods which are there in an interface should be implemented successfully. If you do not implement any method or if we implement one method as a private or something other type then it should give an error.

So, now, let us have the complete program here. Here in this code we see the first few declarations about Interface I1. In this interface, we declare one field called PI and declared public static final that is ok. Also, we declare another member element lambda the floating type. Now, so, this is fine and then also in this method in this interface we declared on a public abstract method by default namely the void method 2; method I2 here ok.

So, all these things are perfectly it will work because is as per the specification of the interface declaration. Now, as we know in the case of the interface only the member that will be declared as a static no instance variable is not possible. Now, let us see if you want to declare a variable say x of type integer and it is declared as 100 and so, these basically considered as an instance variable. So, this leads to an error if we run this program. Fine, we run it, later on, let us have a few more discussions about it here.



(Refer Slide Time: 12:12)

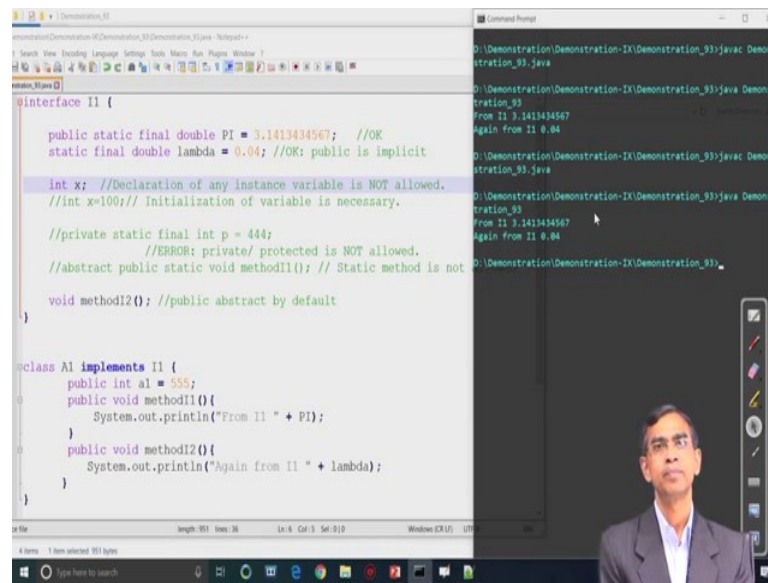


```
9 //private static final int p = 444;
10 //ERROR: private/ protected is NOT allowed.
11 //abstract public static void methodI1(); // Static method is not allowed
12
13 void methodI2(); //public abstract by default
14 }
15
16
17 class A1 implements I1 {
18     public int a1 = 555;
19     public void methodI1() {
20         System.out.println("From I1 * + PI");
21     }
22     public void methodI2() {
23         System.out.println("Again from I1 * + lambda");
24     }
25 }
26
27 // Here, is the main class
28 public class Demonstration_93 {
29     public static void main(String[] args) {
30         A1 a = new A1();
31         a.methodI1();
32         a.methodI2();
33     }
34 }
35
36
```

Now, here class A1; class A1 implements I1 and this is a simple implementation where we have the class A itself its own member A1 and then method 1 it's own and then method 2 is an implementation of the interface method. And as you see as the class A implements I1 all the elements those are there in an interface are readily accessible to this class A. For example, PI this is accessible via the method of its own method I1. And now the main class as we see in the main class here we just create an object of class A1 and then all the methods which basically implements the A1 including the class A1 own method is invoked here.

So, this will run successfully, but before running this program that definitely we should comment the int x. comment int x should be commented here no next one, now fine. So, this program compilable, as well as executable so, let us quickly compile it and then we will come back to this program.

(Refer Slide Time: 13:25)



```
interface I1 {
    public static final double PI = 3.1413434567; //OK
    static final double lambda = 0.04; //OK: public is implicit

    int x; //Declaration of any instance variable is NOT allowed.
    //int x=100; // Initialization of variable is necessary.

    //private static final int p = 444;
    //ERROR: private/ protected is NOT allowed.
    //abstract public static void methodI1(); // Static method is not
    void methodI2(); //public abstract by default
}

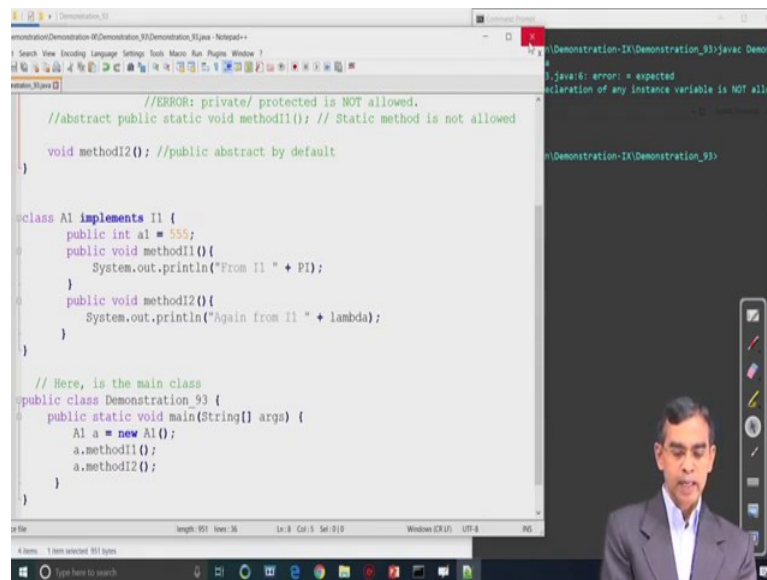
class A1 implements I1 {
    public int a1 = 555;
    public void methodI1(){
        System.out.println("From I1 " + PI);
    }
    public void methodI2(){
        System.out.println("Again from I1 " + lambda);
    }
}
```

```
D:\Demonstration\Demonstration-IX\Demonstration_93>javac Demonstration_93.java
D:\Demonstration\Demonstration-IX\Demonstration_93>java Demonstration_93
From I1 3.1413434567
Again from I1 0.04
D:\Demonstration\Demonstration-IX\Demonstration_93>javac Demonstration_93.java
D:\Demonstration\Demonstration-IX\Demonstration_93>java Demonstration_93
From I1 3.1413434567
Again from I1 0.04
D:\Demonstration\Demonstration-IX\Demonstration_93>
```

Again making the x as an instance variable and then see whether it work or not. So, compilation is successful, no error has been reported and then this is the execution of the program. So, it basically calls method I1 and method I2 from the program itself. As you see method I1 can access the pi which is defined in the interface now let us come back to the program again and here we are in the process of creating the instance variable here.

So, in int right now. So, here int x equals to 100 as we know that this is basically if you do not specify the keyword that is basically public what is called the public static will be default there. What about the int x declaration so, this comment keep it here no problem. this one. Now, let us see that is what this basically creates an instance variable basically run compile.

(Refer Slide Time: 14:46)



```
//ERROR: private/ protected is NOT allowed.
//abstract public static void methodI1(); // Static method is not allowed
void methodI2(); //public abstract by default

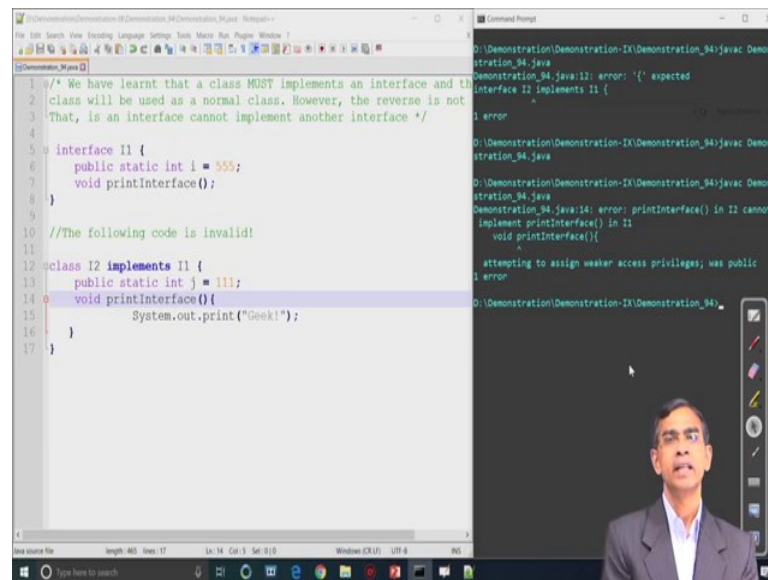
class A1 implements I1 {
    public int a1 = 555;
    public void methodI1() {
        System.out.println("From I1 " + PI);
    }
    public void methodI2() {
        System.out.println("Again from I1 " + lambda);
    }
}

// Here, is the main class
public class Demonstration_93 {
    public static void main(String[] args) {
        A1 a = new A1();
        a.methodI1();
        a.methodI2();
    }
}
```

Now, here we can see `int x` without any initialization it indicates that it is basically as it works as an instance variable. So, if we declare without any `public static final` keyword it will automatically specify this one but this is only applicable after the initialization. You should if it is a static variable you should initialize this one. As there is no initialization it is as an instance variable and no instance variable declaration is allowed in an interface method. So, this example explains that we cannot declare any instance variable this one.

Now, here so, we have learned about that a class can implement an interesting and interface. Now, it is interesting to note whether an interface can implement another interface or not.

(Refer Slide Time: 15:38)



```
1  /* We have learnt that a class MUST implements an interface and the  
2  class will be used as a normal class. However, the reverse is not  
3  That, is an interface cannot implement another interface */  
4  
5  interface I1 {  
6      public static int i = 555;  
7      void printInterface();  
8  }  
9  
10 //The following code is invalid!  
11  
12 class I2 implements I1 {  
13     public static int i = 111;  
14     void printInterface() {  
15         System.out.print("Geek!");  
16     }  
17 }
```

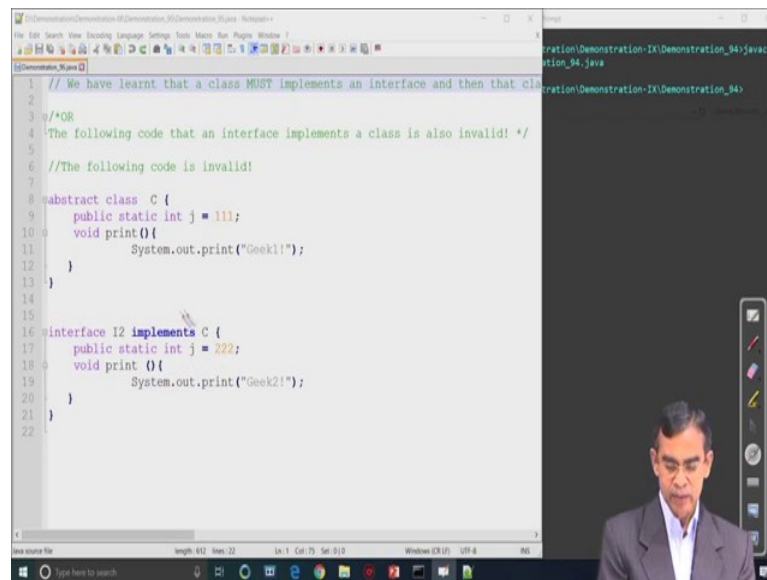
```
D:\Demonstration\Demonstration-IX\Demonstration_84\javac Demonstration_84.java  
Demonstration_84.java:12: error: '{' expected  
interface I2 implements I1 {  
^  
1 error  
D:\Demonstration\Demonstration-IX\Demonstration_84\javac Demonstration_84.java  
Demonstration_84.java:14: error: printInterface() in I2 cannot  
implement printInterface() in I1  
    void printInterface(){  
    ^  
attempting to assign weaker access privileges; was public  
1 error  
D:\Demonstration\Demonstration-IX\Demonstration_84\
```

So, here is an example as we see here I1 is an interface declared with its own member I as 555 and its own public abstract method namely print interface and here is a second is there I2 interface basically attempts to implements I1. In fact, all the entire course is invalid code because I interface I2 cannot implement another interface. Now, if we run this program then it will report a compilation error.

As you see in the statement interface I2 implements I1 it basically is an error indicated here. So, instead of interface I2 if we write class I2 implements I1 it will work possibly and another thing is that in the last statement print interface if we made it we have to make all the method that needs to be implemented as a public. Suppose if we declare as default the print interface method in class I2 yes, just write public term remove ok, fine. So, here it is a default.

Now, let us see no default method implementation is allowed or method implementation with any other access specifier is not allowed. As we see here in the void interface it gives an error so, we have to make it public there then only it will be there. Now, a class can implement another interface if a class implements any methods which are there in an interface that methods should be declared as a public that we have learned it here ok, right. So, we can run it.

(Refer Slide Time: 17:52)

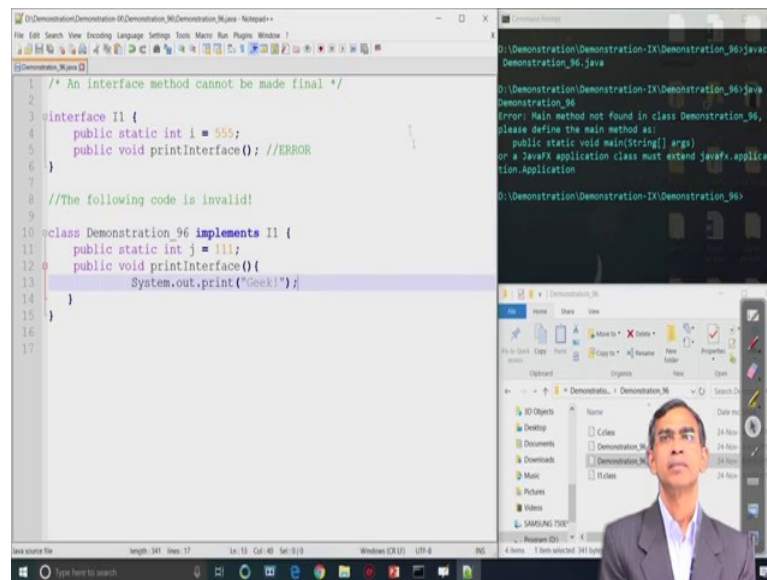


```
1 // We have learnt that a class MUST implements an interface and then that class
2
3 /*OR
4 The following code that an interface implements a class is also invalid! */
5
6 //The following code is invalid!
7
8 abstract class C {
9     public static int j = 111;
10    void print(){
11        System.out.print("Geek1!");
12    }
13 }
14
15
16 interface I2 implements C {
17     public static int j = 222;
18     void print () {
19         System.out.print("Geek2!");
20     }
21 }
22
```

Our next example, whether an interface implements an abstract class? As we know an abstract class in many ways similar to instant interface that abstract class also can have the abstract method that means method without any code and then it also can include the static variable. So, here you can see in this example a class C is an abstract class as we have declared and here the static variable is declared as a and it has one method print, but in case of abstract class we know it can include both abstract method as well as non-abstract method; in this case it is non-abstract method.

Now, if I if we attempt to implement interface this class C that abstract class by using implements it is basically invalid. Now, if we run this then you will see this program will not compilable actually. It will give the compilation error as we see interface I2 implement C is basically it is not allowable to implement this one ok. So, no abstract class also can be implementable using any interface, but a class can be a plan to implement this one.

(Refer Slide Time: 19:19)



Now, one thing is that no method or any variable can be declared final in case of interface it will lead to an error. Here is an example. Here we can see interface I1 which has one static variable public final it equals to 55 by default it is final and now, the method here we can declare you can see we have declared the method as a final and static. Note, these two keywords are not applicable to any method declaration in an interface only public and abstract is applicable. So, by default it is abstract so, final static is not possible. So, this code will work only if we remove the final and static from this one.

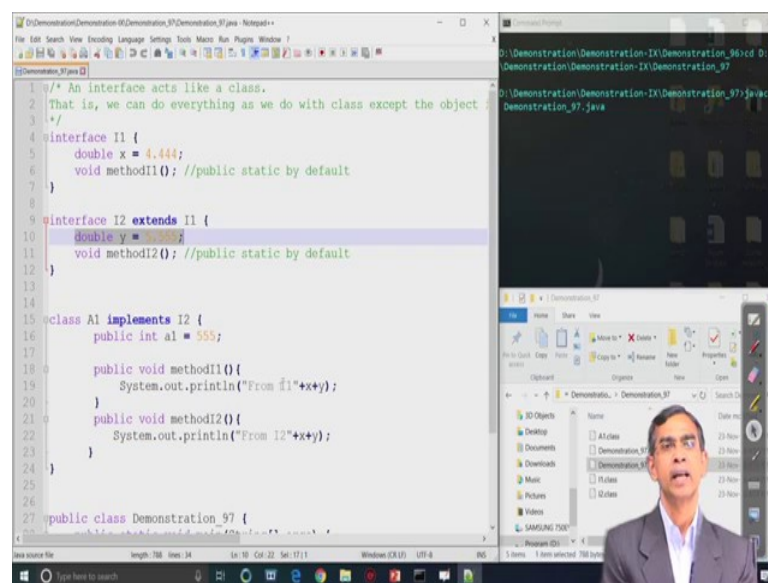
Now, let us have the final static and learn it and one by one if you execute it will see it in the next code basically class C implements I1 it basically declares its own method public and then implements the print interface here. As you see there is an error so, the error is the modifier final not allowed here. Now, let us remove this modifier final and run it again keep it static ok. Now, run it again and let us see whether a static keyword is not there. Here we can see it also reports an error missing method body or declares abstract. As you have declare the abstract it is not taking this one. So, if we remove this static again it will now final, public it is basically, abstract we can write it or not write now it will work. So, void printed you have to declare public so, that is fine.

So, declare the public in the class implementation as we have not declared the public all the implementation of the interface method should be declared as public here ok. So, fine

now this program runs successfully. So, there is a mistake regarding the name of the class file. So, that is why it gives an error that it did not find the class file. Now, we have changed it, yes ok. We are compiling; yes, the compilation is success full in this case running this program anyway.

So, our next illustration anyway. So, the last illustration is not meant for any what is called a class there, only for the explanation that it will require the few properties to be satisfied.

(Refer Slide Time: 22:43)

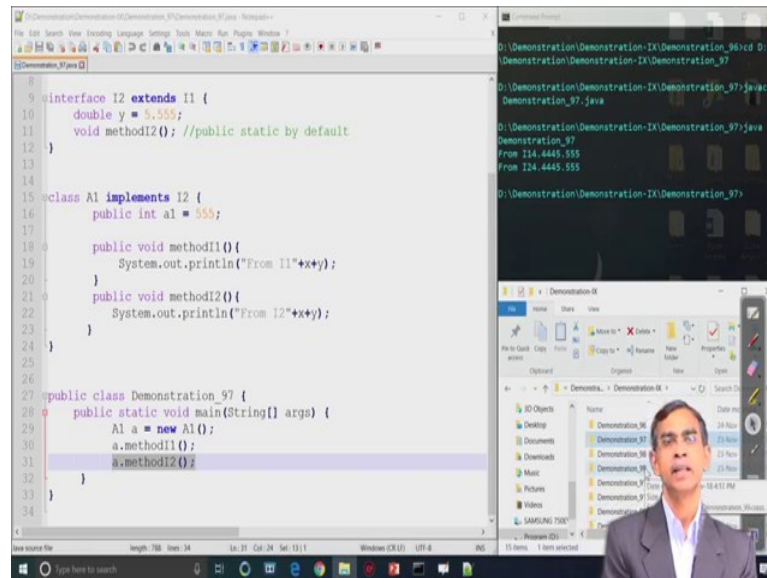


Anyway, so, our next example basically explains the single inheritance. Now, here let us have the loop of the code interface I1 with it is public static final variable x and then it is method 1 and this interface I2 extends I1. So, an interface can be extended from I1, but cannot implements. Now so, extend means I2 inherits I1 and so, as a process of inheritance so, it has its own members y and its own method which is also abstract is declared because it is an interface method. Now, class A1 implements I2 so, whenever I2 comes to this picture. So, by virtual of inheritance all the way method that is there I1 method 1 also now inherited to I2. So, here again, see in this class A1 implements I2 method as we see we declare one variable on its own. This is the class A1 variable and it implements method one which is basically inherited from I1 via I2 and then also method 1 and method 2.



And, as you see here x and y are the two members who belong to the interface I1 and I2 and by the process of implementation class A1 has the readily accessible to these values x and y here.

(Refer Slide Time: 24:18)



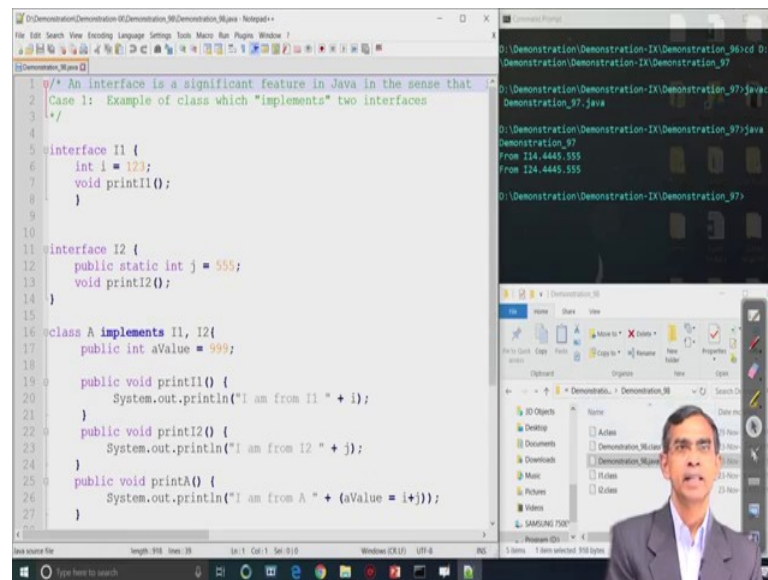
```
8
9 interface I2 extends I1 {
10     double y = 5.555;
11     void methodI2(); //public static by default
12 }
13
14
15 class A1 implements I2 {
16     public int a1 = 555;
17
18     public void methodI1() {
19         System.out.println("From I1*x+y");
20     }
21     public void methodI2() {
22         System.out.println("From I2*x+y");
23     }
24 }
25
26
27 public class Demonstration_97 {
28     public static void main(String[] args) {
29         A1 a = new A1();
30         a.methodI1();
31         a.methodI2();
32     }
33 }
34
```

So, the main program looks like here we create an object of class A 1 and then we call the method, method 1 and method 2. So, if we call so, all the methods and then they are variables those are there in the interface will be readily accessible to this one. So, let us have a quick execution of this program followed by the compilation.

So, we can see yes, we can see that this program is now successfully executed and compilation there is no error there. Now, so, this is the idea about single inheritance; now, let us have the demonstration on multiple inheritances.



(Refer Slide Time: 24:56)



```
1 0/* An interface is a significant feature in Java in the sense that
2 Case 1: Example of class which "implements" two interfaces
3 */
4
5 interface I1 {
6     int i = 123;
7     void printI1();
8 }
9
10
11 interface I2 {
12     public static int j = 555;
13     void printI2();
14 }
15
16 class A implements I1, I2{
17     public int aValue = 999;
18
19     public void printI1() {
20         System.out.println("I am from I1 " + i);
21     }
22     public void printI2() {
23         System.out.println("I am from I2 " + j);
24     }
25     public void printA() {
26         System.out.println("I am from A " + (aValue = i+j));
27     }
28 }
```

Terminal Output:

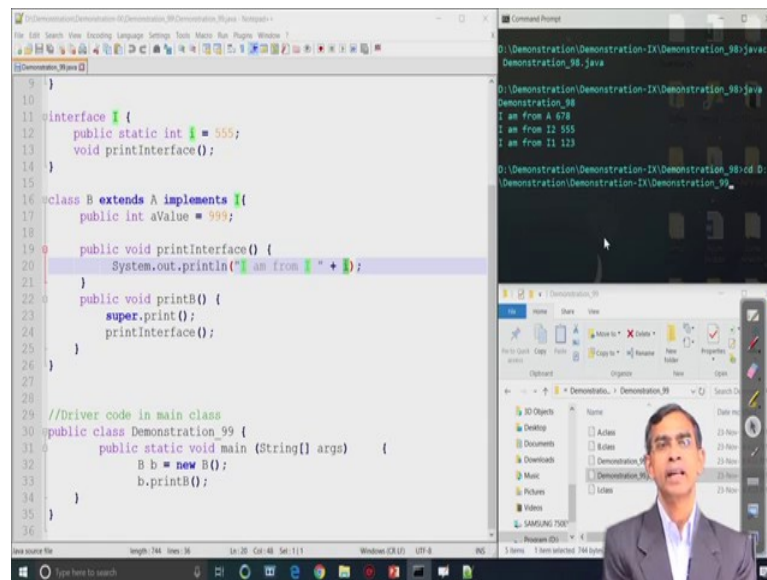
```
D:\Demonstration\Demonstration-IX\Demonstration_96\cd D:\Demonstration\Demonstration-IX\Demonstration_97
D:\Demonstration\Demonstration-IX\Demonstration_97>javac Demonstration_97.java
D:\Demonstration\Demonstration-IX\Demonstration_97>java Demonstration_97
From 114, 4445, 555
From 124, 4445, 555
D:\Demonstration\Demonstration-IX\Demonstration_97>
```

Here again, I1 is an interface as we declare one of the static variable I here and print run method in interface I2 is an again with there is again j as a static variable and print I2 methods are there. Now, multiple inheritance here as we see class A implements I1 and I2. So, this is basically the idea about how to multiply a class that can inherit both from I1 and I2. In the last example, we see that I2 inherits I1 then class implements I2. Here basically class A can implements I1 and I2 here I2 is not necessarily inherited from I1 it basically.

So, here as the number of inheritance can be implemented by a single class; for example, class A implements I1, I2 and I3 and so, on we can write this one. Now, in this class implementation as we see A value is the value of the class variable A and then print 1 is an implementation of class interface I1 print I2 is an implementation of interface method I2 and finally, the print A is a class A methods who basically have the full access of the static variable those are there in interfaces i and j namely.

So, this is the program as we can see the multiple inheritances by the process of multiple inheritances we can access all methods that are there in an interface as well as all variables those are there in the interfaces ok. As we see this program run successfully, now there is an alternative way of doing multiple inheritances our next demonstration is to explain this one.

(Refer Slide Time: 26:47)



```
9 }
10
11 interface I1 {
12     public static int i = 555;
13     void printInterface();
14 }
15
16 class A implements I1 {
17     public int aValue = 999;
18
19     public void printInterface() {
20         System.out.println("I am from " + i + " ");
21     }
22
23     public void printA() {
24         super.print();
25         printInterface();
26     }
27
28 }
29
30 //Driver code in main class
31 public class Demonstration_99 {
32     public static void main (String[] args) {
33         B b = new B();
34         b.printB();
35     }
36 }
```

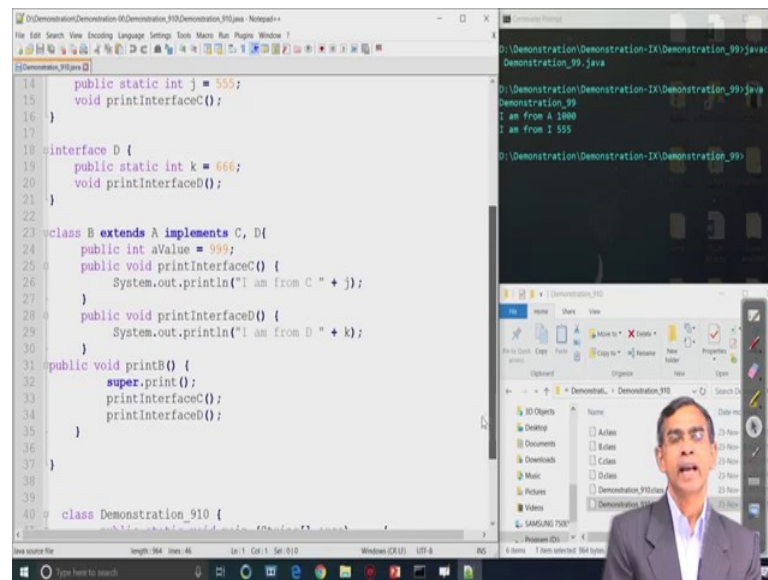
```
D:\Demonstration\Demonstration-IX\Demonstration_98.java
Demonstration_98.java
D:\Demonstration\Demonstration-IX\Demonstration_98.java
I am from A 578
I am from 11 555
I am from 11 123
D:\Demonstration\Demonstration-IX\Demonstration_98\cd D:
D:\Demonstration\Demonstration-IX\Demonstration_99
```

In the last example as we see class A implements I1 and I2. Now, both extends and implements can be added there; this is also one way of inheritance multiple inheritances. In this example as we see class A is a class and interface I is an interface and will see is the class B extends class A; that means, it inherits the superclass A as well as implements I it is also inherited basically the same variable as well as method.

However, this implementation as it is class B should implements I the method print should be defined here in this class as we have seen the public void where is a print interface method. So, as we see interface I the print interface method is being implemented here that print and then call it here and class B basically ok. So, it implements the print interface method in an interface and then print B is basically its own constructor who is basically called the class A constructor; that means, superclass constructor using the super keyword and then also call print interface methods because, this constructor has the access of because of the implementation. And the main class is very simple here. The demonstration underscore 99 is the implementation of this where we can create an object of class B and then we can call the print B the method of class B.

So, this is the one example as we see the multiple inheritances by means of extending I mean inheriting a superclass as well as implementing an interface. Now, there are other cases where it can extend on class A implements two or more interfaces at the same time.

(Refer Slide Time: 28:48)



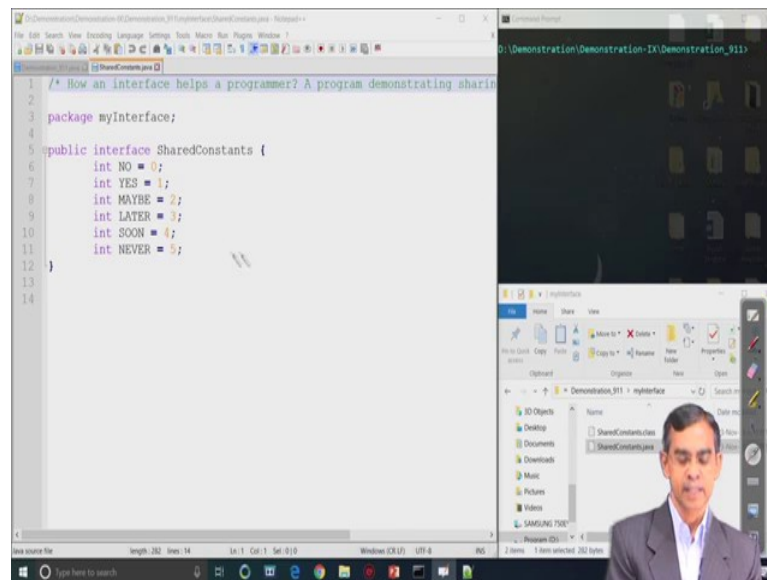
```
14 public static int j = 555;
15 void printInterfaceC();
16 }
17
18 interface D {
19     public static int k = 666;
20     void printInterfaceD();
21 }
22
23 class B extends A implements C, D {
24     public int aValue = 999;
25     public void printInterfaceC() {
26         System.out.println("I am from C " + j);
27     }
28     public void printInterfaceD() {
29         System.out.println("I am from D " + k);
30     }
31     public void printB() {
32         super.print();
33         printInterfaceC();
34         printInterfaceD();
35     }
36 }
37
38
39
40 class Demonstration_910 {
41     public static void main(String[] args) {
42         B b = new B();
43         b.printB();
44     }
45 }
```

```
D:\Demonstration\Demonstration-IX\Demonstration_99.java
Demonstration_99.java
D:\Demonstration\Demonstration-IX\Demonstration_99.java
Demonstration_99
I am from A 1000
I am from C 555
I am from D 666
D:\Demonstration\Demonstration-IX\Demonstration_99.java
```

So, this example is here class A is a class interface C and D are the two interfaces and you see is a multiple inheritance form where class B extends A, that means A is the superclass, class B derived from A as well as is implement C and D that mean class B also inherits from C and D. That means, that all the static variable those are there in C and D namely j and k is also accessible as well as the variable which is protected in class is also accessible. So, we can see this method that is there or this class or main method we can say can access all the elements by means of this multiple inheritance. So, this is a one simple main method as we see here we create an object B which is basically derived based on the multiple inheritance and then call the method print B ok.

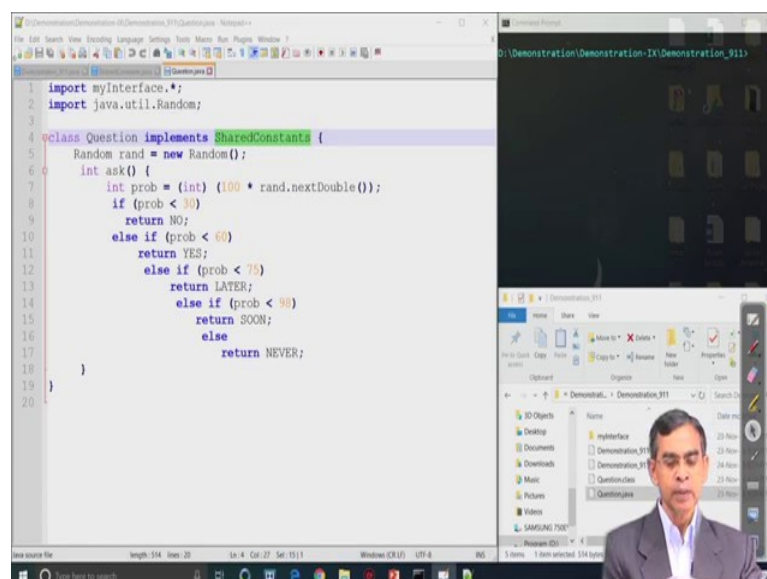
So, this is the different way that we can see that how multiple inheritance is possible with the help of interface as well as class A. Now, what is the usage of this interface we can do the multiple inheritance, this is the one application. Our next example basically explains our next example explain that the usage of the interface; in fact, an interface can help to support the shared variable to use across the many packs' many packages or many classes in different packages.

(Refer Slide Time: 30:38)



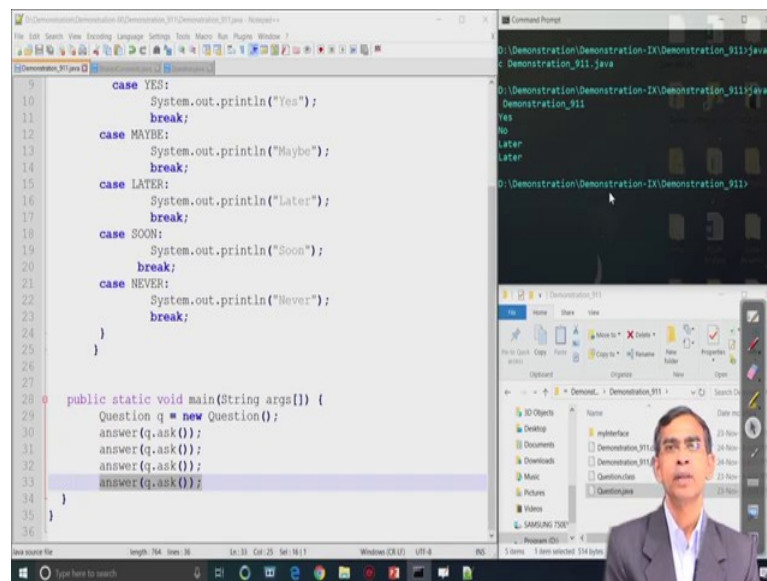
So, here is an example as you see the interface includes declarations of some variable go to the interface in the package my interface go to the my interface. So, we have declared one interface here. This interface includes some static variable which is declared here; NO, YES, MAYBE this one. Now, so once this interface is declared and then stored in the package, then any other class can use them as a global variable look like so; that means, it is called as shared variable in that sense.

(Refer Slide Time: 31:10)



Now, here is an example of one class which uses basically interface; that means the class question implements all this shared constant to the basic global variable we can say. And in this case is basically you see without any declaration it basically used return YES, return LATER, return SOON all these things and here is a simple code we can use the random function which is defined in java.util it will basically generate a random number and based on the random number generation it will print all these values depending on the random probabilities.

(Refer Slide Time: 31:34)



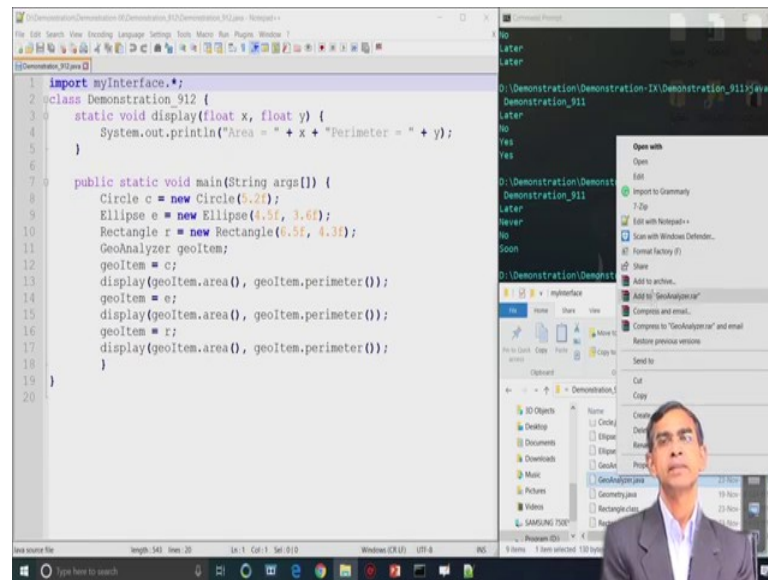
```
9      case YES:
10         System.out.println("Yes");
11         break;
12      case MAYBE:
13         System.out.println("Maybe");
14         break;
15      case LATER:
16         System.out.println("Later");
17         break;
18      case SOON:
19         System.out.println("Soon");
20         break;
21      case NEVER:
22         System.out.println("Never");
23         break;
24      }
25  }
26
27
28  public static void main(String args[]) {
29      Question q = new Question();
30      answer(q.ask());
31      answer(q.ask());
32      answer(q.ask());
33      answer(q.ask());
34  }
35  }
36  }
```

Now, here this is another class that basically uses the same things here and we define another method called answer and it basically take the results first do it and based on the result it will execute this codes are there. Now, let us have a quick method. In this class is as we see go to the next, so this method basically creates an object of question class which is declared which use the interface or implements the interface.

And then for this class, we use the ask method. Ask method generate a random number based on the random number it will print either YES, No, SOON, NEVER, maybe all like this one. If we run this program let us see what is the output it will give. It will give the output as it is a probabilistic one. So, different execution will keep you not necessarily give the same output in this case ok. In this case, we can see this is the output. If we run the same program again we may see different output like this one if we run again it can give another output and so on.

So, it is basically a probabilistic program by means of generating a random number, but it will be discussing about how the shared variable can be used across the different classes. Now, we have to discuss in the ok, what is the difference between interface object and then abstract class interface type and then abstract class declaration.

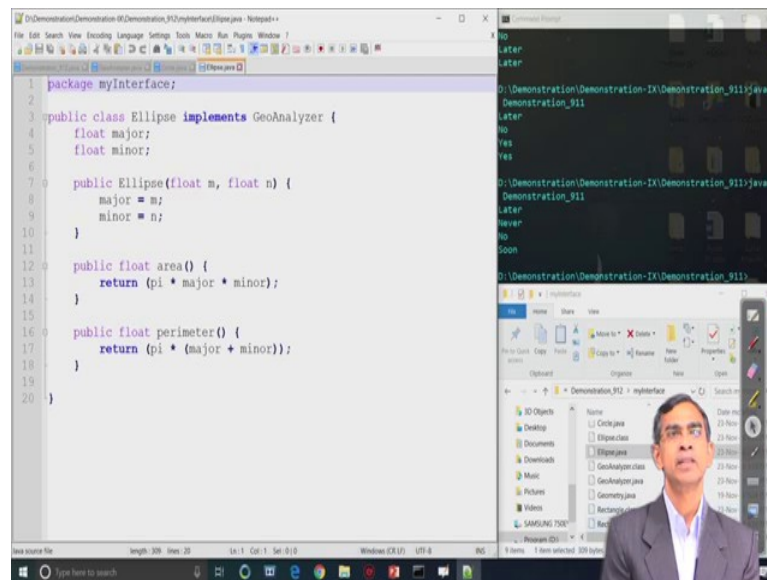
(Refer Slide Time: 33:09)



Now, we have let us have the one program here and it is a basic idea about the inheritance and by means of inheritance we can create many objects there and let us go to the main class ok. First, discuss about the interface class here. Interface class goes to the interface class interface go to the interface, fine no not this is the interface right ok.



(Refer Slide Time: 33:44)



So, now, this interface is defined in my package in the package myInterface. The name of the interface is GeoAnalyzer as you see this geo-interface geo analyzer interface has one static variable namely pi and then two public abstract method areas and perimeter. So, this method is basically the type and now we want to create few classes which implement GeoAnalyzer namely circle, ellipse and rectangle. So, this is the implementation of circle class which implements GeoAnalyzer; this is the implementation of the ellipses class which is an implementation of GeoAnalyzer and GeoAnalyzer this is the implementation of rectangle class which is the implementation of the GeoAnalyzer it has its own this one and this one ok.

So, now, we see the different class implements the GeoAnalyzer interface by the different methods actually in their own way; circle has its own implementation, ellipse has its own implementation, rectangle has its own implementation. Once this implementation is done now we can come to the main program which basically create the objects of this.

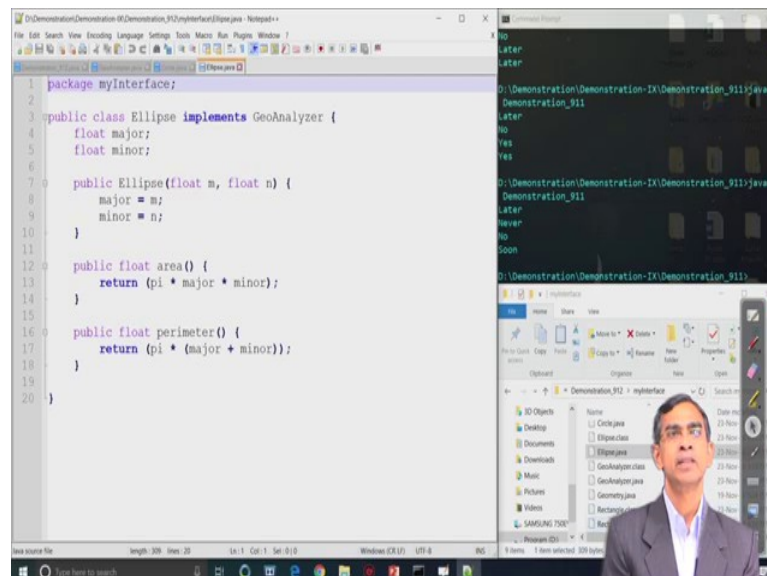
Now, here we can see the demonstration 912 is the main class and there is basically the method includes a display which basically take x and y, namely the area and perimeter of the geo objects. And here we can see we have created three objects; one object c of type class c, another object e of class Ellipse and another object r of class Rectangle.

Now, here Geoanalyzer geo-Item the interface and geo-Item is an object of interface we can just declare. So, we have declared and then once it is declared we see it basically holds the class c, basically the upcasting c is equal to geo item. Now, if we do this then we can call the display method which is defined in the main method by calling this geo-Item.area and geoItem.perimeter. Similarly, if the geoItem holds the reference to the ellipse object, then also same method can print it. But, in this case, this geo the display geoItem area for the referencing of ellipse object call the method area which is declared in ellipse class.

And, then again if it refers to rectangle and then displays area of this one, it also refers to the methods that are there in an interface rectangle in the class rectangle. Now, here we can see the display method is basically binded polymorphism. The different the display methods can display area, perimeter for the different objects as per the references there. So, it is a great example of polymorphism by means of interface.

Now, this program if we run it will work for us, now again repeat the same thing, but using the abstract class. So, in the previous case, we have discussed about Geoanalyzer as an interface we just want to do the same thing by means of abstract.

(Refer Slide Time: 37:08)



Here is basically this abstract class is declared in a package let us have the package it is there ok. So, this abstract class is declared here go to the abstract class their geo geometry. So, here we can see public abstract class geometry is an abstract class here and

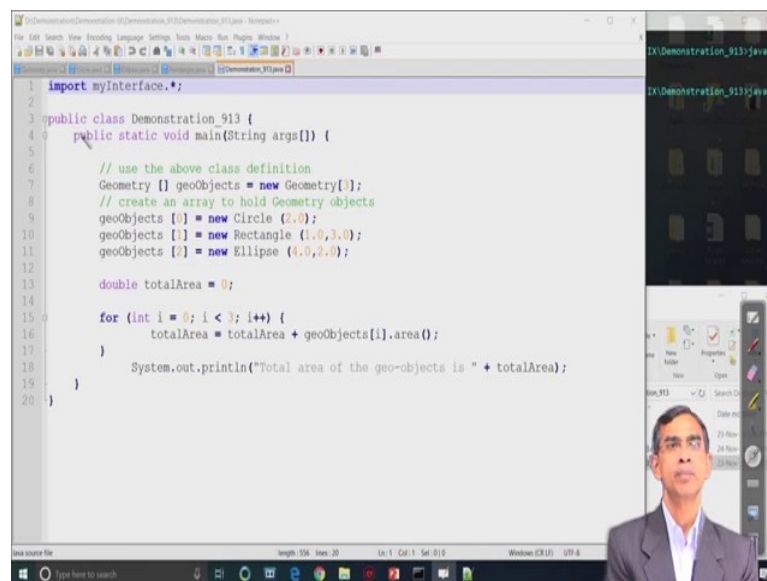


here we define two methods and abstract method abstract as you know the abstract method means no code is there just look like a very similar to an interface.

Now, this basically example to illustrate the similarity and dissimilarity between interface and abstract it is there. Now, let us have the second same way of class implementation it is basically extended because circle should extends geometry here we can see circle. Now, the difference we can note in the previous case circle implements Geoanalyzer, but here geometry being an abstract class we cannot implement, we just simply circle extends geometry. Likewise, Ellipse extends geometry and then Rectangle extends geometry as a process of extension is basically single inheritance as you know.

So, it basically inheritance or you can say override the method those are there in abstract method incase of geometry. So, these are the three implementations of three classes extending the geometry object and finally, the main class here demonstration underscore 9.13 so, this is the main class.

(Refer Slide Time: 38:35)



```
1 import myInterface.*;
2
3 public class Demonstration_913 {
4     public static void main(String args[]) {
5
6         // use the above class definition
7         Geometry [] geoObjects = new Geometry[3];
8         // create an array to hold Geometry objects
9         geoObjects [0] = new Circle (2,0);
10        geoObjects [1] = new Rectangle (1,0,3,0);
11        geoObjects [2] = new Ellipse (4,0,2,0);
12
13        double totalArea = 0;
14
15        for (int i = 0; i < 3; i++) {
16            totalArea = totalArea + geoObjects[i].area();
17        }
18        System.out.println("Total area of the geo-objects is " + totalArea);
19    }
20 }
```

As you see the main class here it is more or less similar to the previous example user interface. Geometry we create we basically declare an array of what is called the objects of geoObjects of type here three array size is three here arrays of the abstract class method we can say just the interface objects we have created in case of interface in case of abstract class also the object can be created as we see here. And, then geoObject 0 we create an instance of the class circle and then it is basically assigned to the location 0 and

similarly a rectangle instance and ellipse instance are created and we call the we pass the parameter to I mean instance share them properly.

So, these are the instantiation and instantiation, but after the instantiation they are basically referred in the abstract class object like geoObject in this case and here again runtime polymorphism as we see in the next for a loop. So, here total area geoObjects i.area. Now, for i equal to 0 so, this will this .area refers to circle objects on the other hand for i equal to 1, this area refers to the Rectangle objects and then for i equal to 2 this area refers to Ellipse object. So, again it is an example of runtime polymorphism. Because it is polymorphically resolved that different method depending on the objects it is there although it looks like same for all calls actually and if you run it the similar to the interface it will run and then give the execution ok.

So, this is the right. So, this is the successful execution of the program. So, what we have learned about here that more or less interface and abstract behaves in the same manner, then why java developer maintains both the thing? The difference only here is that abstract class if you declare it cannot be multiple inheritance or whenever an inheritance interface is there it can be multiple inheritance. So, this is the only difference between the abstract class and the interface otherwise both the concepts are more or less same.

So, we have learned about the interface and if you have further any doubt you are most welcome to post your queries and all the programs that we have used in their demonstrations you are free to use it and then test it so that you can practice it much more.

So, thank you very much.