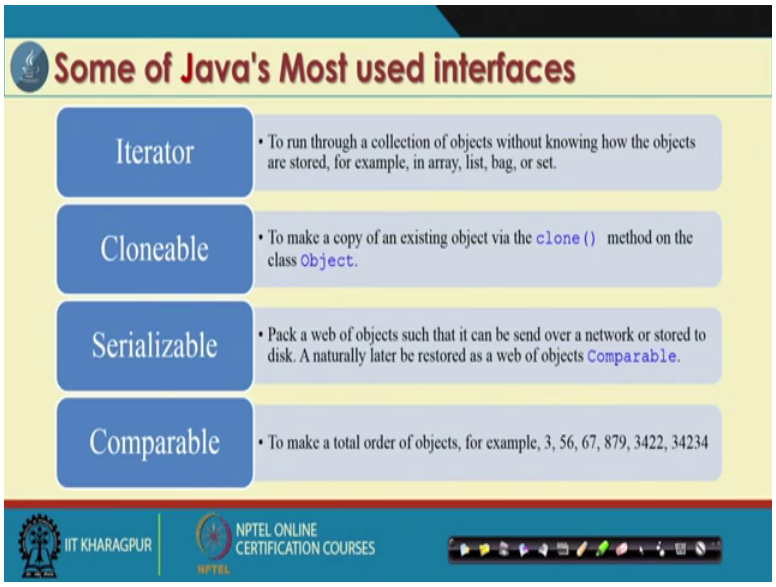
Programming in Java

Prof. Debasis Samanta
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Lecture – 21
Interface -II

So, this is the second part of our discussion on interface, in the last module we have discussed the Interface and basically introduces the basic concepts that is there in interface in Java. So, today we will discuss some more advanced features are there in the interface and finally, we will summarize the lessons that we have learnt so far the interface is concerned. So, there are like different classes built in classes. So, there are also some standard interfaces available in Java.

(Refer Slide Time: 00:52)

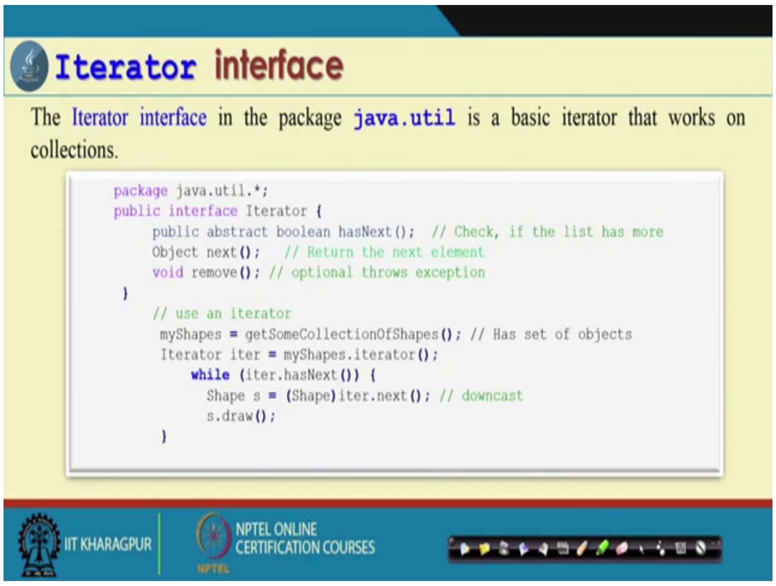


Some of Java's Most used interfaces

Iterator	• To run through a collection of objects without knowing how the objects are stored, for example, in array, list, bag, or set.
Cloneable	• To make a copy of an existing object via the `clone()` method on the class `Object`.
Serializable	• Pack a web of objects such that it can be send over a network or stored to disk. A naturally later be restored as a web of objects `Comparable`.
Comparable	• To make a total order of objects, for example, 3, 56, 67, 879, 3422, 34234

So, the there are few of course not many. So, the important the mostly used interfaces those are already defined they are called the built in interfaces like here, iterator, cloneable, serializable, comparable, these are the most frequently interfaces. So, in our subsequent few slides we will go through each interface quickly and then understand what they can do for us.

(Refer Slide Time: 01:23)



Iterator interface

The Iterator interface in the package **java.util** is a basic iterator that works on collections.

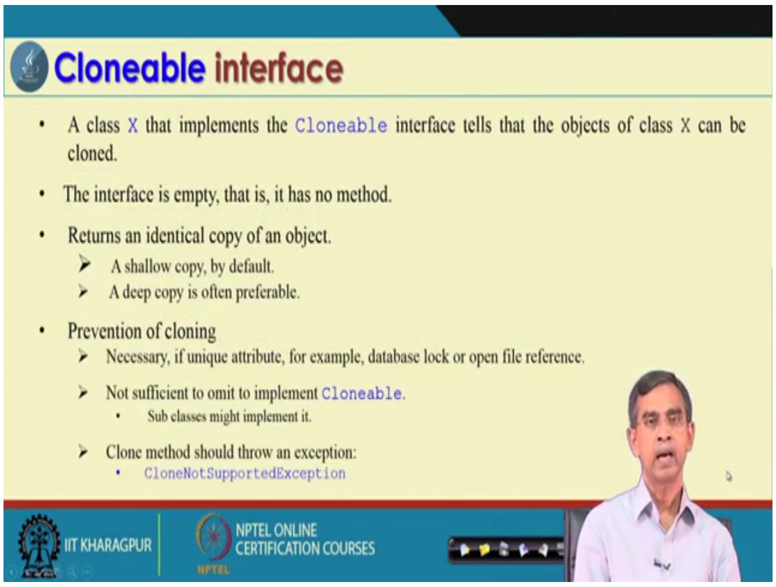
```
package java.util.*;
public interface Iterator {
     public abstract boolean hasNext();  // Check, if the list has more
     Object next();   // Return the next element
     void remove(); // optional throws exception
 }
     // use an iterator
      myShapes = getSomeCollectionOfShapes(); // Has set of objects
      Iterator iter = myShapes.iterator();
          while (iter.hasNext()) {
            Shape s = (Shape)iter.next(); // downcast
            s.draw();
      }
```

So, let us have the discussion on iterator. So, iterator interface is basically if there is a collection of objects right and then how these collection of objects can be processed. So, regarding these things the iterator interface which has been defined in java dot util package. So, in the java dot util package this interface basically help us to manage the set of objects we can say the collection of objects.

Here the interface the name is iterator as we see here this is the name of the Interface Iterator which has few methods as we see here has next the next and the remove. So, there the 3 standard methods are there in iterator and you can use this interface to implement our own class here is an example for example, how these interface can be used in implement our own class. Here we can create we can just implement using the concept this is the class this may be say example Iterator implements Iterator and then here myShapes is the shape of some objects that you want to have it in the shape geometry and then this is the iterator object is created.

Now here if you see for this Iterator, Iterator dot hasNext; that means, it will automatically check the shapes the collection of objects and then it will check that whether the next in the list of collection the element is there or not. So, here hasNext is basically the method that is there in the interface is basically implements it and then this is the implementation how it will work there. So, hasNext next similarly remove is basically another interface to remove an object from a collection. So, this is the one example of iterator interface that is there in java more specifically in java dot util package.

(Refer Slide Time: 03:27)



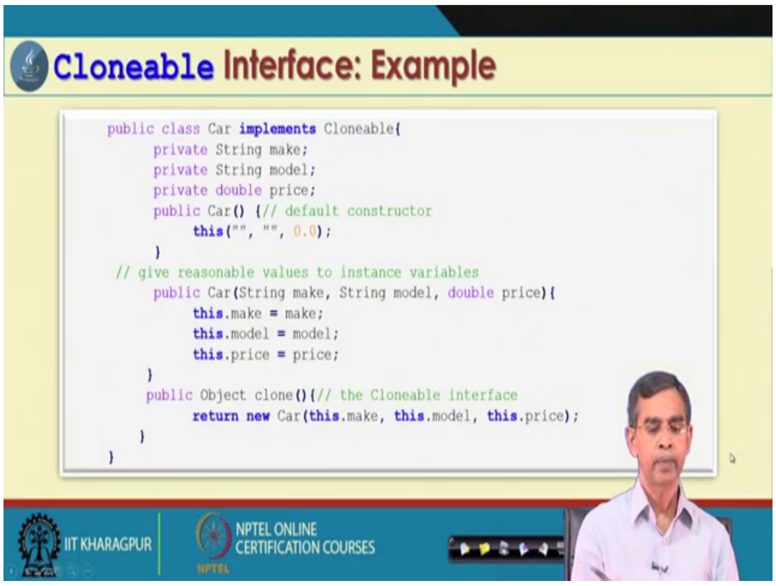
Cloneable interface

- A class `X` that implements the `Cloneable` interface tells that the objects of class `X` can be cloned.
- The interface is empty, that is, it has no method.
- Returns an identical copy of an object.
  - A shallow copy, by default.
  - A deep copy is often preferable.
- Prevention of cloning
  - Necessary, if unique attribute, for example, database lock or open file reference.
  - Not sufficient to omit to implement `Cloneable`.
    - Sub classes might implement it.
  - Clone method should throw an exception:
    - `CloneNotSupportedException`

Now, cloneable is another interface is basically this interface is basically to implement a class which can create a copy of an object. So, it is a cloneable means it will basically make a copy of the objects, as you see here the his interface. In fact, is a new application it does not have any method it is basically empty and it basically helps us to have the copy of an objects their copy maybe 2 types; one shallow copy and deep copy. If the shallow copy it will just logically make a copy and the deep copy means physically make a copy; that means, for all objects reference variables, class variables, instance variable, everything they will make a separate instances or copy of the same object. So, 2 objects will be created having the same things are there.

In some situations the duplicate (Refer Time: 04:26) of the objects needs to be controlled and so, cloneable interface can be called for this purpose and in case suppose this interface does not work it throws exception. So, this exception regarding we will learn about it. So, anyway if there is any mistake regarding the copying an objects, sometimes there may not be any permission or copy is not successful, whatever it is there in that case it will throw an exception which is for the preparing the robust program actually. So, this is the cloneable interface and then example here.

(Refer Slide Time: 05:05)



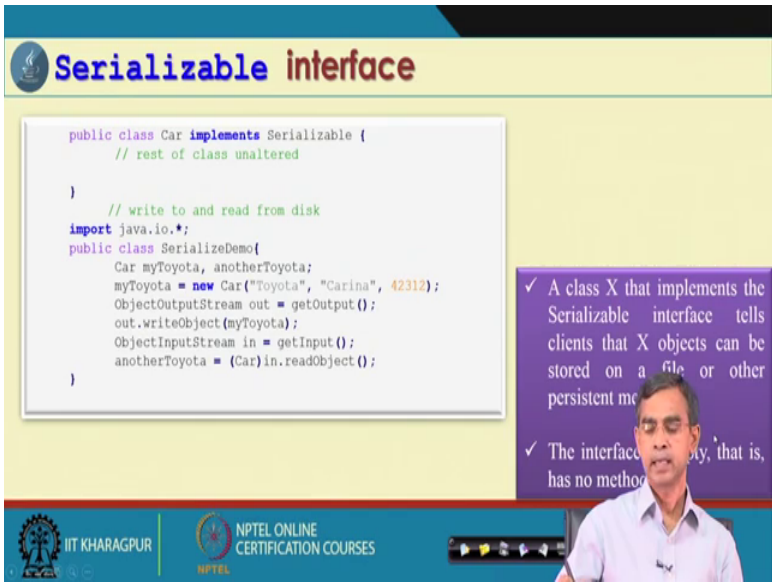
Cloneable Interface: Example

```
public class Car implements Cloneable{
      private String make;
      private String model;
      private double price;
      public Car() {// default constructor
           this("", "", 0.0);
      }
  // give reasonable values to instance variables
      public Car(String make, String model, double price){
           this.make = make;
           this.model = model;
           this.price = price;
      }
     public Object clone(){// the Cloneable interface
           return new Car(this.make, this.model, this.price);
     }
}
```

We can consider one example here this example is basically show how the cloneable interface can be used here.

Say for example, here basically the Car is a class which implements the Cloneable these are the methods in this class Car and then here you can see these objects we create a objects of type and then clone is the method that we have discussed here and this method is basically make a copy, explicitly that is mean by the class declaration by the user. So, this is the one example how the Cloneable interface can be used in Java class declaration.

(Refer Slide Time: 05:44)



Serializable interface

```
public class Car implements Serializable {
      // rest of class unaltered

}
     // write to and read from disk
import java.io.*;
public class SerializeDemo{
      Car myToyota, anotherToyota;
      myToyota = new Car("Toyota", "Carina", 42312);
      ObjectOutputStream out = getOutput();
      out.writeObject(myToyota);
      ObjectInputStream in = getInput();
      anotherToyota = (Car)in.readObject();
}
```

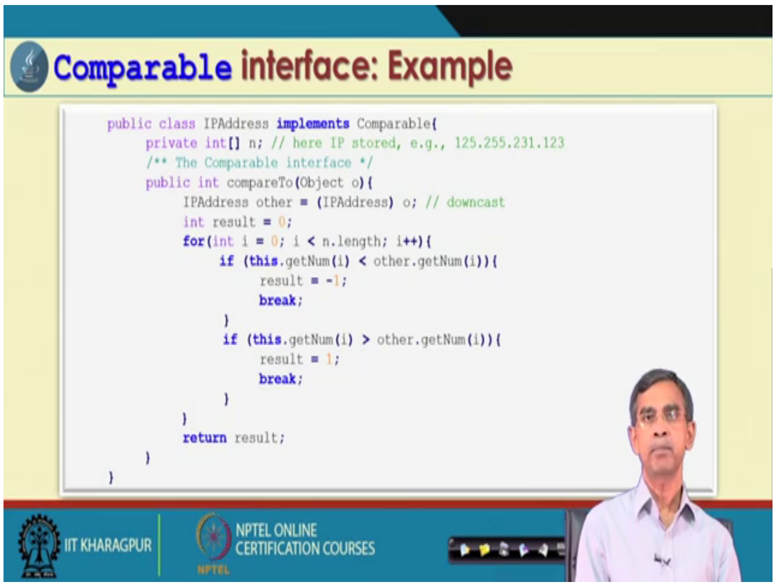
- ✓ A class X that implements the Serializable interface tells clients that X objects can be stored on a file or other persistent me
- ✓ The interface ty, that is, has no metho

And the next is serializable interface, this interface is basically helps a programmer to build their classes while they want to communicate or send some objects over a network to a distance objects or distant pc distant server whatever it is there.

So, this basically helps to make the things called the serializable and here is a quick example of the serializable in again from the Car is the class support it implements serial serializable interface whatever the methods we have discussed in the next class we can copy it verbatim here. So, this will complete the class declaration and here is an example how the Serialization Serializable interface can be used in the machine. Here you can see we create the object for writing or reading, from the writing from the net network channel or reading into the network channel or writing into some file or reading from the file.

So, these basically all the methods that we have discussed that is there as an abstract method interface can be redefined here in the class declaration and can be used here. So, this is an example of serializable interface and then finally, the idea about the comparable interface.

(Refer Slide Time: 07: 08)



Comparable interface: Example

```
public class IPAddress implements Comparable{
    private int[] n; // here IP stored, e.g., 125.255.231.123
    /** The Comparable interface */
    public int compareTo(Object o){
        IPAddress other = (IPAddress) o; // downcast
        int result = 0;
        for(int i = 0; i < n.length; i++){
            if (this.getNum(i) < other.getNum(i)){
                result = -1;
                break;
            }
            if (this.getNum(i) > other.getNum(i)){
                result = 1;
                break;
            }
        }
        return result;
    }
}
```

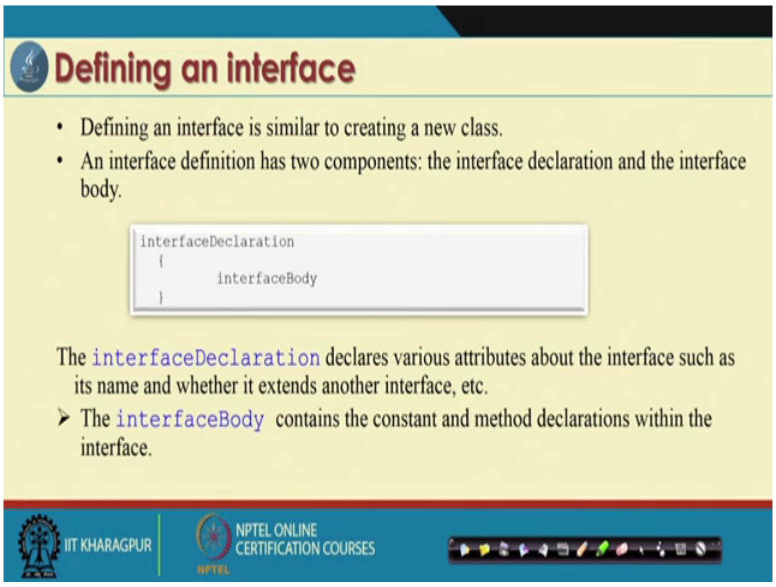
IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

So, here is an example of comparable means sometimes it is required to compare 2 objects whether they are same or different. So, if it is like this. So, Java developer has proposed one interface called the comparable interface and here is an example of this compare interface and we see how it the method that is compared to which is basically interface method we redefined here in our class declaration.

So, this is basically the user defined redeclaration for the interface method and it basically compare objects. So obviously, it is up to the user for comparing the different objects how you can compare one objects with other objects belong to the same class whatever it is there. So, this will written Boolean values whether true or false like. So, this is the interface comparable that is defined in the again all the this is also defined in the java dot util dot package.

Now so, these are the few standard interface usually programmer prefer to have their own interface related to the particular project and use them in their program, other than using the standard interface those are there in the Java system. Now this basically covers all the basic concepts about the interface and before concluding this interface concepts I just want to highlights few more important things which is very important to remember whenever you are handling with interface.

(Refer Slide Time: 08:45)



Defining an interface

- Defining an interface is similar to creating a new class.
- An interface definition has two components: the interface declaration and the interface body.

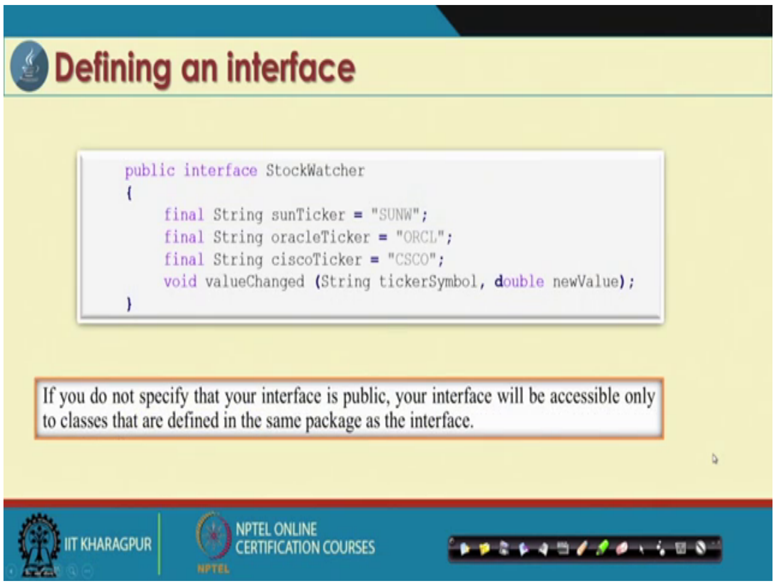
```
interfaceDeclaration
  {
          interfaceBody
  }
```

The `interfaceDeclaration` declares various attributes about the interface such as its name and whether it extends another interface, etc.

➢ The `interfaceBody` contains the constant and method declarations within the interface.

So, the defining an interface as you have learnt that an interface can be defined using the special keyword that is there in the Java is called the interface. So, here basically the basic syntax is the interface declaration by using the interface and giving the name of the interface and then this is the body of the interface. Now, so far the body of the interface is concerned this body includes few things.

(Refer Slide Time: 09:10)



Defining an interface

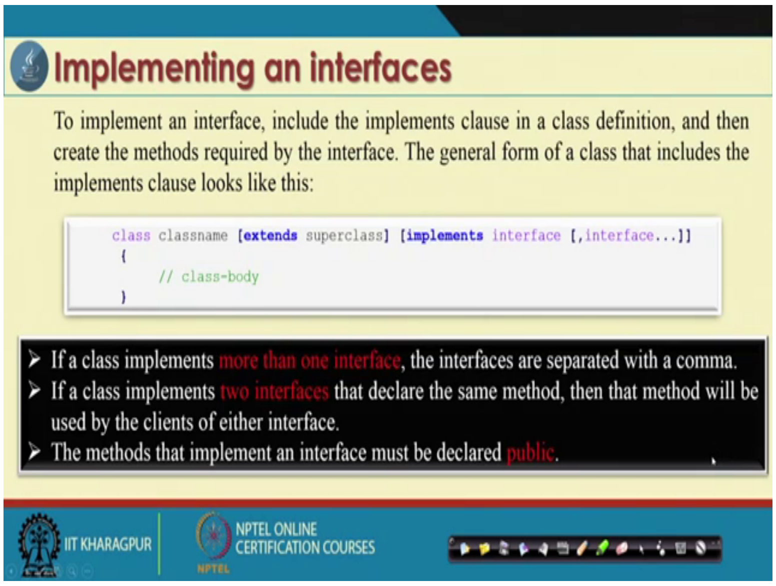
```
public interface StockWatcher
{
    final String sunTicker = "SUNW";
    final String oracleTicker = "ORCL";
    final String ciscoTicker = "CSCO";
    void valueChanged (String tickerSymbol, double newValue);
}
```

If you do not specify that your interface is public, your interface will be accessible only to classes that are defined in the same package as the interface.

The variables or members and the methods and as you have already mentioned that, the methods which should be there in the interface should be declared as a public and abstract there; that means, the method should have the only signature giving the name return type then parameter list and no body of the method. On the other hand so far the members are concerned the member should be declared as public, final, static and they can be initialized by some values actually the methods the variables or members which are there in a interface they should be treated as a global.

So, if any class which implements this interface has the complete access to all the members that is there in the interface and the methods needs to redefine the class which implements an interface. And as I already mentioned while I was discussing about the interface that if you do not declare a method as abstract or is a fine abstract or public by default it will be taken as a public and abstract and more one important concept is that no methods should not be declared as a static. So, static method is not allowed in any interface declaration. So, these are the few rule of thumbs that you should consider while you are declaring your own interface in your program.

(Refer Slide Time: 10:42)



Implementing an interfaces

To implement an interface, include the implements clause in a class definition, and then create the methods required by the interface. The general form of a class that includes the implements clause looks like this:

```
class classname [extends superclass] [implements interface [,interface...]]
{
      // class-body
}
```

- If a class implements more than one interface, the interfaces are separated with a comma.
- If a class implements two interfaces that declare the same method, then that method will be used by the clients of either interface.
- The methods that implement an interface must be declared public.

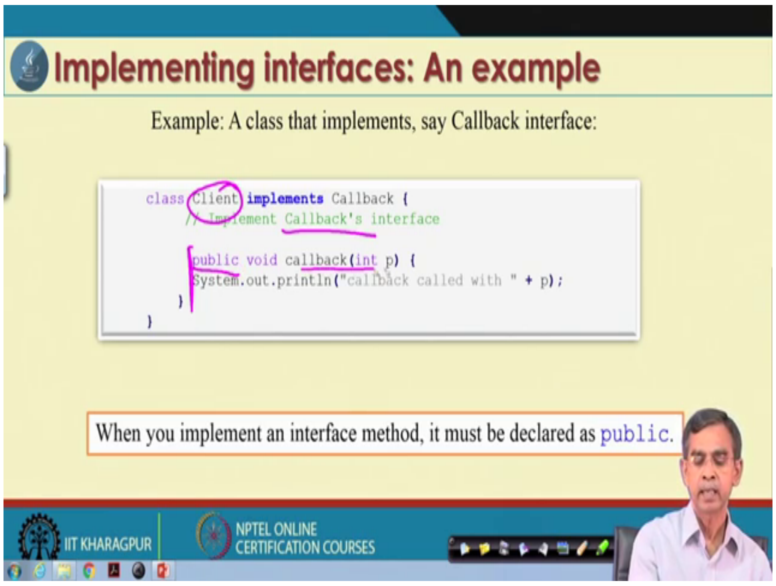
Now once the interface is defined this interface needs to be implemented. So, this implementation is by virtue of declaring a class, here is an example how an interface can be implemented. So, the basic rule is that an inter say this is the class name which basically implements interface and here there are two things are optional extends clause is optional and then also it can at one time interface two or more interfaces implements two or more interfaces. So, this basic idea is that a class can inherit from the other class that is why the extends clause will take care and at the same time it can implements one or more interfaces so, this is the concept.

So, then now we have demonstration in our next module to discuss about how all those things works together and as I mentioned there this class that we should have and that class should be declared as a public there means be this should be public here for example, declare as a default it is not allowed. So, you have to mention that this is the public.

Now there are few cautions that needs to be taken care whenever we implement an interface is that if an interface in if a class implements two or more interfaces which has the same method and then that methods should be overridden by the class and it has only one copy. Now, when we say that same method it means that the methods which are declared as an abstract in the interface they have the same return type same method name and same list of parameters.

If anyone is different the method should be treated as different and then different in implementation in the class is required. So, this is the important thing that you should consider and the methods which basically implemented in your class also that needs to be declared as a public, because these methods should be accessible by anyone. So, no private, no protected method as an implementation is allowed in the class implementation. So, these are few thing that you should note while you are considering the implementing interfaces.

(Refer Slide Time: 13:02)



Implementing interfaces: An example

Example: A class that implements, say Callback interface:

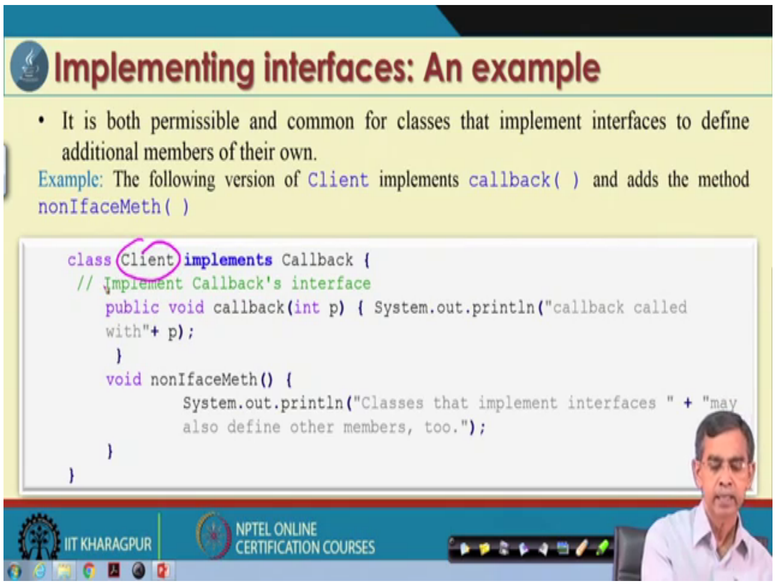
```
class Client implements Callback {
    // Implement Callback's interface

    public void callback(int p) {
    System.out.println("callback called with " + p);
    }
}
```

When you implement an interface method, it must be declared as public.

Now, here is a quick example here you can see that how an interface can be implemented by a class, in this case the name of the class is Client and here you see this is the Callback method which is there in the interface and we implement this method as a public and then whatever that I. And while we implement at the assembly we have to consider the return type and the list of parameter should match these which are there in your interface declaration, otherwise this will treat as a completely new methods the methods of it is own in this class itself, what I want to say here is that. In fact, we are to overwrite the methods, which is declared as an abstract and public method in the interface declaration ok.

(Refer Slide Time: 13:57)



Implementing interfaces: An example

- It is both permissible and common for classes that implement interfaces to define additional members of their own.

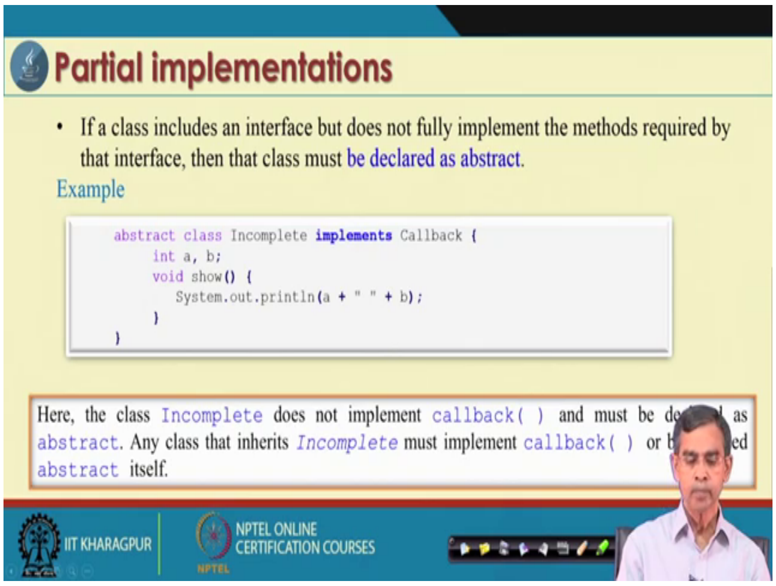
Example: The following version of Client implements callback( ) and adds the method nonIfaceMeth( )

```
class Client implements Callback {
 // Implement Callback's interface
    public void callback(int p) { System.out.println("callback called
    with"+ p);
     }
    void nonIfaceMeth() {
            System.out.println("Classes that implement interfaces " + "may
            also define other members, too.");
    }
}
```

So, this is some standards procedure that needs to be follow while we are using the implementation of an interface by a class. Now here is another example that we can say again this is a class Client which implements the Callback, Callback is an interface here. So, this is the method that we have declared in the interface and implemented here and at the same time the class which implements an interface it may contains, it may contain, it may include it is own method as well the methods that is overridden in the interface.

So, for example, here this is one method which is the own method in this class client; that means, it just like a inheritance concept like. So, it is basically one other way you can say that this class clients inherits the Callback where the Callback is an abstract class we can say in that sense. So, this is the idea about how a class can implement this means that implement means the methods should be overridden and it can includes it is own methods if required and no variable should be declared with the same name as that is there in the interface there. So, the method should be in fact, because it is a static and public declared in the interface. So, we cannot redeclare or redefine the same variable name or members in the class implementation ok.

(Refer Slide Time: 15:31)



Partial implementations

- If a class includes an interface but does not fully implement the methods required by that interface, then that class must be declared as abstract.

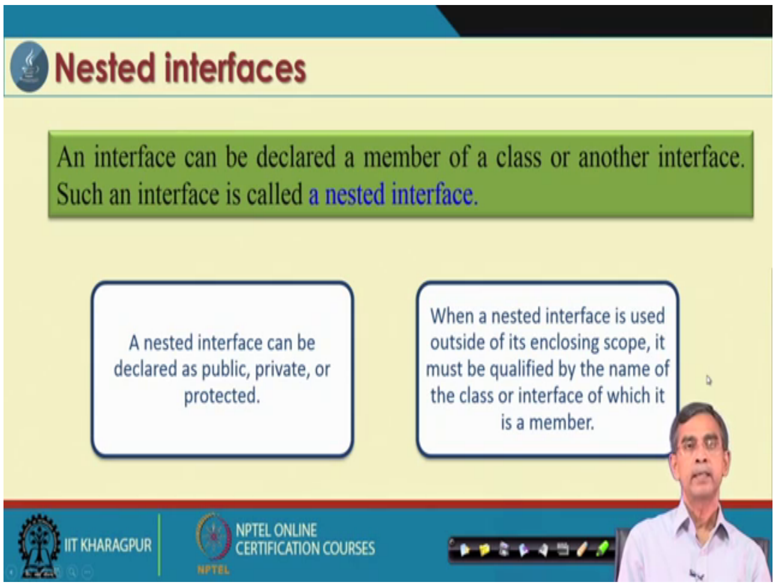
Example

```
abstract class Incomplete implements Callback {
    int a, b;
    void show() {
        System.out.println(a + " " + b);
    }
}
```

Here, the class Incomplete does not implement callback( ) and must be de[illegible] as abstract. Any class that inherits Incomplete must implement callback( ) or b[illegible]ed abstract itself.

So, this is the idea about some implementation of the interface by means of a class and Java also allow partial implementation. Say suppose in an interface there are two methods and you implements this interface by means of a class only one method, then this class can be treated an abstract class; that means, no that means, that means, no objects can be created for this class until you override all the methods which are there in the interface. So, you have to override all the methods so, that you can create an object or you can complete the implementation of an interface in a class.

(Refer Slide Time: 16:15)



Nested interfaces

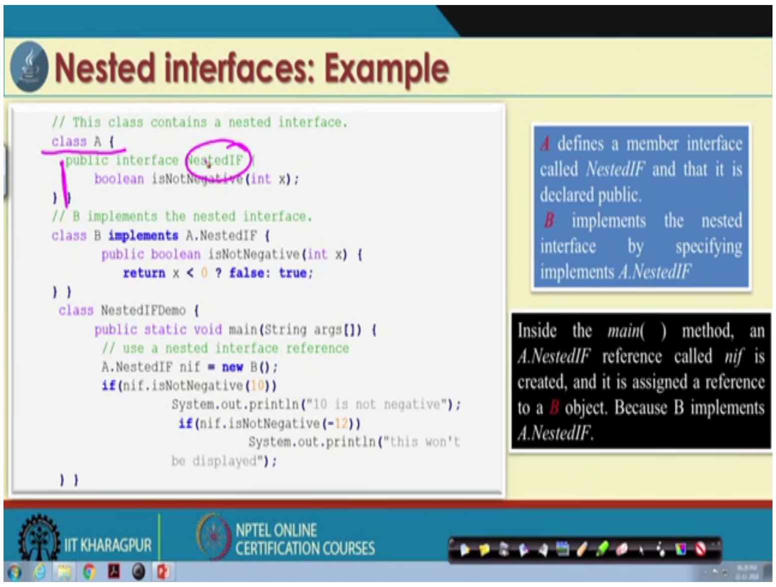
An interface can be declared a member of a class or another interface. Such an interface is called a nested interface.

A nested interface can be declared as public, private, or protected.

When a nested interface is used outside of its enclosing scope, it must be qualified by the name of the class or interface of which it is a member.

And in Java also it is possible to allow nested interface, what is the meaning of nested interface is that an interface can be declared inside a class declaration. This means that this interface is a very much local to this class itself; that means, no one class outside this in class is responsible for implementing this interface. This is the concept of it is called the nested interface, this is a very restrictive use in the program usually we avoid it, but sometimes we want to make an interface which is a very explicit to a particular class only then we can think for this kind of implementation otherwise you can ignore it. Anyway this is an example how the nested if is possible.

(Refer Slide Time: 17:10)



Nested interfaces: Example

```
// This class contains a nested interface.
class A {
  public interface NestedIF {
      boolean isNotNegative(int x);
} }
// B implements the nested interface.
class B implements A.NestedIF {
       public boolean isNotNegative(int x) {
          return x < 0 ? false: true;
} }
 class NestedIFDemo {
      public static void main(String args[]) {
       // use a nested interface reference
       A.NestedIF nif = new B();
       if(nif.isNotNegative(10))
                 System.out.println("10 is not negative");
                  if(nif.isNotNegative(-12))
                           System.out.println("this won't
                 be displayed");
 } }
```

A defines a member interface called *NestedIF* and that it is declared public. *B* implements the nested interface by specifying implements *A.NestedIF*

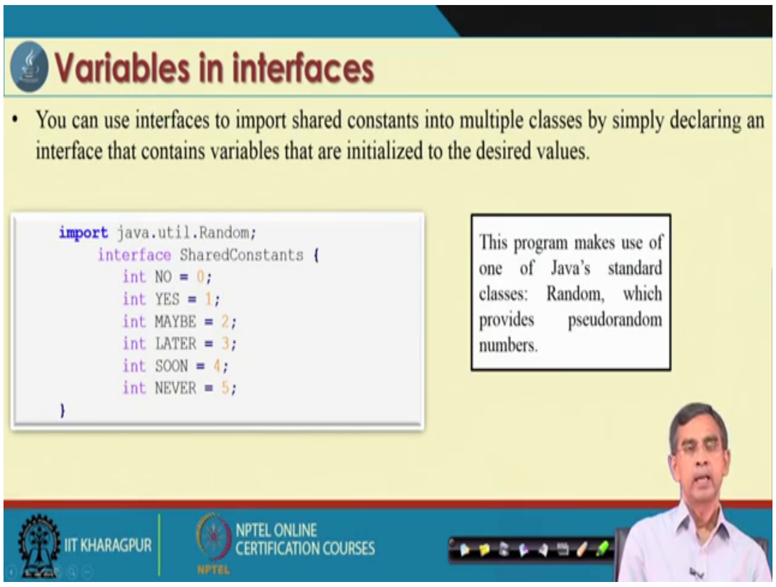
Inside the *main*( ) method, an *A.NestedIF* reference called *nif* is created, and it is assigned a reference to a *B* object. Because B implements *A.NestedIF*.

Now let us look at this small program here and this is the class A declaration as usual the standard normal class and you can see within this normal class we declare an interface this is the interface. So, we declare as an interface as NestedIf and this has the this method is an abstract method and then we can inside this class as you have the interface, interface does not have any utilization until you implements it. So, here is the class B which implements this is the NestedIf and you can see one thing that NestedIf we have expansion A dot, there is a special location specification that A dot means it is the interface which is declaring the class A.

If you do not do in this case it will work, but in some situations if inside A class there more than interface then better to do it like this one. So, otherwise if some interface already in the same name appears somewhere else then it can give a what is called the ambiguity. So, in order to resolve this we have to explicitly mention that this interface belongs to which class. So, that is why the specific location mention that is A dot NestedIf means that this is a NestedIf inside the class A.

So, this way we can define it and then finally, we can implement the interface that is there using the usual concept and then same can be used in your main class wherever you want to use it. So, it is the basic idea about that it is just like a scope of this interface which is a nested interface inside a class is basically the static scope and it can be resolved seeing the program itself and is a local, local to this class itself.

(Refer Slide Time: 19:04)



Variables in interfaces

- You can use interfaces to import shared constants into multiple classes by simply declaring an interface that contains variables that are initialized to the desired values.

```
import java.util.Random;
    interface SharedConstants {
        int NO = 0;
        int YES = 1;
        int MAYBE = 2;
        int LATER = 3;
        int SOON = 4;
        int NEVER = 5;
}
```

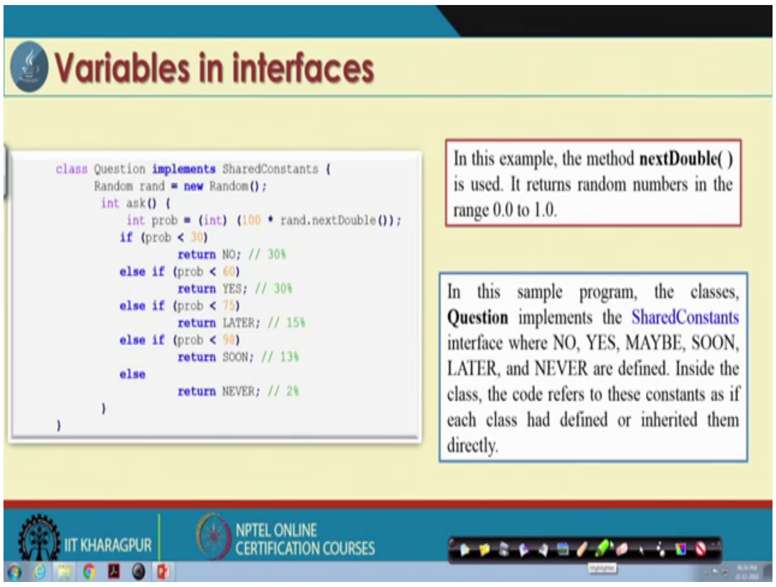
This program makes use of one of Java's standard classes: Random, which provides pseudorandom numbers.

Now, let us have a very simple example about, what is the utilization of an interface? Why we go for an interface? What is the usage of the interface? It has 2 important applications; one is that whenever you use an interface it will it may if it includes some members which are declared as a final, static and then a final static public they can be used as a global variable look like and this variable can be shared. So, it is just like a library of different variables that can be shareable from one class to another class.

So, this is a one example and another example; obviously, the great example the most significant example that we can inherit in a multiple sense as you know Java does not support single inheritance, but in an indirect way Java also helps a programmer to have the multiple inheritance implementation. These are the two main usage and one usage also it is there runtime polymorphism; that means, that will be discussed while we will go for demonstrating the application of interface in our next module. Now let us have the first example that how a variable can be shared across the classes if they are maintained in an interface.

This is an example for like. So, suppose we want to we in this example here we can say we declare an interface the name of the interface as the shared constants and these are the different values and by default they are public static automatically there. So, they are public static int and these are the different value variables and the values are there. So, these are the basically we can say this as they are basically static variable sort of things; that means, they are global look like. So, they can be used one instances in everywhere there. So, these are the global variable look like as you can see in a more simple way. So, whenever we declared all these value were members in an interface they can be considered as a global variable look like.

(Refer Slide Time: 21:25)



Variables in interfaces

```
class Question implements SharedConstants {
      Random rand = new Random();
       int ask() {
           int prob = (int) (100 * rand.nextDouble());
          if (prob < 30)
                   return NO; // 30%
          else if (prob < 60)
                   return YES; // 30%
          else if (prob < 75)
                   return LATER; // 15%
          else if (prob < 98)
                   return SOON; // 13%
          else
                   return NEVER; // 2%
       }
}
```

In this example, the method **nextDouble( )** is used. It returns random numbers in the range 0.0 to 1.0.

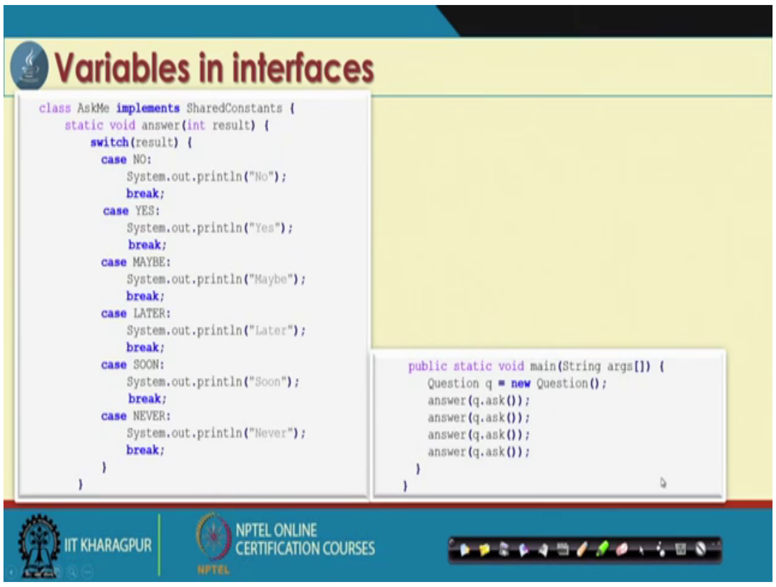
In this sample program, the classes, **Question** implements the SharedConstants interface where NO, YES, MAYBE, SOON, LATER, and NEVER are defined. Inside the class, the code refers to these constants as if each class had defined or inherited them directly.

Now, once these methods are declared we can use them in a program. So, we can just create a program look like. So, this is the one class the name of the class is implements. This class is questions which implements shared constant and basically it uses all these variable names as you can see here. Now here in this class we define one ask method which has this kind of structure.

So, if you go through the program you will be able to understand what exactly the ask question is there, it basically takes a random number and this random number is called prob and depending on the value of the prob it basically return NO. YES, LATER, SOON, all these methods are all these members are which is already declared in the interface. So, this is the idea about a simple example of course, that all these methods as they are they will be used here as if there is a global.

Now so, this is the way that an interface can be used in this case and here is the complete program that you can see how this program can be used.

(Refer Slide Time: 22:28)



Variables in interfaces

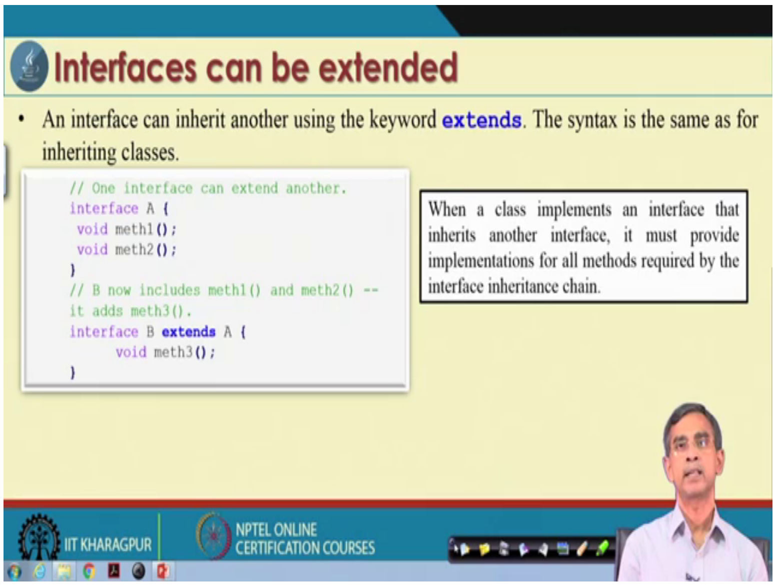
```
class AskMe implements SharedConstants {
    static void answer(int result) {
        switch(result) {
          case NO:
              System.out.println("No");
              break;
          case YES:
              System.out.println("Yes");
              break;
          case MAYBE:
              System.out.println("Maybe");
              break;
          case LATER:
              System.out.println("Later");
              break;
          case SOON:
              System.out.println("Soon");
              break;
          case NEVER:
              System.out.println("Never");
              break;
        }
    }
```

```
    public static void main(String args[]) {
      Question q = new Question();
      answer(q.ask());
      answer(q.ask());
      answer(q.ask());
      answer(q.ask());
    }
}
```

This is a simple example another class ask me which implements shared constant which use the previous asked method and it has it is own body it is there and then this is the main methods which basically utilization of all the method in the last slides the ask method in this slides the answer method and it basically code this one. If you run this program it will be an interesting output which will be discussed while I will go for the demonstration in the next module ok.

So, these basically shows emphasized that if you declare an interface then all the members those are there is basically will be used as a shared variable across the different classes.

(Refer Slide Time: 23:13)



Interfaces can be extended

- An interface can inherit another using the keyword **extends**. The syntax is the same as for inheriting classes.

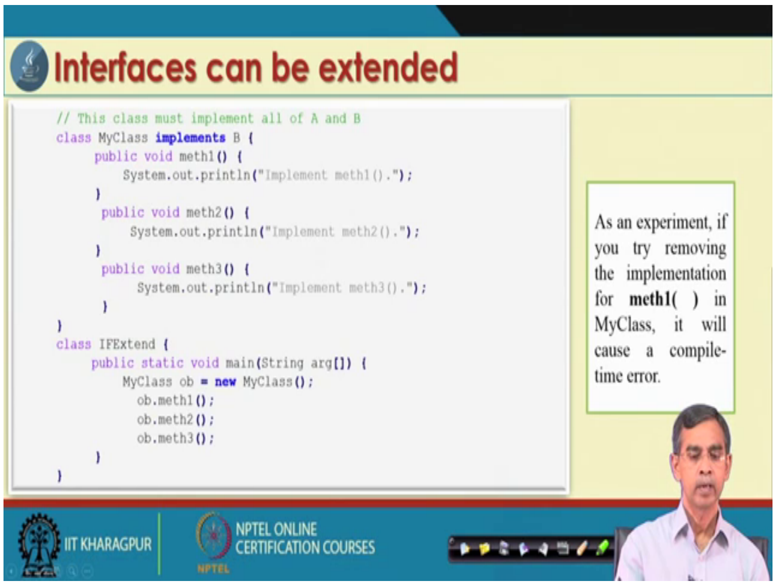
```
// One interface can extend another.
interface A {
 void meth1();
 void meth2();
}
// B now includes meth1() and meth2() --
it adds meth3().
interface B extends A {
      void meth3();
}
```

When a class implements an interface that inherits another interface, it must provide implementations for all methods required by the interface inheritance chain.

And interface can be extended we have already discussed about these that. So, an interface truly works like a class as the class can extend another class so, an interface also can extend another class. So, suppose here the interface A and interface B using the same extends. So, we can extend the class; that means, in this interface all the method that is there or all the members which are there in the interface will be also inherited in this one. So, the basic concept is same as the class inheritance also applicable to the inheritance.

So, again I want to repeat it that an interface in mostly can be treated as a class look like; that means, whatever the procedure that we can follow for class it can be only the exception is that for a class an object can be created; however, for an inter interface no object can be created that is all.

(Refer Slide Time: 24:11)



Interfaces can be extended

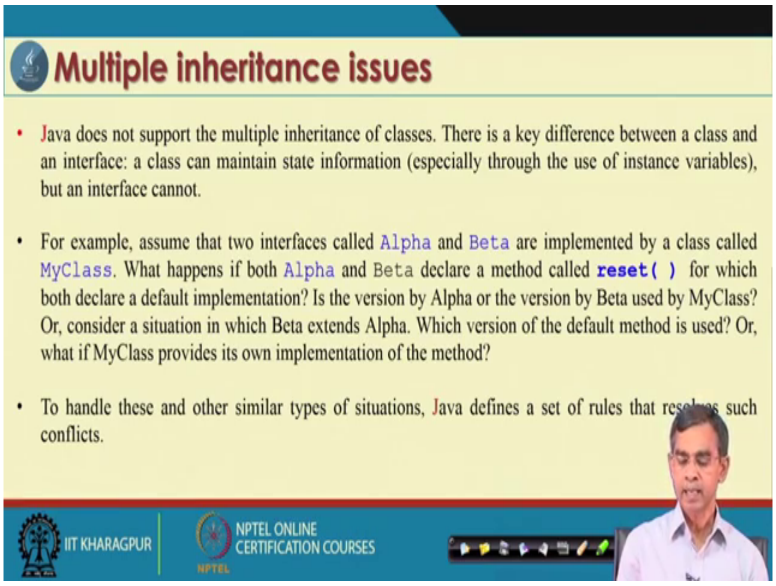
```
// This class must implement all of A and B
class MyClass implements B {
      public void meth1() {
          System.out.println("Implement meth1().");
      }
       public void meth2() {
           System.out.println("Implement meth2().");
      }
       public void meth3() {
            System.out.println("Implement meth3().");
       }
}
class IFExtend {
     public static void main(String arg[]) {
          MyClass ob = new MyClass();
            ob.meth1();
            ob.meth2();
            ob.meth3();
      }
}
```

As an experiment, if you try removing the implementation for **meth1( )** in MyClass, it will cause a compile-time error.

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES

So, this is the one idea where the interface can be extended and here is a complete idea is that on the interface extended the inherited interface can be implemented by means of a class. So, this is the one example that basically explains how the inherited interface can be implements. So, both the superclass interface super interface as well as the derived interface can be used for implementation by another class. And then the multiple inheritance is the significant what is called the use of the inheritance concept here.

(Refer Slide Time: 24:46)



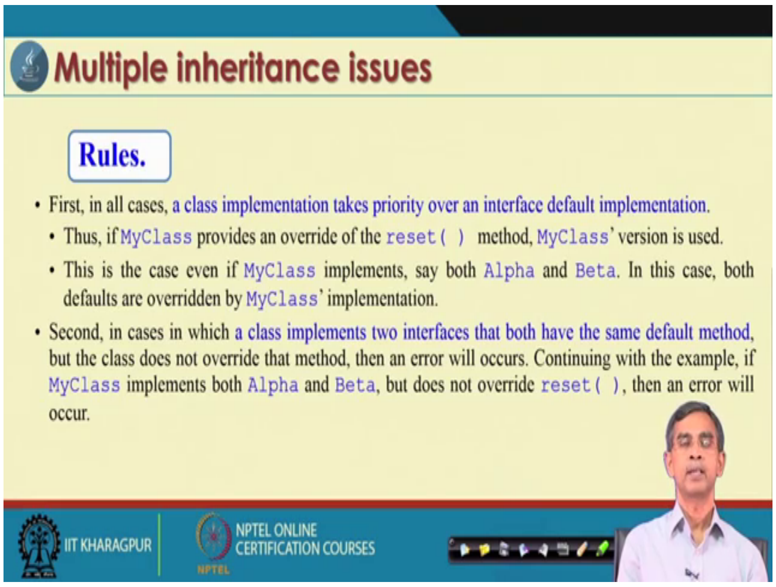
Multiple inheritance issues

- Java does not support the multiple inheritance of classes. There is a key difference between a class and an interface: a class can maintain state information (especially through the use of instance variables), but an interface cannot.
- For example, assume that two interfaces called `Alpha` and `Beta` are implemented by a class called `MyClass`. What happens if both `Alpha` and `Beta` declare a method called **`reset( )`** for which both declare a default implementation? Is the version by Alpha or the version by Beta used by MyClass? Or, consider a situation in which Beta extends Alpha. Which version of the default method is used? Or, what if MyClass provides its own implementation of the method?
- To handle these and other similar types of situations, Java defines a set of rules that resolves such conflicts.

IIT KHARAGPUR NPTEL ONLINE CERTIFICATION COURSES

So, here is a example that multiple inheritance means one interface can extends two or more interface, but it is not exactly the extent rather it is basically the implements actually. So, if we can plan a class which implements two or more interface then we can say that this class in fact, multiply inherits two interfaces. So, the concept is there and the concept is there in the class itself the multiple inheritance can be realized.

(Refer Slide Time: 25:15)



Multiple inheritance issues

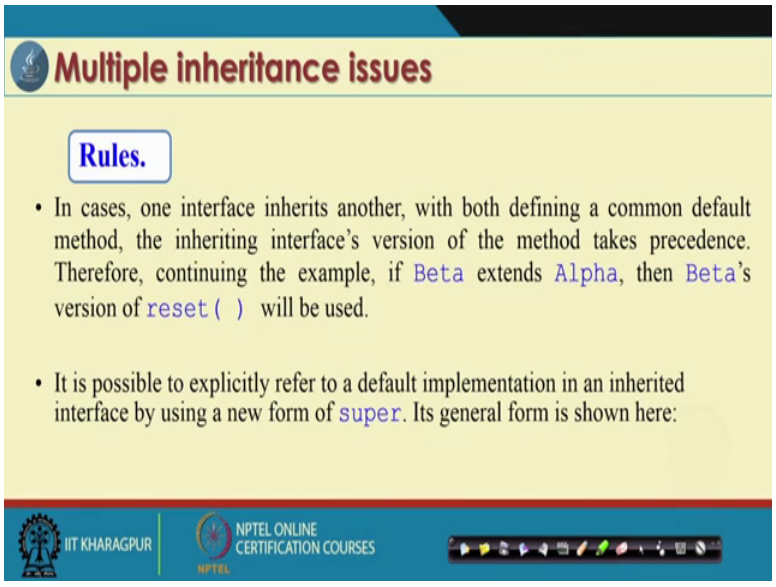
Rules.

- First, in all cases, a class implementation takes priority over an interface default implementation.
  - Thus, if MyClass provides an override of the reset( ) method, MyClass' version is used.
  - This is the case even if MyClass implements, say both Alpha and Beta. In this case, both defaults are overridden by MyClass' implementation.
- Second, in cases in which a class implements two interfaces that both have the same default method, but the class does not override that method, then an error will occurs. Continuing with the example, if MyClass implements both Alpha and Beta, but does not override reset( ), then an error will occur.

Now here few things have to be more few things have to be carefully noted first of all the class suppose implements 2 interface i 1 andi 2 and there is a method say m which is declared in both the interfaces. Then in the implemented class which methods needs to be implemented in this regard I want to say this way that if the 2 methods are same the 2 methods are same in the interface in the sense that they have the same return type and then same list of arguments, having the same type then it basically absolute no problem you have to override only once.

Otherwise, all the methods which are there we have to override in the implementation class implementation of the interface. So, this way it basically multiple implements all the interfaces they are by multiple inheritance it is like this one.

(Refer Slide Time: 26:23)



Multiple inheritance issues

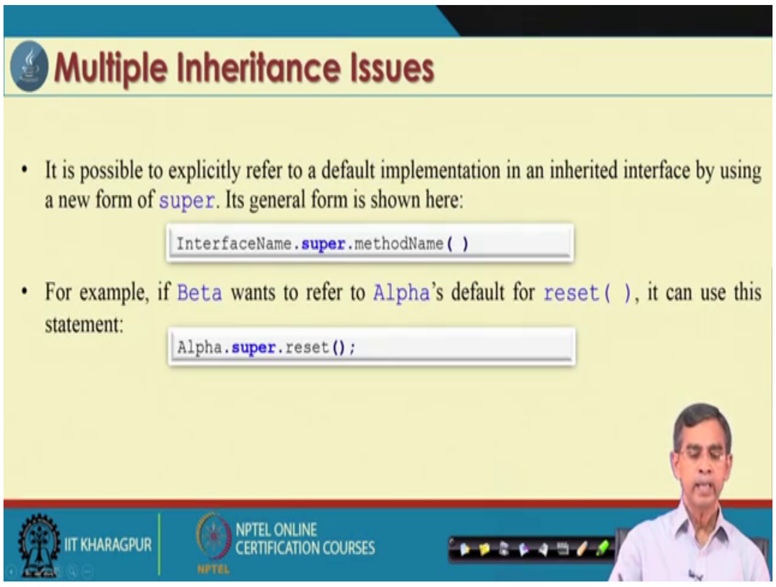
Rules.

- In cases, one interface inherits another, with both defining a common default method, the inheriting interface's version of the method takes precedence. Therefore, continuing the example, if `Beta` extends `Alpha`, then `Beta`'s version of `reset( )` will be used.
- It is possible to explicitly refer to a default implementation in an inherited interface by using a new form of `super`. Its general form is shown here:

There is another also example where we can have the class extends one class and implements another this is also one example of multiple inheritance in there. So, by class extends for example, class B extends class A and implements interface i then it basically is a multiple inheritance concept it is there; that means, it extents class B A means that B will inherit all the methods and members those are accessible to the class A class B it is there in addition to the interface methods and the interface variable also accessible to the class there.

Now, again another important rule that if the two interface have the same variable declaration then it will give a compilation error that you have to somehow take care check that the two interface does not declare the same members variables in duplicate in the two interfaces or more than interfaces which are basically in used in multiple inheritance and multiple inheritance is not limited to only two interfaces any number of interfaces can be considered. So, a class can implements two three or many interfaces at the same time, but extends whenever it come into the picture it can extends only one class that is the important what is called the things that you should note ok.

(Refer Slide Time: 27:47)



Multiple Inheritance Issues

- It is possible to explicitly refer to a default implementation in an inherited interface by using a new form of super. Its general form is shown here:

```
InterfaceName.super.methodName( )
```

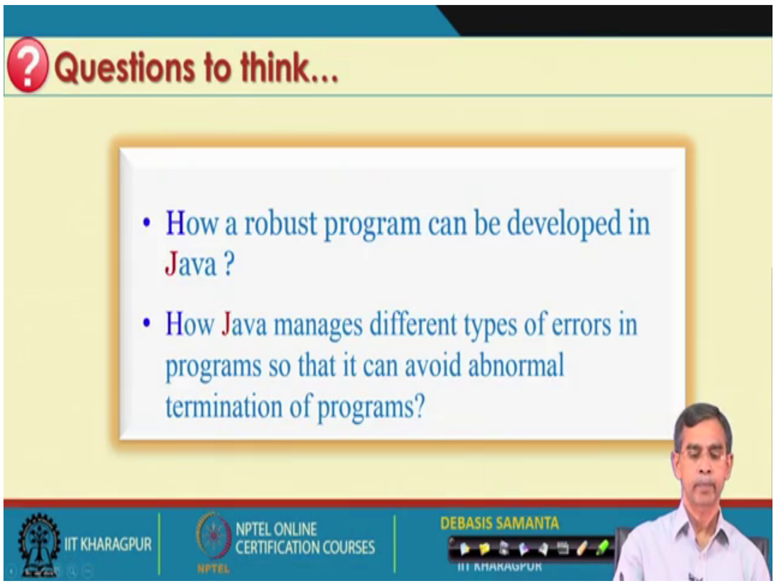
- For example, if Beta wants to refer to Alpha's default for reset( ), it can use this statement:

```
Alpha.super.reset();
```

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES

So, this is a concept that and if you want to specify explicitly some interface and then again the super keyword can be used. So, here is an example for example, super method name is basically in the InterfaceName we can discuss about if it inherits from others name to resolve the ambiguity if any. So, the super concept it is basically same way it is basically namespace collision resolution as well as the method resolution. So, this is the same concept also extendable to the interface here.

(Refer Slide Time: 28:19)



Questions to think...

- How a robust program can be developed in Java ?
- How Java manages different types of errors in programs so that it can avoid abnormal termination of programs?

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
DEBASIS SAMANTA
IIT KHARAGPUR

Now, we have learned about the interface and then more on the interface will be discussed while we will have a quick demo on the interface and we advise you to have the good lessons in the interface demonstration.

Now, our next topic that we are going to cover is very important topic that this topic is basically to address the questions that I have mentioned here. So, there definitely is a big question that how a Java programmer a software programmer can develop the program which is very much robust; that means, fault free tolerant program. And then there are many errors and particularly it is the concern whenever the program size is increased from lows low size low volume to high volume because as the code size will increase the number of errors possibility will increase so, how to deal with this situation. So, all these things will be discussed in our next discussion that discussion is called multiple is called exception handling concept.

Thank you; thank you very much.