Programming in JAVA

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

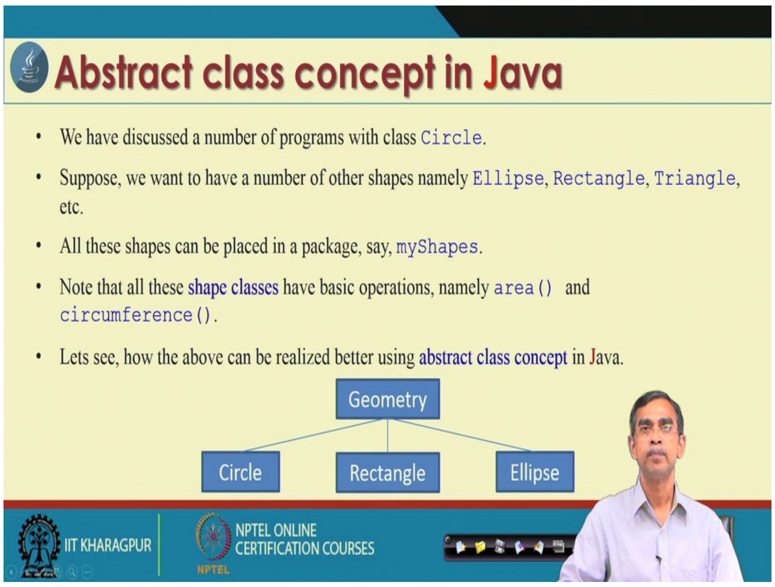
Lecture - 20
Interfaces – I

So, in the last module including the demonstration; we have learned about packages in Java. Now there are two things in Java which makes the Java programming a unique is a package and interface. And this is why the Java programming is basically more popular compared to the other programming languages in particular to build very large and complex software.

So, today we will discuss the concept of interface in today and in the next module. Actually, the things need to I mean lectures hours to discuss complete the discussion so we will take the two modules to complete it. Anyway, today we have some basic concept of interface and then other some other concepts also will be discussed in the next module anyway.

So, let us start about the interface, but before going to take the lesson about the interface we want to start with the abstract class. The abstract class concept we have already discussed while we are discussing inheritance.

(Refer Slide Time: 01:31)



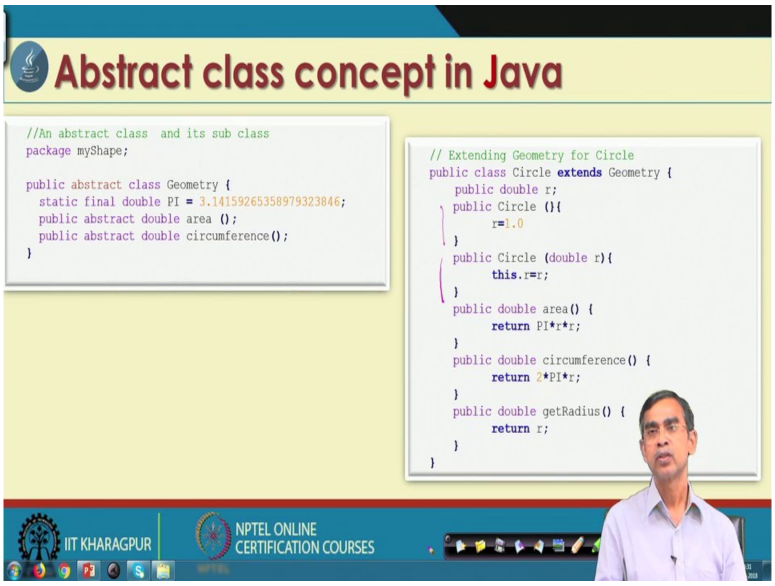
Abstract class concept in Java

- We have discussed a number of programs with class Circle.
- Suppose, we want to have a number of other shapes namely Ellipse, Rectangle, Triangle, etc.
- All these shapes can be placed in a package, say, myShapes.
- Note that all these shape classes have basic operations, namely area() and circumference().
- Lets see, how the above can be realized better using abstract class concept in Java.

So, a class can be declared as an abstract class as we know by defining the keyword abstract. Now, let us consider a very simple example this example, say, we declare a class says geometry as an abstract class. And then we want to have some classes inherited from this abstract class namely circle, rectangle and ellipse.

Here, geometry basically says that a set of objects geometrical objects rather and then there are different special objects belong to this all geometrical objects namely circle, rectangle, and ellipse. Now, let us have a quick look at how such an implementation can be done in Java using the concept of abstract class.

(Refer Slide Time: 02:21)



Abstract class concept in Java

```
//An abstract class  and its sub class
package myShape;

public abstract class Geometry {
  static final double PI = 3.14159265358979323846;
  public abstract double area ();
  public abstract double circumference();
}
```

```
// Extending Geometry for Circle
public class Circle extends Geometry {
   public double r;
   public Circle (){
         r=1.0
   }
   public Circle (double r){
         this.r=r;
   }
   public double area() {
         return PI*r*r;
   }
   public double circumference() {
         return 2*PI*r;
   }
   public double getRadius() {
         return r;
   }
}
```

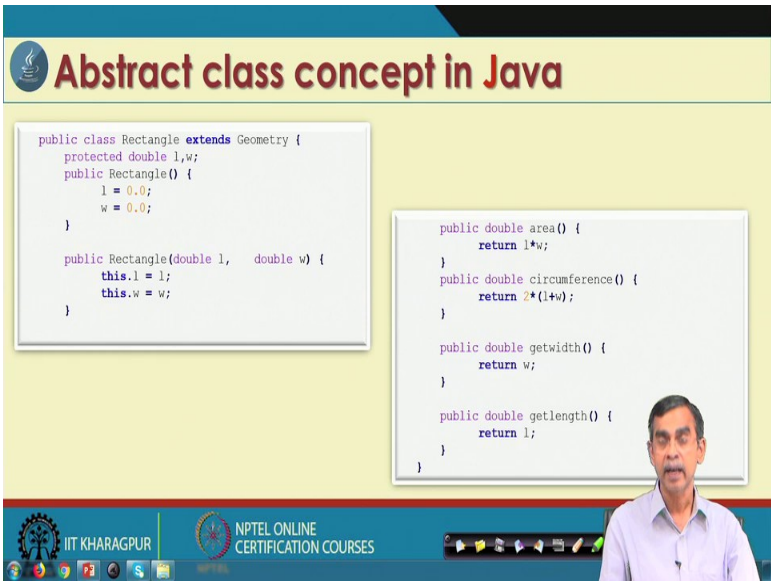
So, here we can see we built a built an abstract class namely class geometry here. And then we put this class in a package. The name of the package as we see MyShape is the package. And under this package, so there are so this class abstract class is defined by these three members. One is PI declared as a static and final and then area and circumference are the two abstract methods that we have discussed.

As you know an abstract class can contain both abstract methods as well as known abstract method. In this example, we include only two abstract methods here. So, they are declared as a public abstract double area, public abstract double circumference. So, this is the declaration of an abstract class. Now, what is our objective is that our objective is to create the other classes which can be inherited from this abstract class.

So, here is the one code what we can see here the circle class which basically inherited from the superclass geometry. It has the constructors of its own. Here are the dipole constructors and there is another constructors is only one argument. So, the two constructors will define the method area here and then the method circumference. And some method which is unique in this class itself. As we see these are the two methods are the implementation of the method abstract method which is there.

So, whenever some abstract methods are there in abstract class it is a responsibility of the programmer to implement it fully. If you do not implement then you will not be able to create any object of this class that we have created just now. Anyways so this completes the creation of the class circle which is an inheritance from the class abstract class.

(Refer Slide Time: 04:51)



Abstract class concept in Java

```
public class Rectangle extends Geometry {
    protected double l,w;
    public Rectangle() {
          l = 0.0;
          w = 0.0;
    }

    public Rectangle(double l,    double w) {
          this.l = l;
          this.w = w;
    }
```

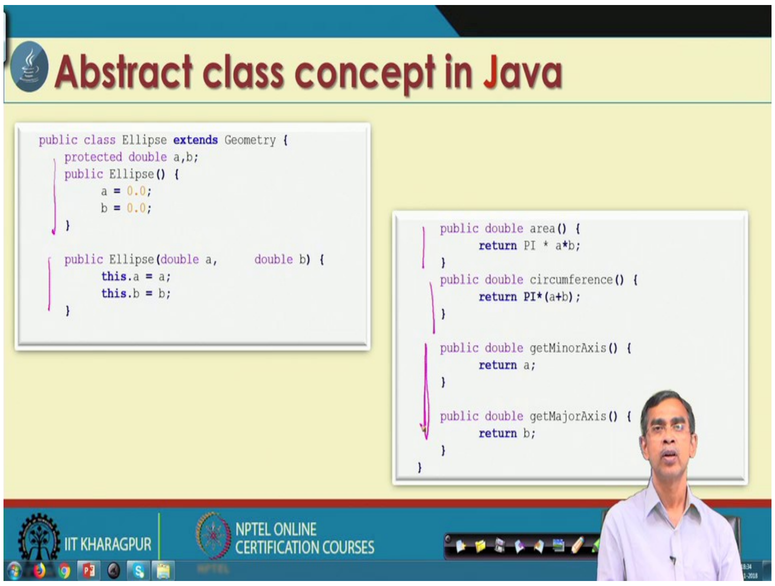
```
    public double area() {
          return l*w;
    }
    public double circumference() {
          return 2*(l+w);
    }

    public double getwidth() {
          return w;
    }

    public double getlength() {
          return l;
    }
}
```

Now, after this class is over and the next is how the ellipse class can be again inherited from this abstract class. So, we are going to inherit the three classes from the abstract class namely circle. And now we are going to discuss we are going to discuss this is the circle class. And next is it is an inheritance from the circle another is the called the rectangle class. So, this class has this kind of form we see this is the one member that is declared as a protect it; that means, this l and w will be accessible to its inherited class.

And these are the method the constructor in this case. And this is another constructors with the two arguments to pass it is a dipole constructor and this is the special constructor. So, this is the class declaration and as we see here this is basically inheritance from the geometry class. So, it is an inheritance of the geometry as a rectangle class.

Now, next class that we are going to discuss this class has come other methods which are not put into here. So, this is a other method as we see the area which is again declared as an abstract in inheritance. So, it is defined here for this class, and this is a circumference which is for this class it is the redefinition and these are the two methods of its own that are special to this class. So, this completes the declaration of the class rectangle which is an inheritance of the class geometry.

(Refer Slide Time: 06:24)



Abstract class concept in Java

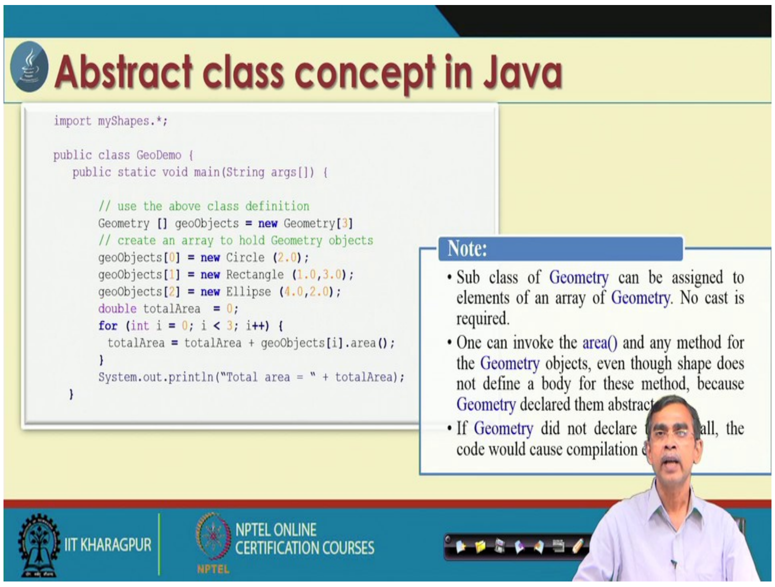
```
public class Ellipse extends Geometry {
    protected double a,b;
    public Ellipse() {
        a = 0.0;
        b = 0.0;
    }

    public Ellipse(double a,      double b) {
        this.a = a;
        this.b = b;
    }
```

```
    public double area() {
        return PI * a*b;
    }
    public double circumference() {
        return PI*(a+b);
    }

    public double getMinorAxis() {
        return a;
    }

    public double getMajorAxis() {
        return b;
    }
}
```

Now, our next class is an example this is an ellipse. So, we again derive the ellipse class inheriting from the geometry it has the methods like this. And it is another constructor there are few more things are also here. So, this is basically completed the declaration of this class as we see here area is redefined for this class circumference and the two methods that are of its own.

Anyway, so we have a full what is called the declarations of three classes namely circle rectangle and ellipse which are derived classes from the superclass geometry here. And the geometry class, in this case, is an abstract class. Now, after this definition of these three classes is there then we are now in a position to create the object of these classes. Now let us see how we can create the object of these classes.

(Refer Slide Time: 07:25)



Abstract class concept in Java

```
import myShapes.*;

public class GeoDemo {
   public static void main(String args[]) {

      // use the above class definition
      Geometry [] geoObjects = new Geometry[3]
      // create an array to hold Geometry objects
      geoObjects[0] = new Circle (2.0);
      geoObjects[1] = new Rectangle (1.0,3.0);
      geoObjects[2] = new Ellipse (4.0,2.0);
      double totalArea  = 0;
      for (int i = 0; i < 3; i++) {
       totalArea = totalArea + geoObjects[i].area();
      }
      System.out.println("Total area = " + totalArea);
   }
```

Note:

- Sub class of Geometry can be assigned to elements of an array of Geometry. No cast is required.
- One can invoke the area() and any method for the Geometry objects, even though shape does not define a body for these method, because Geometry declared them abstract
- If Geometry did not declare [illegible] all, the code would cause compilation [illegible]

So, here is the one application program that is basically the contain the main class; so that we can create the objects of the classes that we have to define Now, we give the name of this class is GeoDemo and we can see all these classes that we have declared we have store them in the package form and the name of the package is MyShape. So, we have to import all the classes that we have stored in the package MyShape and this is the import statement.

Now, so this is the main program as we see we create a geo object of the abstract class here. So, this is basically is a basically container we can say geoObject is a container which holds the different objects of type geometry like this. Now then we create three different objects of three different classes like circle, rectangle, ellipse. And we store all this geoObject into this container like this one. So, it is basically an array of geoObject we can say so this is stored.

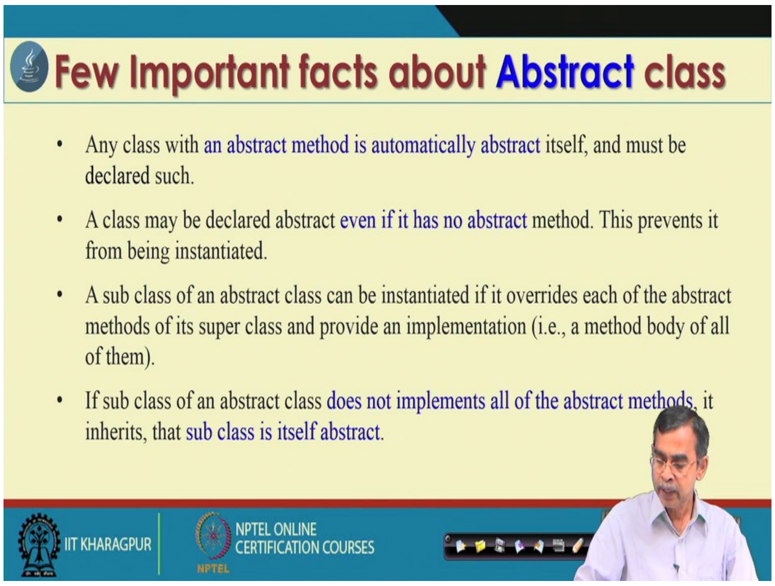
Now, let us see how we can process. So, suppose whatever the objects that we have created in this case we have created three objects with these are the different parameter pass to the objects. And we want to create the total area consumed by all three objects that are there. So, we can have a very simple for loop; so we can create the total area these are the total area is the some of all the areas of the objects this is there and then it will print.

Now, here we can think is looks very simple. What we have done? We have created an abstract class for this abstractor we have derived three sub classes namely circle ellipse and then and then we store this thing in an application and then used them process them. Now here, few things very important to note in this here: now geoObjects.area so for if you do not thing about this code. Now only this code now geoObjects i.area. So, whenever the loop will role it will basically bind itself that which area corresponding to which object.

Now, whenever i equals to 0 this area will bind to the circle objects whenever i equals to 1 this area will bind into the rectangular object, whenever i equal to 2 this area will bind to the ellipse object. So, geoObjects i and it is just automatically binding with a corresponding object this is an example is called dynamic binding or it is also called run time polymorphism.

Because this geometric object polymorphically works for any area irrespective of the type of the object it is. Now, this is one very good example of usage of the abstract class so we have used earlier. Now like this type of concept we are now going to discuss other things. So fine, this we can see from the geometry object we have to derive three classes, but this derives by means of single inheritance.

(Refer to Slide Time: 10:43)



Few Important facts about Abstract class

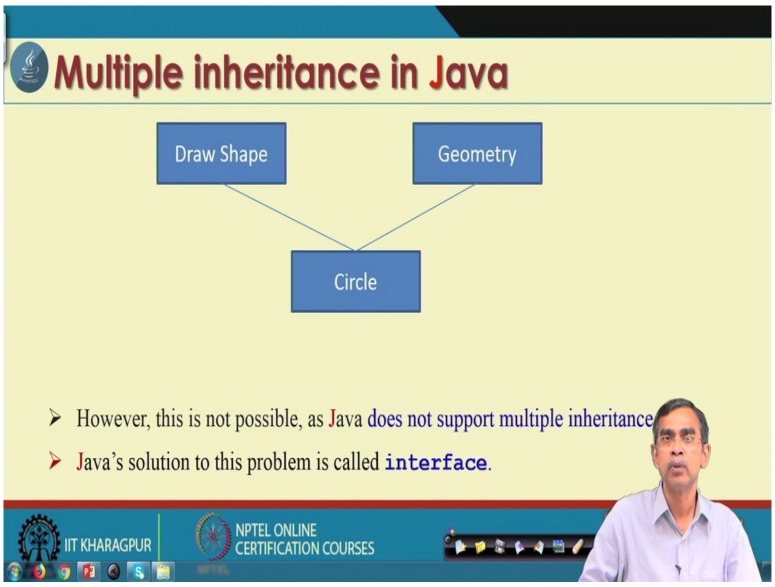
- Any class with an abstract method is automatically abstract itself, and must be declared such.
- A class may be declared abstract even if it has no abstract method. This prevents it from being instantiated.
- A sub class of an abstract class can be instantiated if it overrides each of the abstract methods of its super class and provide an implementation (i.e., a method body of all of them).
- If sub class of an abstract class does not implements all of the abstract methods, it inherits, that sub class is itself abstract.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

But in some situations you know we have to have the concept of multiple inheritance. Now how this concept is there now regarding abstract class and everything as we have discussed this thing that those things are already discussed. So, I do not want to repeat it fine. So, the abstract method is automatically abstract itself, and then if a class may be declared as an abstract even if it has no abstract method also this prevents it from being instantiated. Now here instantiation regarding this instantiation we will discuss the details and then we should have a good demo on it.

So, that you will be able to understand it and a sub class of an abstract class can be instantiated if it overrides each of the abstract methods of its superclass. Otherwise, the subclass object cannot be created that is what I want to say here. And here is basically if a subclass of an abstract class does not implement all the abstract method which is declared then; the abstract class then the sub class itself created as an abstract because we cannot create any object of that abstracts. So, these are the few properties of the abstract class it has a hold.

(Refer Slide Time: 11:52)


Multiple inheritance in Java

- However, this is not possible, as Java does not support multiple inheritance
- Java's solution to this problem is called `interface`.

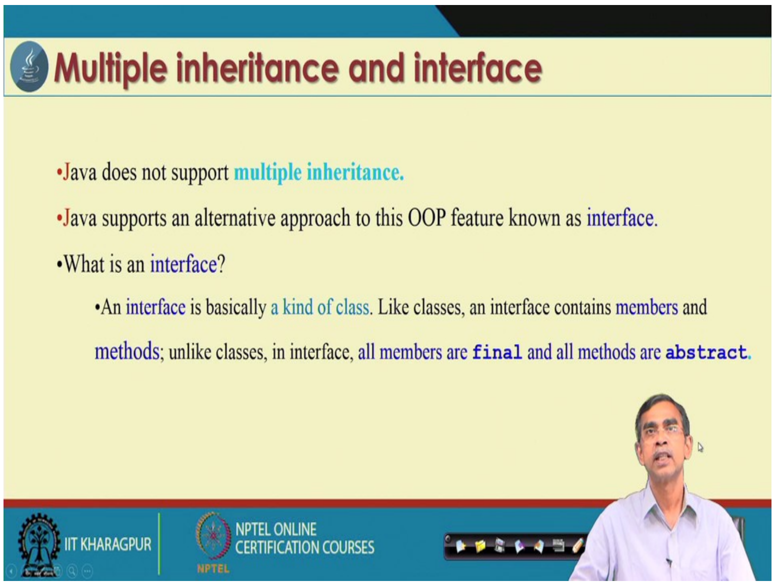
Anyway, now let us see about the concept that is the multiple inheritance is our objective then how the multiple inheritance is possible. They consider multiple inheritance is like this say suppose circle object should have characteristics from the geometry object that we have discussed. Now in addition to this, it will create some other characteristics from some other classes which are defined there in their draw shape.

So, suppose you want to create the circle with different what is called the colors, with different shading, with different zooms like this one. All these things suppose it is defined in this draw shape method. This means we want to define one circle which take advantage of both the geometry classes that we have already learned and then also in addition to draw circle.

So, this is the concept that if we are able to do; that means, if you able to inherit the characteristics from two or more classes then it is not a single inheritance it is called the multiple inheritance. However, Java does not support any multiple inheritance then how we can achieve this thing? So, there is an alternative way of doing these things; this is possible using an interface. Now, so here is the task is at what exactly an interface it is.

So, we are going to discuss the interface. In fact, I have introduced I have started this discussion we giving an introduction I mean discussion on abstract class this because you will learn about that interface an abstract class is in many ways very similar is very difficult to find a difference between the two. But the difference is that abstract class allows only single inheritance whereas, the interface which is very similar to an abstract class, but it supports inheritance multiple inheritances. So, this is the key difference between the abstract class and then interface in Java.

(Refer Slide Time: 13:47)


Multiple inheritance and interface

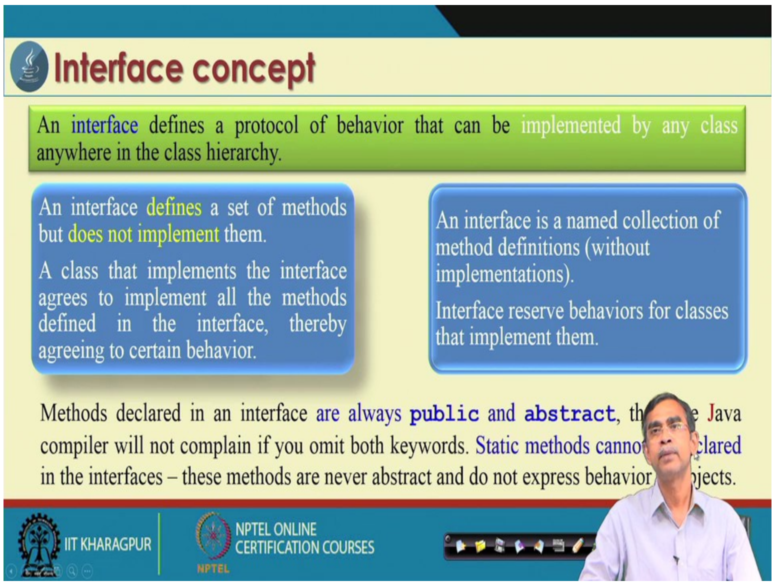
- Java does not support **multiple inheritance.**
- Java supports an alternative approach to this OOP feature known as interface.
- What is an interface?
  - An interface is basically a kind of class. Like classes, an interface contains members and methods; unlike classes, in interface, all members are `final` and all methods are `abstract`.

Now, let us have a full concept of the inheritance for multiple inheritance rather and the interface. As I told you so an interface is basically very similar to a class it is very similar to an abstract class more specifically. This means that like classes an interface can contain members and methods. However, the difference between the class and interface is that in case of interface all members are final and all methods are abstract.

There should not be any members which are nonfinal and any method which is non abstract. Now, this is the difference between the class and then interface whereas, if you see the class abstract class and then interface as you have already learned that the membership will be final. And then inter abstract even can container abstract method as well known abstract method.

This is also on the difference between the abstract class and interface concept and another thing is that for both abstract class and interface no object can be instantiated. So, this is a similar this thing we in that sense there are two things are similar. So, the difference is like this and then the similarity is also like this ok. So, this is the concept of interface in a very broad sense.

(Refer Slide Time: 15:06)



Interface concept

An interface defines a protocol of behavior that can be implemented by any class anywhere in the class hierarchy.

An interface defines a set of methods but does not implement them.

A class that implements the interface agrees to implement all the methods defined in the interface, thereby agreeing to certain behavior.

An interface is a named collection of method definitions (without implementations).

Interface reserve behaviors for classes that implement them.

Methods declared in an interface are always **public** and **abstract**, th e Java compiler will not complain if you omit both keywords. Static methods canno clared in the interfaces – these methods are never abstract and do not express behavior jects.

Now, let us have a more detail concept about it. As you have already discussed what exactly an abstract class is or rather what exactly an interface is. Now, as we see so what is the usage of abstract class or interface if you cannot instantiate an object, if you do not build an object of that class what is the use of it. Actually, all abstract class or an interface in Java basically gives a template for a class.

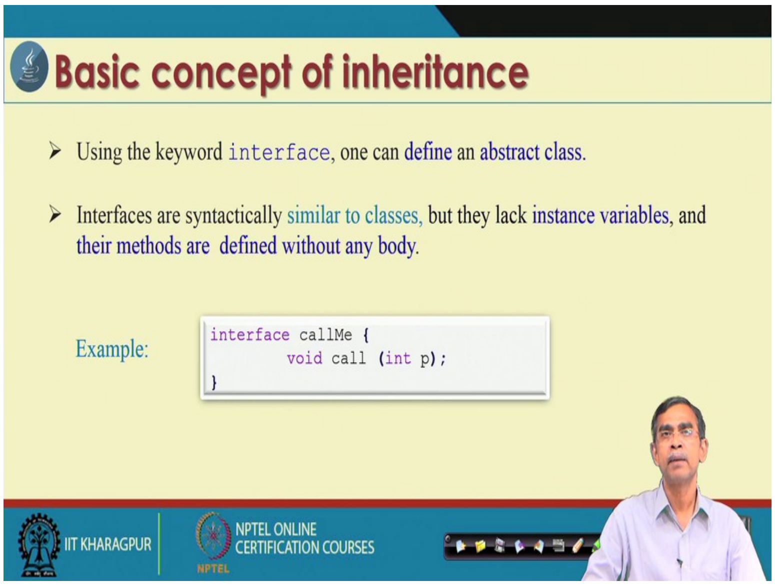
A framework for a class: so it basically gives a profile or a view that; how a class that you should have if you want to create your own program; if you follow the concept according to the abstract class and interface. So, abstract class and interface give a concept rather it gives a protocol of behavior that should be implemented by your Java programs that you want to develop it. So, it gives basically the rule of thumb about creating your own class taking the rule from the abstract class or interface this is the concept.

And as I have already told you an interface defines a set of methods but does not implement them, because they are declared as an abstract it is just like an abstract class also. And a class should implement that methods which is declared theirs as an abstract method in an interface. That means, the interface is a structure, and then if you want to create a class with support of interface then it is a responsibility of that class to implement.

That means, clearly defined all the method that if there in that sense. And as you have already told you all methods those are there in an interface they should be declared as a public and abstract. And by default, if you do not declare any access specification then it will be considered as a public and abstract. And otherwise so it is a thing and then static methods cannot be declared in an interface. So, that is one important restriction so far the thing is a concern there in.

But however, in the abstract class also you can declare a static method as well as. Anyway so interface it is like this all methods will be abstract more clearly all interface will be public and abstract and no static method should be declared there. However, members are a concern they should be declared a public final and also static. So, these are the thing that is equal so for the interface is concerns are there.

(Refer Slide Time: 17:44)



Basic concept of inheritance

- Using the keyword `interface`, one can define an abstract class.
- Interfaces are syntactically similar to classes, but they lack instance variables, and their methods are defined without any body.

Example:

```
interface callMe {
        void call (int p);
}
```

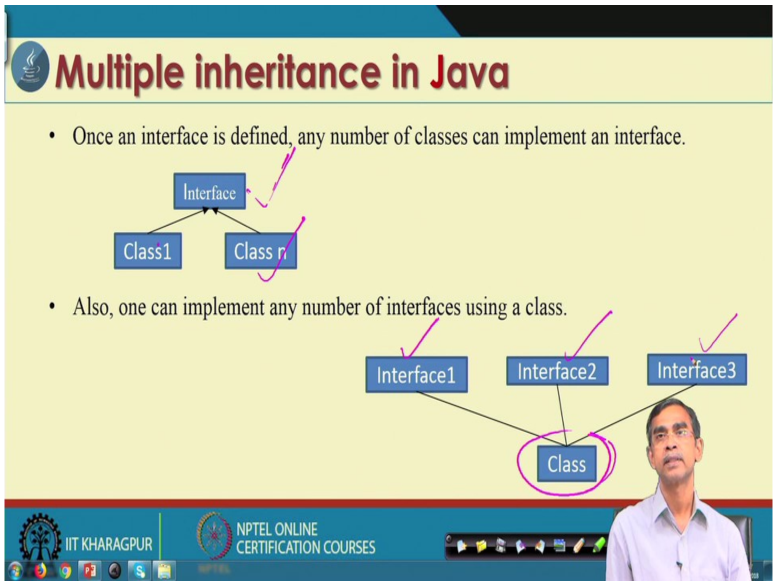
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES

Now, how an interface can be created. So, in order to create an interface, Java provides say keyword it is called the interface keyword. So, it is this is just similar to like the abstract class to create an abstract class we use the keyword abstract. So, it is like this, but there are syntactically a few more things are different that we should discuss it. Now here is an example; so you can see we have declared an interface the name of the interface is called.

So, interface keyword followed by the name of the interface and as it is similar to the declaration of a class with in the curly brackets. And then inside this as we see here it contains one method void call int p and then semicolon, As we see after the method declaration the immediately there is a semicolon this means that this method does not have any definition anybody any code.

So, this means that this is an abstract method and again I told you that as it is the default that this method is a public and abstract. So, we do not have to specify any keywords explicitly we mean you do not do it. So, compliable understand that this calling method is a public and abstract. That means any class can implement this interface.

(Refer to Slide Time: 19:13)



Multiple inheritance in Java

- Once an interface is defined, any number of classes can implement an interface.

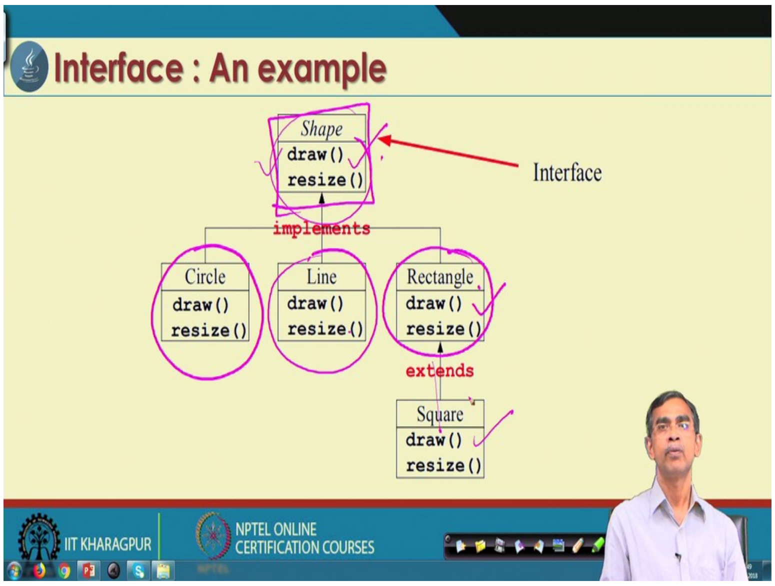
- Also, one can implement any number of interfaces using a class.

So, this is the syntax for declaring an interface. Now regarding interface if you have an interface using that interface you can build any class from that interface; that means. Here, for example, the interface is a class here we see the interface is you define one interface. Then this class 1 and then class 2 are the 2 implementations of this interface in the two different ways two different codes two different programs.

Now, so, this is one idea about it likewise here if we see if we have two or more interfaces like say interface 1, interface 2, interface 3. Like then a class you can plan or you can create you can build. So, that it can implement all this interface. Now we can see this is just like inheritance. What means this class inherits from this all the methods all the variables that are declared here it will be inherited here or like this one.

So, it is just like a multiple inheritances looks like that is the multiple inheritance concept in Java it is coming on the way. That if it interfaces you can do it, but if it is an abstract class you cannot do that all these things should not be an abstract class or any class rather ok. So, this is the idea that we will discuss more the multiple inheritances in Java using the interface in due time.

(Refer Slide Time: 20:44)



Interface : An example

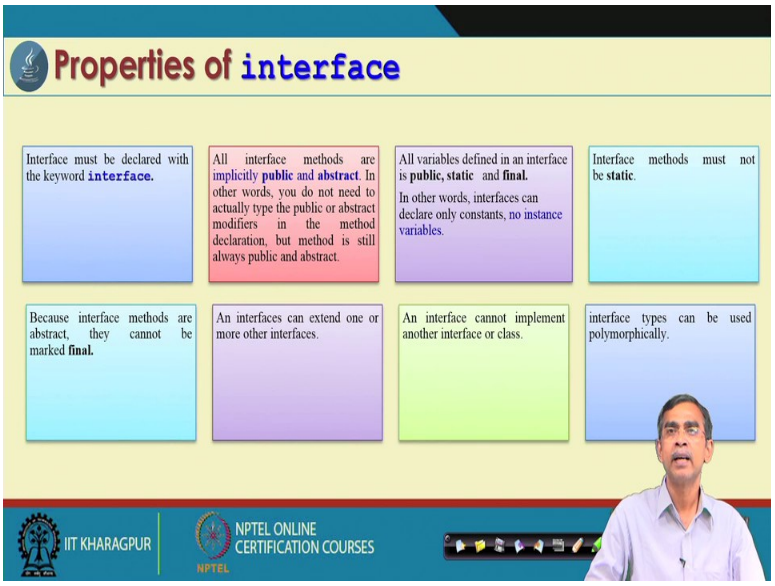
Now, let us have an example quickly; so that we can understand the concept of interface more clearly. As we see here, so this is a structure as we can see and led this structure we interface. Now in this structure what we have defined two methods draw and resize. They basically the method declaration, but not the body this means that all these methods are basically structured a framework a basically template a protocol so this is the interface suppose.

And then we can create our own class namely circle here which basically implement draw a resize method which is defined there. Now, it also can implement for the objects line. So, different methods draw and this one; that means, here the circle and line or rectangle. If we see the draw method here in circle or line or draw they are the different way of implementation. So, that is why the different implantation we can say.

So, this way what we can understand is that so, three different implementations, but it follows the template or protocol that is given their interface. So, this way the interface is coming into the way that we can use it. Now, further, if it is a class now this class can be used a normal class like. So, from this class, we can inherit some other class also with some other method overwriting whatever it is there.

So, all these things are quiet possible makes it more flexible, more versatile, and write any way whatever you want actually. Anyway, so this is the idea about that if you have an interface it gives you a I mean structure. So, that you can follow this structure to create a similar type of objects similar to what is called type of objects rather and then we can use this in our program. Now, so this is the concept that is there and we have a good example so that we can understand about it.

(Refer Slide Time: 23:00)



Properties of interface

Interface must be declared with the keyword **interface**.

All interface methods are implicitly **public** and **abstract**. In other words, you do not need to actually type the public or abstract modifiers in the method declaration, but method is still always public and abstract.

All variables defined in an interface is **public, static** and **final.** In other words, interfaces can declare only constants, no instance variables.

Interface methods must not be **static**.

Because interface methods are abstract, they cannot be marked **final.**

An interfaces can extend one or more other interfaces.

An interface cannot implement another interface or class.

interface types can be used polymorphically.

Now, before going to have the full code about the interface and everything; let us have briefly again summarized regarding the interface and its characteristics. As we have discussed interface must be declared with the keyword interface. All interface methods are implicitly public and abstract that I have already told you because you know need to mention explicitly.

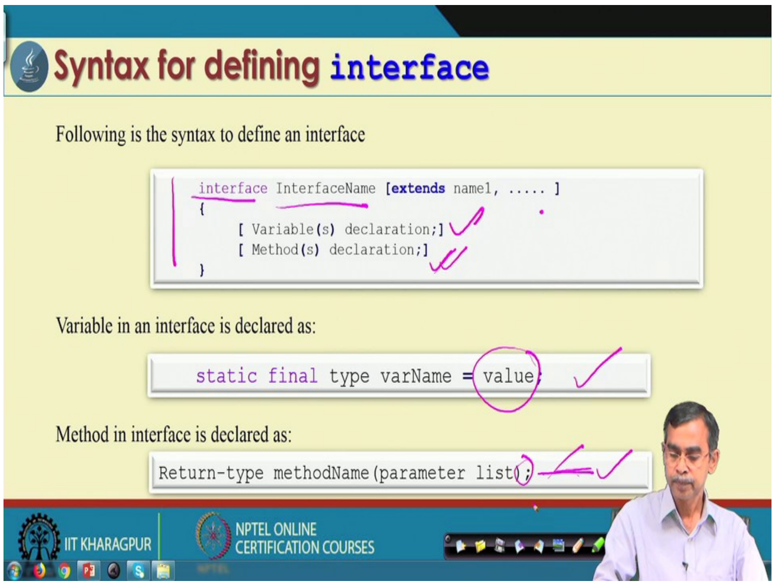
All variables those are there in an interface should be public, static, and final this is obvious; that this means that the variables those are there in an interface they should be treated as a constant. No class any method in other class which basically implements an interface cannot change their values. And interface methods must not be static any static method in an interface is allowed.

And because interface methods are abstract they cannot be marked final. So, you cannot do it because this needs to be implemented or defined in any other class which implements it. So, they should not be final then we all methods are abstract, but not final. And you can find a note that these are the difference between your abstract class and this interface class again.

Now, enter in addition to this and interface class can extend one or more other interfaces we have already discussed. An interface cannot implement another interface or class that is not possible. So, interface only a class can implement it, but the reverse is not possible actually. And interface types can be used polymorphically.

So, this is the concept is that an interface; if it is there is a type then this type can run timely bind is if in the example of extra class we have shown that circle ellipse and then rectangular how dynamically or run time or polymorphically bind with the method called area like this one; so it is like this ok.

(Refer Slide Time: 25:03)



Syntax for defining interface

Following is the syntax to define an interface

```
interface InterfaceName [extends name1, ..... ]
{
    [ Variable(s) declaration;]
    [ Method(s) declaration;]
}
```

Variable in an interface is declared as:

```
static final type varName = value;
```

Method in interface is declared as:

```
Return-type methodName(parameter list);
```

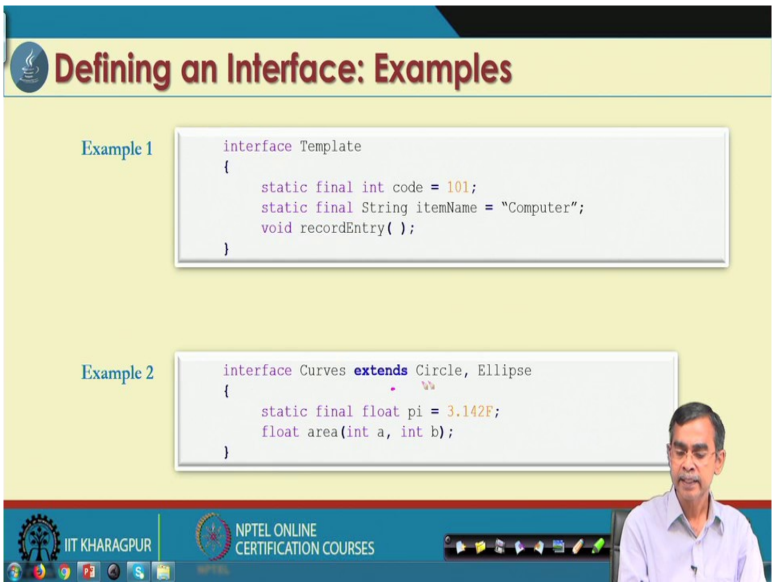
So, now here is a quick syntactical structure about how the interface should be declared in a program. Now so this is the syntax that we can see the interface should be declared with the interface keyword the name of the interface and these are the option. An interface can extend any other interface.

So, these are the name one and everything, but this is the interface, not the class actually. Anyway and these are the variable declaration as we have declared all this variable should be declared static final and abstract as well as, and then these are the public static and final. And these are the method abstract and abstract method, a public abstract method so these are the methods.

Now, so this is basically an example how a variable or a member in an interface look like and these value once it assigns it will remain this you cannot change it because it is static. And then this is a return type that is basically only you can see the semicolon no body it is here. After this declaration that means this method is declared as an abstract method and as it is public also keyword is by default it is there ok.

So, this is the idea about how we can create our interface in a program the syntax that we have discussed it and it is an example.

(Refer Slide Time: 26:28)



Defining an Interface: Examples

Example 1

```
interface Template
{
    static final int code = 101;
    static final String itemName = "Computer";
    void recordEntry( );
}
```

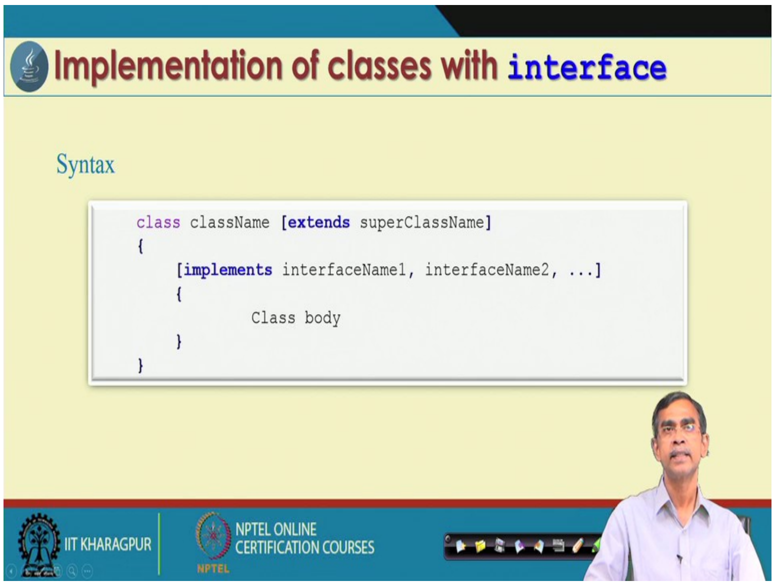
Example 2

```
interface Curves extends Circle, Ellipse
{
    static final float pi = 3.142F;
    float area(int a, int b);
}
```

So, an item is an interface declaration; it has the variable this is the variable which is declared as static final. And this is the one another variable static final and this is the one method that we have discussed here. So this is one example that we can understand how it can be used to create an interface. Now, this is another example so, we can declare here a temperature an interface this is a similar kind of things so it is basically templated. And using this interface we can create a class.

Here we can see in this example interface this is the name of the interface is curves and then extends circle and ellipse. That means, here the curve is an interface which basically extends circle and ellipse types. That means these curves will include all the features which are there in both circle and interface. And in addition to this, it has a method all these things like this.

(Refer Slide Time: 27:46)



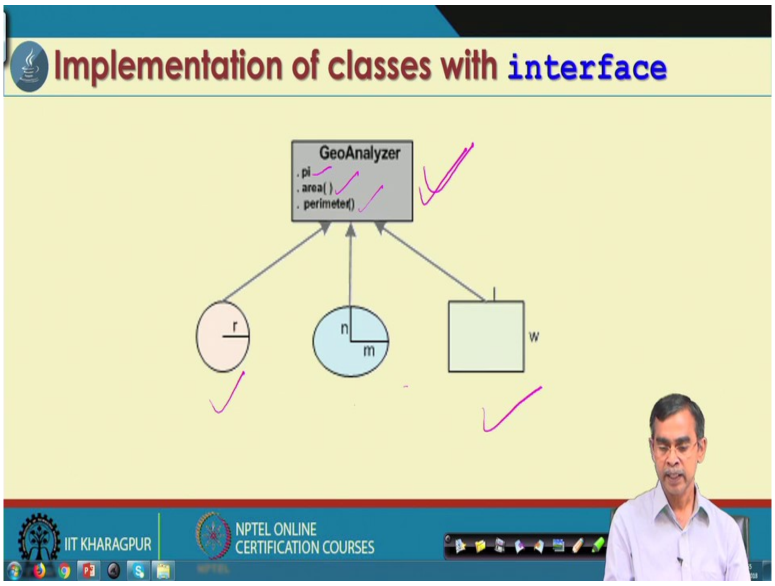
Implementation of classes with `interface`

Syntax

```
class className [extends superClassName]
{
    [implements interfaceName1, interfaceName2, ...]
    {
            Class body
    }
}
```

Now, basic syntax, that after one interface is created how it can be utilized in our program. So, it is the concept here then how to interface can be used. So, we can declare a class name which basically extends this one and implements the interface. So, there is basically a class can extend some other class as well as it implements. So, it is an option. So, this way a class one interface if it is defined then it can be used subsequently in the program.

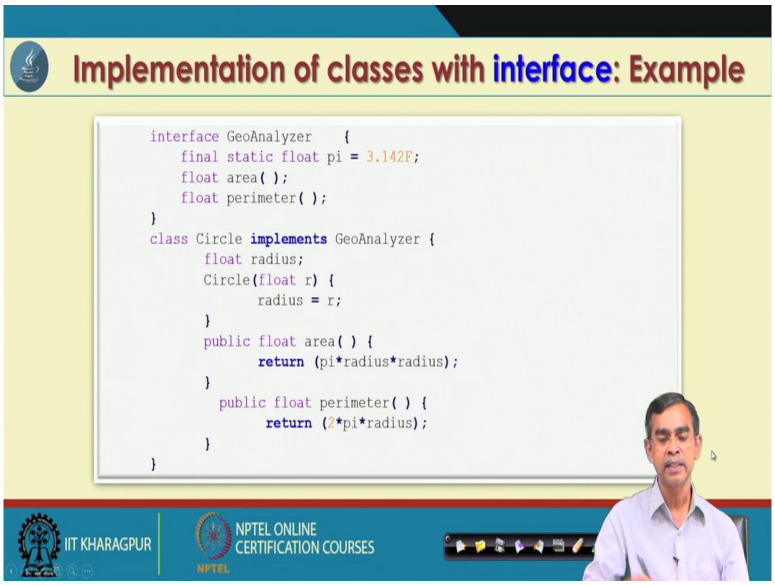
(Refer Slide Time: 28:19)



Implementation of classes with `interface`

Now, here is a quick example this is a very nice example to understand the concept of the interface it is in the same line, as the abstract class example that we have discussed earlier. Now here we can see this is an interface structure as we know it has these are the members static final abstract. And these are the method abstract and public and we want to create three types of object like circle ellipse and rectangle using this concept interface. Now, let us have the quick look of this how it is possible a good program is there.

(Refer Slide Time: 28:51)



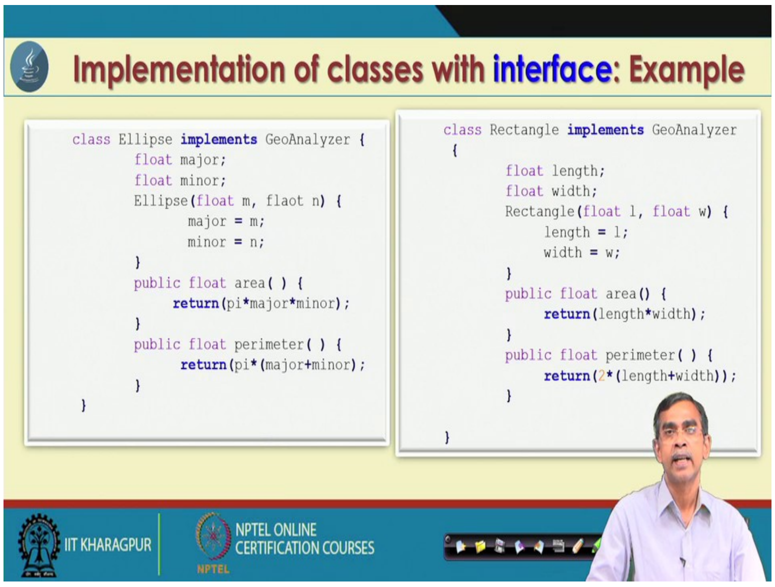
Implementation of classes with interface: Example

```
interface GeoAnalyzer    {
    final static float pi = 3.142F;
    float area( );
    float perimeter( );
}
class Circle implements GeoAnalyzer {
       float radius;
       Circle(float r) {
              radius = r;
       }
       public float area( ) {
              return (pi*radius*radius);
       }
         public float perimeter( ) {
               return (2*pi*radius);
       }
}
```

Now, here we can give the name of that interface as a GeoAnalyzer. This has the methods as we have discussed and circle here you can see this circle implements GeoAnalyzer. And this implantation means it will implement this method it implements this method. So, this is the constructor is automatically for this class we have to do it. And this is an implementation of this area and this is the implementation of this the area. So, this completes the usage of an interface in order to implement the circle.

Similarly, likewise, we can create another class for the geometrical object like an ellipse. So, this is a similar way of creating the class object ellipse and this is the implementation of the rectangle. Now we can understand that using the interface and the implements it does not extend you can see in case of abstract class we may use extends; that means, derived class. But here actually an implementation of the interface which is a totally an abstract concept and then we can have this kind of objects created.

(Refer Slide Time: 29:26)



Implementation of classes with interface: Example

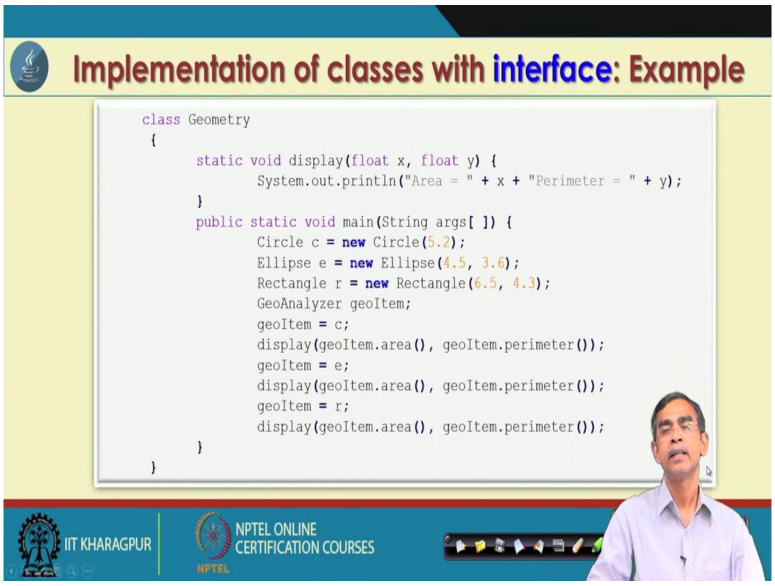
```
class Ellipse implements GeoAnalyzer {
        float major;
        float minor;
        Ellipse(float m, flaot n) {
                major = m;
                minor = n;
        }
        public float area( ) {
                return(pi*major*minor);
        }
        public float perimeter( ) {
                return(pi*(major+minor);
        }
}
```

```
class Rectangle implements GeoAnalyzer
{
        float length;
        float width;
        Rectangle(float l, float w) {
                length = l;
                width = w;
        }
        public float area() {
                return(length*width);
        }
        public float perimeter( ) {
                return(2*(length+width));
        }
}
```

Now, once this kind of object is created we can create the object see our own method. It is very similar to the abstract class concept in the same line we can see let this is the one class that we have declared as a main class name of the main class is geometry. It obviously, input all the classes that we have created earlier so if they are put in a package.

Now, after these things we create one method which is a special method in this class void display. It basically x and y value will display it is like this. Now let us see the main method that we have declared here. So, these main methods create the three objects circle ellipse they store in the form of this object. And then we create one what is called it is not an instantiation we just create we declare we define an object of type GeoAnalyzer that fine. That means, it is an interface type GeoAnalyzer is an interface in our example. So, here we see the GeoItem and interface item basically we have created.

(Refer Slide Time: 28:51)



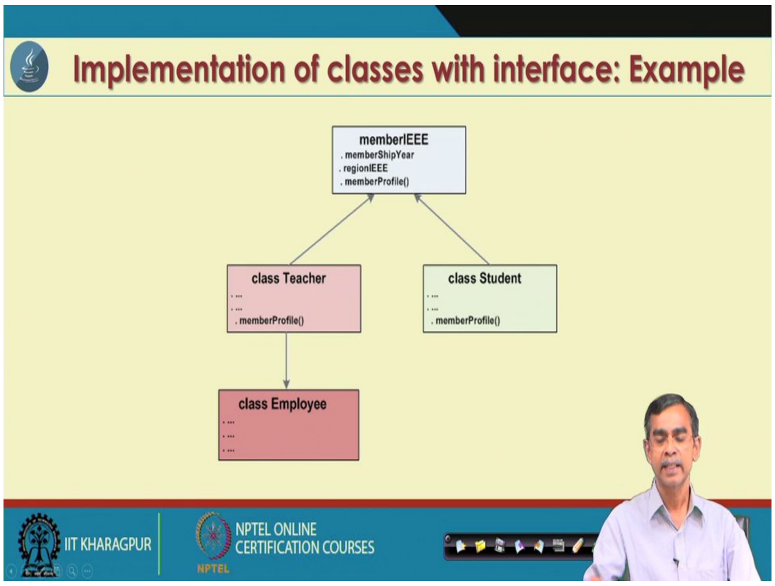
Implementation of classes with interface: Example

```
class Geometry
 {
       static void display(float x, float y) {
               System.out.println("Area = " + x + "Perimeter = " + y);
       }
       public static void main(String args[ ]) {
               Circle c = new Circle(5.2);
               Ellipse e = new Ellipse(4.5, 3.6);
               Rectangle r = new Rectangle(6.5, 4.3);
               GeoAnalyzer geoItem;
               geoItem = c;
               display(geoItem.area(), geoItem.perimeter());
               geoItem = e;
               display(geoItem.area(), geoItem.perimeter());
               geoItem = r;
               display(geoItem.area(), geoItem.perimeter());
       }
 }
```

Now, here once a Circle c is created we can assign this one. That means, GeoItem now points to c; that means, circle and then the display method which is declared here can be called you see whatever be the object the display method looks like the same thing right. If you do not look about this one and this one display a method is basically irrespective of the circle rectangle ellipse it basically over the same.

So, this is again the concept it is called the dynamically binding or it is a polymorphically run time polymorphically binding it is. Now, what we can say is that whatever be the object it is their area and perimeter can be utilized in this program. So, this is the idea about we have learned about how the interface concept can be used to build class. And therefore, subsequently, build the Java program.

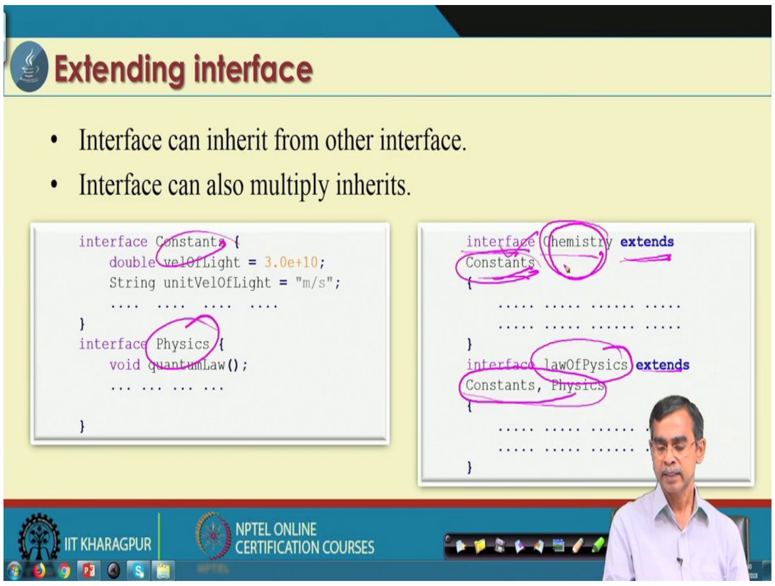
(Refer Slide Time: 31:51)



Implementation of classes with interface: Example

The same way the idea can be extended for the other, this is another example that you can consider.

(Refer Slide Time: 32:01)



Extending interface

- Interface can inherit from other interface.
- Interface can also multiply inherits.

```
interface Constants {
    double velOfLight = 3.0e+10;
    String unitVelOfLight = "m/s";
    ....  ....  ....  ....
}
interface Physics {
    void quantumLaw();
    ... ... ... ...

}
```

```
interface Chemistry extends
Constants
{
    ..... ..... ...... .....
    ..... ..... ...... .....
}
interface lawOfPysics extends
Constants, Physics
{
    ..... ..... ...... .
    ..... ..... ...... .
}
```

Now, inheritance with the interface as I told you an interface also can be used to extend another interface. So, this is one example that inheritance is the same concept as a class. That means, the interface can be created as a class so, for the inheritance is concern interface. As we see here the interface is interfaced chemistry extend this one. That means this is the interface used to extend the constant where constant is an interface.

So, this is an interface and here we can see this interface extends interface creates another interface chemistry inheriting the constant interface. Here, we can see interface create another interface law of physics inheriting the interfaces that are constant and physic. So, it is the same concept it is just if you just like it is a class inherits others like this one. So, it is the concept that inheritance also it is possible.

Now, so this is the idea about a brief idea about the inheritance. And we will discuss more inheritance in our next module.

Thank you.