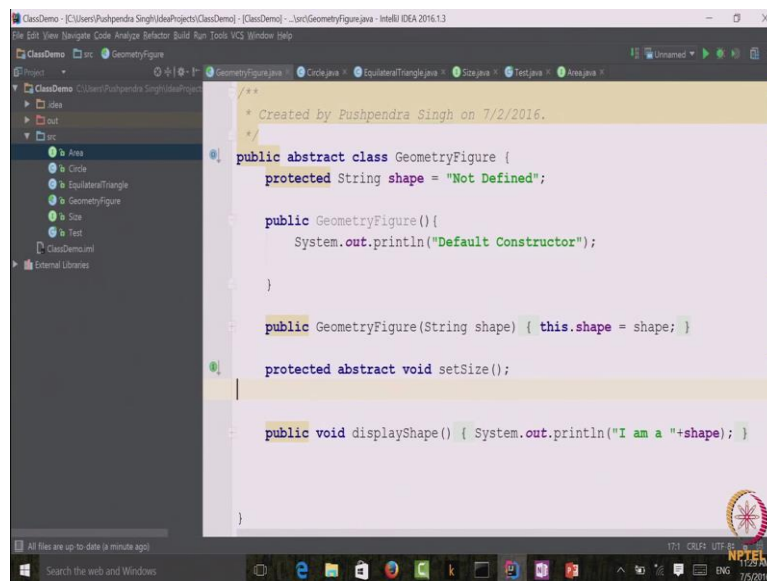


Mobile Computing
Professor Pushpedra Singh
Indraprasth Institute of Information Technology Delhi
Java Basics
Lecture 05

Hello, today we will see a program that will show you a concept of classes, abstract classes, interfaces and inheritance.

(Refer Slide Time: 0:22)



```
/**
 * Created by Pushpedra Singh on 7/2/2016.
 */
public abstract class GeometryFigure {
    protected String shape = "Not Defined";

    public GeometryFigure(){
        System.out.println("Default Constructor");
    }

    public GeometryFigure(String shape) { this.shape = shape; }

    protected abstract void setSize();

    public void displayShape() { System.out.println("I am a "+shape); }
}
```

Here as you can see I have created a project which has certain classes and interface. Let us go to our first class the class name is geometry figure. You may be already familiar with the geometry. So for example triangle, square, rectangle, circle.

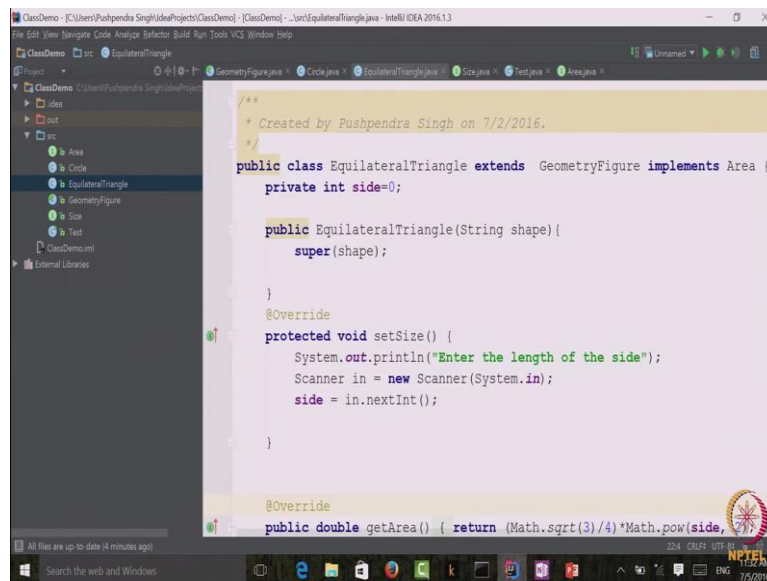
Now let's see so I have created an abstract class called geometry figure. In that abstract class I have created a protected string called shape and I have initialized it as not define. Then I have created a constructor for the geometry figure class which only prints out that it is the default constructor. After that I have created another constructor for geometry figure which takes a string and set the string variable of the geometry figure top the value given.

As you will see that I am using here the key word called this. In the previous lecture I have told you that this is used to refer to the object that you are currently working with. So this dot shape actually refers to the shape variable mentioned here. So when I set this dot shape equal to shape value of this variable will be set equal to value given here. Then I describe an abstract method called set size. And then I describe a public method called display shape. The

purpose of this program is to only show you the concept of an abstract class, interfaces, modifier such as protected public and private and inheritance.

Now because I have defined an abstract class called geometry figure I will be inheriting this class into sub classes. I have created two sub classes. My first sub class is equilateral triangle, you may already know what an equilateral triangle is.

(Refer Slide Time: 2:33)



```
/**
 * Created by Pushpendra Singh on 7/2/2016.
 */
public class EquilateralTriangle extends GeometryFigure implements Area {
    private int side=0;

    public EquilateralTriangle(String shape) {
        super(shape);
    }

    @Override
    protected void setSize() {
        System.out.println("Enter the length of the side");
        Scanner in = new Scanner(System.in);
        side = in.nextInt();
    }

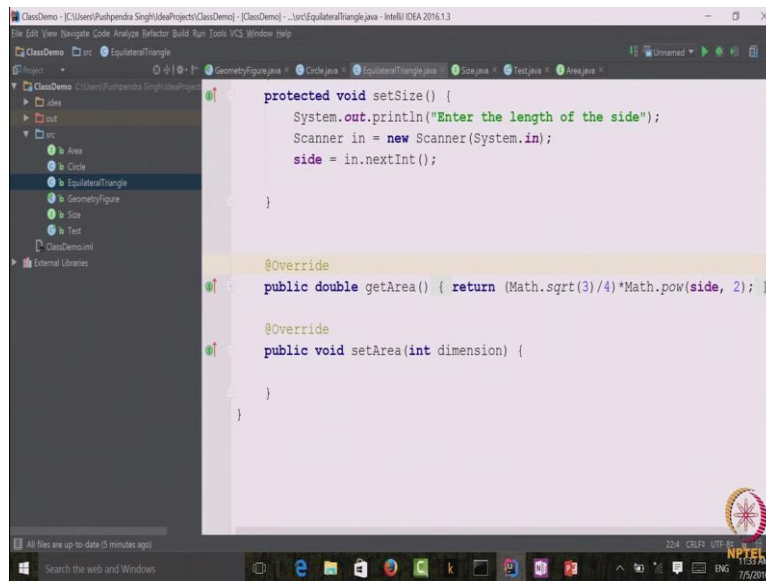
    @Override
    public double getArea() { return (Math.sqrt(3)/4)*Math.pow(side, 2);
}
```

An equilateral triangle is the triangle which has all the side of same length. As you can see that an equilateral triangle is a geometry figure so I can see that there is a very clear inheritance relationship here. Similarly circle is a geometry figure that's why the circle would be the another sub class of the abstract class geometry figure.

Now lets first cover the equilateral triangle. Number one that equilateral triangle has a private variable called side because the side is a property of the triangle only and that's why I did not declare the side variable in the super class which means the geometry figure class but I am declaring it in the subclass. Equilateral triangle constructor calls the constructor of the super class using the super key word.

And then it goes on to the right to over write the method called set size. You may also see that equilateral triangle implementing area which is an interface. Lets first cover the equilateral triangle then we will go to the area interface and then we will come back to the equilateral triangle class to understand.

(Refer Slide Time: 3:56)



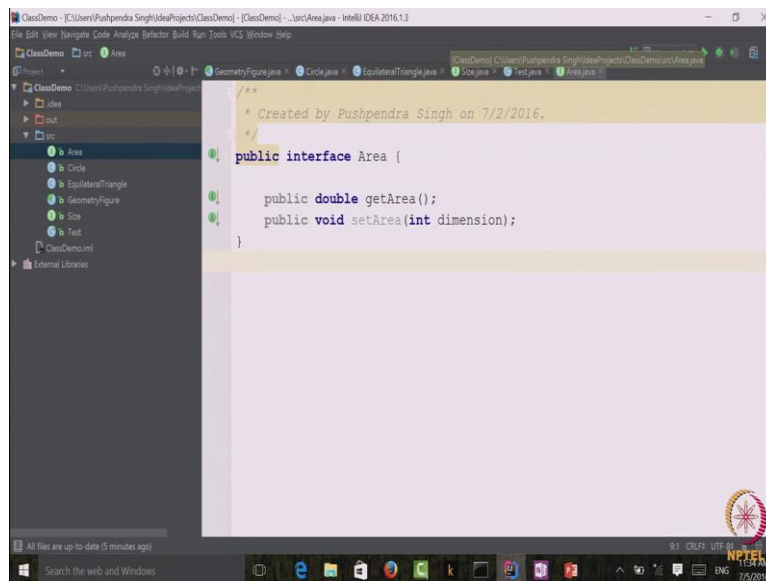
```
protected void setSize() {
    System.out.println("Enter the length of the side");
    Scanner in = new Scanner(System.in);
    side = in.nextInt();
}

@Override
public double getArea() { return (Math.sqrt(3)/4)*Math.pow(side, 2); }

@Override
public void setArea(int dimension) {
}
```

Then equilateral triangle is again overriding two other methods called get area and set area. Set area actually does nothing. Let see where these methods are coming from.

(Refer Slide Time: 4:15)



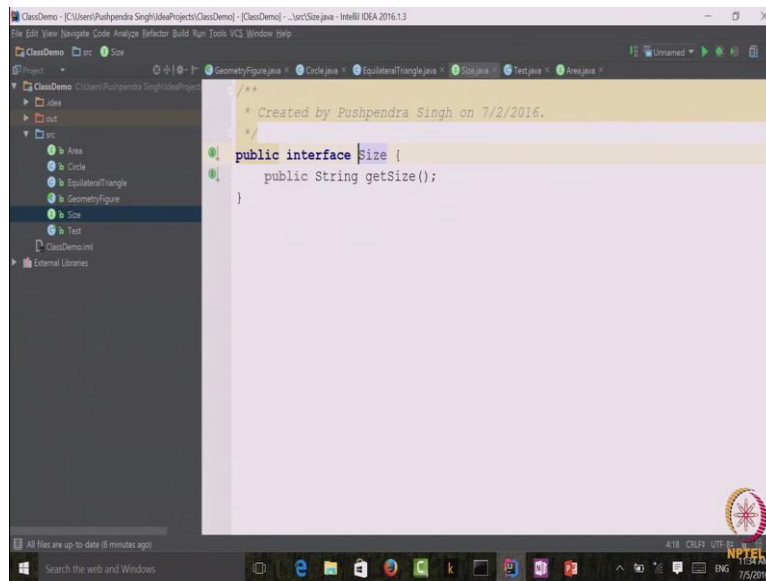
```
Created by Pushpendra Singh on 7/2/2016.

public interface Area {

    public double getArea();
    public void setArea(int dimension);
}
```

Area, so I have defined an interface called area the centre face has two methods one is the get area and one is a set area. I have not I have only declared these methods but have not given any implementation. In interface we cannot give implementation of a method and we can only declare that.

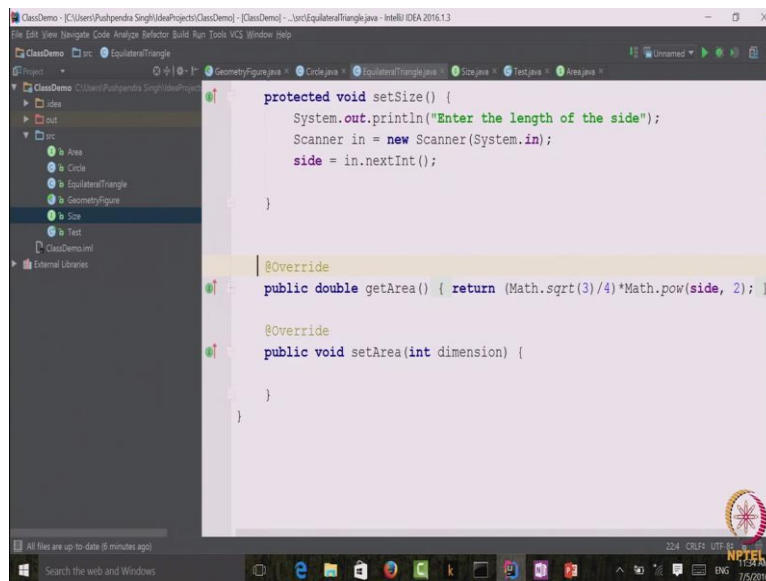
(Refer Slide Time: 4:46)



```
Created by Pushpendra Singh on 7/2/2016.  
  
public interface Size {  
    public String getSize();  
}
```

Similarly I have another interface here which is called size. This interface only one method called get size.

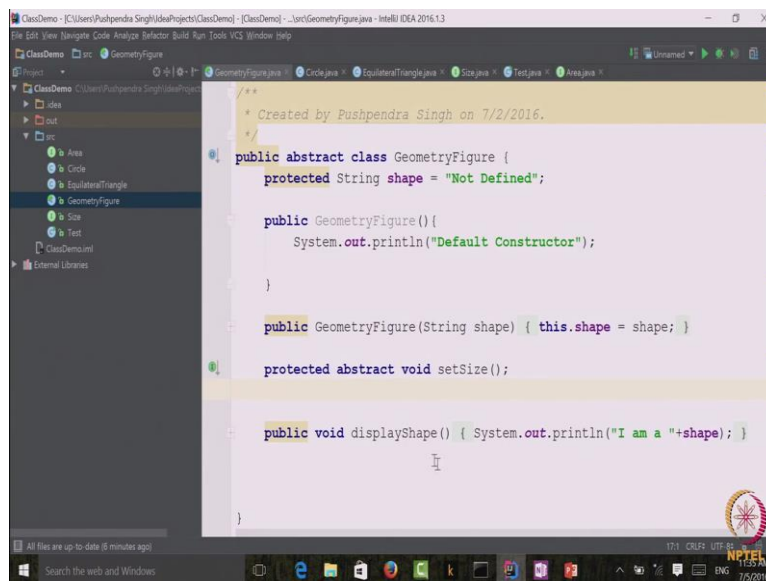
(Refer Slide Time: 5:05)



```
protected void setSize() {  
    System.out.println("Enter the length of the side");  
    Scanner in = new Scanner(System.in);  
    side = in.nextInt();  
}  
  
@Override  
public double getArea() { return (Math.sqrt(3)/4)*Math.pow(side, 2); }  
  
@Override  
public void setArea(int dimension) {  
  
}
```

Now every class which will implement these interface must over write these methods that's the reason why equilateral triangle was over riding methods called get area and set area because equilateral triangle class is implementing the interface called area.

(Refer Slide Time: 5:25)



```
Created by Pushpendra Singh on 7/2/2016.

public abstract class GeometryFigure {
    protected String shape = "Not Defined";

    public GeometryFigure(){
        System.out.println("Default Constructor");
    }

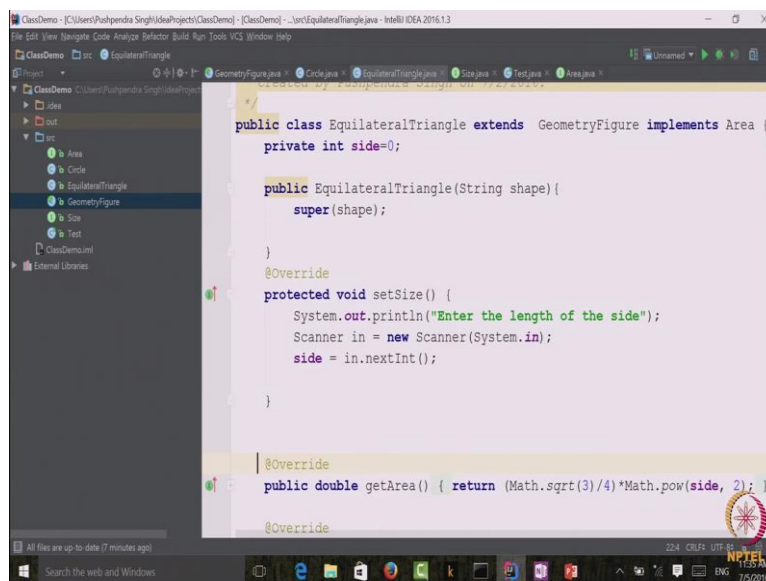
    public GeometryFigure(String shape) { this.shape = shape; }

    protected abstract void setSize();

    public void displayShape() { System.out.println("I am a "+shape); }
}
```

Equilateral triangle also is having axis to its parent class method called display shape without having any need to over write or extends it. However the equilateral triangle class must extend the method called set size.

(Refer Slide Time: 5:43)



```
Created by Pushpendra Singh on 7/2/2016.

public class EquilateralTriangle extends GeometryFigure implements Area {
    private int side=0;

    public EquilateralTriangle(String shape){
        super(shape);
    }

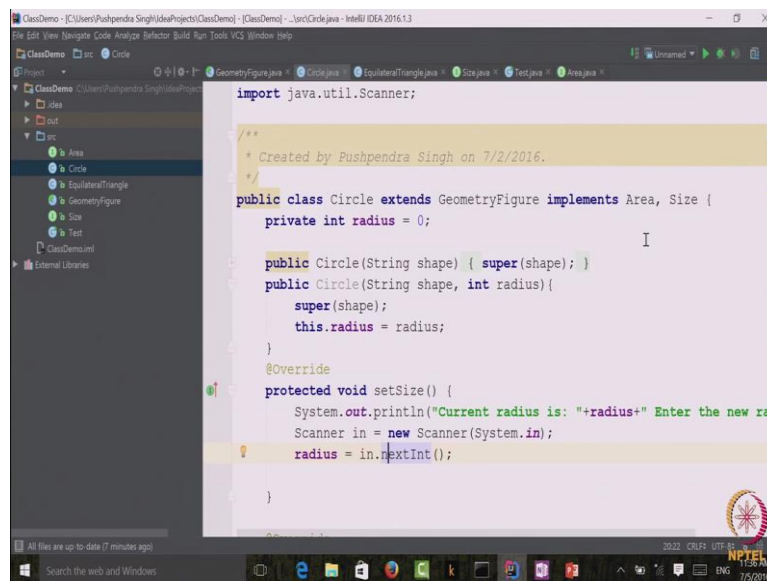
    @Override
    protected void setSize() {
        System.out.println("Enter the length of the side");
        Scanner in = new Scanner(System.in);
        side = in.nextInt();
    }

    @Override
    public double getArea() { return (Math.sqrt(3)/4)*Math.pow(side, 2); }

    @Override
}
```

Because set size is described as an abstract method in the class geometry figure. So equilateral triangle is indeed extending the method set size and implementing it. Now let us see what we have in another sub class called circle.

(Refer Slide Time: 5:56)

A screenshot of an IDE window showing the source code for a Java class named 'Circle'. The code is as follows:

```
import java.util.Scanner;

/**
 * Created by Pushendra Singh on 7/2/2016.
 */
public class Circle extends GeometryFigure implements Area, Size {
    private int radius = 0;

    public Circle(String shape) { super(shape); }
    public Circle(String shape, int radius) {
        super(shape);
        this.radius = radius;
    }

    @Override
    protected void setSize() {
        System.out.println("Current radius is: "+radius+" Enter the new rad
        Scanner in = new Scanner(System.in);
        radius = in.nextInt();
    }
}
```

The IDE interface includes a project explorer on the left, a main editor area with the code, and a Windows taskbar at the bottom showing the date and time as 7/5/2016, 11:58 AM.

As you see a circle is also extending geometry figure and circle is implementing area as well as size means circle is implementing both of the interfaces. Circle has a private variable called radius. If you recall equilateral triangle had the private variable called side. This private variable indicates that they are private to the sub class and this is a correct approach because here I made a radius a variable in the super class and side also variable in the super class. The side would not have been useful for the circle sub class. While radius would not have been useful for equilateral sub class.

Now the circle has constructor which is doing nothing but calling (th) the constructor of the super class. Then circle has another constructor which is calling the constructor of the super class but is also setting the radius of the circle.

(Refer Slide Time: 6:59)

```
ClassDemo - [C:\Users\Pushpendra Singh\IdeaProjects\ClassDemo - [ClassDemo] - \src\Circle.java - IntelliJ IDEA 2016.13
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
ClassDemo src Circle
Project
ClassDemo C:\Users\Pushpendra Singh\IdeaProjects
idea
out
src
Area
Circle
EquilateralTriangle
GeometryFigure
Size
Test
ClassDemo.iml
External Libraries
All files are up-to-date (8 minutes ago)
2022 CRLF UTF-8
NPTEL
1:37 AM
7/5/2016

public Circle(String shape) { super(shape); }
public Circle(String shape, int radius) {
    super(shape);
    this.radius = radius;
}
@Override
protected void setSize() {
    System.out.println("Current radius is: "+radius+" Enter the new rad
Scanner in = new Scanner(System.in);
radius = in.nextInt();
}
@Override
public String getSize() { return("I am a circle of radius: "+ radius); }
@Override
public double getArea() { return Math.PI*radius*radius; }
@Override
public void setArea(int dimension) {
```

```
ClassDemo - [C:\Users\Pushpendra Singh\IdeaProjects\ClassDemo - [ClassDemo] - \src\EquilateralTriangle.java - IntelliJ IDEA 2016.13
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
ClassDemo src EquilateralTriangle
Project
ClassDemo C:\Users\Pushpendra Singh\IdeaProjects
idea
out
src
Area
Circle
EquilateralTriangle
GeometryFigure
Size
Test
ClassDemo.iml
External Libraries
All files are up-to-date (8 minutes ago)
2022 CRLF UTF-8
NPTEL
1:37 AM
7/5/2016

public class EquilateralTriangle extends GeometryFigure implements Area {
    private int side=0;
    public EquilateralTriangle(String shape) {
        super(shape);
    }
    @Override
    protected void setSize() {
        System.out.println("Enter the length of the side");
        Scanner in = new Scanner(System.in);
        side = in.nextInt();
    }
    @Override
    public double getArea() { return (Math.sqrt(3)/4)*Math.pow(side, 2); }
    @Override
```

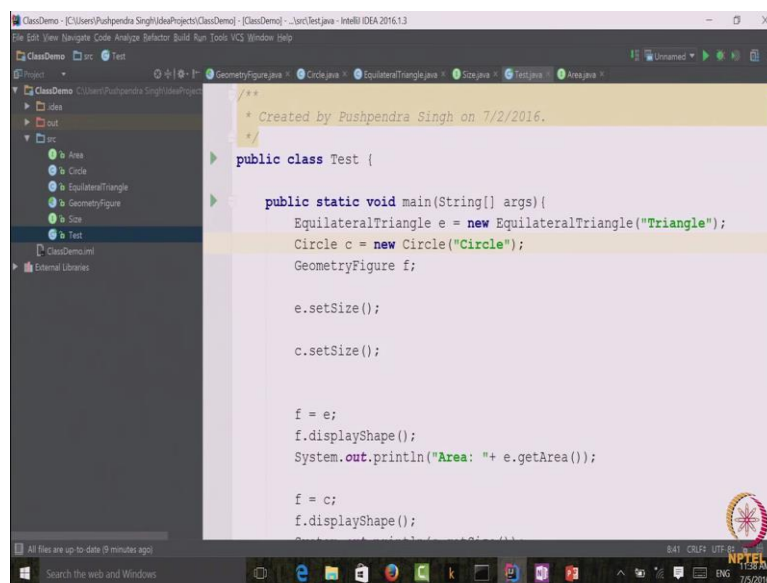
```
ClassDemo - [C:\Users\Pushpendra Singh\IdeaProjects\ClassDemo - [ClassDemo] - \src\Circle.java - IntelliJ IDEA 2016.13
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help
ClassDemo src Circle
Project
ClassDemo C:\Users\Pushpendra Singh\IdeaProjects
idea
out
src
Area
Circle
EquilateralTriangle
GeometryFigure
Size
Test
ClassDemo.iml
External Libraries
All files are up-to-date (8 minutes ago)
2022 CRLF UTF-8
NPTEL
1:37 AM
7/5/2016

@Override
protected void setSize() {
    System.out.println("Current radius is: "+radius+" Enter the new rad
Scanner in = new Scanner(System.in);
radius = in.nextInt();
}
@Override
public String getSize() { return("I am a circle of radius: "+ radius); }
@Override
public double getArea() { return Math.PI*radius*radius; }
@Override
public void setArea(int dimension) {
}
}
```

After that the circle also need to extend or implement the set size function or the set size method which is described in the super class. For circle the implementation of set size is different than the implementation of set size in equilateral triangle. As you can see that both methods are implemented differently.

After that circle needs to implement all the methods of both of the interfaces that is area and size. Which circle does? Now we have got our basic program ready. Which means that we have two classes circle and equilateral triangle they are extending a super class called geometry figure and one of them is implementing one interface and the another of them is implementing both of the interfaces. Now let us see polymorphism in action.

(Refer Slide Time: 7:56)



```
Created by Pushpendra Singh on 7/2/2016.  
public class Test {  
  
    public static void main(String[] args){  
        EquilateralTriangle e = new EquilateralTriangle("Triangle");  
        Circle c = new Circle("Circle");  
        GeometryFigure f;  
  
        e.setSize();  
  
        c.setSize();  
  
        f = e;  
        f.displayShape();  
        System.out.println("Area: "+ e.getArea());  
  
        f = c;  
        f.displayShape();  
    }  
}
```

I have created a separate class called as test for testing the functionality of the classes that we have previously created. Because we want the test class to run that's why we need to have a main method in the test class as discussed in the previous lecture the main method should be declare static and public. Which means that in order to use the main method I need not to create a object of the test class.

Static ensures that I can run main method without creating an object. Public ensures that I can run main e method without being a part of the test class. Now let us see what we have we declare (e) object of (e) equilateral triangle E. E is the variable that refers to the object similarly we do for the circle where C is the variable that refers to the circle as you see that we are calling the constructor which takes the string as an input and we also declare a

variable of abstract class called geometry figure. As we saw earlier we cannot create object of an abstract class. But we can create variable of an abstract class.

Now I am calling the method E dot set size and C dot set size. Which essentially means that I am calling the same method for the variable of equilateral triangle class and the variable of the circle class. Okay. This is what we refer to when we mention method over writing the set size is the same name. Method over writing allows your program to be more readable and understandable. Now I only need to remember the method called set size. And whatever variable I will use java dynamic binding will make sure that the method from that class as well.

(Refer Slide Time: 9:56)

```
Circle c = new Circle("circle");
GeometryFigure f;

e.setSize();

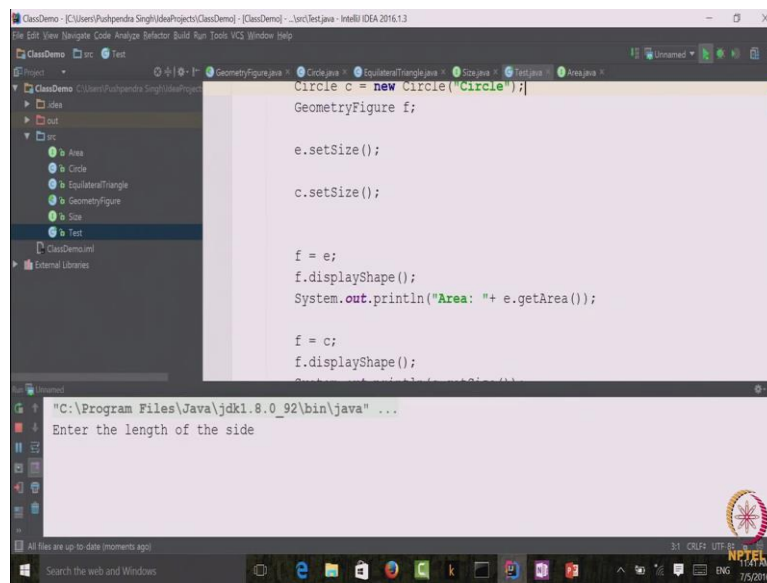
c.setSize();

f = e;
f.displayShape();
System.out.println("Area: " + e.getArea());

f = c;
f.displayShape();
System.out.println(c.getSize());
}
```

In order to show polymorphism I assign F to E and then I run the method called display shape. I also print out the area using the get area. After that I assign F to C and I run display shape and I print out the size of C by using the method called get size. Lets run this program and see what kind of output we get and then we should come back to the program to understand how that the output is obtained.

(Refer Slide Time: 10:37)



```
Circle c = new Circle("circle");
GeometryFigure f;

e.setSize();

c.setSize();

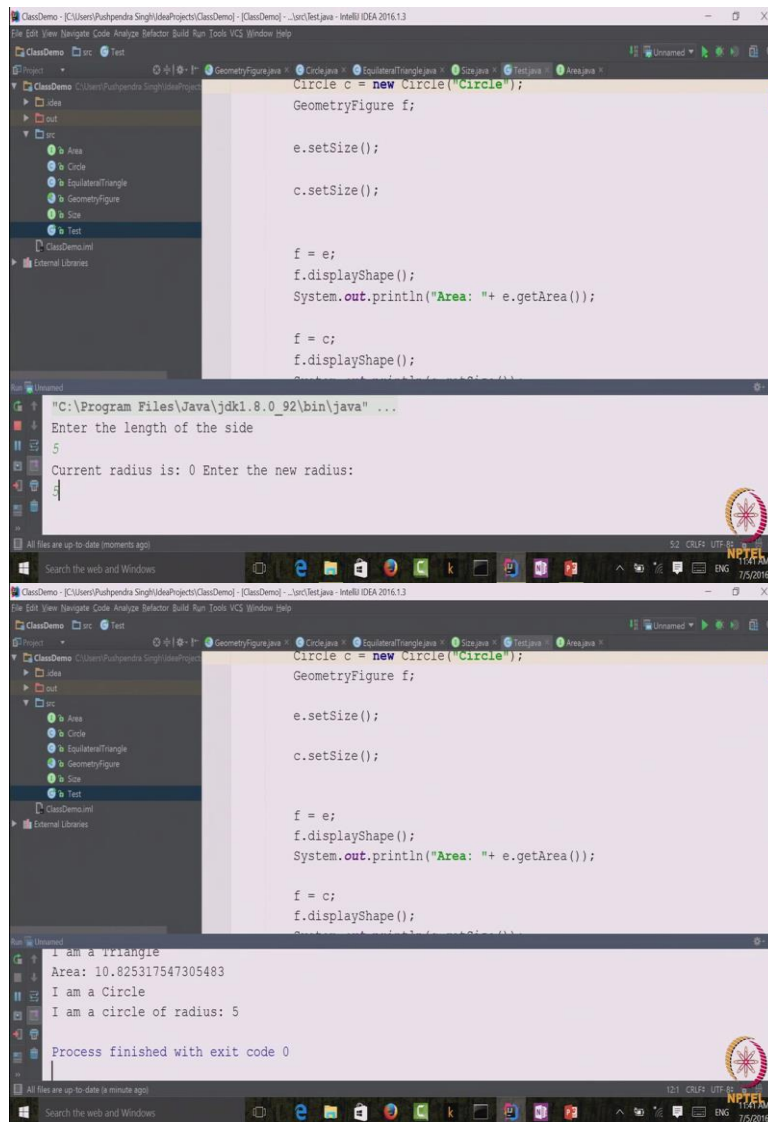
f = e;
f.displayShape();
System.out.println("Area: " + e.getArea());

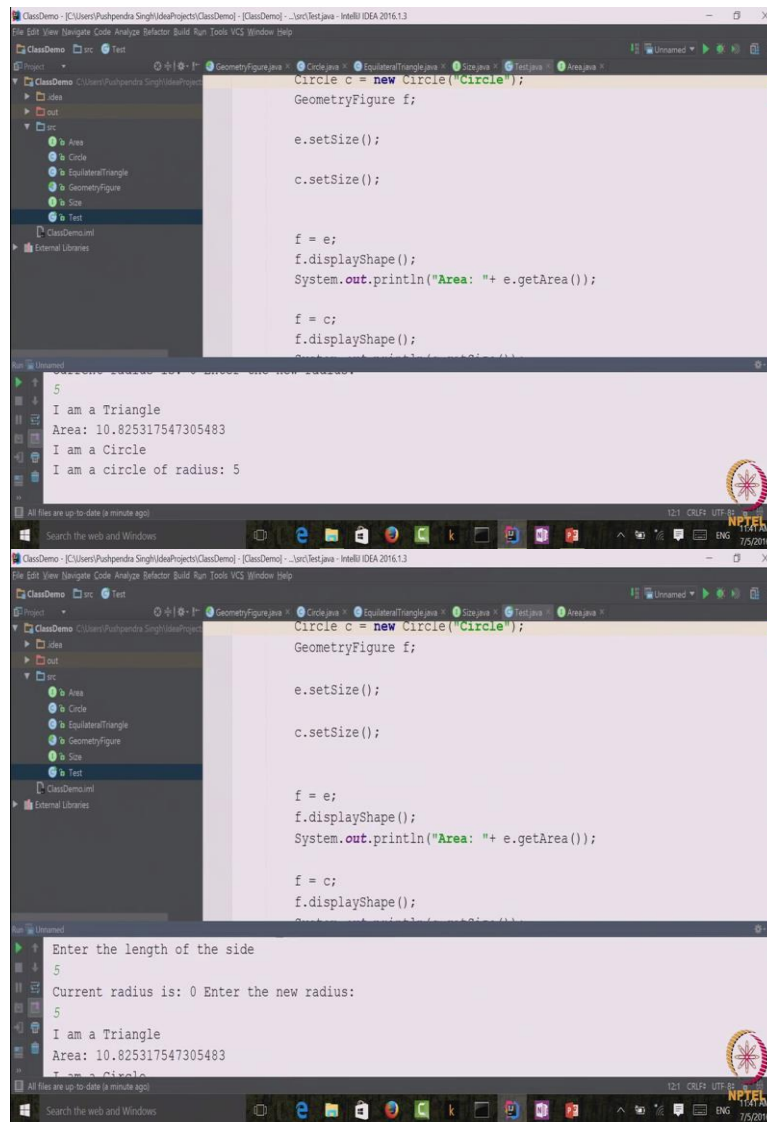
f = c;
f.displayShape();
```

Enter the length of the side

Our program is prompting us for an input because the set size method requires an integer input. I will give it an input let's say five. Then the set size method for the circle also requires an input. I will again enter five. Now let's see what happened.

(Refer Slide Time: 11:01)





First you saw that the correct set size methods were called for the variable of equilateral triangle and of the circle. And then F dot display shape was called F was a variable of the F set class geometry figure but at the time of this method called F was referring to E which means that the display shape of equilateral triangle should be called.

(Refer Slide Time: 11:38)

The image displays three sequential screenshots of an IDE (IntelliJ IDEA 2016.13) showing the development of a Java class named `EquilateralTriangle`. The class extends `GeometryFigure` and implements the `Area` interface.

Top Screenshot: Shows the initial class structure. The code includes a private integer `side` initialized to 0, a constructor `EquilateralTriangle(String shape)` that calls `super(shape)`, and an overridden `setSize()` method that prompts the user to enter the length of the side and updates `side`.

```
public class EquilateralTriangle extends GeometryFigure implements Area {
    private int side=0;

    public EquilateralTriangle(String shape) {
        super(shape);
    }

    @Override
    protected void setSize() {
        System.out.println("Enter the length of the side");
        Scanner in = new Scanner(System.in);
        side = in.nextInt();
    }
}
```

Middle Screenshot: Shows the addition of `getArea()` and `setArea()` methods. The `getArea()` method returns the area calculated as $\frac{\sqrt{3}}{4} \times \text{side}^2$.

```
@Override
public double getArea() { return (Math.sqrt(3)/4)*Math.pow(side, 2); }

@Override
public void setArea(int dimension) {
}
```

Bottom Screenshot: Shows the final code with the `import java.util.Scanner;` statement and a comment: `Created by Pushpendra Singh on 7/2/2016.`

```
import java.util.Scanner;

/**
 * Created by Pushpendra Singh on 7/2/2016.
 */
public class EquilateralTriangle extends GeometryFigure implements Area {
    private int side=0;

    public EquilateralTriangle(String shape) {
        super(shape);
    }

    @Override
    protected void setSize() {

```

The Run console in all screenshots shows the following output:

```
Enter the length of the side
5
Current radius is: 0 Enter the new radius:
5
I am a Triangle
Area: 10.825317547305483
I am a Circle
```

```
ClassDemo - [C:\Users\Pushpendra Singh\IdeaProjects\ClassDemo] - [ClassDemo] - ...src\GeometryFigure.java - IntelliJ IDEA 2016.1.3
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help

ClassDemo src GeometryFigure
Project
  ClassDemo
    src
      out
    src
      Area
      Circle
      EquilateralTriangle
      GeometryFigure
      Size
      Test
      ClassDemo.iml
  External Libraries

public GeometryFigure(String shape) { this.shape = shape; }

protected abstract void setSize();

public void displayShape() { System.out.println("I am a "+shape); }

}
```

```
Run Unnamed
Enter the length of the side
5
Current radius is: 0 Enter the new radius:
5
I am a Triangle
Area: 10.825317547305483
I am a Circle
All files are up-to-date (2 minutes ago)
```

```
ClassDemo - [C:\Users\Pushpendra Singh\IdeaProjects\ClassDemo] - [ClassDemo] - ...src\GeometryFigure.java - IntelliJ IDEA 2016.1.3
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help

ClassDemo src GeometryFigure
Project
  ClassDemo
    src
      out
    src
      Area
      Circle
      EquilateralTriangle
      GeometryFigure
      Size
      Test
      ClassDemo.iml
  External Libraries

/* Created by Pushpendra Singh on 7/2/2016. */
public abstract class GeometryFigure {
    protected String shape = "Not Defined";

    public GeometryFigure(){
        System.out.println("Default Constructor");
    }

    public GeometryFigure(String shape) { this.shape = shape; }

    protected abstract void setSize();
}
```

```
Run Unnamed
Enter the length of the side
5
Current radius is: 0 Enter the new radius:
5
I am a Triangle
Area: 10.825317547305483
I am a Circle
All files are up-to-date (2 minutes ago)
```

```
ClassDemo - [C:\Users\Pushpendra Singh\IdeaProjects\ClassDemo] - [ClassDemo] - ...src\EquilateralTriangle.java - IntelliJ IDEA 2016.1.3
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help

ClassDemo src EquilateralTriangle
Project
  ClassDemo
    src
      out
    src
      Area
      Circle
      EquilateralTriangle
      GeometryFigure
      Size
      Test
      ClassDemo.iml
  External Libraries

/* Created by Pushpendra Singh on 7/2/2016. */
public class EquilateralTriangle extends GeometryFigure implements Area {
    private int side=0;

    public EquilateralTriangle(String shape){
        super(shape);
    }

    @Override
    protected void setSize() {
        System.out.println("Enter the length of the side");
        Scanner in = new Scanner(System.in);
        side = in.nextInt();
    }
}
```

```
Run Unnamed
Enter the length of the side
5
Current radius is: 0 Enter the new radius:
5
I am a Triangle
Area: 10.825317547305483
I am a Circle
All files are up-to-date (2 minutes ago)
```

The image displays three sequential screenshots of an IDE (IntelliJ IDEA 2016.1.3) showing the development and execution of a Java application for calculating the area and size of geometric shapes. The application uses a design pattern where a base class (GeometryFigure) is abstracted into a concrete class (EquilateralTriangle) and a separate class (Circle) is also implemented. The Test class contains the main logic for user interaction and calculations.

Code Snippets:

```
Circle c = new Circle("Circle");
GeometryFigure f;

e.setSize();
c.setSize();

f = e;
f.displayShape();
System.out.println("Area: "+ e.getArea());

f = c;
f.displayShape();

public class Test {

    public static void main(String[] args){
        EquilateralTriangle e = new EquilateralTriangle("Triangle");
        Circle c = new Circle("Circle");
        GeometryFigure f;

        e.setSize();

        c.setSize();

        f = e;
        f.displayShape();
        System.out.println("Area: "+ e.getArea());

        f = c;
        f.displayShape();
        System.out.println(c.getSize());
    }
}
```

Console Output:

```
Enter the length of the side
5
Current radius is: 0 Enter the new radius:
5
I am a Triangle
Area: 10.825317547305483
I am a Circle
```

The screenshots show the code being edited in the IDE, the console output during execution, and the system tray at the bottom of the window. The system tray includes the Windows taskbar, a search bar, and the NPTEL logo.

```
ClassDemo - [C:\Users\Pushpendra Singh\IdeaProjects\ClassDemo] - [ClassDemo] - IntelliJ IDEA 2016.1.3
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help

ClassDemo [src] Test
Project
  ClassDemo [C:\Users\Pushpendra Singh\IdeaProjects\ClassDemo]
    idea
    out
    src
      Area
      Circle
      EquilateralTriangle
      GeometryFigure
      Size
      Test
    ClassDemo.iml
    External Libraries

c.setSize();

f = e;
f.displayShape();
System.out.println("Area: " + e.getArea());

f = c;
f.displayShape();
System.out.println(c.getSize());
}
```

```
Run | Unnamed
Current radius is: 0. Enter the new radius:
5
I am a Triangle
Area: 10.825317547305483
I am a Circle
I am a circle of radius: 5
```



```
ClassDemo - [C:\Users\Pushpendra Singh\IdeaProjects\ClassDemo] - [ClassDemo] - IntelliJ IDEA 2016.1.3
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help

ClassDemo [src] EquilateralTriangle
Project
  ClassDemo [C:\Users\Pushpendra Singh\IdeaProjects\ClassDemo]
    idea
    out
    src
      Area
      Circle
      EquilateralTriangle
      GeometryFigure
      Size
      Test
    ClassDemo.iml
    External Libraries

Created by Pushpendra Singh on 7/2/2016.

public class EquilateralTriangle extends GeometryFigure implements Area {
    private int side=0;

    public EquilateralTriangle(String shape){
        super(shape);
    }

    @Override
    protected void setSize() {
        System.out.println("Enter the length of the side");
        Scanner in = new Scanner(System.in);
        side = in.nextInt();
    }
}
```

```
Run | Unnamed
Current radius is: 0. Enter the new radius:
5
I am a Triangle
Area: 10.825317547305483
I am a Circle
I am a circle of radius: 5
```



```
ClassDemo - [C:\Users\Pushpendra Singh\IdeaProjects\ClassDemo] - [ClassDemo] - IntelliJ IDEA 2016.1.3
File Edit View Navigate Code Analyze Refactor Build Run Tools VCS Window Help

ClassDemo [src] EquilateralTriangle
Project
  ClassDemo [C:\Users\Pushpendra Singh\IdeaProjects\ClassDemo]
    idea
    out
    src
      Area
      Circle
      EquilateralTriangle
      GeometryFigure
      Size
      Test
    ClassDemo.iml
    External Libraries

Scanner in = new Scanner(System.in);
side = in.nextInt();

@Override
public double getArea() { return (Math.sqrt(3)/4)*Math.pow(side, 2); }

@Override
public void setArea(int dimension) {
}
}
```

```
Run | Unnamed
Current radius is: 0. Enter the new radius:
5
I am a Triangle
Area: 10.825317547305483
I am a Circle
I am a circle of radius: 5
```



Let's see what does the display shape method is we have defined the display shape method in the geometry figure and all display shape method does is print out the value of the shape variable.

Now the shape variable in the abstract geometry figure is defined as not defined. However when we create an object of an equilateral triangle class we use the constructor which was taking the value of shape as in our program. We gave the value of shape to the triangle. Now when the display shape method is called it prints I am and the value of the shape variable because it was referring to the equilateral triangle and where the value of the shape variable was triangle it printed I am a triangle.

Then we ran the method called the get area which was a method of the interface implemented by equilateral triangle class. It calculated the area of the triangle given the length of the side and printed it. Similarly we did the same thing for the circle we first assign the variable of geometry figure to the object of the circle after that we printed display shape and correctly. Now the print out says I am a circle. After that we print out the value of the get size which prints nothing but the radius that was given.

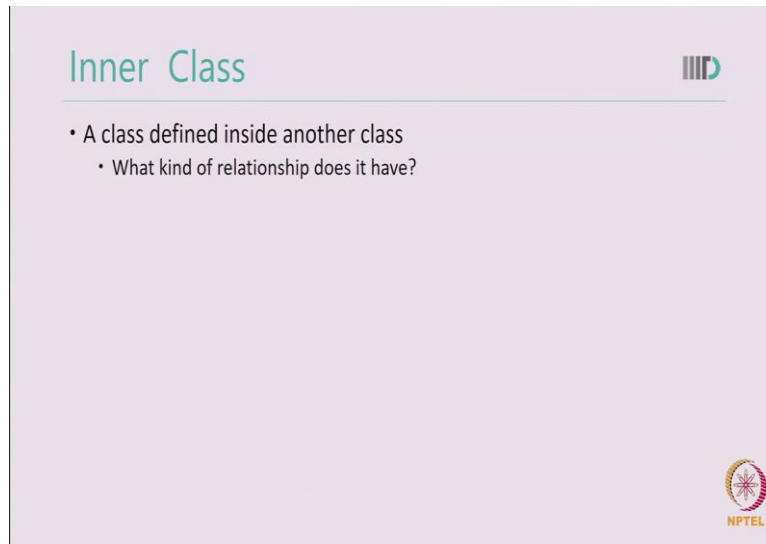
As you will see that for the get area size we need separate implementation because there are different methods to calculate area of a triangle and area of a circle or if declare other sub classes such as rectangle or a square or a polygon the method get area will be different from each one of them. That's the advantages of using interface or abstract classes. We only need to remember the method name called get area which (they) which we can then called for different object and get the correct output.

So in this program we have seen concepts of abstract classes sub classes, interfaces, polymorphism and inheritance. I advised you to write a similar program may be using your college as an example or you may use any (())(14:16) example. For example you may define an abstract class called animal. You can then extend it to let us say dog and cat then you can say you can implement different methods (wir) which are specific to different animal category.

Try to use all the concepts of inheritance and polymorphism in your program that you are clear that when you are using polymorphism or inheritance what would be the expected program output look like. We have already covered basic concepts about java classes, object

abstract classes and interfaces. Today we will look at some other concepts called inner classes and anonymous inner classes.

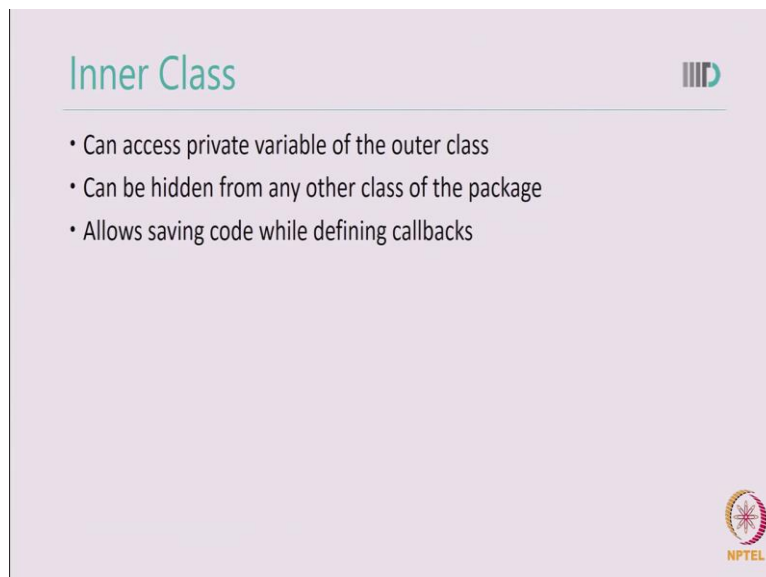
(Refer Slide Time: 14:59)



The slide is titled "Inner Class" in a teal font at the top left. In the top right corner, there is a logo consisting of three vertical bars of increasing height. Below the title, there is a horizontal line. Underneath the line, there are two bullet points: "• A class defined inside another class" and "• What kind of relationship does it have?". In the bottom right corner, there is a circular logo with a star-like pattern and the text "NPTEL" below it.

The best way to learn this concept is to write the program that displays these concepts in action. Let's see what an inner class is. In java an inner class is a class defined inside another class.

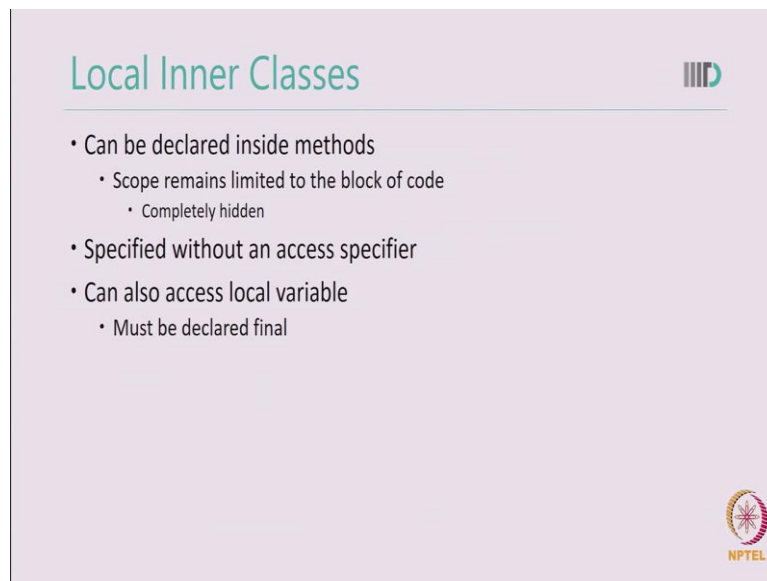
(Refer Slide Time: 15:13)



The slide is titled "Inner Class" in a teal font at the top left. In the top right corner, there is a logo consisting of three vertical bars of increasing height. Below the title, there is a horizontal line. Underneath the line, there are three bullet points: "• Can access private variable of the outer class", "• Can be hidden from any other class of the package", and "• Allows saving code while defining callbacks". In the bottom right corner, there is a circular logo with a star-like pattern and the text "NPTEL" below it.

An inner class can access private variable of the outer class. And it can be hidden from any other class of the package. An inner class is usually used to save the amount of code that you need to write while you are defining call back methods.

(Refer Slide Time: 15:33)

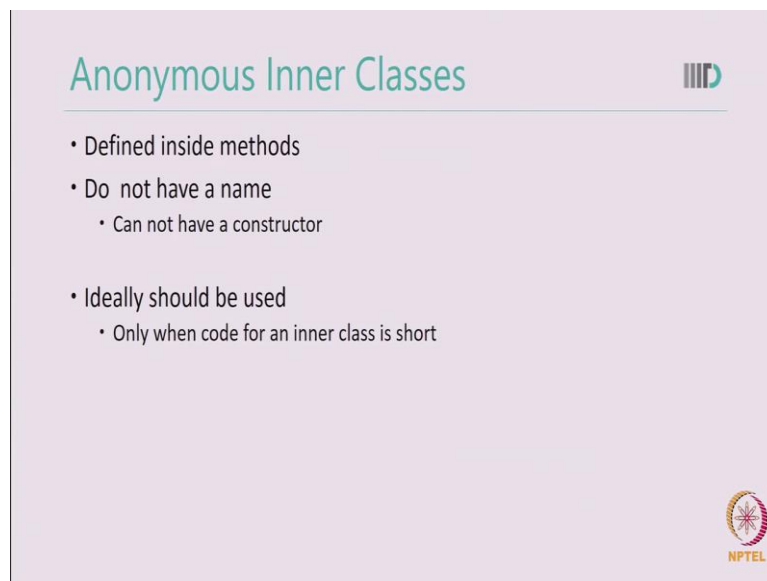


The slide is titled "Local Inner Classes" in a teal font. It features a list of four bullet points: "Can be declared inside methods" (with sub-bullets "Scope remains limited to the block of code" and "Completely hidden"), "Specified without an access specifier", "Can also access local variable" (with sub-bullet "Must be declared final"), and "Can be declared inside methods". The NPTEL logo is in the bottom right corner.

- Can be declared inside methods
 - Scope remains limited to the block of code
 - Completely hidden
- Specified without an access specifier
- Can also access local variable
 - Must be declared final

A local inner class can be declared inside methods and the scope of that inner class remains limited to that block of code which is completely hidden from outside the block of code. Inner class can be specified without an access specifier and it can also access local variables.

(Refer Slide Time: 15:58)



The slide is titled "Anonymous Inner Classes" in a teal font. It features a list of three bullet points: "Defined inside methods", "Do not have a name" (with sub-bullet "Can not have a constructor"), and "Ideally should be used" (with sub-bullet "Only when code for an inner class is short"). The NPTEL logo is in the bottom right corner.


- Defined inside methods
- Do not have a name
 - Can not have a constructor
- Ideally should be used
 - Only when code for an inner class is short

An anonymous inner class similar to inner class except that it does not even have a name or constructor for that reason it should only be used when the code is very short in size. In android programming we will be extensively be using anonymous inner classes.

(Refer Slide Time: 16:21)

Static Inner Classes

- Used when you want to have an inner class which does not require a reference of the outer class i.e. no need to use an object of the outer class
- E.g.
 - Compute minimum and maximum number in an array




Static inner class is used when you want to have an inner class which does not require a reference of the outer class that is there is no need to use an object of the outer class. For example you may want to compute minimum and maximum number in an array.

(Refer Slide Time: 16:40)

Object Cloning

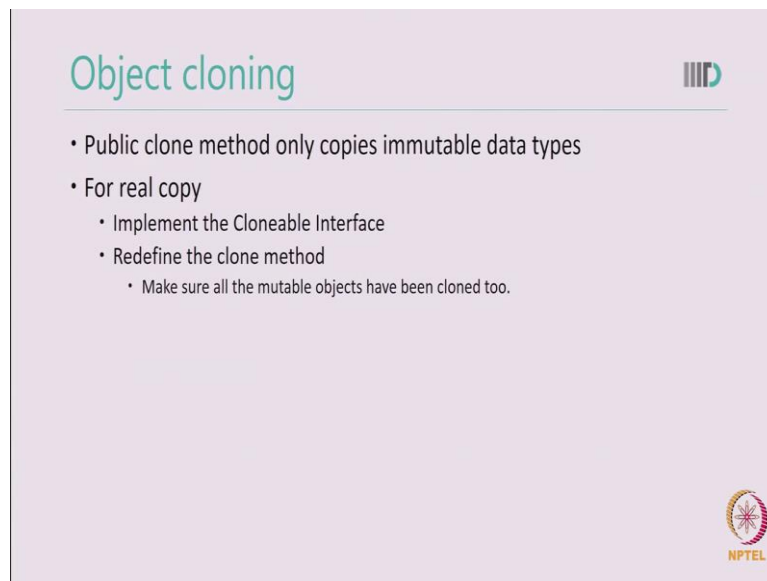
```
Student s = new Student()  
Student s1 = s;
```

- Only references are being copied
- To make a real copy
`s1 = s.clone();`



Another concept in java is of object cloning. Because you create object of a class sometimes you may want to copy that object information. However you may use the single equal to sign only reference is copied. Which means that now the two variables referred to the same object and any change in the object will be reflected in the two variables. If you really want to make a copy then you should use the method called clone

(Refer Slide Time: 17:15)

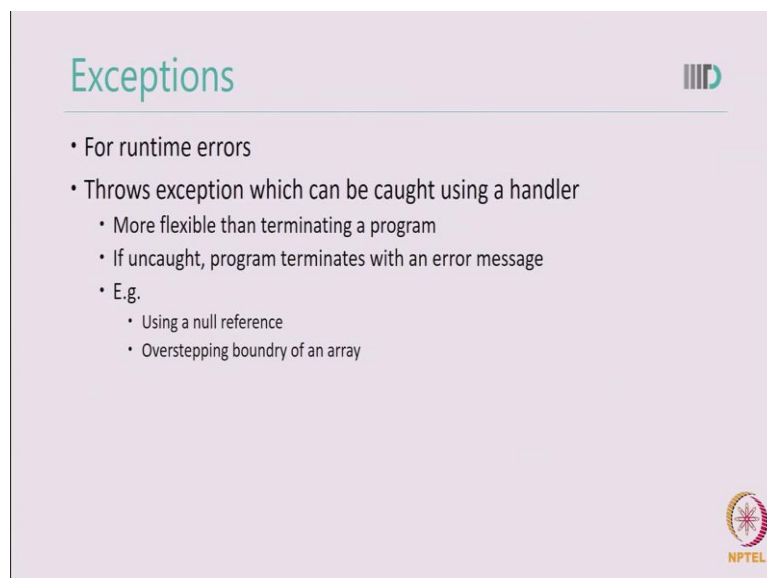


The slide is titled "Object cloning" in a teal font. It features a list of bullet points: "Public clone method only copies immutable data types", "For real copy" (with sub-bullets "Implement the Cloneable Interface" and "Redefine the clone method" which includes "Make sure all the mutable objects have been cloned too."), and the NPTEL logo in the bottom right corner.

- Public clone method only copies immutable data types
- For real copy
 - Implement the Cloneable Interface
 - Redefine the clone method
 - Make sure all the mutable objects have been cloned too.

The public clone method that is available only copies the immutable data types. If you want the real copy of an object you must implement the clone able interface as provided by the java and then you should define the clone method yourself. This is the way to clone all the mutable objects.

(Refer Slide Time: 17:42)



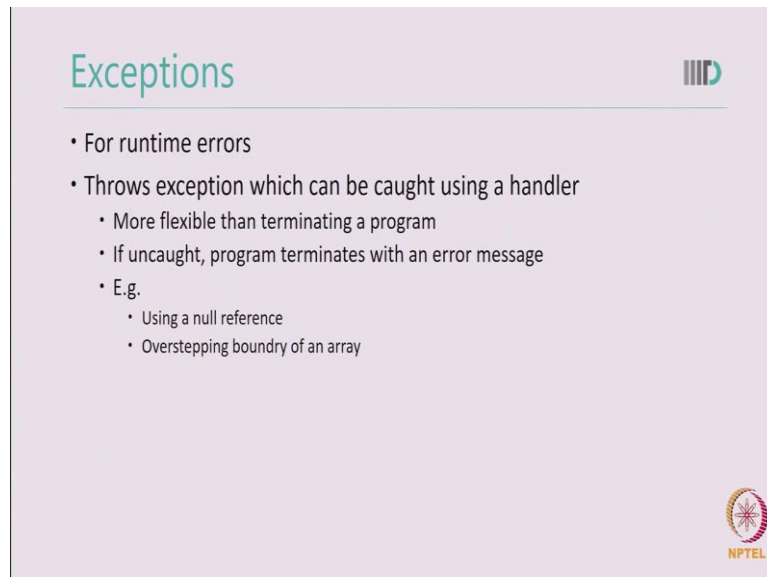
The slide is titled "Exceptions" in a teal font. It features a list of bullet points: "For runtime errors", "Throws exception which can be caught using a handler" (with sub-bullets "More flexible than terminating a program", "If uncaught, program terminates with an error message", and "E.g." which includes "Using a null reference" and "Overstepping boundry of an array"), and the NPTEL logo in the bottom right corner.

- For runtime errors
- Throws exception which can be caught using a handler
 - More flexible than terminating a program
 - If uncaught, program terminates with an error message
 - E.g.
 - Using a null reference
 - Overstepping boundry of an array

Another important concept in java is of exceptions. An exception is a way to handle the errors in java program and to define behaviour of a java program when errors do occurs in your program. For example in C plus plus or in C if you access an element outside the boundary of an array you receive an unexpected behaviour.

For example you may have a segmentation fault which will clear your program or you may receive a random garbage value. You never know what kind of behaviour you will get. In java you may throw an exception whenever the program attempts to access any value which is outside the boundary of a given class.

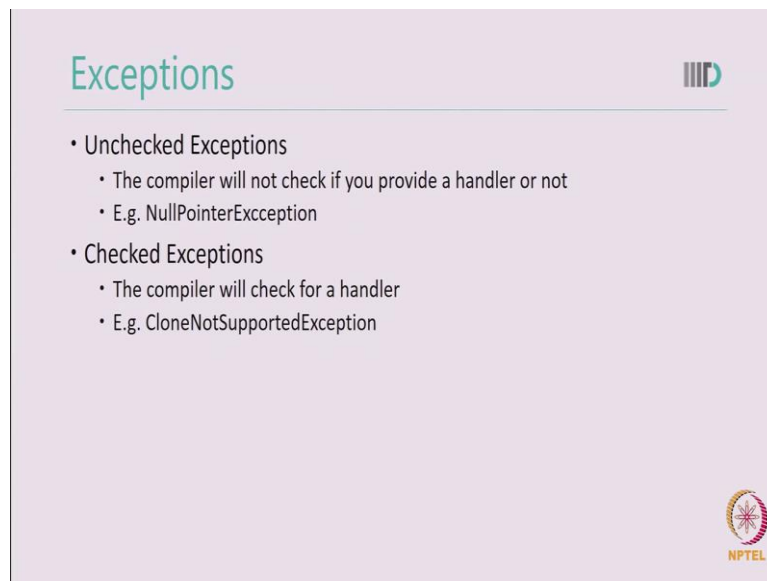
(Refer Slide Time: 18:34)



The slide is titled "Exceptions" in a teal font at the top left. In the top right corner, there is a logo consisting of three vertical bars of varying heights. Below the title, there is a list of bullet points: "• For runtime errors", "• Throws exception which can be caught using a handler", "• More flexible than terminating a program", "• If uncaught, program terminates with an error message", "• E.g.", "• Using a null reference", and "• Overstepping boundary of an array". In the bottom right corner, there is a circular logo with a star-like pattern and the text "NPTEL" below it.

By defining expectation you can limit the behaviour of the java program. Exceptions are very useful for run time errors. In order to use exception you throw an exception which is then caught by handler. Exceptions provide you more flexibility than terminating a program. Using an exception you can keep your program running while handling an error. However if you do not catch a exception then your program may terminate. Such as for the example given of overstepping the boundary of an array.

(Refer Slide Time: 19:11)



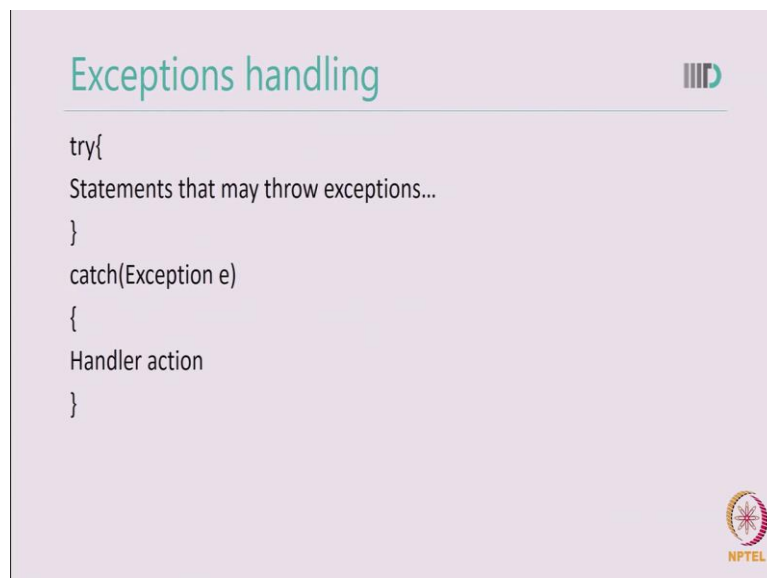
The slide is titled "Exceptions" in a teal font at the top left. In the top right corner, there is a logo consisting of three vertical bars of increasing height. The main content is a bulleted list:

- Unchecked Exceptions
 - The compiler will not check if you provide a handler or not
 - E.g. NullPointerException
- Checked Exceptions
 - The compiler will check for a handler
 - E.g. CloneNotSupportedException

In the bottom right corner, there is a circular logo with a star-like pattern and the text "NPTEL" below it.

Java does provide you exceptions which are checked and unchecked for unchecked exception the compiler will not check if you have provided a handler or not. And it is responsibility of you as a programmer. For checked exception the compiler will check for a handler and if you have not providing that handler the compiler will give a warning and an error where you try to compile the program.

(Refer Slide Time: 19:42)



The slide is titled "Exceptions handling" in a teal font at the top left. In the top right corner, there is a logo consisting of three vertical bars of increasing height. The main content is a code block showing the structure of a try-catch statement:

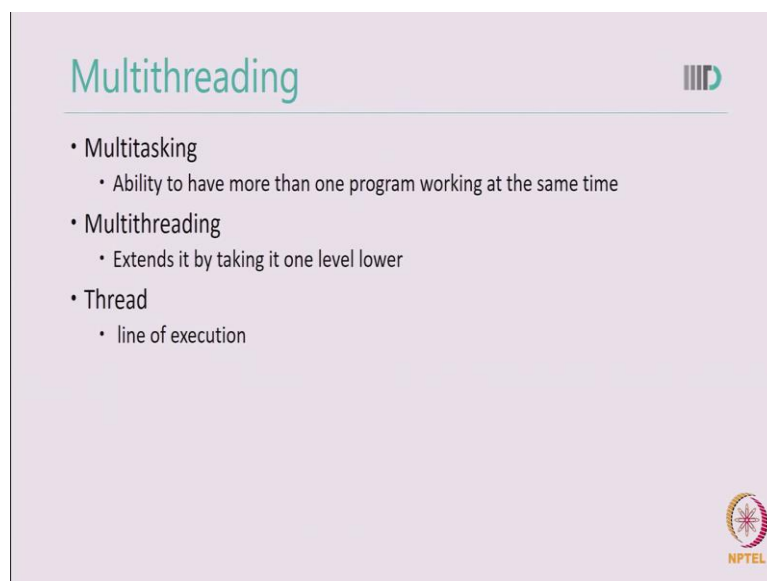
```
try{
Statements that may throw exceptions...
}
catch(Exception e)
{
Handler action
}
```

In the bottom right corner, there is a circular logo with a star-like pattern and the text "NPTEL" below it.

A simple way of declaring exception is using a try and catch block. You start a try block with a curly bracket then you continue all the statement that may throw an exception after that you try to catch those exception in catch block. The catch block defines the action that you will

take if such an exception is worth thrown by your program. So far we saw the exception and the classes now we will look at threads in java.

(Refer Slide Time: 20:13)



As you see that today's computer are very powerful which means they can do multiple task at the same time. The technology that enables is called multitasking or multithreading. Multithreading allows you to run parallel lines of execution in your program as you may have studied in your operating system codes that are thread is line of execution and a program may have multiple threads which signifies multiple parallel lines of execution. So why do we use threads?

(Refer Slide Time: 20:52)



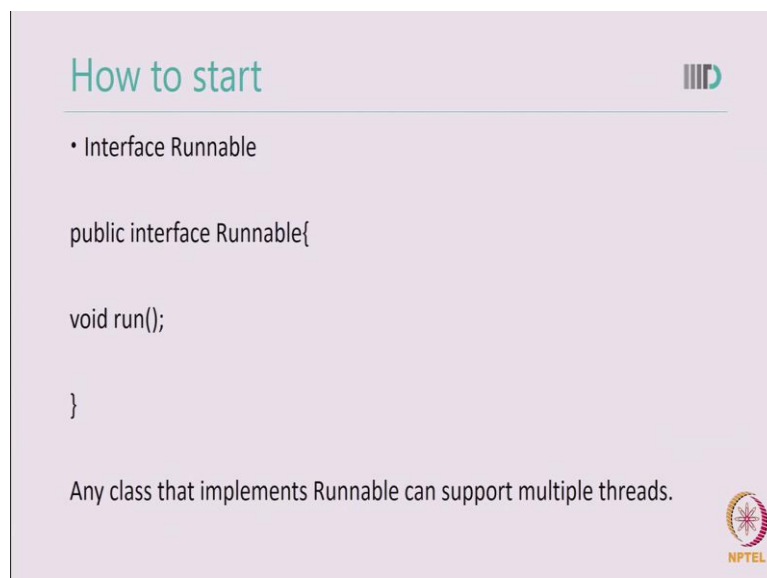
Why threads

- To support concurrent execution paths in your program e.g.
 - GUI: to support many GUI components at the same time
 - Network: receive data in one thread and process it in another thread
 - ...

NPTEL

Well we use threads to support concurrent execution paths in our program. For example if you are running a network program. You may assign a thread to receive the data while another thread is displaying some information on the display. Similarly for GUI applications you may assign different threads to different GUI components.

(Refer Slide Time: 21:15)



How to start

- Interface Runnable

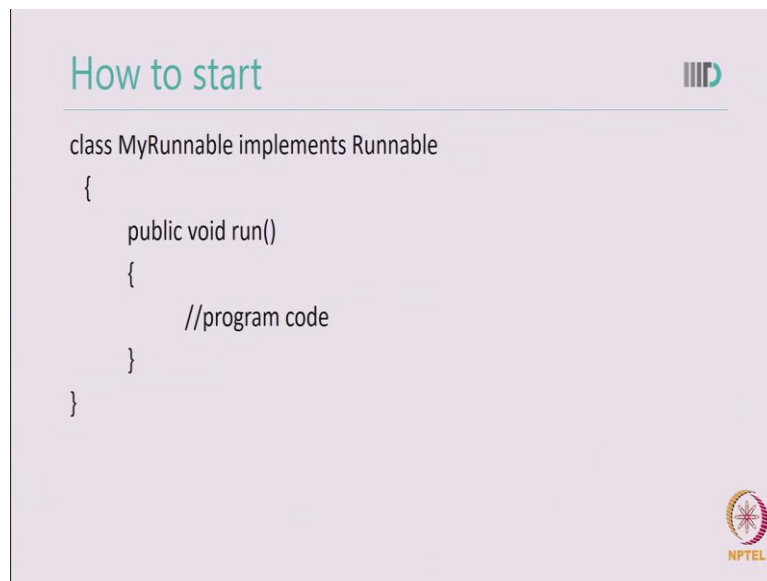
```
public interface Runnable{  
  
    void run();  
  
}
```

Any class that implements Runnable can support multiple threads.

NPTEL

In java you can extend an interface called runnable to implement the thread functionality in your program. The interface runnable has a method called run which you need to implement for your class. A class that implements runnable can support multiple threads.

(Refer Slide Time: 21:36)



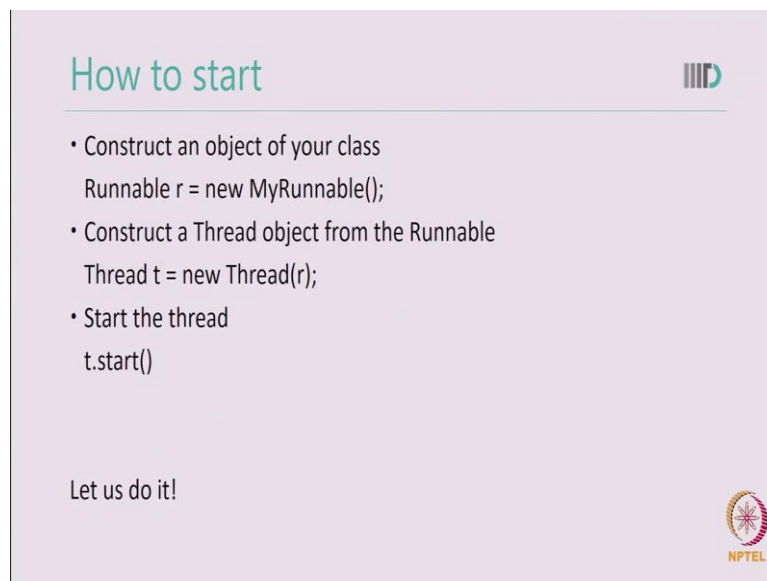
How to start

```
class MyRunnable implements Runnable
{
    public void run()
    {
        //program code
    }
}
```

NPTEL

Here is a simple example. Suppose I have a class called my runnable in that I will have to then define a public word run method and that's it. Now my program can support multiple threads.

(Refer Slide Time: 21:53)



How to start

- Construct an object of your class
`Runnable r = new MyRunnable();`
- Construct a Thread object from the Runnable
`Thread t = new Thread(r);`
- Start the thread
`t.start()`

Let us do it!


NPTEL

So first I can create an object of a runnable type then I create a thread object and then I can start the thread please note that when you use the command thread dot start or t dot start your control will pass to the method called run.

(Refer Slide Time: 22:14)

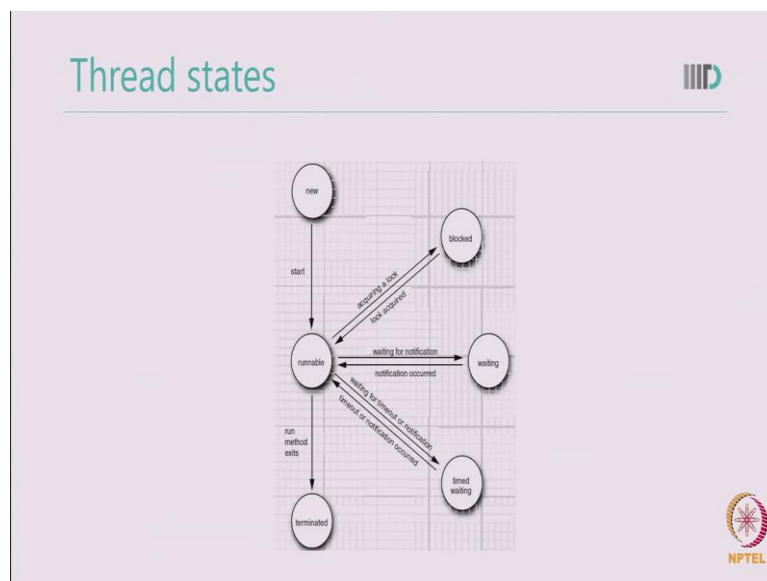
How to start

```
class MyRunnable implements Runnable
{
    public void run()
    {
        //program code
    }
}
```



That you have already defined.

(Refer Slide Time: 22:17)




From your operating system class you may already know that thread may live in different states. A thread may be created (a) after that it becomes runnable. From runnable it may be blocked, it may be waiting or it may be timed waiting and once the work of the thread is completed it may be terminated.

(Refer Slide Time: 22:47)

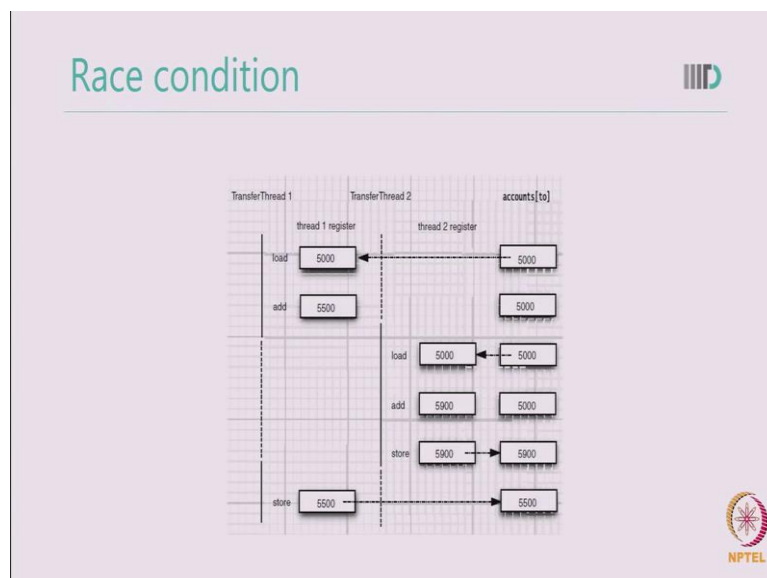
Synchronization

- When two or more threads need to share access to the same data
 - Order in which data is accessed/modified can result in corrupted objects:
Race Condition
 - e.g.
 - Multiple thread modifying a bank account at the same time



Synchronization, it's very important to manage a controlled access to the same data in your program when there are more than one thread is active. You may have studied about race condition in your operating system codes. A race condition occurs when more than one thread try to access the same data and tried to modify it.

(Refer Slide Time: 23:09)

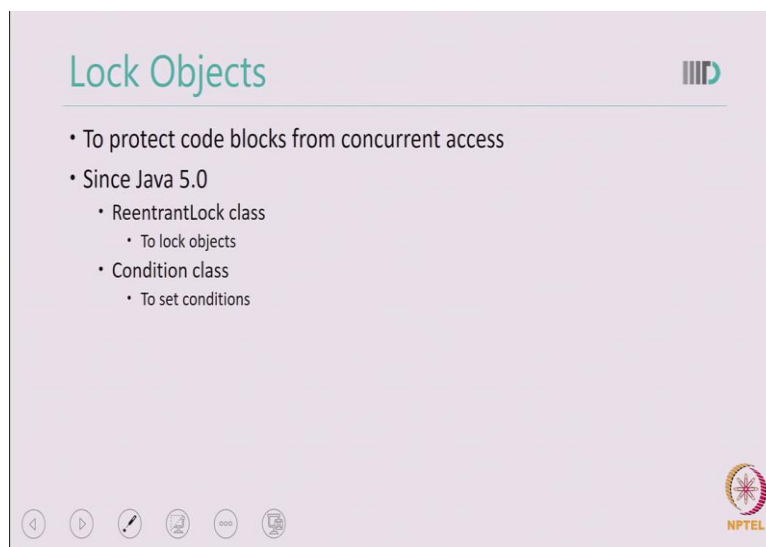


Here is the simple example showing race condition. Suppose there are 2 threads, thread 1 and thread 2 so this is the thread 1 and the thread 2. They copy the same variable which has a value of a 5000 into the register of thread 1 and thread 2 also copies the same values because the thread may have been switched by the operating system scheduler at that point of time.

Now as you see that the variable is copied into the internal register for the threads so when the control comes back to the threads it works on that value. Suppose the first thread add 500 to it. However before it could store the value it is again switched. Similarly for another thread which is copied the value of 5000 it also adds 900 to it.

It is able to store this value again but then the control is switched by the scheduler and then the thread one which had the value 5500 stores the new value. This means that the value modified by thread two is over written by the thread one. This is simple example of race conditions. Race condition may occur in many other program and they may be very complex to detect.

(Refer Slide Time: 24:38)



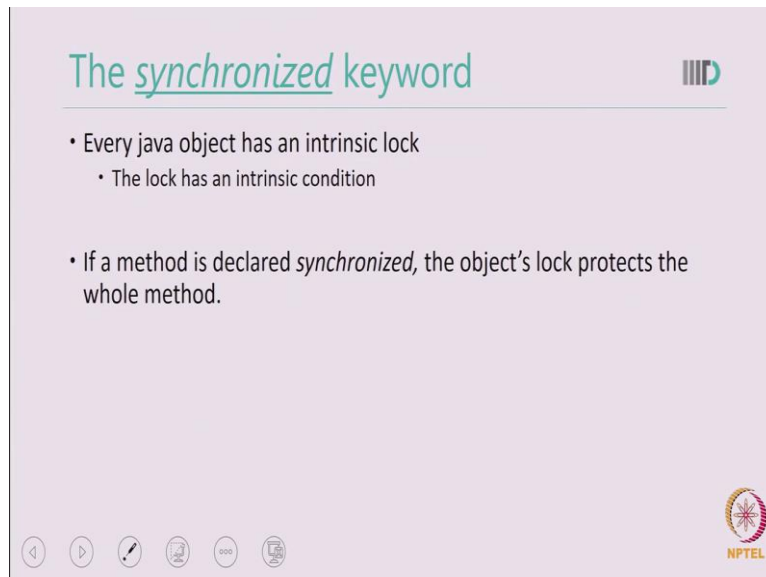
The slide is titled "Lock Objects" and features a list of Java lock classes and their purposes. The list is as follows:

- To protect code blocks from concurrent access
- Since Java 5.0
 - ReentrantLock class
 - To lock objects
 - Condition class
 - To set conditions

The slide also includes a navigation bar at the bottom with icons for back, forward, search, and other controls, and the NPTEL logo in the bottom right corner.

In order to handle race conditions java provides you locks. From java five point zero there are multiple types of locks are available in java. You may have studied about logs, condition, semaphores, view tax and monitors in your operating system codes. So we will not repeat it here. However you are not sure you should try to make a java program that uses these logs, semaphores and monitors to see the fact of each one of them.

(Refer Slide Time: 25:12)



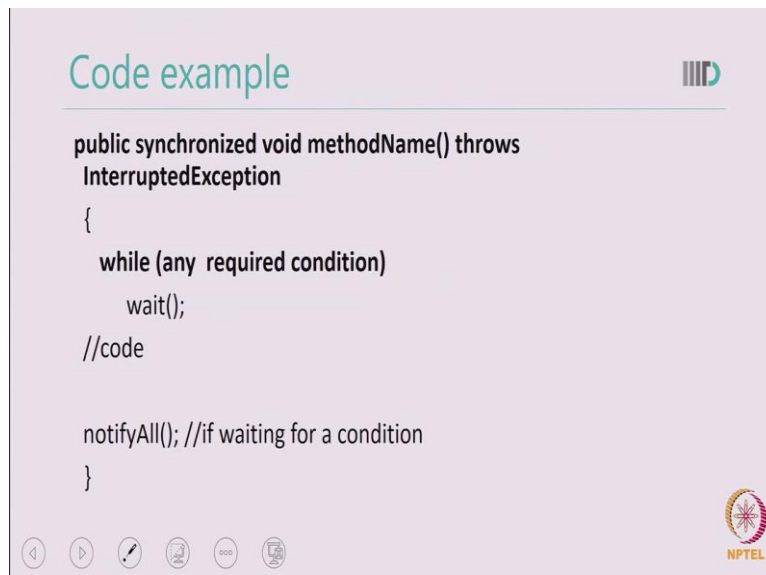
The synchronized keyword

- Every java object has an intrinsic lock
 - The lock has an intrinsic condition
- If a method is declared *synchronized*, the object's lock protects the whole method.

NPTEL

Java also provides a very simple method to make sure that you are access is synchronized. Just by using a synchronized keyword you can ensure that all the access to M to a data objects inside a methods are synchronized across the multiple threads that are trying to access. Synchronized is like a monitor.

(Refer Slide Time: 25:36)



Code example

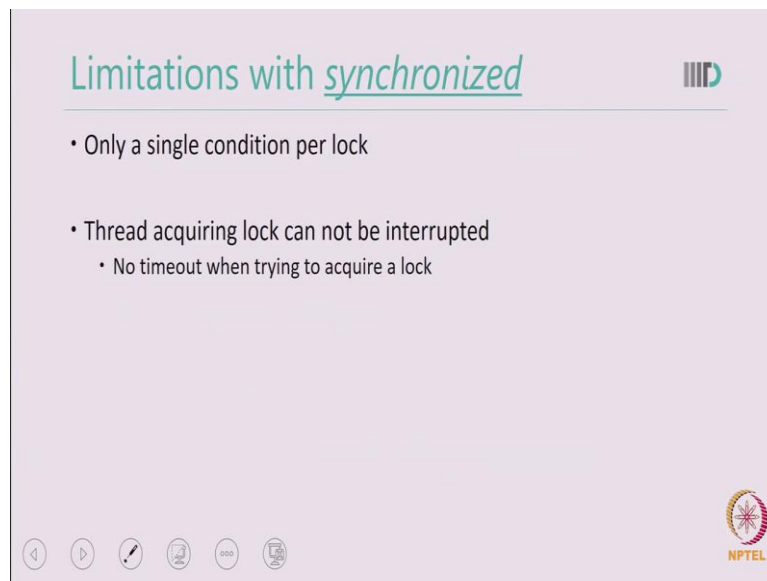
```
public synchronized void methodName() throws
InterruptedException
{
    while (any required condition)
        wait();
    //code

    notifyAll(); //if waiting for a condition
}
```

NPTEL

Here is a simple code example using synchronize. Suppose I want to make sure that in my method called method name all the data accesses are synchronized. Only thing that I need to do is put a synchronized keyword in front of it.

(Refer Slide Time: 25:51)



The slide is titled "Limitations with synchronized" and features a list of three bullet points. The first bullet point is "Only a single condition per lock". The second bullet point is "Thread acquiring lock can not be interrupted", which has a sub-bullet point: "No timeout when trying to acquire a lock". The slide includes a navigation bar at the bottom with icons for back, forward, search, and other controls, and the NPTEL logo in the bottom right corner.

- Only a single condition per lock
- Thread acquiring lock can not be interrupted
 - No timeout when trying to acquire a lock

However there are several limitations with synchronize as you may have already known from your operating system codes. Synchronize act as a monitor which means that it has inherently inefficient compare to a lock. Thank you! This completes our discussion on threads and now we also complete our required background in java for taking the course on the android.

There are many more features of java which are used in android programming however after the series of lectures I expect you to complete that on your own. For these series of lectures I have used the book Core java by Horstmann and Cornell. This is an excellent book which is available at multiple book stores. You may also use any other java book that you may be familiar with. Thank you!