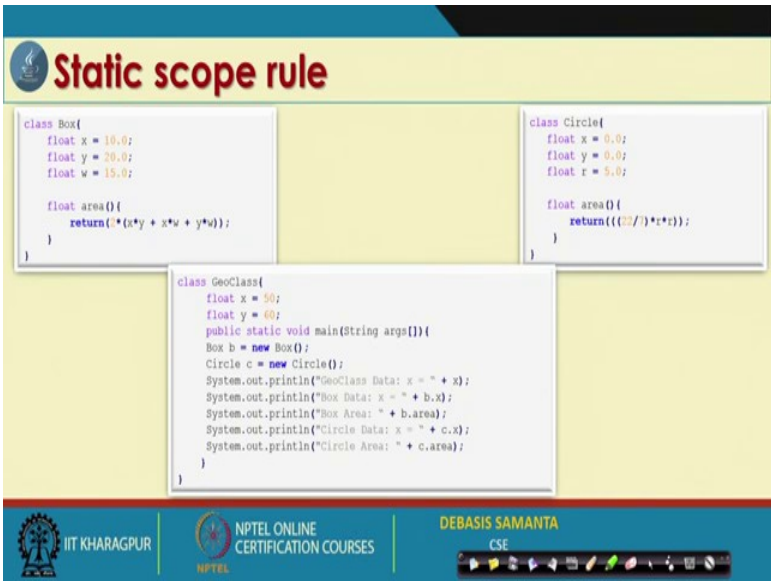
Programming in Java
Prof. Debasis Samanta
Department of Computer Science Engineering
Indian Institute of Technology, Kharagpur

Lecture – 11
Java Scope Rule

Scope rule is a very important concept in any programming language. So, Java is no exception for this. There are mainly 2 scope rules static and dynamic. In Java also we follow both the scope rules static and dynamic. In today's lecture we will discuss about Static Scope Rule.

(Refer Slide Time: 00:41)



Static scope rule

```java
class Box{
    float x = 10.0;
    float y = 20.0;
    float w = 15.0;

    float area(){
        return(2*(x*y + x*w + y*w));
    }
}
```

```java
class Circle{
   float x = 0.0;
   float y = 0.0;
   float r = 5.0;

   float area(){
       return(((22/7)*r*r));
   }
}
```

```java
class GeoClass{
     float x = 50;
     float y = 60;
     public static void main(String args[]){
     Box b = new Box();
     Circle c = new Circle();
     System.out.println("GeoClass Data: x = " + x);
     System.out.println("Box Data: x = " + b.x);
     System.out.println("Box Area: " + b.area);
     System.out.println("Circle Data: x = " + c.x);
     System.out.println("Circle Area: " + c.area);
    }
}
```

Now, let us see a small program here is a class declaration called the class Box. So, this class is you already know that it has the data as well as the method in this case area is a method x y w is another class is a member of this class.

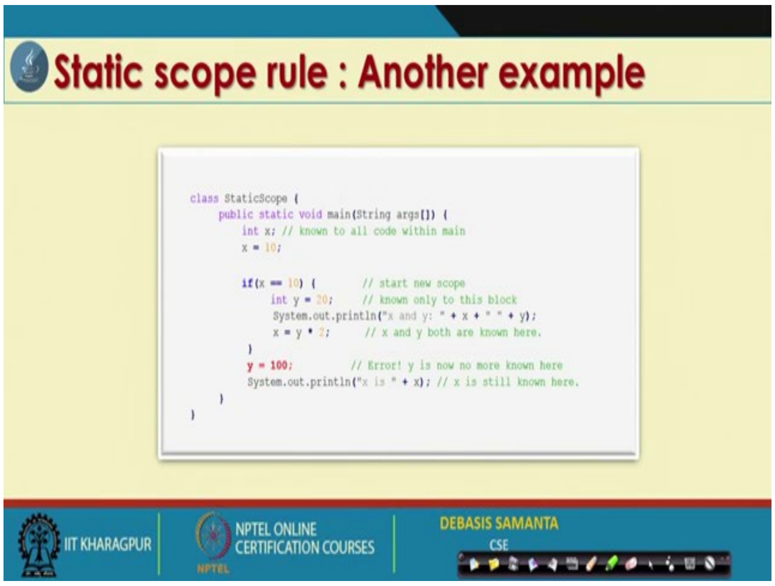
Now, let us consider another class, here is the Circle. The Circle has data like x y r and then another method is called the area. Now if we consider the 2 classes; namely Box and Circle then we can see that x y both member data members are declared as float in both the classes, but the thing is that x y which is defined in Box it has scope within the Box class. Similarly, the x y which is defined in the class Circle it has the scope only within this class Circle.

Now if we write a main program, let us name this main program as GeoClass and we declare here the 2 members, the x and y which is declared as 50 and 60. So, x and y is basically member which belongs to GeoClass. Again this x y and the x y in class Box or x y in class Circle are totally different.

Now, here again if we see in this method we create 2 objects of type Box and then Circle b and c respectively, and then if you see System.out.println this x this x is basically, x equals to 50. Similarly if we call this one b.x, so b.x is basically this x in float. Similarly b. area of the method b and like this c. x is the x of the class c and c.area is the area of the method c.

Now, so with this scope we usually resolve by specifying the name of the object. So, b.x ,c.x, b.area c.area like, so this way we can resolve the scope and the scope that we can follow it is called static scope, because by saying the program we can tell that which method or which member has the scope of what.

(Refer Slide Time: 03:23)



Static scope rule : Another example

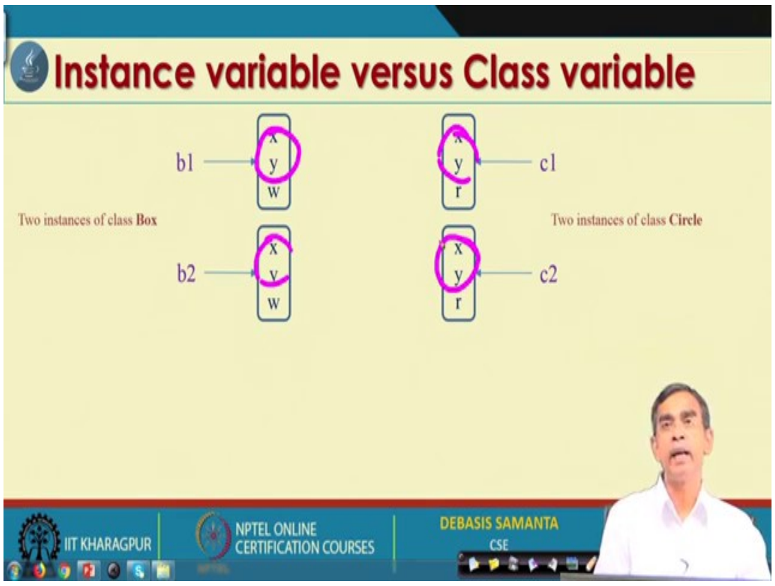
```
class StaticScope {
    public static void main(String args[]) {
        int x; // known to all code within main
        x = 10;

        if(x == 10) {          // start new scope
            int y = 20;        // known only to this block
            System.out.println("x and y: " + x + " " + y);
            x = y * 2;         // x and y both are known here.
        }
        y = 100;             // Error! y is now no more known here
        System.out.println("x is " + x); // x is still known here.
    }
}
```

Now, this is the one example of static scope rule. Another example: now let us see another class that we have declared here. So, static scope the name of the class and in this in this class we declare one variable x as integer and x is declared as 20. Now if we see it is declared within this main method; that means this x has the scope of the entire method.

So, here x is the one class that we have the declared, it is declared as an integer the scope of this is basically the entire method, the main method. On the other hand, we have declared another variable integer y equals to 20 and as it is declared here not here; this means that, this y has the scope of this part. So, this scope the scope of this is this one. So, the 2 variables which we have declared has the 2 different scope, x has the scope of the entire whereas, the entire whereas, the y has the scope of this one.

Now, so this is basically another example of static scope rule and in this case, if we see in this case if we can attempt to access y is equal to 100 is basically report a compilation error. This is because the scope of y is beyond this right. So, y does not have any scope outside this region. So, that is the case of the static scope rule and it is in case of simple variable that we have discussed.

(Refer Slide Time: 05:49)



Instance variable versus Class variable

b1
x
y
w
x
y
r
c1
Two instances of class Box
Two instances of class Circle
b2
x
y
w
x
y
r
c2
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
DEBASIS SAMANTA
CSE

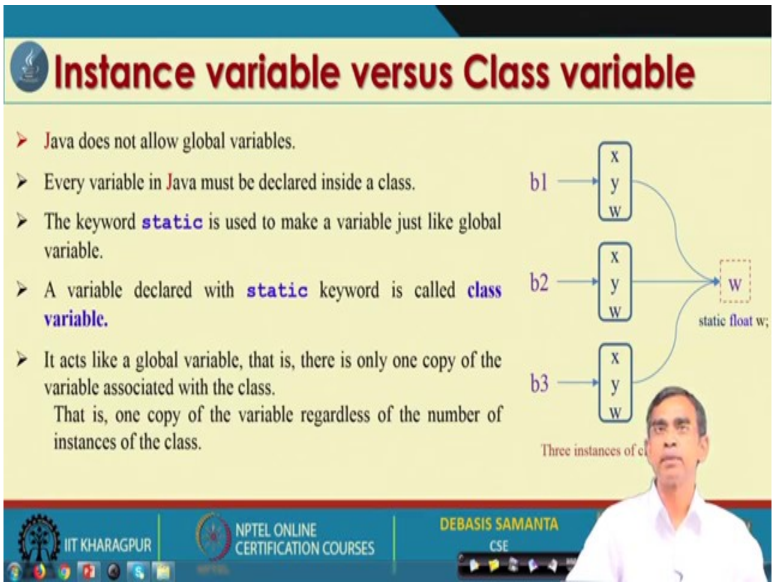
Now, so we have learned about the simple scope rule. Now there are 2 concept in Java related to the scope rule it is called the instance variable and the class variable.

Now, let us first discuss about what is the instance variable. Now let us refer to the concept of class box, we have created 2 objects say b1 and b2. So, b1 and b 2 are the 2 objects and the 2 objects have the 2 variable 3 variables x y w. So, if we create b1, it has instances and this is the one instance corresponding to the object b1 this is the another instance corresponding to the object b2. So, there are 2 instances of the object that we have created. Now let us consider another instances of another object say class 1. c1 is another objects of class Circle, c 2 is another object of class Circle again.

And again here we created the 2 instance of class Circle, where x y r are the 3 instances. So, here this is the one instances and this is the other instances of the right. So, whenever we maintain the memory. So, basically this x is the separate and this x is the separate. Similarly this x they are all separate existence. So, this is the idea about instance variable.

Now if so instance variable we have learn about. So, whenever an object of a particular class is created, we consider instance variable for that objects are created.

(Refer Slide Time: 07:49)



Instance variable versus Class variable

- Java does not allow global variables.
- Every variable in Java must be declared inside a class.
- The keyword **static** is used to make a variable just like global variable.
- A variable declared with **static** keyword is called **class variable.**
- It acts like a global variable, that is, there is only one copy of the variable associated with the class.
  That is, one copy of the variable regardless of the number of instances of the class.

So now, we will discuss the static variable declaration.

We have mention the concept of static, while we are discussing about main method declaration. So, public static void main, so they are the static concept was declared that without ok, if we declare a method as static. So, this method can be called without creating any object. This is the concept that we have learn about, but the concept of static keyword in Java also has its own implications.

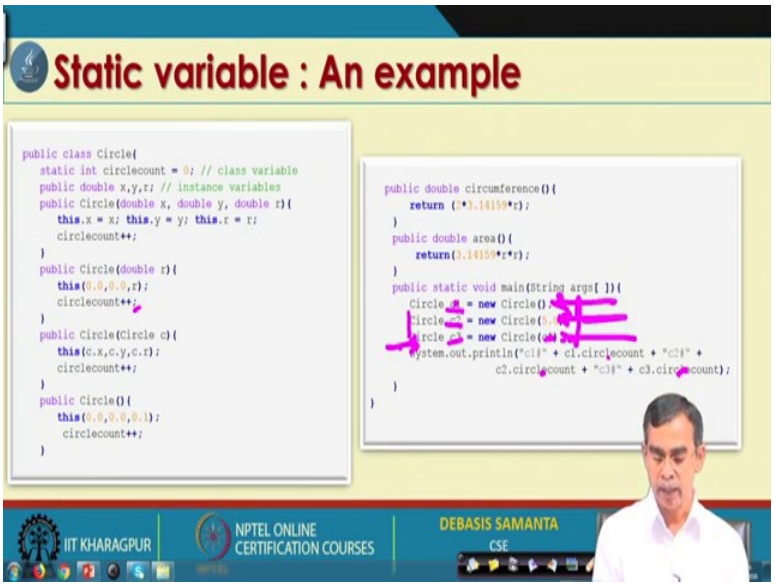
Now we are going to learn about it. So, in Java we know there is no concept of global variable declaration. All the variable; that means data that will be declared will be encapsulated in that class, but sometimes we may require the global variables also. So, Java developer can provide this one by means of static keyword. So, if we declare a variable or a method as a static, then such a variable is called class variable or such a method is called class method.

Now, we are going to discuss about class variable and class method one by one. So, if we declare a variable as a static; that means, if we declare a variable as a class, the difference from this class variable to the instance variable is that instance variable has its own local copy whereas, class variable has the global copy. That means, for all instances of all objects belong to a class, if it is a class variable then it has only one copy whereas, for all variables which are declared as instance variable they have the different copies.

Now, so here is a one example. Here suppose b 1 b 2 b 3 are the 3 objects belong to the class Box and if we declared w as a static, so that declaration like say static keyword before this, so static float w, if we declare w as a static right this way, then w become the class variable.

So, it means that, in this example all x y they are the instance variable, where as this w and the other static variable of class variable; this means that it has for all objects is the only one copy is there. So that means, it shared w is shared among all 3 objects that we have created. So, it is a class variable, that is the static variable then it will have only one instance for all classes; that means, shareable to all classes. So, this is it about 2 concepts that instance variable versus class variable.

(Refer Slide Time: 10:51)



Static variable : An example

```
public class Circle{
   static int circlecount = 0; // class variable
   public double x,y,r; // instance variables
   public Circle(double x, double y, double r){
      this.x = x; this.y = y; this.r = r;
      circlecount++;
   }
   public Circle(double r){
      this(0.0,0.0,r);
      circlecount++;
   }
   public Circle(Circle c){
      this(c.x,c.y,c.r);
      circlecount++;
   }
   public Circle(){
      this(0.0,0.0,0.1);
       circlecount++;
   }
```

```
 public double circumference(){
     return (2*3.14159*r);
  }
  public double area(){
      return(3.14159*r*r);
  }
  public static void main(String args[ ]){
     Circle c1 = new Circle();
     Circle c2 = new Circle(5.0);
     Circle c3 = new Circle(c1);
     System.out.println("c1#" + c1.circlecount + "c2#" +
                       c2.circlecount + "c3#" + c3.circlecount);
  }
}
```

Now, let us discuss an example to clear the concept of static variable more understandable way.

So, here we discussed one class name, of the class is Circle here. So, we declare a 1 class Circle_and we declare one variable here, you can note the variable is a circlecount and which is declared as a static. These variable is declared as a static; that means, circle count is a static variable or class variable in this case. On the other hand, here we declare other 3 variables. Simple as a public double x, y ,r they are basically instance variable.

So, in this in this class example, we declare 3 2 types of variable; class variable and instance variable. Now, let us these are the as usual we have already familiar to this is a constructor, there are many way constructor using this. So, this is the one constructor which is defined explicitly and these are the constructor which basically it defined using this, this means called the superclass constructor this constructor using other parameters. So, those are the concepts that we have already learned about it.

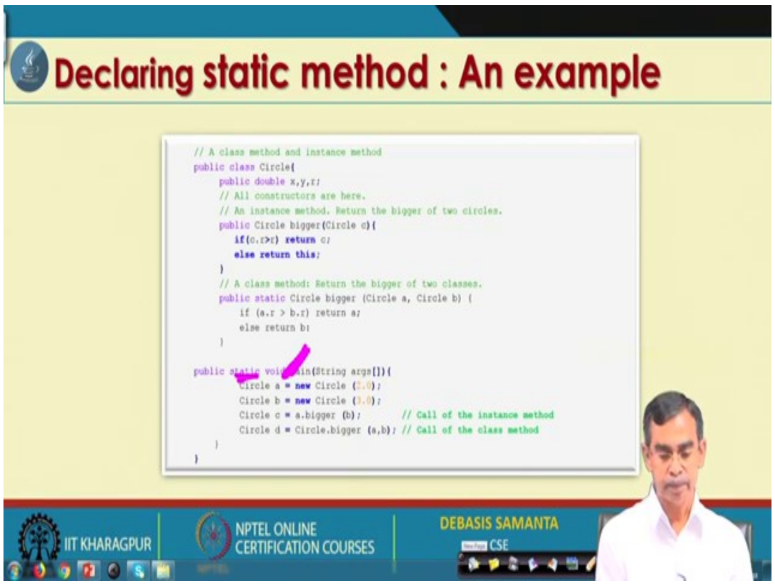
So, this basically compressor, the declaration of a class Circle, so, this is a class circle that can be initialized an objects or create an object with different parameter that can be passed here. Now in addition to this, this method has this class has the 2 more method circumference and area. So, this completes the declaration of the class Circle.

Now let us come to the declaration of this method public static void main string args method. Now, here we declare 3 objects c 1, c 2 and c 3, so 3 objects are created. Now for 3 objects therefore, so for the instance variable x y r are our concern, they have the 3 instances separately whereas, for the circlecount for all the 3 objects c 1 c 2 and c 3, it has only one instance.

So, this means that if we print the c1.circlecount and then c2.circlecount and c3. circlecount, for each circle it will give the value. As it is basically whenever this is increased, so circlecount will be increased, when one object is created. So, initially it is 0, when this is created circlecount is 1. This is created circlecount is 2 and when this is created circle count is 3. So, this c one has its circlecount one c 2 has circlecount 2 and c 3 has circlecount 3 and at the end of these things the value of the circlecount when the all 3 objects are created is basically 3.

So, in this case, if we print c1.circlecount c 2.circle count c 3.circle count it will print 3 3 3 in all cases. However, if we print the circlecount print c1.circlecount here it will print 1, if we print here then it will print 2 and if we print here it will print 3. So, this basically this is because the circle count being a class variable is a global variable. So, if the value is changed by any object, it will be reflected automatically to the global object. So, this is a one good example of static variable.

(Refer Slide Time: 14:51)



Declaring static method : An example

```
// A class method and instance method
public class Circle{
    public double x,y,r;
    // All constructors are here.
    // An instance method. Return the bigger of two circles.
    public Circle bigger(Circle c){
       if(c.r>r) return c;
       else return this;
    }
    // A class method: Return the bigger of two classes.
    public static Circle bigger (Circle a, Circle b) {
        if (a.r > b.r) return a;
        else return b;
    }

public static void main(String args[]){
        Circle a = new Circle (2.0);
        Circle b = new Circle (3.0);
        Circle c = a.bigger (b);        // Call of the instance method
        Circle d = Circle.bigger (a,b); // Call of the class method
    }
}
```

Now, let us consider the discussion of we have learned about static variable likewise a method also can be declared as a static method.

Now, a method can be declared as a static method, if we use is a static keyword again. So, static keyword is used to declare the method. For example, here again come to the Circle class, it has instance variable these are the instance variable and this is a one method the constructor this is a one method called the bigger. This bigger takes an input and argument as a circle and return another circle. So, this is a method and here another method you can see, this method we have declared again as a bigger, but this method is declared as a static.

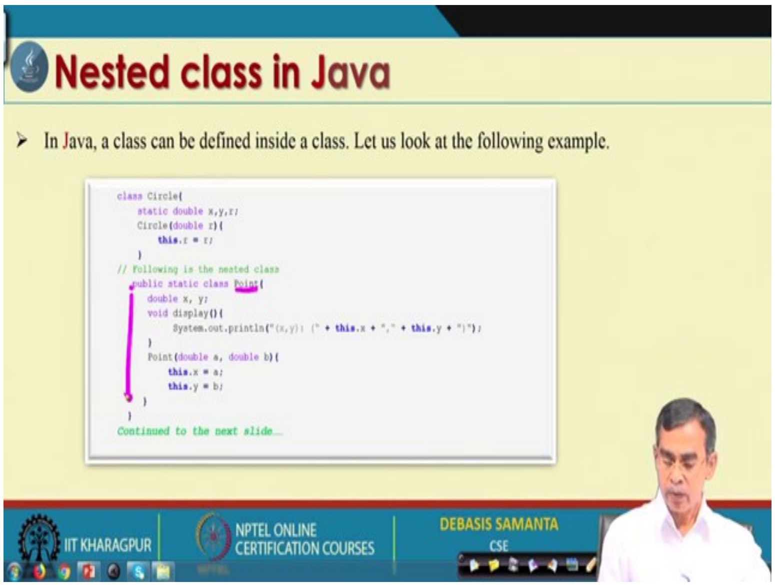
Now, these 2 methods are called overloading, overloading method means this bigger and this bigger although same, but they have the different task and operation, because its argument is 1 circle, it is having 2 it code is different than this one. So, these 2 methods are overloading method.

However, in the second overriding method, overloading method that we have discussed here as a static; this means, that this is the simple instance method. That means, whenever one object is created, we can call this method for that object. However, if we declare a method as a static, then it is called the class method. That mean this kind of method can be accessed without creating an object.

Now, let us see one example in the main method it can clear your idea when I mean how the 2 methods instance method and class method is different. So, here is the one example that we can see here. So, we can create a circle circle and so a b are the 2 circles objects are created and then c basically is another circle object is created, but it will be returned by bigger.

Now, you see this bigger method this bigger is basically the instance method because in that it is accessed by accessing objects. However, in the next example if you see, here this bigger a b is called for the circle class without creating any object. So, this is a idea with the class method. This means that a class method can be invoked without creating any object and that is why our main method is static which belong to this class Circle this is because without creating any object class Circle, we can call this method a main is called without getting any object of that class. So, this is idea about the static method.

(Refer Slide Time: 17:57)



Nested class in Java

- In Java, a class can be defined inside a class. Let us look at the following example.

```
class Circle{
    static double x,y,r;
    Circle(double r){
        this.r = r;
    }
// Following is the nested class
  public static class Point{
     double x, y;
     void display(){
          System.out.println("(x,y): (" + this.x + "," + this.y + ")");
     }
     Point(double a, double b){
         this.x = a;
         this.y = b;
     }
  }
Continued to the next slide…
```

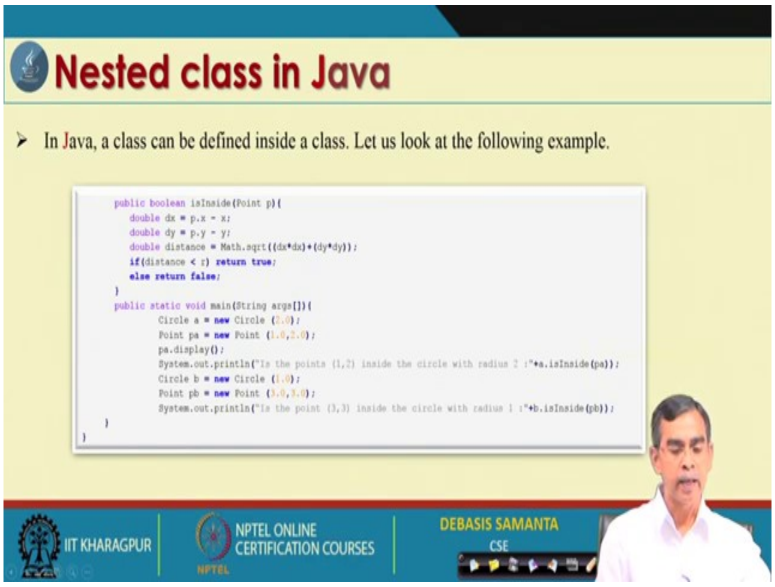
So, we have discussed about static scope rule, and then the class variable and instance variable as a class method and instance method.

Now, we will discuss about the nested class in Java program. The nested class means a class can be defined within inside another class. So, it is called a nested class and in Java there is no limit about how many nesting is there. It can be going and, but nested class is not a good practice because it basically creates a lot of i mean a source of many errors are there, but sometimes if we see that this class is only limited to the this part only then in that part we can discuss that class. So, being defined a class as a nested, it is basically very local to either that class or belong to that method or belong to that block.

So, nested class concept is it is although rarely used, but it has some usage in some context. So, let us discuss about the idea of the nested class concept. Again come to the declaration of class Circle that we have discussed here and here you see the circle class is declared, circle class is declared and within the circle class, we declare one class called point. We declare this class as static; we can declare simply public class Point. If we declare static means it is only one instance for all the circle of that we created that is why the concept is there anyway.

So, this is a one point that we have declared within this class. So, this class concept, this declaration, this declaration is a nested declaration, and it is the example of nested class. Now let us see the complete code for this.

(Refer Slide Time: 19:55)



Nested class in Java

- In Java, a class can be defined inside a class. Let us look at the following example.

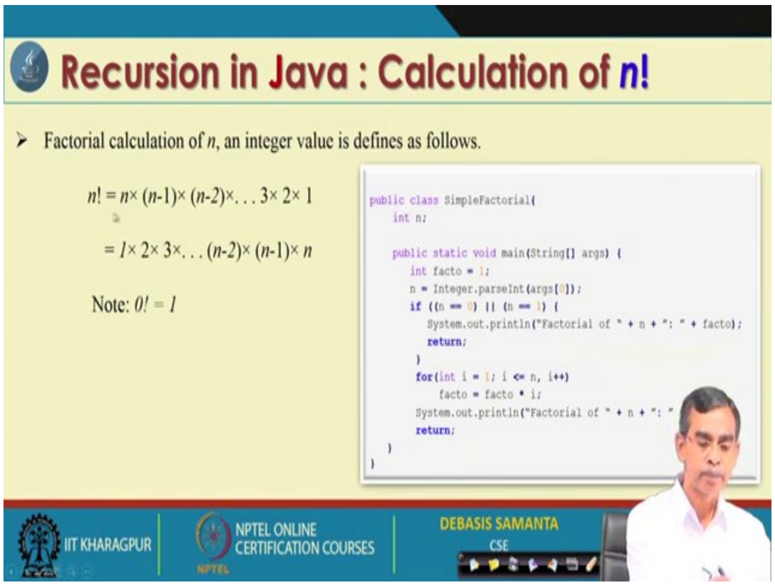
```
    public boolean isInside(Point p){
       double dx = p.x - x;
       double dy = p.y - y;
       double distance = Math.sqrt((dx*dx)+(dy*dy));
       if(distance < r) return true;
       else return false;
    }
    public static void main(String args[]){
            Circle a = new Circle (2.0);
            Point pa = new Point (1.0,2.0);
            pa.display();
            System.out.println("Is the points (1,2) inside the circle with radius 2 :"+a.isInside(pa));
            Circle b = new Circle (1.0);
            Point pb = new Point (3.0,3.0);
            System.out.println("Is the point (3,3) inside the circle with radius 1 :"+b.isInside(pb));
  }
}
```

Here is the complete code and there simple one code here. So, we declare one method within the class Circle, the name of the method that we have declare here is a boolean method; Boolean is a variable, the name of the method is inside and these are the code of this method, very simple code it basically the method is to decide whether a point inside a circle or on the circle or outside the circle. It is a simple logic that you can follow it will create a distance from the centre of the circle to that point.

And this distance is less than radius of the circle then it is inside. If it is equal to the radius then it is on and then if it is greater than the radius then it is there. So, this concept it is used here. So, this Isinside method and let us see the main method. The main method creates 2 objects a is the object of class Circle, pa is the point and then pa dot display is basically call the display method which is defined in the class itself. And then here basically it will check that whether this point pa is inside the circle or not. So, is basically we create another circle b and p b and then we can call whether this point 3, 3 is inside the circle or not and here this point p is inside the circle this circle or not.

So, this is the idea about that here the class p is to be local to the circle class. This means that this class point cannot be accessed by any other class either in this file or outside this file. That mean, if we use any other program where we can use these class Circle, but we cannot use the class point there exclusively, it can be used only the class Circle is used like this one. So, class circle class point p class point is totally local to the class circle only. So, it has limited access in that sense.

(Refer Slide Time: 22:13)



Recursion in Java : Calculation of $n!$

- Factorial calculation of n, an integer value is defines as follows.

$$n! = n\times(n-1)\times(n-2)\times\ldots 3\times 2\times 1$$
$$= 1\times 2\times 3\times\ldots(n-2)\times(n-1)\times n$$

Note: $0! = 1$

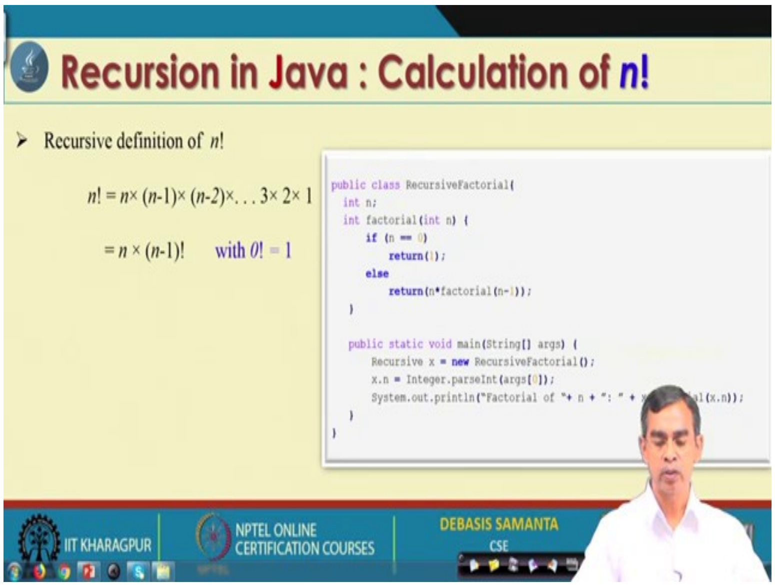
```
public class SimpleFactorial{
    int n;

    public static void main(String[] args) {
       int facto = 1;
       n = Integer.parseInt(args[0]);
       if ((n == 0) || (n == 1) {
          System.out.println("Factorial of " + n + ": " + facto);
          return;
        }
        for(int i = 1; i <= n, i++)
            facto = facto * i;
        System.out.println("Factorial of " + n + ": "
        return;
    }
}
```

So, after knowing the nested programming, nested class concept, the Java also provides us the recursive program.

Now, recursion is an important concept and it solve many complex problem very quickly, only the thing that is required is that to solve the problem which needs to be solved using recursion, we have to have a very clear definition of the problem itself. Now let us see how the recursive program can be written in Java, we will consider a simple example; we know the calculation of factorial. So, a factorial n factorial will look like this one, so the factorial n which look like this one.

So, n factorial is basically as you know this is a common declaration n factorial is look like this ok. And we know that 0 factorial equals to 1, so it is there. So, this basically this basically the concept of the factorial, and if we write a iterative program using some for loop, the calculation of this recursion is not a big job and this code is basically explain how recursive program, how the calculation of factorial can be done using an iterative program, so using for loop, but the same thing can be done using a recursive program, let us see how the recursion is there.

(Refer Slide Time: 23:59)



Recursion in Java : Calculation of *n*!

- Recursive definition of $n!$

$$n! = n \times (n-1) \times (n-2) \times \ldots 3 \times 2 \times 1$$
$$= n \times (n-1)! \quad \text{with } 0! = 1$$

```
public class RecursiveFactorial{
  int n;
  int factorial(int n) {
      if (n == 0)
          return(1);
      else
          return(n*factorial(n-1));
   }

   public static void main(String[] args) {
       Recursive x = new RecursiveFactorial();
       x.n = Integer.parseInt(args[0]);
       System.out.println("Factorial of "+ n + ": " + x[illegible]al(x.n));
   }
}
```

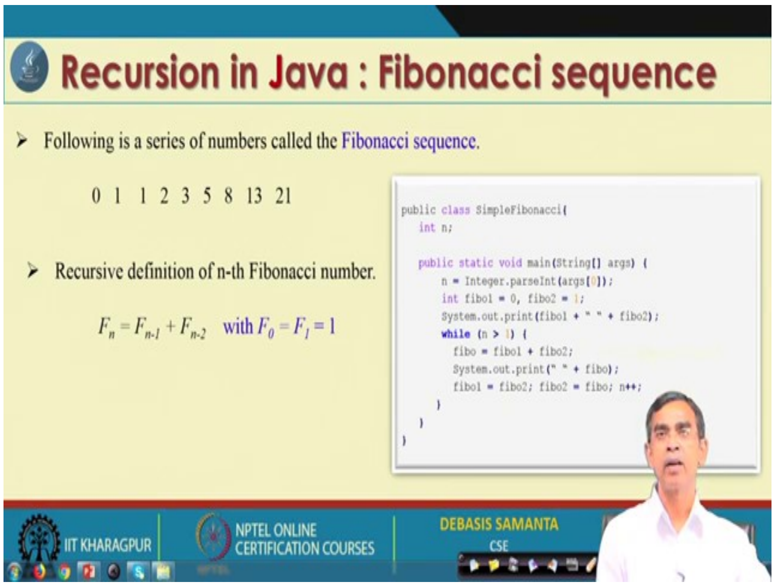
So, the factorial calculation can be defined recursively using this definition.

So, here is the definition of recursion n=n*(n-1)!. It is called the recursive definition, because n! is defined in terms of (n-1)!. So, this is why that mean the same thing is define by itself. So, that is why the concept of recursive. Now, this definition can be straight away implemented in a recursive program and now here is an example how we can see. So, here you just let us look at the code. So, we have declared here the one class Recursivefactorial and then integer n is a data for this class and factorial is the method which basically will calculate the factorial of any integer n.

Now, here is the code, basically complete codes for calculating the factorial of any integer and if you see n factorial this is a factorial n=n*(n-1)!, so this definition. Now, here also for every recursive program, you should consider the termination condition. So, is the termination condition is that 0 factorial equals to one because this factorial will call factorial n minus 1, to calculate the factorial n minus 1, it will call the factorial of n-2 and so and so on. So, call will go on this way, n to n-1 to n-2 to …and whenever it will come to 0 it basically return 1. So, no more call of the factorial function is there. So, this is a concept of factorial calculation.

And here is the basically we create an object. So, x is an object of this class. So, it is basically recursive factorial. The object is created and we read an integer value from the common line. So, x dot n is basically the value of the class at a (Refer Time: 26:24) n and then it will basically call the factorial x .n as a output it will give you the factorial of n basically. So, this is idea about factorial calculation. If you look this program little bit carefully you will be able to understand. So, this is the idea about, now if we practice more recursive program, then a concept of recursion will be good. So, I will just clear few more example.

(Refer Slide Time: 26:57)



Recursion in Java : Fibonacci sequence

- Following is a series of numbers called the Fibonacci sequence.

  0 1 1 2 3 5 8 13 21

- Recursive definition of n-th Fibonacci number.

  $F_n = F_{n-1} + F_{n-2}$ with $F_0 = F_1 = 1$

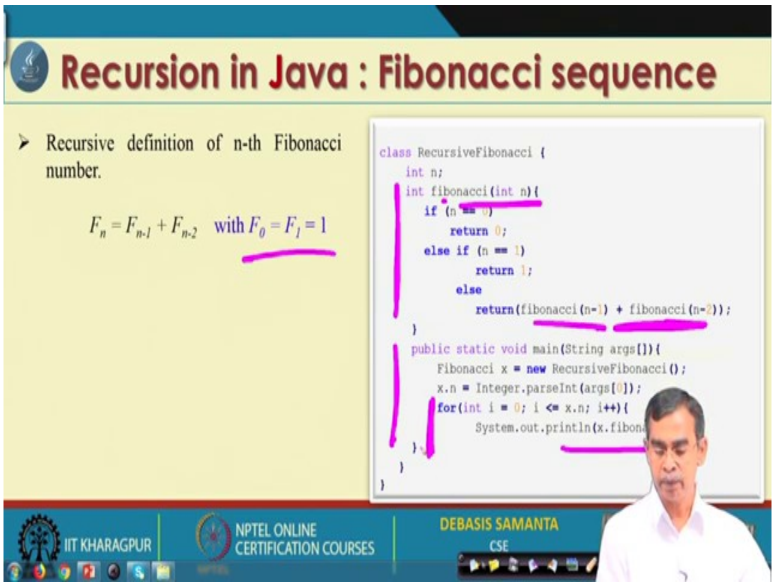
```
public class SimpleFibonacci{
   int n;

   public static void main(String[] args) {
      n = Integer.parseInt(args[0]);
      int fibo1 = 0, fibo2 = 1;
      System.out.print(fibo1 + " " + fibo2);
      while (n > 1) {
        fibo = fibo1 + fibo2;
        System.out.print(" " + fibo);
        fibo1 = fibo2; fibo2 = fibo; n++;
      }
   }
}
```

Another example, this is also good example for to discuss the recursive program writing. Now the series I have listed here. If you see the series, then if I ask you what is the next number, the next number you can guess that this next number will be 34. So, here if we see any number is basically sum of the previous 2 numbers. So, this way the recursion is here. Recursion means, here again I just discuss about how the recursion come into the picture here. So, the idea about this, and so if I ask you to write a program which will print all the numbers according to this series, this series particularly called the Fibonacci series or Fibonacci sequence. And this is the iterative program is very simple loop, that loop is basically declared here, you can understand that this loop will create this kind of. So, these are iterative approach of calculating Fibonacci series.

Now, let us see how the same can be done using writing a recursive function. So, in that case you have to understand that how this Fibonacci series can be expressed in a recursive way. Now, so here is basically the recursive definition, so far the nth Fibonacci number is concerned where n maybe 0 1 whatever it is there. So here basically this basically the recursive definition of any nth Fibonacci number, if you see if you have to calculate the nth Fibonacci number it is basically Fibonacci(n-1)+ Fibonacci(n-2). Similarly, n-2 can be recursively calculated n minus 2 can be recursively calculated. So, this way a recursion can be grown out and this basically is the termination condition.

Now having this definition within us we are now in a position to calculate the recursive version of the Fibonacci series calculation.

(Refer Slide Time: 29:19)



Recursion in Java : Fibonacci sequence

- Recursive definition of n-th Fibonacci number.

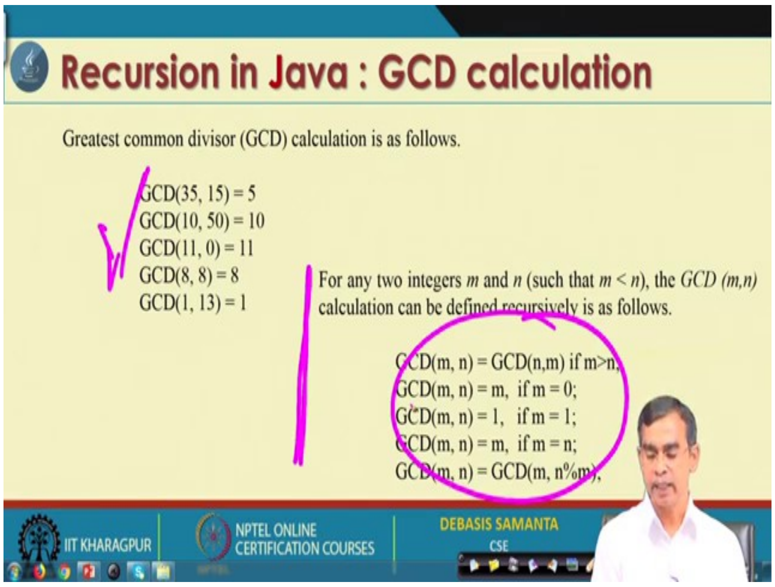
$F_n = F_{n-1} + F_{n-2}$ with $F_0 = F_1 = 1$

```
class RecursiveFibonacci {
    int n;
    int fibonacci(int n){
       if (n == 0)
          return 0;
       else if (n == 1)
             return 1;
           else
             return(fibonacci(n-1) + fibonacci(n-2));
     }
     public static void main(String args[]){
        Fibonacci x = new RecursiveFibonacci();
        x.n = Integer.parseInt(args[0]);
        for(int i = 0; i <= x.n; i++){
             System.out.println(x.fibona
        }
   }
}
```

So, here is the recursion recursive version of the program. Now let us look the program here we have so, this is basically the termination condition. So, this is basically here is a termination condition as we see, this is corresponding this one is implemented and this basically Fibonacci(n)= Fibonacci(n-1)+ Fibonacci(n-2). is basically Fibonacci n minus 1 plus Fibonacci n minus 2. So, this way the recursion will be carried out and this is a simple main program, which create a which read a value from the keyboard and then calculate its Fibonacci and then print the things.

So, these basically, print the entire series of the Fibonacci sequence. So, this is a one program that you should practice with your own time. So that you can understand both the recursion program that we have discussed for the factorial as well as Fibonacci, so practice.

(Refer Slide Time: 30:19)



Recursion in Java : GCD calculation

Greatest common divisor (GCD) calculation is as follows.

GCD(35, 15) = 5
GCD(10, 50) = 10
GCD(11, 0) = 11
GCD(8, 8) = 8
GCD(1, 13) = 1

For any two integers *m* and *n* (such that *m* < *n*), the *GCD (m,n)* calculation can be defined recursively is as follows.

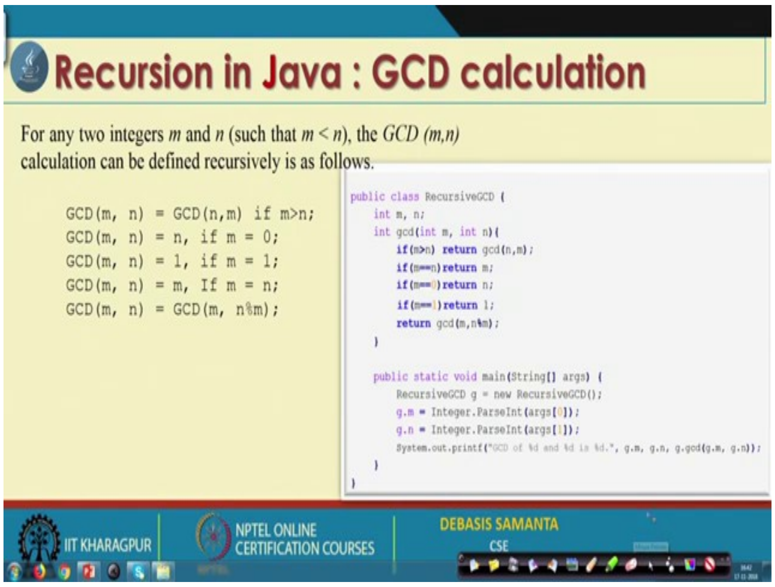
GCD(m, n) = GCD(n,m) if m>n;
GCD(m, n) = m, if m = 0;
GCD(m, n) = 1, if m = 1;
GCD(m, n) = m, if m = n;
GCD(m, n) = GCD(m, n%m);

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
DEBASIS SAMANTA
CSE

Now, let us before conclusion, let us have one more example and this is the GCD calculation, GCD stands for greatest common divisor. As we see GCD of 2 numbers integer number rather can be calculated like this ok. So, GCD of 113, one this 8 and 8 is like this one. So, this is the as per the simple definition of GCD. Now let us see how the same the GCD can be defined recursively.

So, here is the recursive definition these basically, include the recursive definition of the GCD it is basically same as the same as this one right almost same this one; so this basically recursive. Now if we convert this part, if we convert this part into a Java program is a method in a Java program it basically calculation.

(Refer Slide Time: 31:11)



Recursion in Java : GCD calculation

For any two integers *m* and *n* (such that *m* < *n*), the *GCD (m,n)* calculation can be defined recursively is as follows.

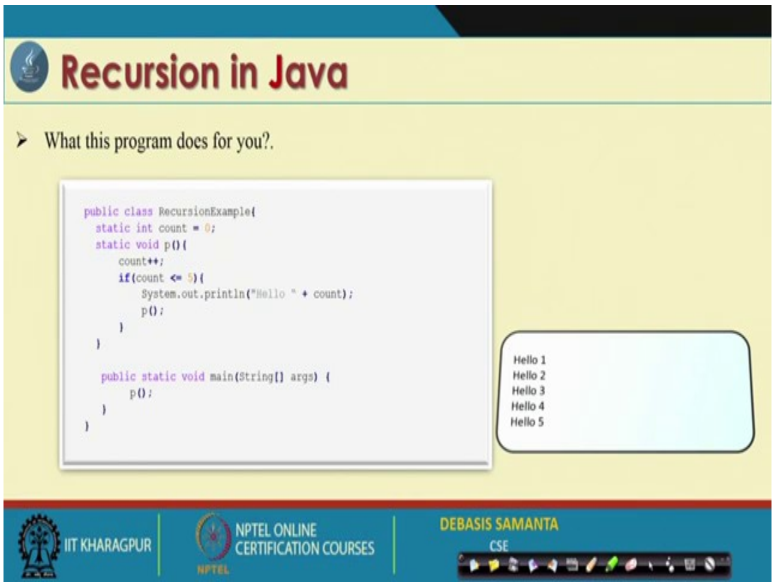
```
GCD(m, n) = GCD(n,m) if m>n;
GCD(m, n) = n, if m = 0;
GCD(m, n) = 1, if m = 1;
GCD(m, n) = m, If m = n;
GCD(m, n) = GCD(m, n%m);
```

```
public class RecursiveGCD {
    int m, n;
    int gcd(int m, int n){
        if(m>n) return gcd(n,m);
        if(m==n)return m;
        if(m==0)return n;
        if(m==1)return 1;
        return gcd(m,n%m);
    }

    public static void main(String[] args) {
        RecursiveGCD g = new RecursiveGCD();
        g.m = Integer.ParseInt(args[0]);
        g.n = Integer.ParseInt(args[1]);
        System.out.printf("GCD of %d and %d is %d.", g.m, g.n, g.gcd(g.m, g.n));
    }
}
```

Now, let us see how the method for the same in Java program will look like. So, this is basically is the method recursive version of the GCD calculation. The definition those things that we have discussed here, here it is basically implemented here, and then here the recursive version is that is GCD is basically GCD of this one, so this concept it is there.

So, this way the recursion can be carried out and this is the main method where we just call this recursive method to calculate the recursion for any 2 integer read from the command line argument from the keyboard. So, this is the idea about the GCD calculation and so we have discussed about few example of GCD calculation.

(Refer Slide Time: 32:01)



Recursion in Java

- What this program does for you?.

```
public class RecursionExample{
  static int count = 0;
  static void p(){
      count++;
      if(count <= 5){
          System.out.println("Hello " + count);
          p();
      }
  }

   public static void main(String[] args) {
       p();
   }
}
```

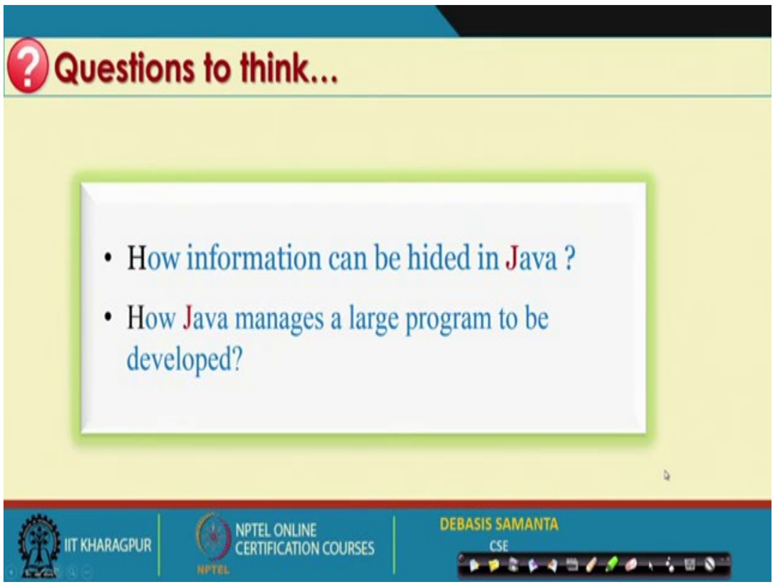
```
Hello 1
Hello 2
Hello 3
Hello 4
Hello 5
```

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | DEBASIS SAMANTA CSE

Now, let us consider another simple program. Can you guess it? Although it will take some time right maybe 30 seconds you can think about. So, what this code will give results?

If you think it then you can understand that recursion is little bit understandable to you. Anyway, if you run this program with your own machine and then it will you will see that it will give this kind of output. Can you explain why this output for this program? So, if you know the recursion then you will be able to explain these things very carefully.

(Refer Slide Time: 32:35)



Questions to think...

- How information can be hided in Java ?
- How Java manages a large program to be developed?

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | DEBASIS SAMANTA CSE

Anyway, this is about the concept of static scope rule in Java we have discussed about. And there then in our next lecture we will learn about information hiding, this is very important concepts. And then also we should learn about how a very big Java program can be developed.

So, these are the next topics that will be covered in the next discussion.

Thank you very much.