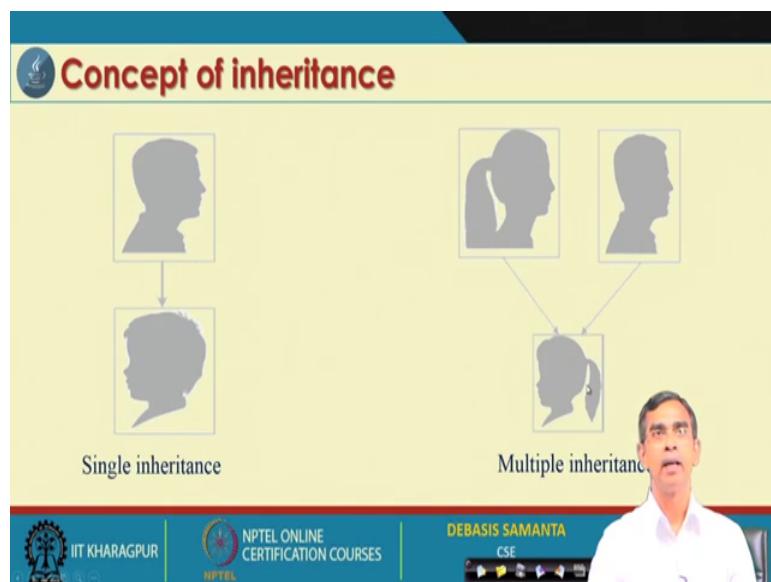


Programming in Java
Prof. Debasis Samanta
Department of Computer Science Engineering
Indian Institute of Technology, Kharagpur

Lecture – 13
Inheritance

So, in this lecture we will discuss one important object oriented paradigm is called the Inheritance. We have discussed the encapsulation, then inheritance is another important object oriented paradigm; in today's lecture we will discuss about the Inheritance. So, first we should learn about the concept and then how this concept is basically implementable in Java program.

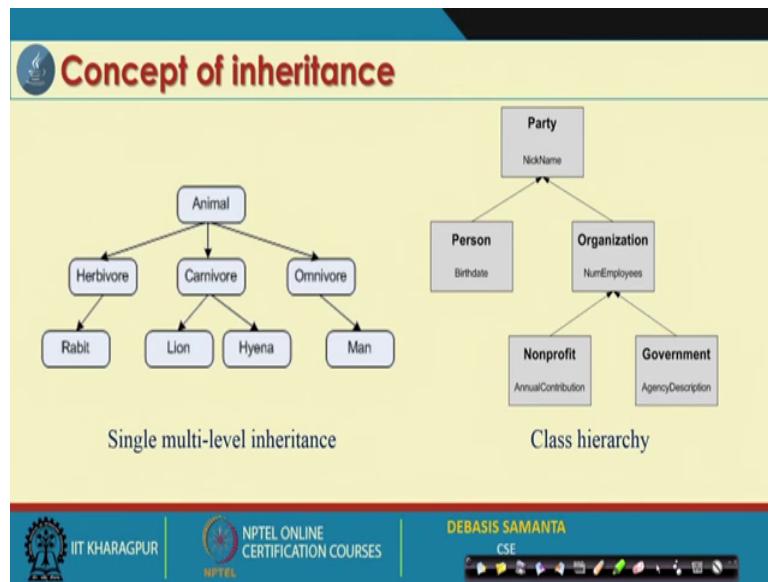
(Refer Slide Time: 00:41)



Now, inheritance is very common concept is a biological term although and you know exactly the inheritance means basically inheritance a child children inherits from its parents order it is the (Refer Time: 00:54). So, the concept it is like this and then single inheritance means, if it inherits from on only on entity and multiple inheritance means if one entity inherits from the multiple entities.

So, concept it is there the both single as well as multiple inheritance and then inheritance can be also hierarchically with multiple levels; so, multi level inheritance. So, children inherits from the parent grandchildren inherits from the children like this one.

(Refer Slide Time: 01:23)



So, these are concepts actually very common it is there, but so far the object is concerned the concept it is also there. Here for example, if we consider animal is a kind of object then animal has the different other type of objects. So herbivore, carnivore like this one so, here animal is a general whereas, herbivore is a special and if we see the lion and hyena they are more special.

So, from generalized to more specialization is the concept of inheritance and this is called the ej concept; that means, lion is an animal hyena is an animal; so, is ej concept. So, they are followed they are basically related to the ej relationship. And there may be many hierarchy; that means, that hierarchy means it is in the different level.

(Refer Slide Time: 02:11)

Inheritance in Java

- Inheritance is one of the cornerstone of object-oriented programming because it allows the creation of hierarchical classification.
- Using inheritance, one can create a general class that include some common set of items.
- This class then can be used to create more specific classes which has all the items from the base class, in addition to some items of its own.

```
graph TD; Vehicle[Vehicle] --> CAR[CAR]; Vehicle --> TRUCK[TRUCK]
```

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | DEBASIS SAMANTA CSE

So, this is a concept of there now, here inheritance in Java is very useful for many reason. The first reason is that using inheritance we can create given a class another class. So, the concept of this thing is very important here.

(Refer Slide Time: 02:27)

Terms used in inheritance

- **Superclass:** A class that is inherited is called a superclass.
- **Subclass:** The class that does inheriting is called a subclass.
 - A subclass is a specialized version of a superclass
 - It inherits all of the instance variables and methods defined by the superclass and add its own, unique elements (i.e., variables and methods)
- **Reusability:** It is a mechanism which facilitates you to reuse the data and methods of the existing class when one create a new class.
 - One can use the same data and methods already defined in the previous class.

```
graph TD; Vehicle[Vehicle] --> CAR[CAR]; Vehicle --> TRUCK[TRUCK]; CAR --> Wagon[Wagon]; TRUCK --> FireTruck[Fire truck]
```

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | DEBASIS SAMANTA CSE

Now, so this concept is basically given a class we usually call it is a super class. And, if we can create another class from this class then this is called a sub class. As an example here we can see vehicle is a super class whereas, car and the truck are the two sub classes. Similarly, fire truck is another subclass of the class truck. So, here truck is the

super class of the sub class fire truck. So, this basically says that we can create the number of classes inheriting from its super classes.

Now what is the concept of inheriting? We will come to this discussion about what basically a sub class will inherit from its parent class. Now, this inheritance concept is very important in Java program, it is because reusability. So, if we have a class say truck then we can share this code of the class truck, and then we can implement some other code in fire truck. So, it is basically code share ability or the reusability even the maintainability, the code maintainability is very important aspect which can be done using the concept of inheritance.

(Refer Slide Time: 03:43)

The slide has a blue header bar with the title 'Inheritance syntax'. Below the title is a bulleted list:

- The **extends** keyword is used to define a new class that derives from an existing class. The meaning of "extends" is to increase the functionality.

Below the list is a code snippet in a white box:

```
class <Subclass-name> extends <Superclass-name> {
    //Data and methods in this sub-class
}
```

At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, NPTEL Online Certification Courses logo, and the name DEBASIS SAMANTA CSE.

Now, how the inheritance of a class is possible. So, there is a syntax in Java program. The inheritance in a class is possible by means upon key word called extends. So, here the class this is the name of the subclass which basically you want to inherit. And this is the name of the super class that from here you have to inherit and this is basically the code of the subclass in addition to the code or the method or members which is there in this subclass.

So, here basically all the codes which is there in super class is also accessible here in this method. So this way, basically the idea of the code readability come into the picture. So, this is the idea about that these things can be done using inheritance. Now let us see some example so, that we can understand about this idea about the inheritance.

(Refer Slide Time: 04:39)

The slide title is "Example of a simple Inheritance". It features a class hierarchy diagram where a "2D Point" class is shown as the base class, and a "3D Point" class is shown as the derived class, which inherits from the 2D Point class. Below the diagram are two code snippets in Java. The first snippet shows the definition of the 2D Point class:

```
class Point2D{  
    int x;  
    int y;  
    void display(){  
        System.out.println ("x=" + x + "y=" + y);  
    }  
}
```

The second snippet shows the definition of the 3D Point class, which extends the Point2D class:

```
class Point3D extends Point2D{  
    int z;  
    void display(){  
        System.out.println("x=" + x + "y=" + y + "z=" + z);  
    }  
}
```

The slide also includes the IIT Kharagpur logo, NPTEL Online Certification Courses logo, and the name Debasis Samanta, CSE.

Very simple example, that we are going to discuss about say suppose there is a class called the 2D point. So, 2D point is a class and we want to inherit another class called the 3D point. This means that 2D point, if it has some members and methods then all these member and method may be available to this 3D point or we can say that some method can be overridden; that means, it can be redefined in this method. Now, let us see one example. So, here is basically the statement of a class Point 2D it has two members x and y and display is the method.

So, these are the composition of this class. Now, here Point 3D which basically inherits point this is basically Point 3D inherits Point 2D. So, it is so it should be corrected this is Point 2D. So, the Point 3D extends Point 2D; that means, Point 3D inherits from the class point 2D. And, here we can see we declare integer z as a new elements on it. This means that for this class Point 3D all these things which is there also use accessible here. So, this means that x y and z all the three members are available to this or members of this class Point 3D. How again you can note it that display method which is there in Point 3D also it is there. However, the method which is defined here and there they are different. This means that this display overrides the super class method display.

(Refer Slide Time: 06:25)

The slide title is "Example of a simple Inheritance". It features a class hierarchy diagram where a "2D Point" class is generalized into a "3D Point" class. Below the diagram are two code snippets in Java:

```
class Point2D{  
    int x;  
    int y;  
    void display1(){  
        System.out.println("x=" + x + "y=" + y);  
    }  
}  
  
class Point3D extends Point2D{  
    int z;  
    void display2(){  
        System.out.println("x=" + x + "y=" + y + "z=" + z);  
    }  
}
```

The slide also includes the IIT Kharagpur logo, NPTEL Online Certification Courses logo, and the name DEBASIS SAMANTA CSE.

Now, on the other hand if we write say display 1 and display 2 then the two methods are there and here with this change, what is the idea is that like x y is available to this class Point 3D. Similarly, display 1 is also accessible to this method this one. So, this basically a Point 2D is a generalization concept is a general, then Point 3D is a special one; that means, it has many more things other than the Point 2D itself.

So, this is the concept of inheritance. And this is obviously, an example of single inheritance and as it is a very simple example we usually call it a simple inheritance example ok. So, this is the idea about the simple inheritance.

(Refer Slide Time: 07:19)

The slide title is "Example of a simple Inheritance". On the left, there is a class hierarchy diagram showing a "2D Point" class above a "3D Point" class, with an arrow pointing from 2D Point to 3D Point. On the right, there is a Java code listing:

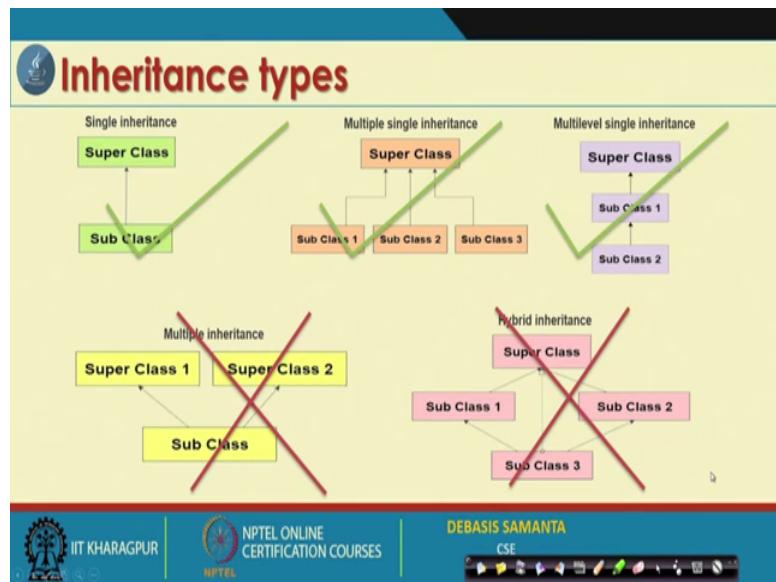
```
class simpleSingleInheritance{
    public static void main(String args[]){
        Point2D new P1();
        Point3D new P2();
        P1.x = 10;
        P1.y = 20;
        System.out.println("Point2D P1 is" + P1.display());
        /* Initializing Point3D
        P2.x = 5;
        P2.y = 4;
        P2.z = 15;
        System.out.println("Point3D P2 is" + P2.display());
    }
}
```

The slide footer includes the IIT Kharagpur logo, NPTEL Online Certification Courses logo, and the name DEBASIS SAMANTA CSE.

Simple inheritance and here is a main method that you can declare. Now if you can look at this method little bit carefully you will be able to see these are the objects that we have created for 2D class Point 2D and Point 3D. And these are the basically initialization. Similarly, here the initialization of the point 3 there is basically point 2 ok. So, this is basically initialization of point 2. Now, here if you see in P 2 being a point in class basically Point 3D x y is accessible.

So, this is the idea is that by means by virtual inheritance all the members and methods are accessible with the object of subclass. So, this is an example of the simple intent that we have discussed. Now, we quickly discuss about what are the different type of inheritance that is possible in the Java program.

(Refer Slide Time: 08:23)

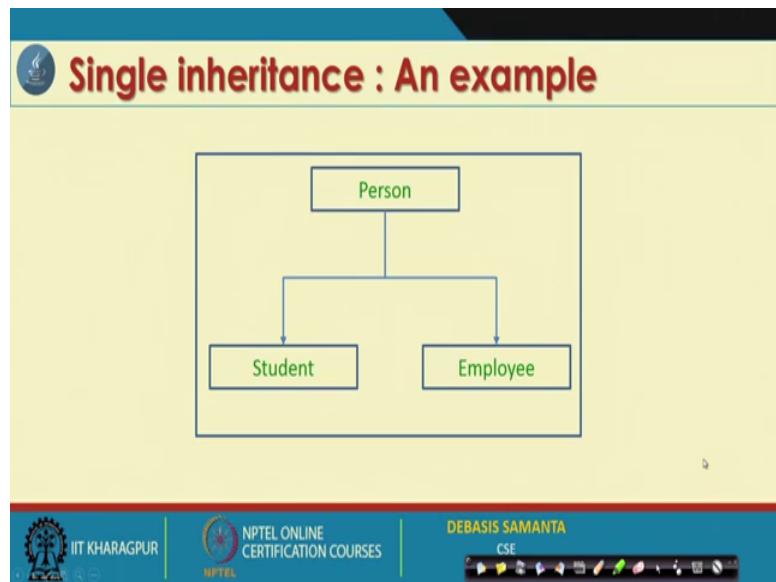


Now, this is an example of single inheritance. So, if we can derive a subclass from a super class these are simple inheritance, we can say other than simple and intense that they are may be more than one class can be derived from the same class. So, it is called a multiple single inheritance. For example, here subclass 1 is derived from the super class, subclass 2 is also derived from the super class subclass 3. So, this is another type of inheritance and again a sub class can be derived from a super class another subclass can be derived from another subclass (Refer Time: 08:56). So, this also is called a multi level single inheritance.

Other than this multilevel single inheritance there is a concept of called the multiple inheritance. Here for an example this class inherit from this and this. So, if a class inherits from the two classes then it is the example of multiple inheritance. And, here is basically the hybrid example here for example; this class inherit from this one this. So, these are single however, this class inherits from this one. So, it is called a hybrid inheritance that mean both single as well as multiple inheritance are there.

Now however, the super Java programming is concerned all inheritance are not possible rather in Java we have only single inheritance, multiple single inheritance, multi level single inheritance are possible. Whereas, the other two are not possible Java does not support multiple inheritance as well as hybrid inheritance.

(Refer Slide Time: 10:02)



Now, let us have little bit bigger example about inheritance maybe multiple multi multiple single inheritance we can say. Here for example, person is a general class and then student is another subclass which can be derived from the person. Similarly, employees of say another subclass can be derived from the person. Now, what is the actual idea about this inheritance that can be understand about it. Here basically person may have some data as well as method of its. Here for example, these are the different composition of the person. Now, let us come to the student it has another right and these are another. Now what it does mean, that what it does mean that that the student class has all these things in addition to this one.

Similarly, employee class has all these things in addition to this one. So, for an employee class all these things are available, for a student class all these things are available. So, this is the idea about that by means of inheritance all the course which is there; for example, these methods are nothing, but code can be accessible to this one. All the codes which is there also accessible to the employee class so, code (Refer Time: 11:23) is there. So, by means of inheritance we can have this one. Now, here quick a Java programming features our programming concept of this one.

(Refer Slide Time: 11:33)

```
class Person{  
    String name;  
    Date dob;  
    int mobileNo;  
    void readData(String n, Date d, int m){  
        name = n;  
        dob = d;  
        mobileNo = m;  
    }  
    void printData(){  
        System.out.println("Name : "+name);  
        dob.printDate();  
        System.out.println("Mobile : "+mobileNo);  
    }  
}  
  
class Student extends Person{  
    String institution;  
    int[] qualif = new int[20];  
    int rollNo;  
    int[] marks = new int[5];  
  
    void printBioData(){  
        printData();  
        System.out.println("Institution : "+institution);  
        System.out.println("Roll : "+rollNo);  
        for(int q=0; q<qualif.length;q++){  
            System.out.println("Marks "+q+": "+ qualif[q]);  
        }  
        for(int m=0; m<marks.length;m++){  
            System.out.print("Result "+m+": "+marks[m]);  
        }  
    }  
}
```

IT Kharagpur | NPTEL Online Certification Courses | DEBASIS SAMANTA CSE

As an example we can say about say a class person that we have declared whatever the idea that we have mentioned here written in a Java code like ok. You can relate to that figure pictures, that we have shown in the last slide is basically related to this one. This is a curve class person is a general class and now student class can be defined like. So, having this person is to us then we can have the student class which basically extends person class; so, it is there.

Next so, this way the person class can be declared about and now here you can see here the printData which is declared in this method, here is also used here to print the biodata of the student class like this one. Now so, this is the class student which has been inherited from the class person.

(Refer Slide Time: 12:31)

```
class Person{  
    String name;  
    Date dob;  
    int mobileNo;  
    void readData(String n, Date d, int m){  
        name = n;  
        dob = d;  
        mobileNo = m;  
    }  
    void printData(){  
        System.out.println("Name : "+name);  
        dob.printDate();  
        System.out.println("Mobile : "+mobileNo);  
    }  
}  
  
class Employee extends Person{  
    int empNo;  
    int[] salaryHistory = new int[10];  
    String organization;  
    String designation;  
    Date doj;  
    void printSalary(){  
        for(int s=0; s<salaryHistory.length;s++){  
            System.out.println("Salary "+s+": "+salaryHistory[s]);  
        }  
    }  
}
```

DEBASIS SAMANTA
CSE

Now, likewise we can have another inheritance the employee, employee also can be inherited from the class person who is basically has this one and whatever the different methods are there we can declare here. So, this way the class student and class employee is more what is called a code than the class person itself and inheritance is the utilization of this one. So, once the all the classes are used then in your main class we can use those things and you can process them.

(Refer Slide Time: 13:01)

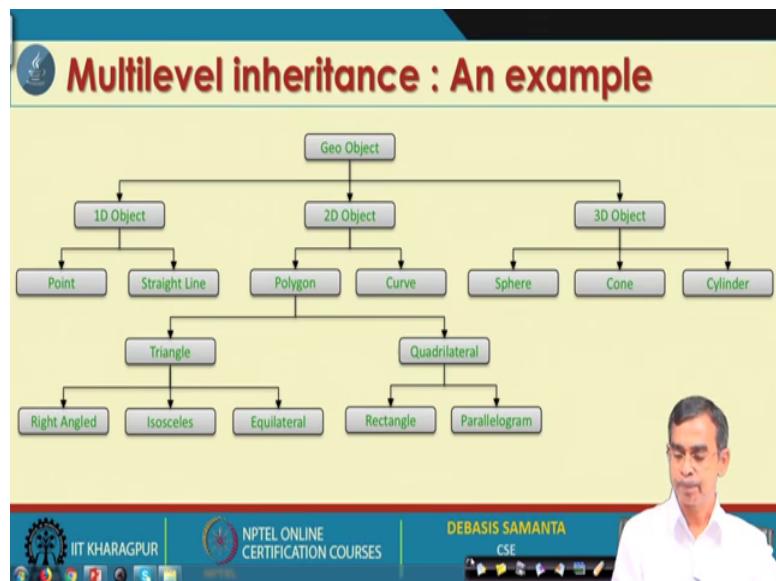
```
class inheritanceDemo{  
    public static void main(String args[]){  
        Person p = new Person();  
        //Code with the objects p.  
        Student s = new Student(100);  
        //Code with the objects s.  
        Employee e = new Employee(50);  
        //Code with the objects e.  
    }  
}
```

DEBASIS SAMANTA
CSE

Here for an example, we use the main method where the different objects of the class can be declared individually as if they are the new class of their own. So, this way the single inheritance is possible. Now, consider inheritance is like this is very simple not very difficult to understand that is why Java makes the thing so simple. Only allowing single inheritance and this inheritance can be gone into another level.

The same thing if I say suppose regular employee extends employee, then permanent employee temporary employee extends employee. So, that other two different classes can be inherited from the employee after the part employees inherited from personalization. So, this way extension has no limit to any level we can go.

(Refer Slide Time: 13:51)



Now, here is another good example so, that you can understand about it. So, there are many geometrical objects. So, all objects are of general type and then they can be a special type. So for example, 1D object 2D object 3D object. On the other hand if we consider the 3D object they are again different type, 2D object these are different type, 1D object is also different type. Now, further what is called the specialization also can be done. For example, 2D object can be of the further specialization like triangle quadrilateral and so on; quadrilateral maybe another specialization rectangle, parallelogram this one.

Now, this kind of inheritance hierarchy; now if you want to create a program for manipulating all geometrical objects so, first we can create the go objects which

basically has all the common attributes in it. Then whatever the special attributes slowly can be added into its inherited classes. And then finally, the classes at the bottom level can be obtained, they are basically the more refined or more specialized class that needs to solve your problem. So, this is a concept that is followed there and now you can write the programs or implementing all the type of objects that we have listed in a this section only. So, this concept can be extended like this one. Now, we will discuss about one concept the method overloading.

(Refer Slide Time: 15:13)

The slide has a blue header bar with the title 'Method overriding concept' in red. Below the title is a sub-section titled 'Usage of Java Method Overriding' with two bullet points:

- Method overriding is used to provide the specific implementation of a method which is already provided by its superclass.
- Method overriding is used for runtime polymorphism

Below this is another sub-section titled 'Rules for Java Method Overriding' with three bullet points:

- The method must have the same name as in the parent class
- The method must have the same parameter as in the parent class.
- There must be an IS-A relationship (inheritance).

At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the text 'NPTEL ONLINE CERTIFICATION COURSES', and a portrait of a man named DEBASIS SAMANTA, CSE.

In the example of Point 3D inherited from the Point 2D, we have discussed about method overriding; that means, that they are there is a method display which is basically defined in both the classes. If we declare a method which is already defined in a super class then the method called the overriding method. That means you have to override the method. So, this concept is called a method overriding. So, method overriding it is basically required that the method that we have declared in a super class can we can sometimes needs to be redefined here.

(Refer Slide Time: 15:53)

```
class Point2D{  
    int x;  
    int y;  
    Point2D(int a, int b){  
        x = a;  
        y = b;  
    }  
    void display(){  
        System.out.println("x = "+x+"y = "+y);  
    }  
}  
  
class Point3D extends Point2D{  
    int z;  
    Point3D(int c){  
        z = c;  
    }  
    void display(){  
        System.out.println("x = "+x+"y = "+y+"z = "+z);  
    }  
}  
  
class MethodOverridingTest{  
    public static void main(String args[]){  
        Point2D p = new Point2D(3, 0, -1, 0);  
        p.display(); // Refers to the method in Point2D  
  
        Point3D q = new Point3D(3, 0);  
        q.display(); // Refers to the method in Point3D  
  
        Point2D x =(Point2D) q; // Cast q to an instance of class Point2D  
        x.display();  
    }  
}
```

IT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | DEBASIS SAMANTA CSE

So, if we want to do that if redefinition then we can do it like this one. So, this is again continuation of the previous example that we have discussed about. Here the display method which is declared here these display basically override these display and then we can use it here in this program as we can see it is here. So, this method as we can as you can see here this method as you can see here.

So, Point 2D p is a point of 2D class and Point 3D q is a point of 3D class. And, here you can see x is another class which we have created which basically up casting; that means, q is a point of 3D, but we can cast into 2D using this kind of special features are there. So, typecasting we will discuss details here later on. So, then x dot display then this basically we will call the display method of this class although in q it is overriding. So, it is like this way we can have the access of this one this concept is called the dynamic binding.

So, dynamic binding is the one kind of runtime what is called a scope rule. So, scope will be decided from which, because if we cast with others then binding will be different and so on. So, these the dynamic a binding will be demonstrated in our practical class; so, that we can understand these features more clearly. So, this is the idea about method overriding it is there.

(Refer Slide Time: 17:27)

The slide has a blue header bar with the word "Note" in red. The main content area is yellow. It contains two bullet points:

- A sub class object can reference a super class variable or method if it is not overridden.
- A super class object cannot reference a variable or method which is explicit to the sub class object.

At the bottom of the slide, there is a video player interface. On the left, there are logos for IIT Kharagpur and NPTEL, followed by the text "NPTEL ONLINE CERTIFICATION COURSES". To the right of the video player, the name "DEBASIS SAMANTA" is displayed above a small image of a man speaking. Below the video player are several small icons.

Now, regarding this inherited inheritance concept one thing that we should note that a subclass object can refer a super class variable or method if it is not overridden. That means, all the methods and variables are accessible to the subclass if it is not defined in that class itself. On the other hand a super class; that means the reverse is not possible. That means, is super class cannot access any variable or any method which is defined in the subclass.

So, one way traffic it is actually. So, we can access in from the subclass platform only, but super from the super class platform other than the method are variables defining the super class we cannot access anything from the subclass.

(Refer Slide Time: 18:19)

The **super** keyword in Java is a reference variable which is used to refer immediate parent class members.

Whenever you create an instance of a sub class, an **instance of its parent class is created implicitly**, which is referred by **super** keyword.

Usage of Super Keyword

- 1 Super can be used to refer immediate parent class instance variable.
- 2 Super can be used to invoke immediate parent class method.
- 3 Super() can be used to invoke immediate parent constructor.

DEBASIS SAMANTA
CSE

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES

Now, there is another very important concept, it is called the super keyword which basically used for many purpose. So, super keyword has the many implication in this; using the super keyword in Java one can use one can refer immediate parent class variables. They are basically instance variability is there; super also this keyword also can be used to invoke parent class method and super also can be used to invoke parent class constructor.

So, there are many use of this super class. In our subsequent slides we will see how the super class can service the three different facilities: referencing variables, referencing method and referencing the constructor those are there in super class.

(Refer Slide Time: 09:15)

The slide title is "super : Referring parent class instance variable". The code示例 is as follows:

```
class Animal{    String color="white";}class Dog extends Animal{    String color = "black";    void printColor(){        System.out.println(color);        System.out.println(super.color);    }}class TestSuper1{    public static void main(String args[]){        Dog d = new Dog();        d.printColor();    }}}
```

The output is shown in a box: "black" and "white".

A text box on the right explains: "Animal and Dog both classes have a common property color. If you print the color property, it will print the color of the current class by default. To access the parent property, you should use super keyword."

At the bottom, there are logos for IIT Kharagpur and NPTEL, and a name DEBASIS SAMANTA CSE.

Now, this is an example if you can look at this example little bit carefully, you will be able to understand that this is basically an example using the super keyword, where we can refer a parent class of a instance variable. Now, let us see in this example we can see this color is a variable which is declared in a class animal and dog is another class which extends animal and in the dog class we can declare again the color variable. So, this color is basically overridden variable; that means, this color and this color they have the two different scope. Now, here if we see in the super class the variable color is white value is white whereas, in dog it is black.

Now, in the print color method which is basically new methods in the class dog it basically if we refer this color then this refer to this color. But, if we want to mention that this color I want to mention which belong to the super then I should write super dot color. So, this super dot color will refer to this value whereas, this color will refer to this one. Now, having this kind of concept now let us see this is basically on the main program. So, here d an object of type this classes and then d dot printColor we will call it and whenever it call it then it will basically show the output which will look like this. So, output is because of this first print color black, because this one and then super color white.

So, these are the two output is there. So, this is a concept that super keyword is used to reference the parent class instance variable. Now, this is this has another also use it is

called like the referencing the parent class variable, the super also can be used to reference the parent class method. The method is the concept is same as the previous one. Now, again look at this example here the animal is another class which has the method eat. Again in the dog which is an inherited from the inherit in subclass of the super class animal, it also declared eat.

This mean, that this method is basically an overriding method than this one. So, this method a has its own which has the scope within the class dog and this method has this kind of print statement. Now, bark is their totally new method belong to this and work is also another method which is defined this. So, bark and work are the newly added method in the class dog. Now, wherever now you see the work method which is a new method in the class dog it called this one. Now, is super dot eat you can understand what it does mean. It means that this is the method of this whereas, bark is the as usual because there is no resolution and then it is basically this one eat.

So, these basically resolve the namespace. So, eat method belongs to this if it is prefixed by super dot and these are the method it is there. So, using this super method we can refer that this method belongs to whether it is a super class method or belongs to the subclass method if it is overridden. And, this is an example of this one very simple you can understand d that dog object is created and d dot work, if we call you can guess that what output it should give it you; obviously, you can check that the output that it should give you this one. So, dog work super dot eat it is basically eating then, bark it will barking and then eat again eating great.

(Refer Slide Time: 23:17)

super : Invoking parent class method

```
class Animal{  
    void eat(){System.out.println("eating...");}  
}  
class Dog extends Animal{  
    void eat(){System.out.println("eating bread...");}  
    void bark(){System.out.println("barking...");}  
    void work(){  
        super.eat();  
        bark();  
        eat();  
    }  
}  
class TestSuper2{  
    public static void main(String args[]){  
        Dog d = new Dog();  
        d.work();  
    }  
}
```

Animal and Dog both the classes have eat() method. If you call eat() method from Dog class, it will call the eat() method of Dog class by default because priority is given to local.

To call the parent class method, you need to use super keyword.

eating...
barking...
Eating bread...



IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | DEBASIS SAMANTA CSE

So, this kind of output you can see it if you run this program. So, this concept is the concept of that using super class we can resolve the parent class method then the base class method.

(Refer Slide Time: 23:29)

super : Invoking parent class constructor

```
class Animal{  
    Animal(){System.out.println("animal is created");}  
}  
class Dog extends Animal{  
    Dog(){  
        super();  
        System.out.println("dog is created");  
    }  
}  
class TestSuper3{  
    public static void main(String args[]){  
        Dog d = new Dog();  
    }  
}
```

The super keyword can also be used to invoke the overloaded parent class constructor, if arguments are there, then they should be specified accordingly.

animal is created
dog is created



IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | DEBASIS SAMANTA CSE

Now, this is another example of use of super class is basically invoking the parent class constructor. We will use this kind of construct frequently in our subsequent program. So, we should understand it very carefully again this is a super class, this has one constructor animal. So, this is the super class constructor, dog is another class is an inherited from

the super class in animal. So, it is a subclass of animal and here if we see dog is a constructor. And, this draw constructor we use the super.

This means that if we use this say means dog will also call the super class constructor that is then animal is called here. So, is basically if we writing this one means it is a super class constructor is called here and then finally, dog his own method. So, this means that in this constructor we has the two print statement this and this as well as this one. Now, if we run this test case and then if we run this one then you can see that this kind of output you can get it. Animal is created and dog is created because of these two things are there.

So, this is the concept of the use of super here the super keyword; so, that if we write super within parentheses this indicates that it will basically call the parent class constructor. So, super we can see that super is a very important keyword. We have used similar kind of key word this earlier and then another key word new also earlier. So, those new these and super are very important keyword, we will understand also few more keywords later on.

(Refer Slide Time: 25:15)

super : Invoking parent class constructor

```
class Point2D{  
    double x, y;  
    Point2D(){x = 0.0; y = 0.0} //Default initialization  
    Point2D(double x, double y){this.x = x; this.y = y;}  
}  
class Point3D extends Point2D{  
    double z;  
    Point3D(){super(); z = 0.0} //Default initialization  
    Point3D(double x, double y, double z){  
        super(x, y);  
        this.z = z; }  
}  
class TestSuper4{  
    public static void main(String args[]){  
        Point3D p = new Point3D(2.0, 3.0, 4.0);  
    }  
}
```

If there is a number of overloading constructors in the super class, then you have to define the **super constructors** matching with each constructor.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | DEBASIS SAMANTA CSE

Now, this is another example of invoking parent a parent class constructor using the super. This example similar to the previous example, previous example was pretty simple. Here you can see Point 2D is the super class, 3D is an subclass and here whenever we create a constructor Point 3D we will construct this one; that means, we

can. Now, here again you can note that in Point 2D there are two constructors. So, super if we call then which constructor it will refer to, actually it depends on what kind of arguments are there.

If we call this kind of argument then this is basically reflect to that that constructor who is matches its argument. For example, here super is a default constructor. On the other hand we could write that super using these are the say three different value; then that constructor will be called here. So, it depends on so using the parsing proper argument which matching to that constructor, now in the super will refer to that constructor. So, this is the idea about the super constructor is there.

(Refer Slide Time: 26:23)

Dynamic method dispatch concept

Dynamic method dispatch is a process in which a call to an overridden method is resolved at runtime rather than compile-time. Also, it is called Runtime polymorphism.

In this process, an overridden method is called through the reference variable of a super class. The determination of the method to be called is based on the object being referred to by the reference variable.

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | DEBASIS SAMANTA CSE

And whenever we use this kind of inheritance one very important concept is that dynamic method resolution. So, this concept is basically runtime polymorphism if we create many object, many object many object finally, which belongs to the method belongs to which object it little bit confusing. So, that confusion can be avoided by means of ok, if you can understand this concept little bit carefully.

(Refer Slide Time: 26:45)

```
class Bike{  
    void run(){System.out.println("running");}  
}  
class Splendor extends Bike{  
    void run(){System.out.println("running safely with 60km");}  
}  
  
public static void main(String args[]){  
    Splendor b1 = new Splendor();  
    b1.run();  
    Bike b2 = new Bike();  
    b2.run();  
    Bike b3 = new Splendor(); //Up casting  
    b3.run();  
}
```

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | DEBASIS SAMANTA CSE

Now, one example can be given so, that you can understand about that which binding; that means, which are the method reflected to a particular call. Now, we can consider this example here. So, Bike is a super class, the Splendor is another subclass which has the run method, you can see run is overeating method here. And, after this declaration of the two classes super class and subclass we have the main method declared here ok. So, it is very simple, so b 1 is an object of type this class; that means, if we run b dot run so, it will run this code method.

On the other hand here bike b 2; that means, we create an object of type b 2 and b 2 dot run the resolution is quite see that this method will be run in this case. On the other hand now come here this is little bit tricky. Now, here we create an object by means of this memory allocator then slender, but actually we cast it. And, then we store it b 3, but its type of the Bike object and then if we call this b 3 dot run. So, then which method will be called here. Actually in this case as it is the object of splendor although it is b 3 run, we should not confuse that this b 3 as the object of class bike then this run is this one, it is not like that and it is basically Splendor; so, it is run.

This means that dynamically so, b 3 dot run it changed from this method to this method. So, it is called a runtime polymorphism and it has many utilization, those utilization we will discussed when discuss about the packages and others. So, if we have to store many objects in an array and then objects of different type then better idea is that that array can

be declared of the super class object type x. And, then if it is declare super class any subclass object can be put into that or array and it can be process in y irrespective of the different object. So, this is the one good example of the runtime polymorphism in the Java.

(Refer Slide Time: 29:05)

The screenshot shows a slide titled "Dynamic binding in Java: Example". The code is as follows:

```
class A {  
    void callMe ( ) {  
        System.out.println ( "I am from A " );  
    }  
}  
  
class B extends A {  
    void callMe ( ) {  
        System.out.println ( "I am from B " );  
    }  
}  
  
class Who {  
    public void static main (String args [ ] ) {  
        A a = new B ( ) ;  
        a.callMe();  
        B b = new B( );  
        b.callMe();  
    }  
}
```

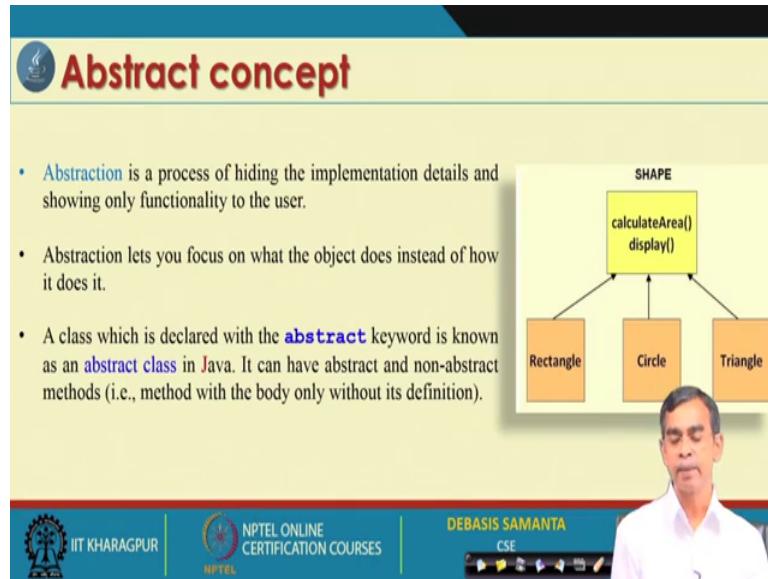
The output window shows two lines of text: "I am from B" and "I am from B".

At the bottom, there are logos for IIT Kharagpur, NPTEL, and DEBASIS SAMANTA CSE, along with other course-related icons.

Now, this is another example is you can guess that what output it will give it to for you. So, A this is a super class, this is a derived class, inherited class, subclass and this is the main method and you can understand that how it can. So, you just look at this point and then you can try to give the answer then you can understand that whether you have understood it or not.

So, if you run it this program and it will give this kind of output you can say and that you can resolve it how it is basically giving this kind of output. So, this basically the idea about that if inheritance is there, you have to little bit clear about that how the different method is called there.

(Refer Slide Time: 29:51)



The slide has a blue header bar with the title "Abstract concept". Below the header, there is a bulleted list of three points about abstraction:

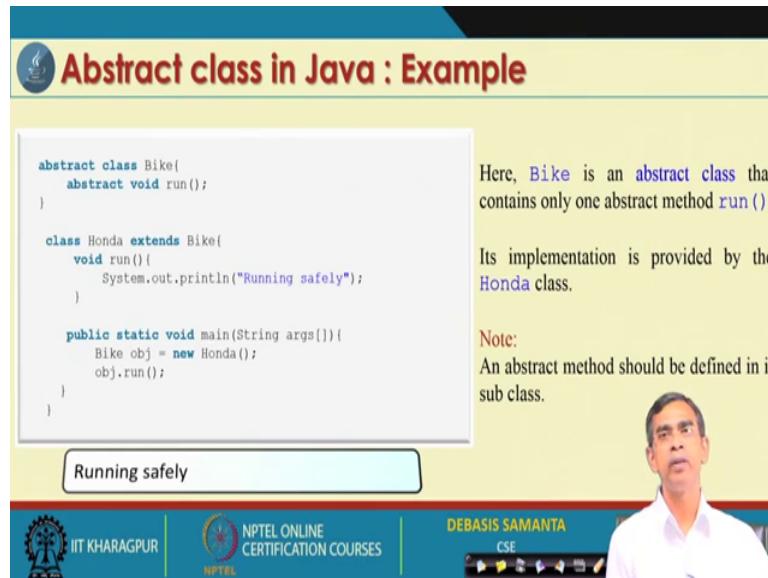
- Abstraction is a process of hiding the implementation details and showing only functionality to the user.
- Abstraction lets you focus on what the object does instead of how it does it.
- A class which is declared with the **abstract** keyword is known as an **abstract class** in Java. It can have abstract and non-abstract methods (i.e., method with the body only without its definition).

To the right of the list is a diagram illustrating inheritance. A yellow box at the top is labeled "SHAPE" and contains two methods: "calculateArea()" and "display()". Three orange boxes below it represent subclasses: "Rectangle", "Circle", and "Triangle". Arrows point from each subclass box up to the "SHAPE" box, indicating that the subclasses inherit from the abstract class.

At the bottom of the slide, there are logos for IIT Kharagpur and NPTEL, followed by the name "DEBASIS SAMANTA" and "CSE". There is also a small video player interface.

Now, I will quickly discuss about two more important concepts in Java: one is called the abstract class and another is the final class. A class can be declared an abstract if we declare an abstract class then its all method and all data member also can be declared an abstract actually.

(Refer Slide Time: 30:09)



The slide has a blue header bar with the title "Abstract class in Java : Example". Below the header, there is a code block showing an example of an abstract class "Bike" and its subclass "Honda".

```
abstract class Bike{
    abstract void run();
}

class Honda extends Bike{
    void run(){
        System.out.println("Running safely");
    }
}

public static void main(String args[]){
    Bike obj = new Honda();
    obj.run();
}
```

A callout box with the text "Running safely" is positioned over the line of code that prints the message.

To the right of the code, there is explanatory text:

Here, **Bike** is an **abstract class** that contains only one abstract method **run()**. Its implementation is provided by the **Honda** class.

Note:
An abstract method should be defined in its sub class.

At the bottom of the slide, there are logos for IIT Kharagpur and NPTEL, followed by the name "DEBASIS SAMANTA" and "CSE". There is also a small video player interface.

So, abstract class is basically the class which does not have any method to be defined clearly or the method can be kept as a voyage that mean without any code. Now, abstract

class as it does not have any code or any other thing. So, that any object of that class cannot be created.

(Refer Slide Time: 30:31)

The slide has a blue header bar with the title "Abstract concept". Below the title is a circular icon containing a stylized lamp. The main content area is yellow and contains the heading "Points to remember" in bold. A bulleted list follows:

- An **abstract class** must be declared with an **abstract** keyword.
- It can have abstract and non-abstract **methods**.
- It **cannot be instantiated**.
- It can have constructors and static methods also.
- It can have **final methods** which will force the sub class not to change the body of the method.

At the bottom of the slide, there is a footer bar with the IIT Kharagpur logo, the text "NPTEL ONLINE CERTIFICATION COURSES", and the name "DEBASIS SAMANTA CSE". To the right of the footer, there is a small video frame showing a man speaking.

So, it has certain properties like and abstract class it is declared by means of a specifier is called a abstract. And, it can have again abstract method and non-abstract method. All the method that we have discussed so far, non-abstract if a method is prefixed with the abstract keyword then it is called the abstract method. It cannot be instantiated this means that no object can be created for an abstract class. And, it can have final method if the abstract class has a final method then that object cannot be that method cannot be overridden in its derived class.

So, this is the idea about the abstract class and then so, abstract class is like this. So, if we can declare a method no objective is can be created, but it can it can be used to derive many classes from it. It is basically gives a template, template means this is a generalization a concept that this one whose ultimate implementation will be done when we derive the subclass.

(Refer Slide Time: 31:41)

The `final` keyword in Java is used to restrict the access of an item from its super class to a sub class. The Java `final` keyword can be used in many context.

- Variable : a variable cannot be accessed in sub class
- Method : a method cannot be called from a sub class object
- Class : a class cannot be subclassed.

Note:
If you make any class as `final`, you cannot extend it.

Hey, I'm final!
You can not
change my value,
You cannot
override me
you can not
inherit me

IIT KHARAGPUR | NPTEL ONLINE CERTIFICATION COURSES | DEBASIS SAMANTA CSE

Then the final keyword: the final keyword is very one is a strict keyword we can say that if we can declare a final, then this final class cannot be used for inheritance. So, no class can be derived from the final class that mean final class cannot be a super class. And, in addition to the declaration of a class as a final we can declare any variable any method in a class as a final also.

If we declare a variable as a final so, that variable cannot be overridden in any derived class. And, if a method is declared as a final so, same method cannot be overridden in any class objects. So, final in the sense final that it is basically no more implementation in any derived class is possible.

(Refer Slide Time: 32:31)

The slide has a title 'Final class in inheritance : An Example'. It contains the following Java code:

```
final class Bike{}  
class Hondal extends Bike{  
    void run(){  
        System.out.println("Running safely with 100kmph");  
    }  
}  
public static void main(String args[]){  
    Hondal honda = new Hondal();  
    honda.run();  
}
```

To the right of the code, a note says: 'Extending a class which is declared as **final** will cause **compile time error**. A man in a white shirt is visible on the right side of the slide.

Now, here is an example here we can see the class Bike is declared as a final. This means that this will give an error because it is not permissible. So, this is an error ok.

(Refer Slide Time: 32:49)

The slide has a title 'Question to think...'. It contains the following list of questions:

- Can we inherit a class from other class which is defined in other package?
- How information access can be restricted in a class?

A man in a white shirt is visible on the right side of the slide.

So, we have discussed about the basic concepts that is related to the inheritance of classes in Java programs. Now, there are many more questions that can be answered in subsequent classes. For example, can you inherit a class from other class which is defined in other package? Now, that here concept of package first should be learnt; so,

that we can give answer to this question. And, then information hiding that is on another pending job that we will discuss in our next lecture hours.

Thank you very much.