

FUNDAMENTALS OF OBJECT ORIENTED PROGRAMMING

Lecture13

Lecture 13: Inheritance - Multilevel Inheritance

Program in case super is not required

```
1. class Person{
2.     void message(){System.out.println("welcome");}
3. }
4.
5. class Student17 extends Person{
6.
7.     void display(){
8.         message();//will invoke parent class message() method
9.     }
10.
11. public static void main(String args[]){
12.     Student17 s=new Student17();
13.     s.display();
14. }
15. }
```



Welcome to Lecture 13: Inheritance. So, in the last class, we talked about super. Super can be used for instances, that means the member data, also for the constructor and the member function. So, there are cases where super is not required. Program, so where it is not required.

For example, you do not have the same function name, right? So, when you do not have the same function name, obviously, when I am calling, for example, Person is a superclass. Right? Person is a superclass. And here you have void message.

Right? Void message is printing System.out.println welcome. So now you have Student17, which is extending Person. So Person is a superclass, and Student17 is a subclass. Right?

So, let us assume you have a display member function. So here, you have a message. Right? So, the display member function is calling the message. The display member function is calling the message.

Right? So here, you have Student17. Person is the base class. I will just draw the diagram. And here, you have Student17 as the subclass.

Person is the base class or superclass. Student17 is the subclass. So here, you have a member function called display. Right? Student17 has a member function called display.

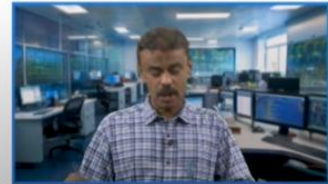
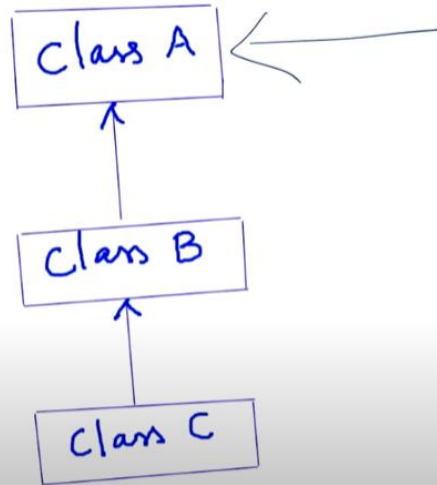
So under this, we are calling the message. Alright? So now you have an object s. Small s. Whose class is Student17? The Student17, right, class, the object s, which is invoking display.

So, when it is invoking display, the message will be invoked, right? So, the message is in the superclass. So, it goes automatically, right? It is the usual, the inheritance. So here, you do not need to use the super, right.

So that means it will automatically invoke this particular member function, and welcome will be printed. This is exactly how it is happening, right. When I run this code, you can find welcome is being displayed, right. So this is how the super or this can be used. So these two we have seen.

Maybe extensively, we will see some examples of this pointer. Okay. So now, super is understood. Maybe since we have had to see four different inheritances. So during that course,

Multilevel Inheritance



we will talk about that. So now, multi-level inheritance. In the last class, we had seen single or simple inheritance. So, multi-level inheritance. Assume that you have a parent class A, right?

Assume that you have parent class A. Now, class B is inheriting. Class B is a subclass, which is inheriting class A, right? Which is inheriting class A. That means it is using all the properties of class. And class C is inheriting class B. When you have a scenario like this, you call this multi-level inheritance, right?

Multilevel Inheritance

```
16     class BabyDog: public Dog
17     {
18         public:
19         void weep() {
20             cout<<"Weeping...";
21         }
22     };
23     int main(void) {
24         BabyDog d1;
25         d1.eat();
26         d1.bark();
27         d1.weep();
28         return 0;
29     }
```



So, this is an example of multi-level inheritance. So, under this, let us say we will take an example, right? So, you have class Animal, right? So, class Animal is the base class, right? So, under class Animal, you have one member function called eat, right?

Right. So, you have one member function called eat. Only one member function, and your class is over. Right. So, next you have Dog.

So, Dog is inheriting Animal. Right. So, Dog is inheriting Animal. So, class Dog is publicly inheriting Animal. So, this has one member function called bark.

Right. Your class is over here. Right. Class Dog is over here. End of the class Dog.

So now you have another class, BabyDog, which is publicly inheriting Dog. Right. So here you have BabyDog. Right. So here you have BabyDog.

So BabyDog is publicly inheriting Dog. Right. So this is what is happening. BabyDog is publicly inheriting Dog. So here you have one member function.

So that member function is nothing but weep. You have one member function. So that member function is called weep. So, you are seeing cout weeping. Alright.

So, that means it is displaying weeping. Alright. So, you have a scenario like this. Animal, Dog, and BabyDog. So, BabyDog is inheriting Dog, and Dog is inheriting Animal.

So, when you have such a scenario or a program like this, you call this multi-level inheritance in C++. So, now Assume that you are creating an object d1 under BabyDog. Okay. So, I am creating an object d1 under BabyDog.

Right. So, the d1. Right. Which is invoking eat. Right.

Where is this eat function available? Eat member function, line number 5. Right. So, that means eat is available in Animal. Yes.

I mean, d1 can inherit the property of Dog. Dog can inherit Animal. Right. So, in that process, d1 can access Animal because both are publicly inherited. So, d1.eat.

Under Dog, you have bark. So bark is under Dog. So the bark member function is under Dog. So now when it is invoking the eat function. So when d1 is invoking the eat function.

So this will be called. Right. Eating will be printed. So this will be my first output statement. Right.

So I can write. Maybe I'll use another. So this will be my first output statement. And then if you look, d1 is invoking bark. So bark, which is available in the class Dog.

Bark is available in class Dog. Right. So void bark. Void bark is barking. So this will be printed next.

All right. So this will be printed next. All right. And then the third one is d1 dot weep. All right.

So weep is being invoked. So d1 is invoking weep. So when it is invoking weep, weeping, the third output will be printed. Correct. So you can see three outputs.

So the first output is eating dot dot dot. All right. So that will be printed. And the second output is bark that will be printed, right? And then you can see the third output.

This will be printed, right? So, when I run the code, you can see as expected eating was our first output, right? If you look at the green in color, so that will be displayed, right? And then we... know that barking will be printed, the second one.

And the third one is weeping will be printed. So this is how the multi-level inheritance works, right? So you have created an object under d1. So now you can see that, right, under d1 we have created. So now we can see that this d1 can access the Dog class, right?

And the Dog class can access Animal. So that means BabyDog. Right. When you are creating an object under BabyDog and d1, you are trying to access, let us say, bark. Right.

It can be accessed. Right. d1.bark is not a problem. It is a parent class for BabyDog. Right.

So when it is trying to access d1.eat. Right. It is looking for the Dog class. So the Dog class is available. Now Dog can access the Animal.

Right. So, therefore, your void eat, the line number 5 is being invoked, and when line number 5 is being invoked, the first output you are getting is eating. So, d1.barj, d1 you

are creating under BabyDog, and BabyDog is trying to access your parent class. The parent class you have is Dog. And Dog, you have a member function bark.

So bark is being printed. And the third one is weep, which is available in the BabyDog class itself. All right. So which is being printed. So this is how exactly the multi-level inheritance is working.

Multilevel Inheritance in JAVA

```
class Class1{
    public void method1(){
        System.out.println(" Base Class Method ");
    }
}

class Class2 extends Class1{
    public void method2(){
        System.out.println(" inside Class2, method2");
    }
}

class Class3 extends Class2{
    public void method2(){
        System.out.println(" inside Class3,method2");
    }
}
```

```
class Main{
    public static void main(String[] args){
        Class3 c = new Class3();
        c.method1(); //from class1
        c.method2(); // overriding class2 and
                     // printing class3
    }
}
```

Output:

Base Class Method
Inside Class3, method 2



I hope it is clear for everyone. We will see how this will work in Java. All right. So we will consider an example. So let us say Class1.

So Class1, I will write C1. C1 is your base class. You mean to say it is your parent class. Under the parent class, you have one method. Method1.

System.out.println base class method. The base class method. And Class2 is extending Class1. I put C2. Class2 is extending Class1.

So here you have method2, a member function or method. So that method is invoking, right? When it is being invoked, in fact, it is defined inside Class2 method2. You have one print statement, right? So that print statement inside Class2, you have method2, correct?

And you have Class3. So that means Class2, C1, right? C2 is extending C1. Class2 is extending C1. Class1 is the superclass, and Class2 is the subclass.

And you have one more class. Let us say Class3. You call it C3. So, C3, you have one method. The method is, again, you have method2.

Okay. So here you have method2. Slowly, we are going to talk about overriding. That means polymorphism. Next, we are going to see.

So anyway, which is not a problem. So, Class3, you have the same method name. In fact, we use the superclass, right? Super, the keyword super, right? So, you will not have any problem.

So, let us see which one will be called and how it is being tackled. So, Class3, you have the same name, method2, right? Class3, you have the same method, method2. So, `System.out.println` inside Class3, method2. Inside Class3, method2.

So, now go to the main program, which is the driver class main function main method. You are creating an object c under Class3. I am creating an object c under Class3. So, dynamically you are creating, and then you are having the constructor over here. Correct.

So, now this Object c, right, this object c, which is invoking method1, right. So, this object is invoking method1. So, when it is invoking method1, right, the method1 which is available in Class1, right, which is not a problem. So, method1 will be accessed, right, invoked, and you have a print statement: `System.out.println` base class method.

So, the base class method will be printed. So, this one will be printed, which is not very difficult, right? It is printing. Now, when it is invoking method2, the question arises: which function will be invoked? Which method will be invoked, right?

So, we are coming to an interesting situation. Sooner or later, we are going to call this overriding, right? We are going to call this overriding, right? So, your object c is under Class3. Your object c is under Class3.

So, c is invoking method2. So first, it will go overriding, right? It is overriding Class2 and printing Class3, right? It is overriding Class2. So, it will not go over here.

That means this method2 will not be called. What I mean is this method2 will not be called. So, you are creating an object c under Class3, alright. So, overriding Class2 and then it is trying to invoke this method, alright. So, it is trying to invoke this method.

Case Study

In a university, various members belong to the institution, such as students and teachers. The university keeps track of basic information like name, age, and ID for every member. For students, additional information such as department and GPA is maintained, while for teachers, the department and subject they teach are tracked.

To manage this efficiently, we can implement a multilevel inheritance system using C++.

- The base class, **Person**, will store the basic information common to both students and teachers.
- The **Student** class will inherit from Person and add specific student-related information.
- The **GraduateStudent** class will inherit from Student and handle additional details related to graduate-level students.



When it is trying to invoke this method, you can see here inside Class3, method2 will be printed. So, this will be printed. So, this is what we call overriding. When we talk about polymorphism, if you look at the last lecture, we introduced super. So, under this super keyword, we have two methods, message, if you recall.

Right. In the last class, we saw the message. Right. We use the super keyword to go to the superclass. Right.

So, what will happen if I don't use the super keyword? Correct. So, which one will be overridden? For example, this is the situation. So, in this situation, what is happening?

You are trying to call method2. Right. You are trying to call method2, and method2 is available in both Class2 and Class3. So, in such a situation, So, what do I have to do?

So, which function should be called? So, this we have to be careful about. So, the concept is overriding. So, here the Class2 method2 will be overridden. And it is invoking method2 of Class3.

So, this is going to be called overriding. Method overriding in Java. The same if you use in C++. You call this as function overriding. So, it is overriding Class2.

And it is going or invoking Class3's method2. And then it is printing, right, inside Class3, method2. So, this is being printed. So, the concept is called overriding, in fact. So, this is an example of multi-level inheritance in Java.

So, you have a base class or superclass C1. You have a subclass C2. So, that means C2 is inheriting the properties of C1. And similarly, C3 is inheriting the properties of C2. And when you are creating an object C under C3.

One class C1 you have called. I mean, it is not a problem, right? See, when you are invoking method1, method1 will be invoked. The base class method will be printed. So, that we had seen, all right?

So, now the tricky point is when C is invoking method2, we have to be a little careful. Which one will be called, all right? So, this is the concept of overriding. In fact, we are going to see many examples, and in fact, the overloading concept we had seen. Basic points in polymorphism we had seen when the compiler knows during compile time, you call it as overloading.

And here, during runtime, which method will be chosen? So, this will be decided during runtime. So, therefore, your method2 is there, which will be overridden. And then it is invoking Class3's method2. And then, inside Class3, method2 will be printed.

So, this is exactly how multi-level inheritance works. So, we have seen one example in C++. Another example in Java. So now, based on this multi-level. Let us say ABC.

We stick to ABC. So, you can go. You can go to whatever level you want. So, let us consider this case study. Right.

```

1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  // Base class: Person
6  class Person {
7  protected:
8      string name;
9      int age;
10     int id;
11
12 public:
13     // Constructor for the base class
14     Person(string personName, int personAge, int personId)
15         : name(personName), age(personAge), id(personId) {}
16

```



So, let us consider this case study. In a university, various members belong to the institution. Right. So, overall, they will be in the institution. Students will be there, and teachers.

So, we consider the university, a class university. So, under the class university, we have students and teachers. Right. So, we have students and teachers. Right.

So, let us consider the university keeps a track record of basic information like The name, age, ID for every member, right? So, name, age, ID for every member. For students, additional information such as department, GPA, right, grade point average is maintained. While for teachers, the department and subject they teach are tracked, right?

So, which department do they belong to, and what are all the subjects they are teaching, right? So, to manage this efficiently, let us try to implement multi-level inheritance. So, we can see the example of a multi-level inheritance system using C++. So, the base class, Person, will store the basic information about the students and the teachers. So, that is the class Person.

And you have a class Student. You have a Student class. So, the Student class will inherit information from Person, right? So, this will inherit Person and add specific related information.

We will see what all the information we can add for the students, right? And you have another class, Graduate Student. So, this class will inherit from Student. So, Person is a

base class, Student is a subclass, and you have Graduate Student, right? So, it is like a grandchild.

```
31 public:
32     // Constructor for the Student class
33     Student(string studentName, int studentAge, int studentId,
34             string dept, float studentGPA)
35         : Person(studentName, studentAge, studentId),
36           department(dept), gpa(studentGPA) {}
37
38     // Method to display Student details (including Person info)
39 void displayStudentInfo() const {
40     displayPersonInfo();
41     cout << "Department: " << department << endl;
42     cout << "GPA: " << gpa << endl;
43 }
44 };
45
```



So, that means the parent of a graduate student is Sorry, a graduate student is a student, right? The parent of a graduate student is a student. So, this graduate student class will inherit from the student, right? And handle additional details related to graduate-level students, right?

So, this is how we are going to write a program in C++. So, Person is the base class. Student is a subclass, and you have the graduate student. So, let us see how this is working. So, you have a base class called Person.

Right. So, assume that you have the base class called Person. Right. So, here you have the base class. So, I have a protected member data.

All right. So we have already seen what is meant by protected member data. So you have string name. I have included string. Integer age and integer ID.

So you have member data like this. So protected. So in the derived class, it will become private. Right. So that is what we have already seen.

And you have public. So under public, you have a constructor. Right. The three-member constructor. Right.

Parameterized constructor. You have `personName`, which is a string. And you have `personAge`, which is an integer. And you have `personId`. All right.

So now you are copying. `personName` will be copied into `name`. `personAge` will be copied into `age`. And `personId` will be copied into `id`. All right.

I hope it is clear for you. All right. `personAge`, `personName`, `personAge`, `personId`. So we are copying into `name`, `age`, and `id`. So now you have one display method.

Right. So, `displayPersonInfo`. So, `displayPersonInfo`. So, I put it in `name`. Age and ID, right? So that I am trying to print over here in line numbers 19, 20, and 21: `name`, `age`, and `ID`, right? So now you have class `Student`, which is publicly inheriting `Person`, right? So, your `Person` is the base class. I use this `Person` as a base class. Now, I have the derived class.

Derived class is `Student`, right? Which is publicly inheriting `Person`, right? So, here you have the protected member data. So, these protected member data can be accessed by `Student`. `Student` class, right?

`Student` class. So, `Student` class again, a protected member data. String `department`, float `GPA`, right? So, this is what exactly we had seen in the problem, right? So, additionally, you should have

The `department` and `GPA` for students. Correct. So, string `department`. Float `GPA`. So now here you have the constructor `Student`.


So, constructor `Student`. It is right. Five parameters you have. `studentName`. Right.

The argument string. That means your data type is string. `studentAge` integer. `studentId` integer. Right.

```

46 // Further derived class: GraduateStudent (inherits from Student)
47 class GraduateStudent : public Student {
48 private:
49     string researchTopic;
50
51 public:
52     // Constructor for the GraduateStudent class
53     GraduateStudent(string gradName, int gradAge, int gradId, string dept,
54                     float gradGPA, string topic)
55         : Student(gradName, gradAge, gradId, dept, gradGPA),
56           researchTopic(topic) {}
57
58     // Method to display GraduateStudent details (including Student and Person info)
59 void displayGraduateStudentInfo() const {
60     displayStudentInfo();
61     cout << "Research Topic: " << researchTopic << endl;
62 }
63 };

```



StudentId is an integer. So, you have a department name. Whose data type is a string, and studentGPA is a float. Right. So now, you have a constructor.

Right. Student is a constructor over here. So, what I will do over here. Right. So here, you have the person.

Right. Person is a base class. Correct. So, under the base class, I have studentName, studentAge, and studentId. Right.

So that means I'm calling the base class constructor, Person. Right. So you have studentName, studentAge, and studentId. And department, you are copying into department. And studentGPA, you are copying into GPA.

Alright. So when I am having this studentName, studentAge, and studentId. Alright. So that will be nothing but your name, age, and ID. Correct.

Because it is called, I mean, it is calling your Person constructor. Okay. So now you have one more member function, displayStudentInfo. So under displayStudentInfo, I have, because when it is calling this, these three will be printed, right? Line number 20, 35, right?

So when it is calling, when you are passing studentName, studentAge, and studentId, and then when you are calling this person constructor, the name, age, and ID will be displayed. So, additionally, you require department and GPA, right? So, that will be

printed. And you have the next class, GraduateStudent, which is publicly inheriting Student, right? So, here you have GraduateStudent.

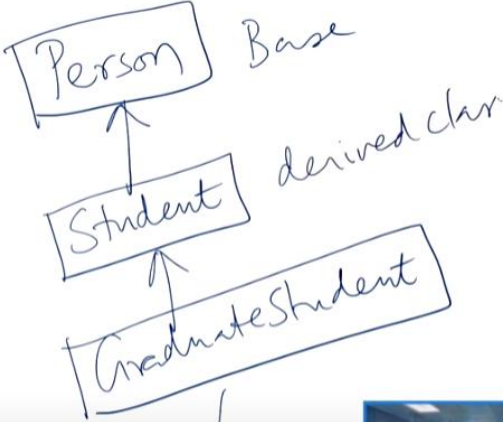
So, the GraduateStudent is publicly inheriting Student. The GraduateStudent is publicly inheriting Student, right? So, here you go, string researchTopic. So, that is one more topic. All right, we are adding because a GraduateStudent has a researchTopic.

So now you have the GraduateStudent constructor. So here you have six, right, six parameters. gradName, gradAge, gradId, department name. GradGPA and topic, correct? So, now when I call this Student constructor, right?

So, the Student constructor is what you require, right? The Student constructor. So, here you go. How many parameters does the Student constructor have? studentName, studentAge, studentId, department, and studentGPA.

Five parameters. So, there are five parameters, right? So, you are passing gradName. Graduate student age, gradId, department, and gradGPA are being passed. Alright.

```
1  #include <iostream>
2  #include <string>
3  using namespace std;
4
5  // Base class: Person
6  class Person {
7  protected:
8      string name;
9      int age;
10     int id;
11
12 public:
13     // Constructor for the base class
14     Person(string personName, int personAge, int personId)
15         : name(personName), age(personAge), id(personId) {}
16 }
```



And then the researchTopic, meaning the topic is assigned to researchTopic. Alright. In the previous class, your department is assigned to department, and studentGPA is assigned to GPA. Alright. So, now you have one additional display for researchTopic.

Alright. So, you have gradName, gradAge, gradId, department, and GPA. Everything will be there. So, when you have the researchTopic, right? So, topic is copied into

researchTopic, so that will be printed in this case in displayGraduateStudentInfo, right? So, basically, you are going to display. So, when you have the base class, you have three parameters, right? The base class has a three-parameter constructor, right? And then, when you have Student, you have additionally department and GPA, five parameters. And then one more, the sixth parameter will be added in the next class.

So, all are being inherited, right? So, GraduateStudent is inheriting Student. Student is inheriting Person, right? So, this is a base class, and this is a child class, derived class. Derived class of base, and this is the derived class of Student, derived class of Student.

```
64
65 ▾ int main() {
66     // Creating an object of GraduateStudent and displaying its details
67     GraduateStudent gradStudent("Ravindra", 35, 8, "Computer Science",
68                                8.3, "Artificial Intelligence");
69
70     // Display information of the Graduate Student
71     cout << "Graduate Student Information:" << endl;
72     gradStudent.displayGraduateStudentInfo();
73
74     return 0;
75 }
76
```



Okay, so multi-level inheritance. So now we will go to the main program. So, in the main, you are having a GradStudent object, right? You have a GradStudent object under the GraduateStudent class, the last one, right? You are passing, see how many parameters: Ravindra, the name, age 35, ID 8, right? The branch, computer science, right. See, GPA 8.3, GPA 8.3, and the topic, Artificial Intelligence, right? So, you are passing 1, 2, 3, 4, 5, 6 parameters. First, initially 3, then 2 is being added, and finally, for the GraduateStudent, 1 is also being added. So, when you are passing this, right?

So here you go, your GraduateStudent constructor will be called, right? You are passing all these, right? And then this will call the Student constructor. Right. So the first five parameters—graduate name, graduate age, graduate ID, department, graduate GPA—will be passed to Student.

Right. So when it is calling the Student constructor, So here you have these five. It will call the Person constructor. In the Person constructor, you are passing three parameters.

Right. Name, age, and ID. Right. So that will be copied. Right.

So that means these three will be initially displayed, right? So then these two will be displayed. Then finally, your research topic information will be displayed. So I will get six outputs. So six outputs will be nothing but Ravindra, age 35, ID 8, right?

You have a Computer Science department. CGPA is 8.3. Research topic is Artificial Intelligence. So the first three are the base class. Next two, Computer Science and 8.3, are the derived class student.

And the last one is under the GraduateStudent constructor. So when you are displaying this, that means the gradStudent object, when it is invoking displayGraduateStudentInfo, All right. So that means displayGraduateStudentInfo will be called. Correct.

So, this will call the displayStudentInfo method. So, the displayStudentInfo method is available over here. That will be called displayPersonInfo. So, the displayPersonInfo is available here. First name will be printed.

Age will be printed. Then, ID will be printed. Right. So, then your department and GPA will be printed from here. And then, finally, the research topic will be printed, right?

So, you will get the output like this. So, when you run the code, you should get the output like this. So, the name is Ravindra, age is 35, ID is 8, department is Computer Science, GPA is 8.3, and the research topic is Artificial Intelligence, alright? So, this is all about your multi-level inheritance.

So, single-level inheritance and multi-level inheritance we have seen. So, in the next class, we will see further inheritance topics. With this, I am concluding this lecture. Thank you very much.