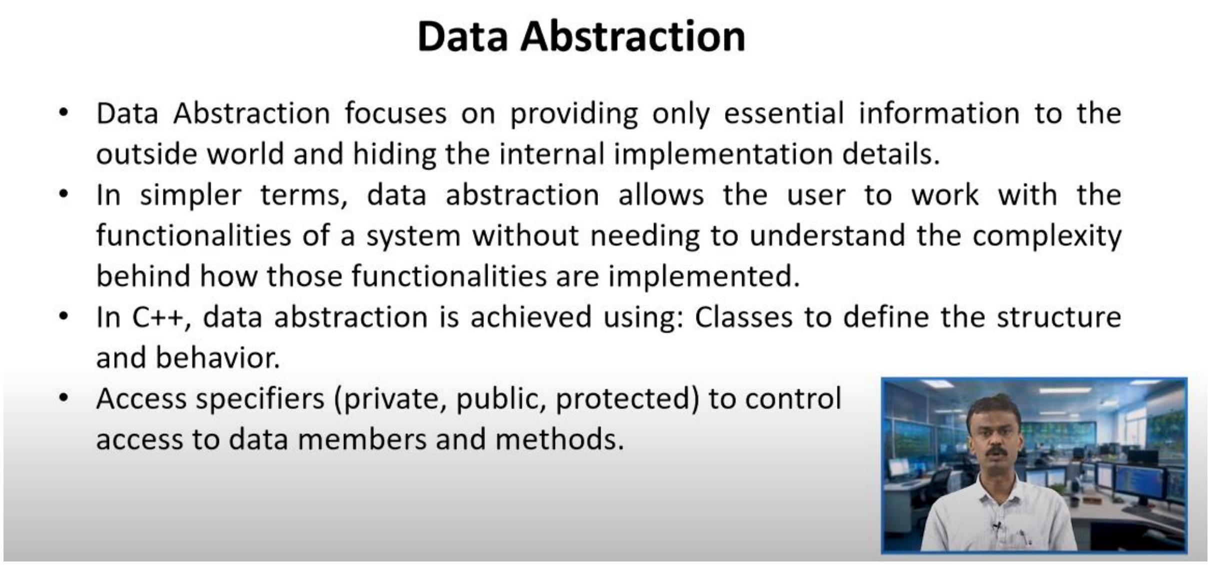
FUNDAMENTALS OF OBJECT-ORIENTED PROGRAMMING

Lecture23

Lecture 23: Data Abstraction



Data Abstraction

- Data Abstraction focuses on providing only essential information to the outside world and hiding the internal implementation details.
- In simpler terms, data abstraction allows the user to work with the functionalities of a system without needing to understand the complexity behind how those functionalities are implemented.
- In C++, data abstraction is achieved using: Classes to define the structure and behavior.
- Access specifiers (private, public, protected) to control access to data members and methods.

Welcome to Lecture 23: Encapsulation and Abstraction. So, in this lecture, we are going to talk about data abstraction. So, when I am talking about abstraction, it is nothing but, for example, when you are writing a research paper, right? So, initially, you start with an abstract after the title. You start with an abstract, which means the detailed implementation you are not going to give in the abstract. All right, it will come later. All right, so suppose I want to prove some theorem or give some experimental results. We will do it later. Correct?

So, initially, we are writing what exactly I am going to do in this paper. In a similar way, so when I am talking about, let us say, the abstract class. Alright? So, there are member data and member functions or methods. So, the implementation details I will define later or in the derived class.

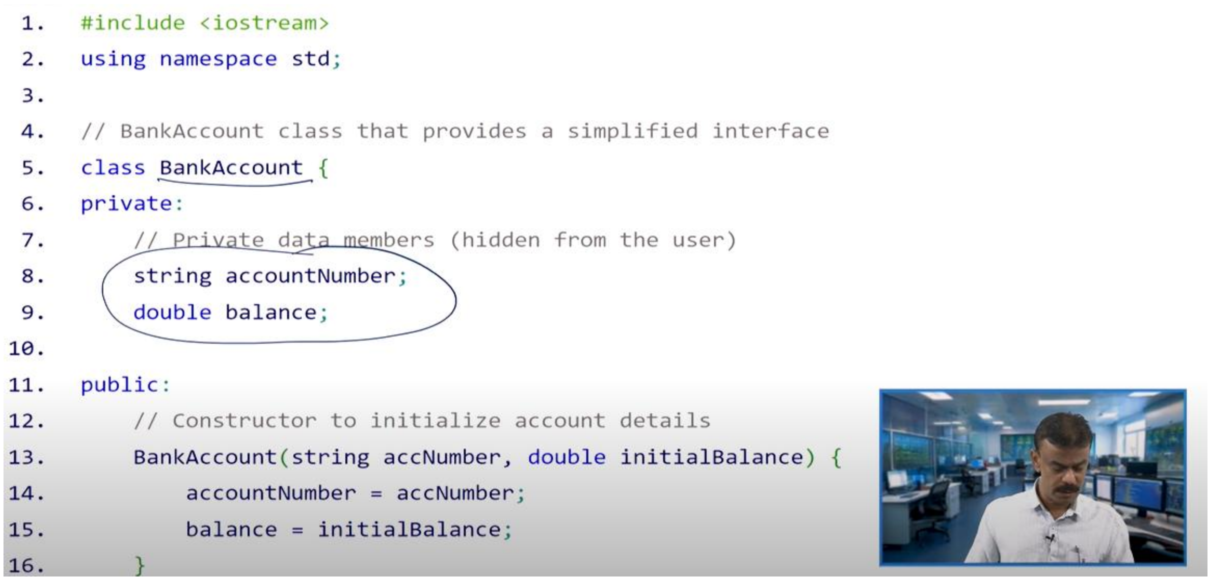
So, initially, when I am talking about the abstract class, So, for example, I want to do the factorial method or factorial member function, right? Or function 1, function 2, function 3, right? So, these are all the functions or the names of the functions. Let us say I am going to calculate the standard deviation or average, right?

Or the maximum of two numbers. So, I am giving the name of the function or just declaring the name of the function. So, then I will define it in the later stage. So, in the derived class, I will define the details, the implementation details.

Initially, in the base class, we will hide it. So, that is called data abstraction. So, in fact, suppose I am using only member data. In today's class, we are going to see only the member class, right? Only the data members.

So the data members, so let us say I am declaring, right? So we know the access modifier. If you look, I can define, let us say, private, public, and protected. Suppose I am defining only private, right? So that data cannot be accessed outside the class, right?

So, that data cannot be accessed outside the class, right? So, what can we do? So, let us say, whether in C++ or in Java, assume that we define the class. So, when I am talking about abstraction, or simply let us say an abstract class, I define the class which will be the abstract class, and I have to derive it later, right? So, that means I have to have a derived class.



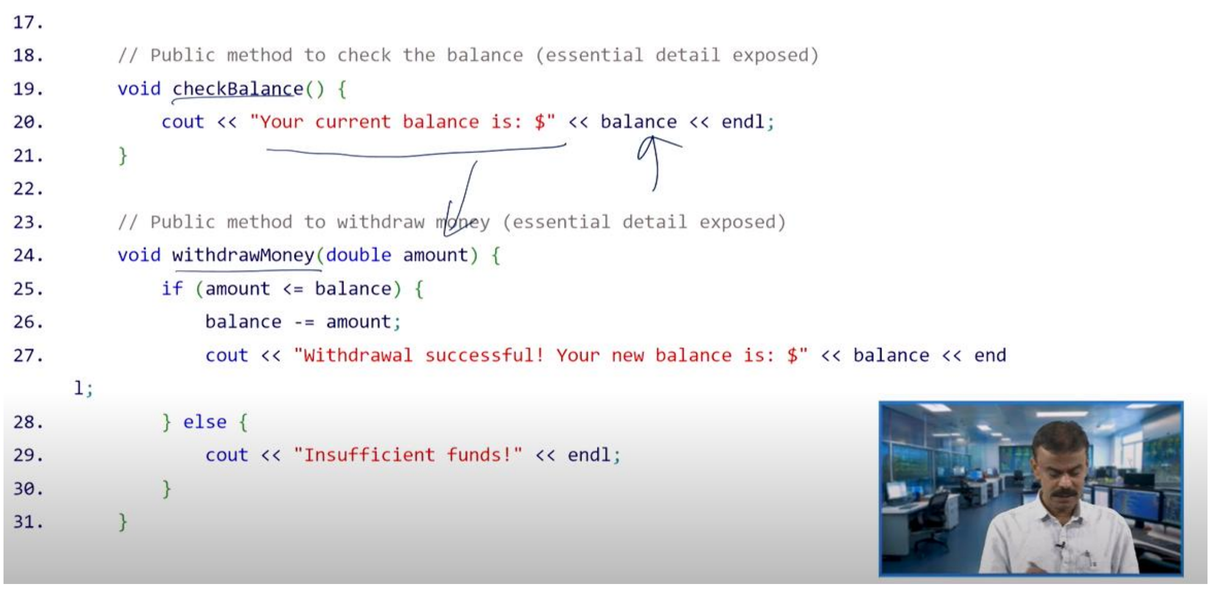
```cpp
#include <iostream>
using namespace std;

// BankAccount class that provides a simplified interface
class BankAccount {
private:
    // Private data members (hidden from the user)
    string accountNumber;
    double balance;

public:
    // Constructor to initialize account details
    BankAccount(string accNumber, double initialBalance) {
        accountNumber = accNumber;
        balance = initialBalance;
    }
```

So, that means when I am having the abstract class, implementation details are hidden. So, this is the concept called data abstraction. So, what we will see is we will look at one example with the help of member data, right? With the help of member data, we will see one example, alright. So, assume that here is the class BankAccount, alright.

So, here I am hiding this data, these two data, right, outside the board, alright. String accountNumber, double balance. So, initially, we will look at only the data members.

So, later we will talk about the abstract class and data. So, when I am defining the class inside the base class, what do you mean by an abstract class?

So, all these we will see slightly later. So, right now, these two are the data members or member data, right? We define them as private, right? The access modifier or access specifier is private. So, in the public, we have a member function or, in fact, this is nothing but a constructor.



```
17.
18.        // Public method to check the balance (essential detail exposed)
19.        void checkBalance() {
20.            cout << "Your current balance is: $" << balance << endl;
21.        }
22.
23.        // Public method to withdraw money (essential detail exposed)
24.        void withdrawMoney(double amount) {
25.            if (amount <= balance) {
26.                balance -= amount;
27.                cout << "Withdrawal successful! Your new balance is: $" << balance << end
     l;
28.            } else {
29.                cout << "Insufficient funds!" << endl;
30.            }
31.        }
```

We have a constructor, bankAccount. So, it is a parameterized constructor; you have an account number and then you have an initial balance. So, accountNumber, whatever we are passing, it will be assigned to, right? accNumber will be assigned to accountNumber. And initialBalance will be assigned to balance, right?

So, this is your parameterized constructor. Two arguments are there. So, this is called the parameterized constructor. And you have a member function called checkBalance. So, checkBalance is returning, right?

So, this is your statement. Your current balance is in dollars. And then, right, the balance. Now, line number 24, we have void withdrawMoney. All right.

Void withdrawMoney. And then you have one argument. Right. And then we are checking the condition. If the amount is less than or equal to the balance.

Right. So, the balance is equal to the balance minus the amount. So, your withdrawal is successful. And what is the balance? Right.

Otherwise, it is going to say it has insufficient funds. So another method called void depositMoney. So balance is equal to balance plus sum. So whatever you are depositing, after depositing, what will be the balance, right? So now, let us say in the main program, how we are going to demonstrate the data abstraction concept, right?

So here we have two private member data. One is accountNumber. Another one is balance, right? So now if you look at the main program, I have a constructor, right? So in fact, I have created an object, instantiated an object.

And then passing two values, right? I am passing two values. So, this will invoke the constructor, right? So, our constructor over here, I have passed two values. So, one will be accountNumber.

So, that 1, 2, 3, 4, 5, 6, 7, 8, 9 is the accountNumber. And the balance is 500, right? So, that will be assigned to the myAccount object value. So, now myAccount is invoking the checkBalance function. The checkBalance member function.

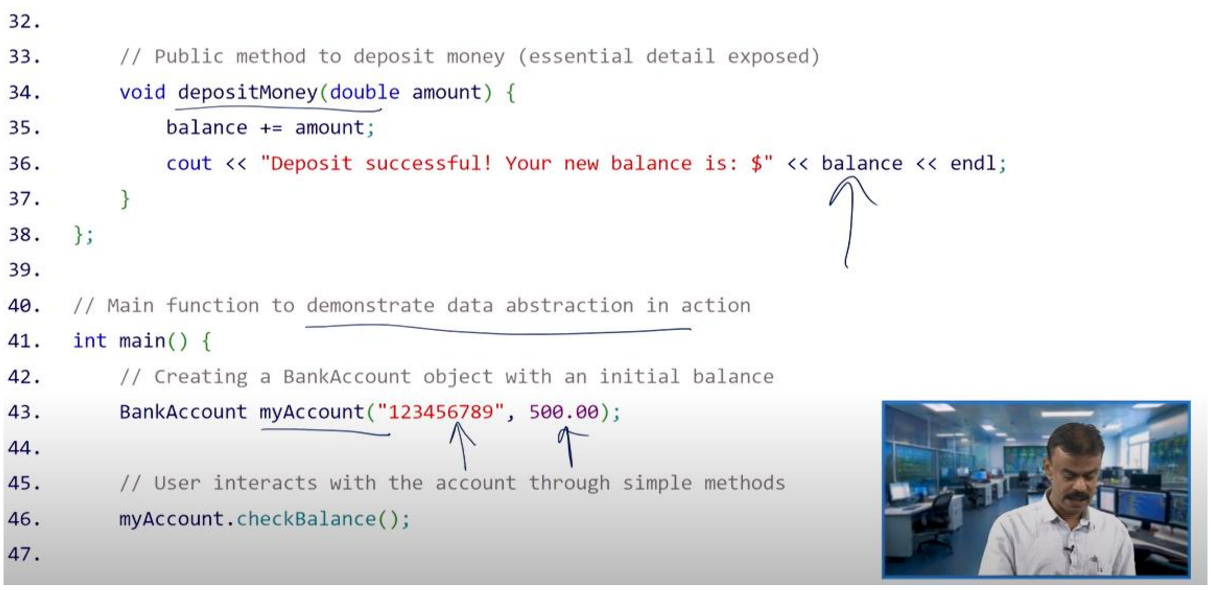
So, checkBalance. So, here you go. Your current balance is in dollars. Right? What is the balance?

500. So, that means if you look at the accountNumber and balance. So, I have written. Right? So, I have written to the outside world.

Right? But whereas outside the world, in the sense of the main program. Okay? So... How can I access those two?

So, indirectly, we are accessing. So, when I call checkBalance, you can see over here. So, this member function can access balance, right? So, when it is accessing balance, you can see that. So, these are all the private data members, right?

So, this function can access. So, 500 is the Balance value for this particular object, my account, when it is invoking checkBalance, right? Because we have passed 500, all right? So, checkBalance, balance will be equal to 500, all right?

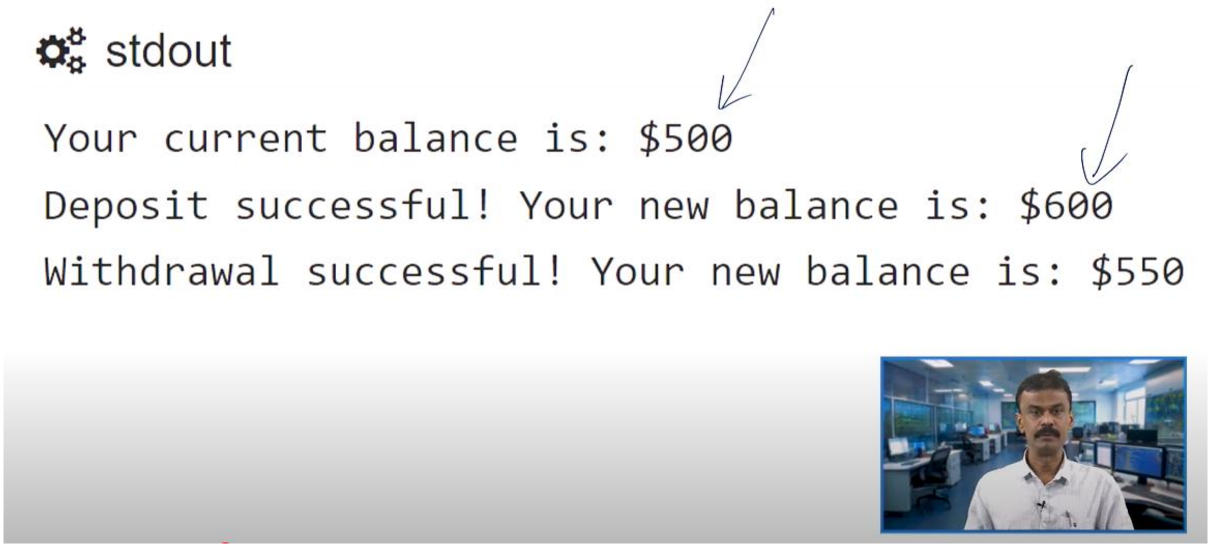


```
    // Public method to deposit money (essential detail exposed)
    void depositMoney(double amount) {
        balance += amount;
        cout << "Deposit successful! Your new balance is: $" << balance << endl;
    }
};

// Main function to demonstrate data abstraction in action
int main() {
    // Creating a BankAccount object with an initial balance
    BankAccount myAccount("123456789", 500.00);

    // User interacts with the account through simple methods
    myAccount.checkBalance();

```

Now, myAccount is invoking the deposit money function. So, for deposit money, you are passing 100.00, right? Right. So, when you are passing 100.00, right, here you go, deposit money, right. So, when you are passing 100.00, the balance is equal to the balance plus the amount, right? The amount you are passing.

So, 500 plus 100, right, you will have 600, right. So, now the balance will be equal to 600, correct? So now myAccount is invoking withdraw, right? So here, the balance is equal to the balance plus the amount.

So, in fact, I am able to access the balance again. That is the meaning, right? So now this is invoking withdrawMoney, 50, right? Dollars we are withdrawing, right? So when you are withdrawing 50, so here you go.



stdout

```
Your current balance is: $500
Deposit successful! Your new balance is: $600
Withdrawal successful! Your new balance is: $550
```

If the amount is less than or equal to the balance. So, what is the balance? 600, 50. 50 is less than or equal to 600, true. So, the balance will be reduced by 600 minus 50. So, it will be 550, right?

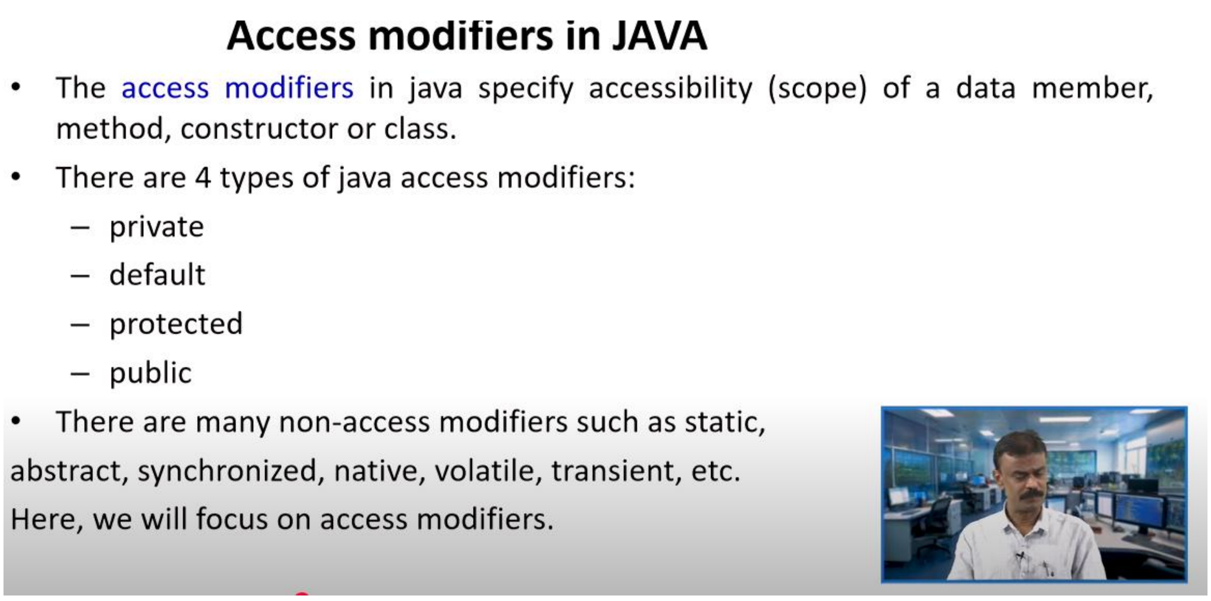
So, your balance will be printed as 550. Suppose I am trying to withdraw 1000 dollars. So, then it is insufficient funds because your balance is 600 dollars, right? So, when I run the code, I have to get the output like this, right? So, this is the complete code when I run it.

So, $500 initially, then the balance is 600, and the withdrawal should be printed, right? The withdrawal is successful, that is what I said, and then 550 will be the remaining balance. So, initially 500, then it became 600, right? And then it became 550. So, these are the outputs we are getting: 1, 2, and then 3, fine. So, in fact, when we run the code, you are getting your current balance as 500. So, then after the deposit, it became 600.

Then, after withdrawal, it is 550. So now, in this program, what we learned, if you look at the private member data, right? So, in fact, we pass accountNumber also, right? accountNumber, because I have talked about only the balance, right? So, we are passing the accountNumber from outside, in fact, in the main.

So, when it is accessing, myAccount is accessing. In fact, the constructor bankAccount is accessing. So, then whatever you have passed will be assigned to accountNumber. So, I can access now because this constructor can access accountNumber and balance. So, that means we have hidden this from the outside world.

Outside world means outside this class. So, I hope it is clear, the concept of data abstraction, right, in terms of member data or data members. So now, in C++, we have already seen, right, the private access specifier. I specified it as a specifier, right?



Access modifiers in JAVA

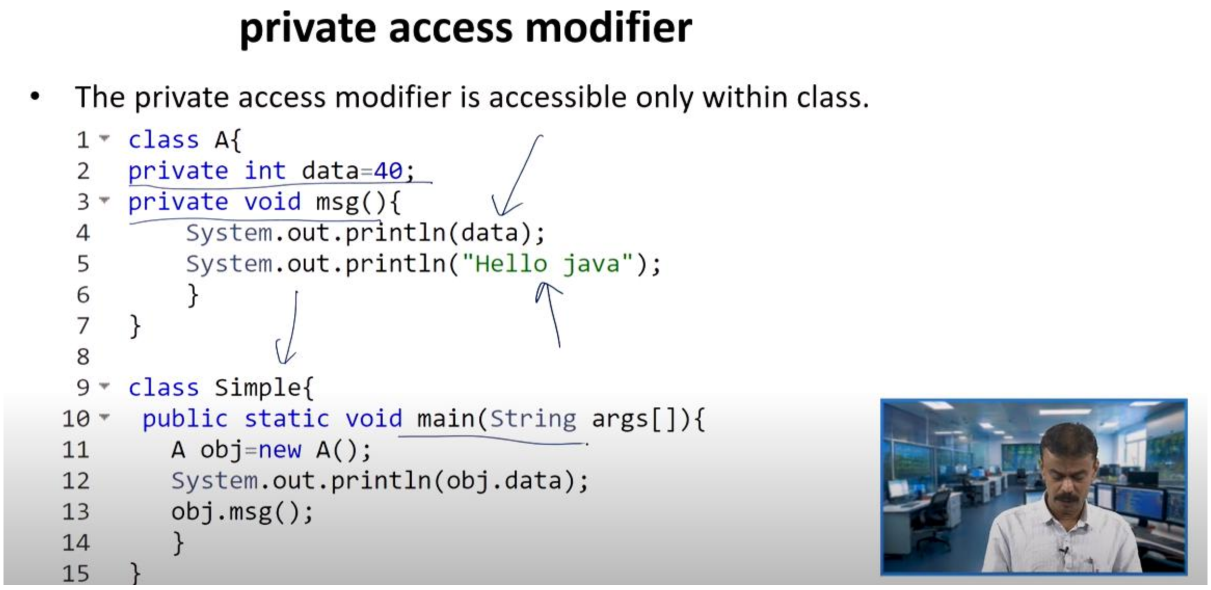
- The access modifiers in java specify accessibility (scope) of a data member, method, constructor or class.
- There are 4 types of java access modifiers:
  - private
  - default
  - protected
  - public
- There are many non-access modifiers such as static, abstract, synchronized, native, volatile, transient, etc. Here, we will focus on access modifiers.

So, access modifier, access specifier, I mean, all are similar, right? So now, suppose I want to consider, what is the scope of a data member? What is the scope of a method or a member function? What is the scope of the constructor? So, can the constructor, can you have a private constructor?

Yes, you can have. Or even a class, right? You can have a private class, public class, right? So now, we have already seen, right, in C++. And in Java also, you have private member data, default, right, protected, and public, right?

So, the access modifiers, you have private, default, protected, and public, right? All right. So, there are several non-access modifiers. Right. The keyword static we had seen.

Abstract, we are going to see. Right. And then, so many. So, they are all called non-access modifiers. Whereas, in this case, what we are going to see in this chapter, we are going to see access modifiers.



private access modifier

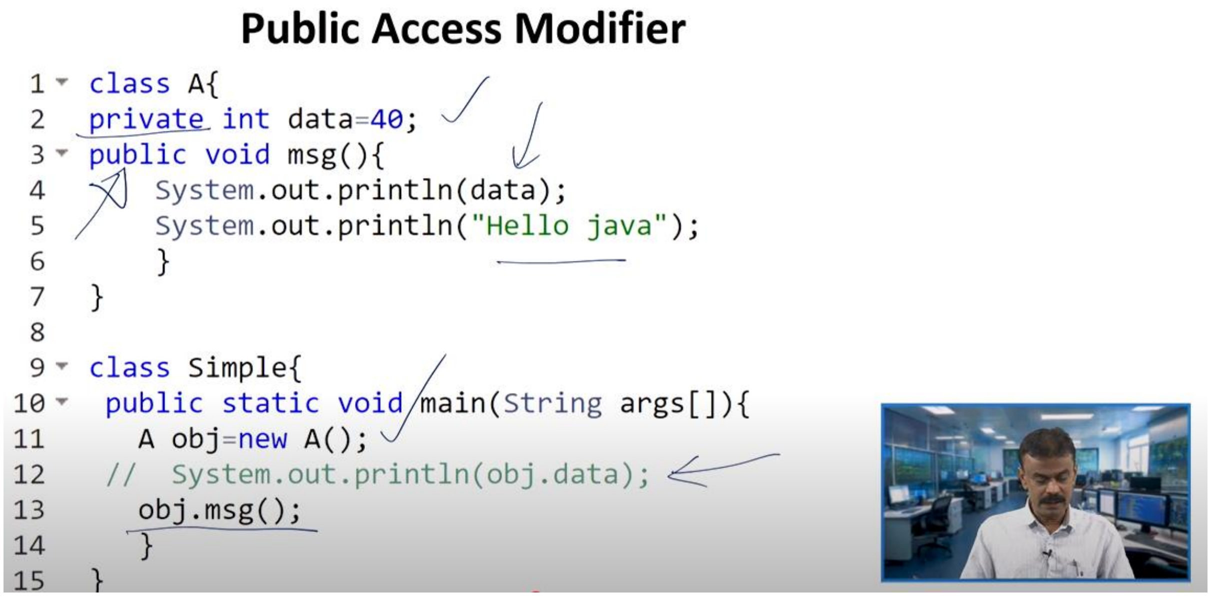
- The private access modifier is accessible only within class.

```
class A{
private int data=40;
private void msg(){
    System.out.println(data);
    System.out.println("Hello java");
    }
}

class Simple{
 public static void main(String args[]){
   A obj=new A();
   System.out.println(obj.data);
   obj.msg();
   }
}
```

Right. So, suppose you consider this particular code in Java. We have class A, and then here I have a private member data. The data is 40. And then, you will have the private method called message.

So, we are expecting System.out.println data and System.out.println hello Java, right? So, now we will go to the driver class, right? We will go to the driver class. You have a main program. I am creating an object, right?

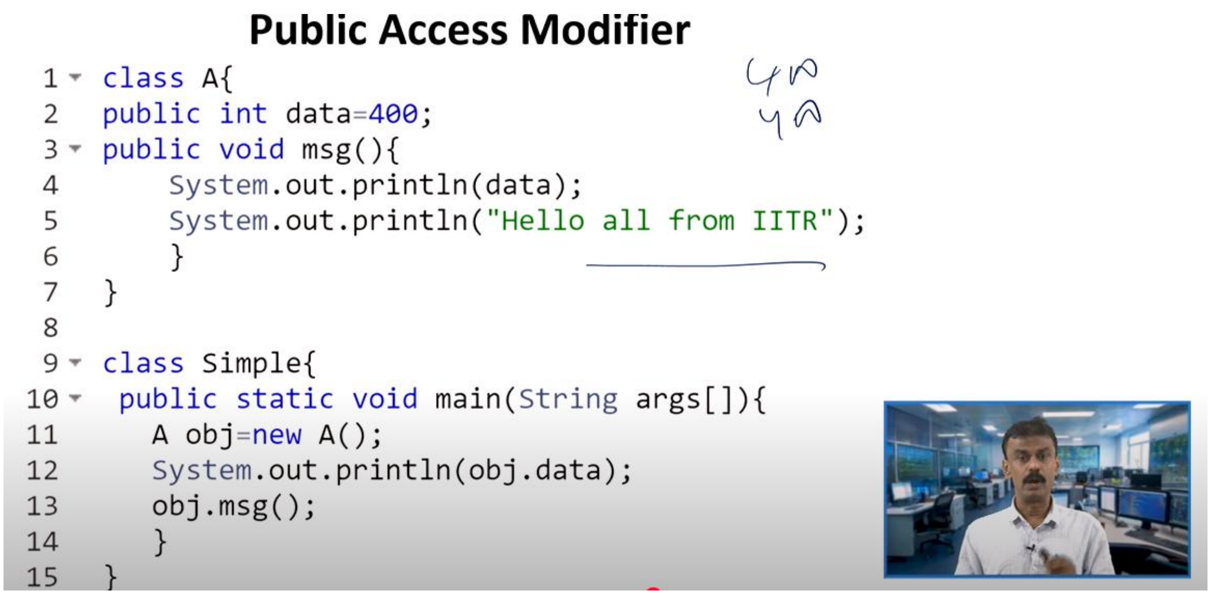
Let us assume I am creating an object under class A, right? And then you have memory allocation, and then you have a constructor, right? So now I am trying to print obj.data and obj.message. That means obj is invoking data, and obj is invoking message. So we know what will happen because this is a private member data and a data member or a private method.



Public Access Modifier

```
class A{
private int data=40;
public void msg(){
    System.out.println(data);
    System.out.println("Hello java");
    }
}

class Simple{
 public static void main(String args[]){
   A obj=new A();
 //  System.out.println(obj.data);
   obj.msg();
   }
}
```

So in that case, what will happen, right? You should get an error, right? So the reason is I cannot access these two outside the class, right? So I am trying to create an object, right? So that is not a problem, but when this object is invoking the data, you will get an error because it is private, right? It is a private access to the class. I cannot use it. In fact, the same concept we studied in C++, and this is an example in the case of Java. So how to rectify this? So make it a private data member, right? Exactly, I am relating



Public Access Modifier

```
1   class A{
2   public int data=400;
3   public void msg(){
4       System.out.println(data);
5       System.out.println("Hello all from IITR");
6       }
7   }
8
9   class Simple{
10   public static void main(String args[]){
11     A obj=new A();
12     System.out.println(obj.data);
13     obj.msg();
14     }
15  }
```

the concept that we have studied: data abstraction. So here you have private int data that is equal to 40, and here you have public void message, right. So in the void message, you are printing data and hello Java. So one, you are making private. Let it be private. I am not changing in the previous program, but here, let us assume we use public. In the previous program, we use that also private. Right.

Now it has become public. I am creating an object. Right. So let us create an object OBJ. Right.

So now, yes, of course, this will give an error. Right. So I am hiding this. Right. Put a comment statement.

And let us assume OBJ is invoking MSG. Right. OBJ is invoking the method MSG. Yes. Now it is possible because it is public.

Right. According to the third line, it is public. Method, so you have to print out a statement, right? Print statement. So here, you have line number four. You are trying to print the data. So what is the output you have to get? 40. You have to get, and the next one, System.out.println("Hello Java"), so that will also be printed. So when I run the code, right, I have to get the output 40 and "Hello Java". All right, so now You can see exactly what we had seen in C++ and in Java as well. When I use in line number 3 public, so I can run the code.

So now I am hiding this data, right? So that is private. It cannot be accessed. So if I just remove, let us say, this 1, it will give an error, right? So in line number 3, I am making it public.

So now obj.message, so it is going inside. Right. So message, it can access, I mean to say. So then data will be printed. Okay.

So, this is the public access modifier. Right. So, here you go. So, suppose if I make both public. Right.

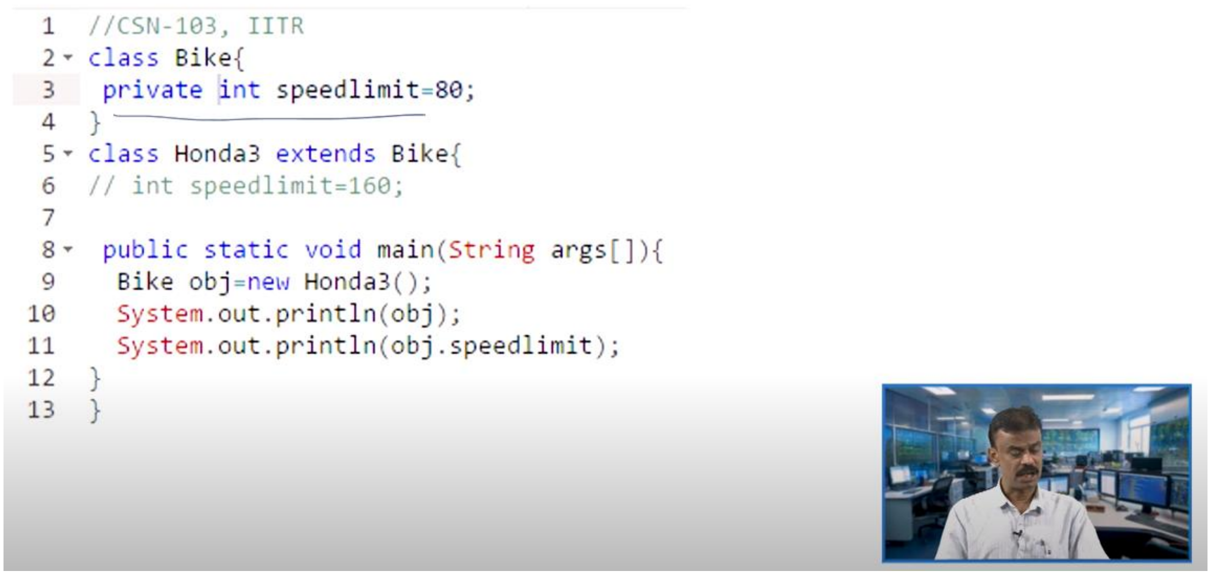
So, in the previous example, it was not working. Line number 12 was not working. So, when I ask, suppose obj.data, that has to print the data. What do you have to do? Make line number 2 also public.

Right. So, when you do that, you can see over here data is 400, and then you have a message. Again, you have to print 400. 400 will print two times, and then this will be printed. Same program.

The only thing I have removed is line number 12. Now I can remove line number 12 because, if you look at line number 2, your data, int data equal to 400, is public. Okay. So now, when I run the code, you can see I will get the output like this. Right?

So, 400, 400, and hello all from IITR. I hope it is clear for everyone. So, now protected, right? So, protected access modifier. So, let us consider this example: class Bike, right?

I have one private member data, right? And then I am using inheritance: class Honda3, which is a subclass, and the superclass is Bike, extends Bike. Right. So you have void main over here, and then you have an object obj under the class Bike. Right.



```
 1  //CSN-103, IITR
 2  class Bike{
 3   private int speedlimit=80;
 4  }
 5  class Honda3 extends Bike{
 6  // int speedlimit=160;
 7
 8   public static void main(String args[]){
 9    Bike obj=new Honda3();
10    System.out.println(obj);
11    System.out.println(obj.speedlimit);
12  }
13  }
```

And then the constructor is Honda3. Right. The constructor is Honda3. So now, System.out.println, I am trying to print obj. That means I am trying to print the identity hashmap.

It is exactly equivalent to the address in C++. All right. Next one is obj.speedlimit. Right. I am expecting it will print 80.

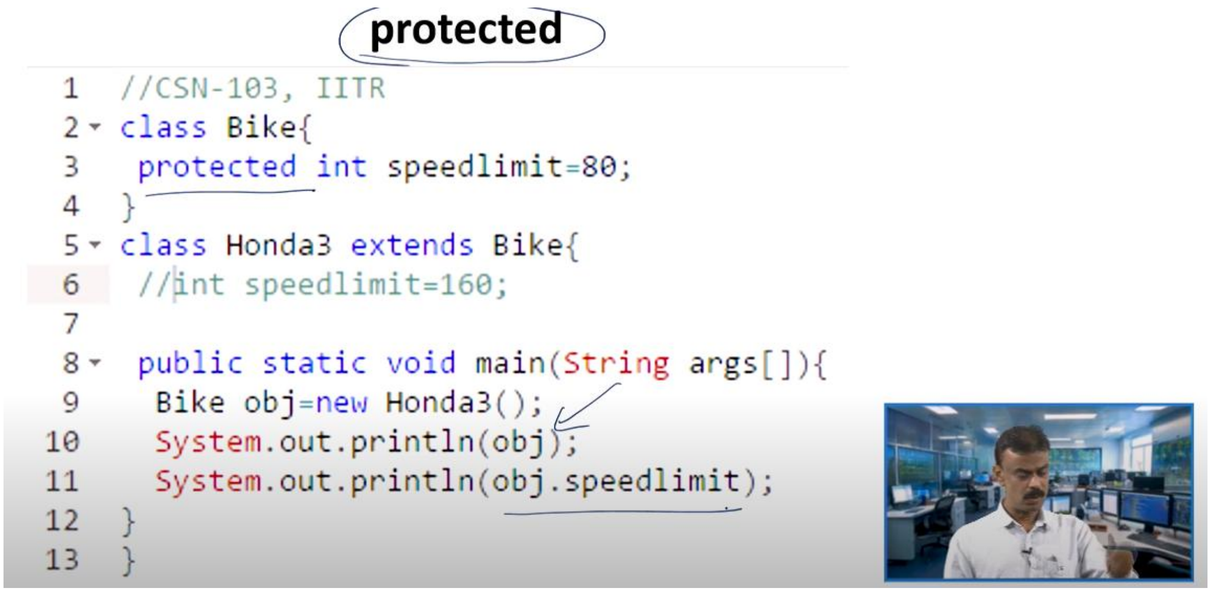
But unfortunately. Right. So you are getting the error. Right. So the reason is, if you look at line number 3, it is the private access.

Right. It is a private data member. So you cannot access it. You cannot access it from outside. Right.

So now, suppose I put, let us say, protected. Right. The same program. Assume that I put protected. We are talking about the protected access modifier.

The same program, I am just renaming it as protected. Instead of public, we have protected. The rest of the code is the same. So now, you are trying to print the identity hash map, like exactly equivalent to the address in C++. And here you go, obj.speedlimit.

So, now this will access. So, now there is a problem. That is the property of protected. Exactly what we have seen in C++. So, in fact, when you are doing publicly, protected will also become protected.



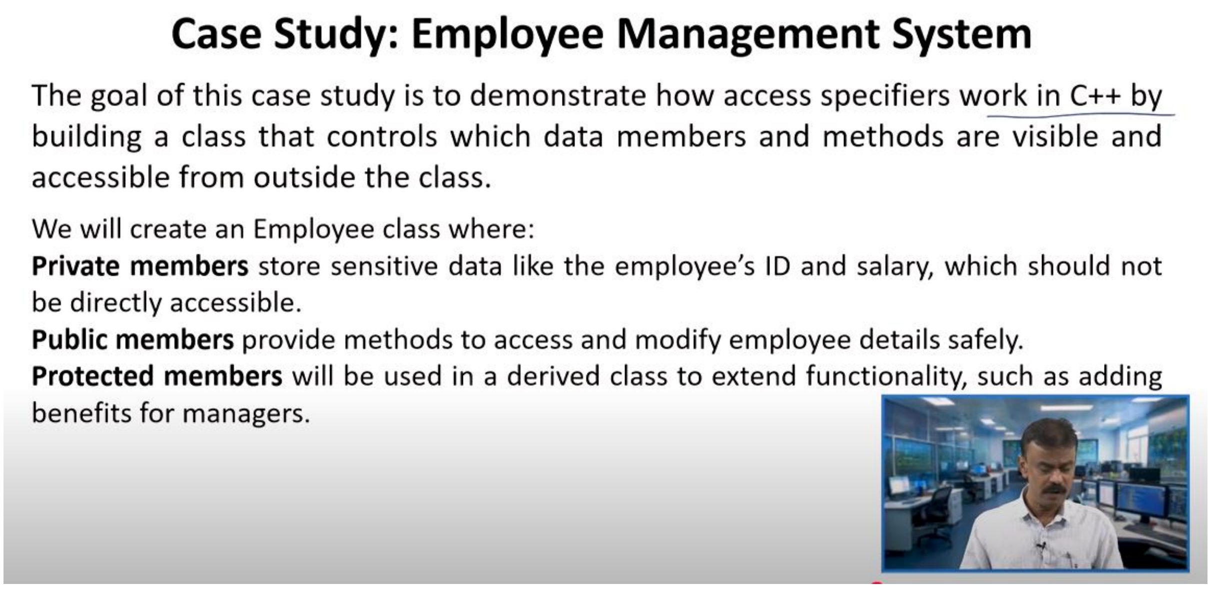
protected

```
//CSN-103, IITR
class Bike{
 protected int speedlimit=80;
}
class Honda3 extends Bike{
 //int speedlimit=160;

 public static void main(String args[]){
  Bike obj=new Honda3();
  System.out.println(obj);
  System.out.println(obj.speedlimit);
}
}
```

So, in that concept, now when it is trying to print obj, it will be printing. And System.out.println speedlimit. So, I will get 80. So, one is address, or I can call it as identity hashmap. And another one is the speed is 80.

So, these will be printed. So, when I run the code, you can see that, right? So, because you have the constructor at Honda3. So, Honda3 at this is nothing but the identity hashmap. The one you are seeing over here is an identity hashmap.

And then we are printing 80, right? So, this is what we are expecting: 80. So, if you look, we are getting 80. So, I hope it is clear in the case of protected as well. So now, to know more about this access specifier or access modifier.



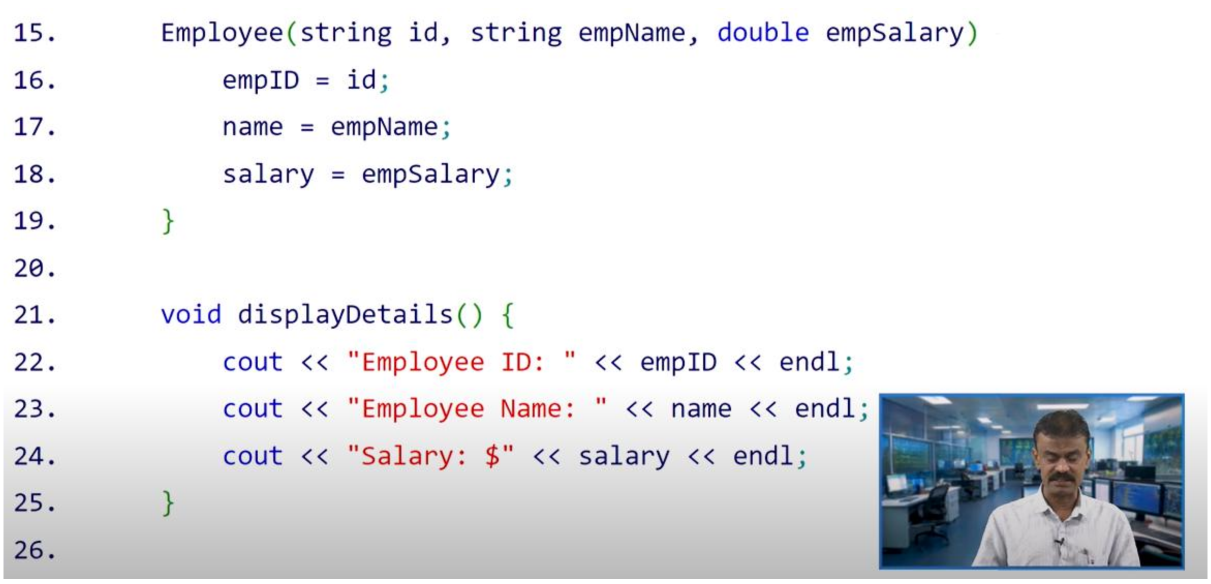
Case Study: Employee Management System

The goal of this case study is to demonstrate how access specifiers work in C++ by building a class that controls which data members and methods are visible and accessible from outside the class.

We will create an Employee class where:
Private members store sensitive data like the employee's ID and salary, which should not be directly accessible.
Public members provide methods to access and modify employee details safely.
Protected members will be used in a derived class to extend functionality, such as adding benefits for managers.

So, let us take this case study in C++. We have already done it. That is why I am not doing this private, protected, and public in C++ again. So, we have already seen, if you look at lecture number 10 and lecture number 11, we have already talked about that. That is why we have seen these access modifiers in Java.

So now, what we will do is, whatever the concepts are, they are almost the same. In fact, they are exactly the same. So in that case, what we can do is, we will take this case study and we will have one example based on private, public, and protected. So, let us consider the employee management system. So that means I am going to use the private member data, public, and the protected members.



```
    Employee(string id, string empName, double empSalary)
        empID = id;
        name = empName;
        salary = empSalary;
    }

    void displayDetails() {
        cout << "Employee ID: " << empID << endl;
        cout << "Employee Name: " << name << endl;
        cout << "Salary: $" << salary << endl;
    }

```

So, for example, if I have an employee ID and salary, these data should be secured. So, I put it as private members and then public members. Right. So, let us say some modify function. Right.

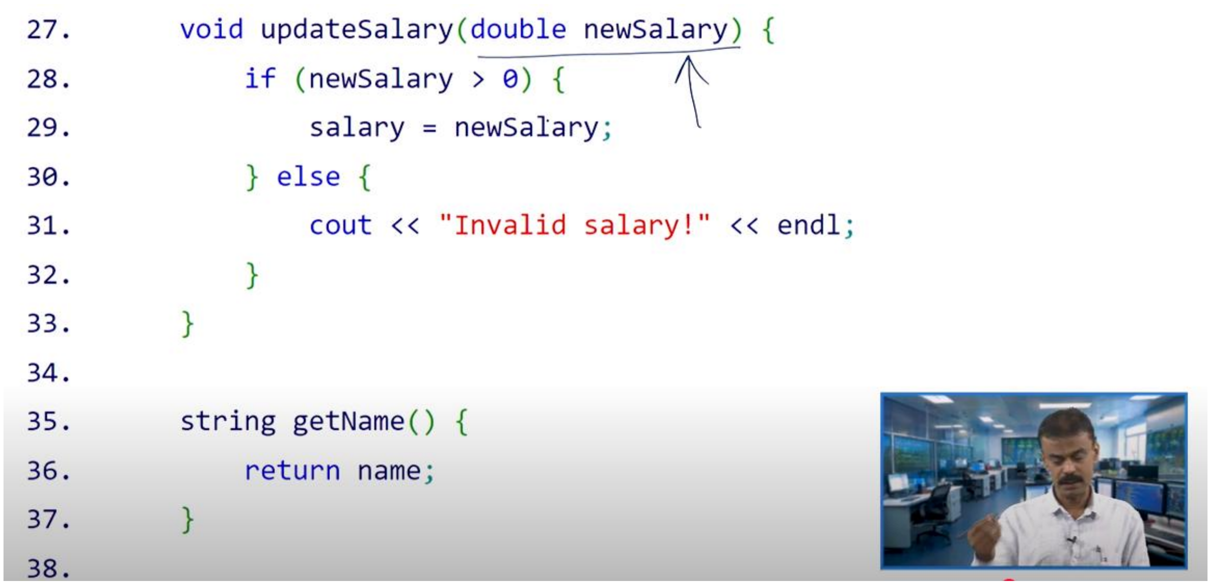
Modify member function that I am going to use. Right. So, these all go under public members. And similarly, we have certain protected members. Correct.

So, we will see how we are going to have the employee management system. So, from this case study, you can further understand how private, public, and protected work. So, let us consider class employee. So, let us consider class employee. So, you have private member data, employee ID, and salary.

So, the employee ID and salary. So, in fact, these should be hidden. And you have a protected data member called name. So, I am using all here: private, protected. And you have

Employee constructor. The employee constructor is a three-argument constructor, right? The employee is a three-argument constructor. ID is assigned to employee ID. Employee name is assigned to name, and employee salary is assigned to salary.

And then the void display, I am displaying all the details, right? So, I am displaying all the details. And here you have the updateSalary, correct? So, here you have the updateSalary. So,



```
    void updateSalary(double newSalary) {
        if (newSalary > 0) {
            salary = newSalary;
        } else {
            cout << "Invalid salary!" << endl;
        }
    }

    string getName() {
        return name;
    }

```

So here you have updateSalary, and you have one argument called newSalary. One argument. If newSalary is greater than salary, then salary will be newSalary. That means you are updating. You are updating this salary.

And string getName, you are returning the name. And you have one member function called getSalary. So getSalary will return salary. And you have class Manager, which is the derived class. So, publicly inheriting Employee, right?

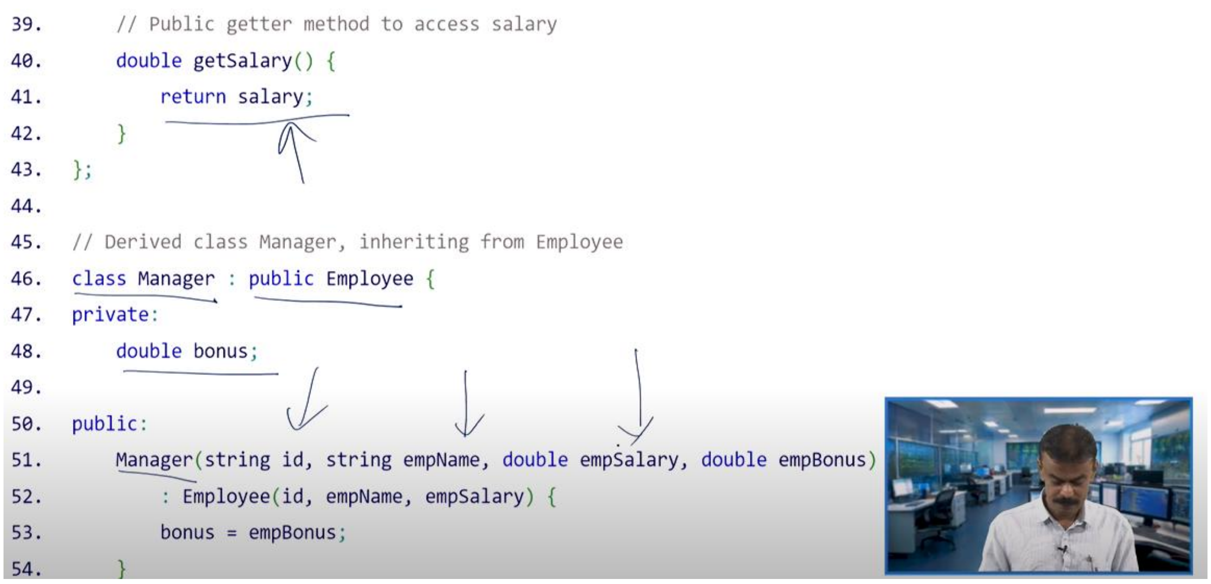
Which is publicly inheriting Employee. And here you have private member data called bonus, right? And then here you have the constructor. So, the constructor has ID, employee name, employee salary, right? And employee bonus, right?

So, where in the base class, This is nothing but the base class constructor. So, the base class constructor has ID, employee name, and employee salary. That means, in the base class constructor, whatever you are passing, so that will be assigned to ID, employee name, and employee salary.

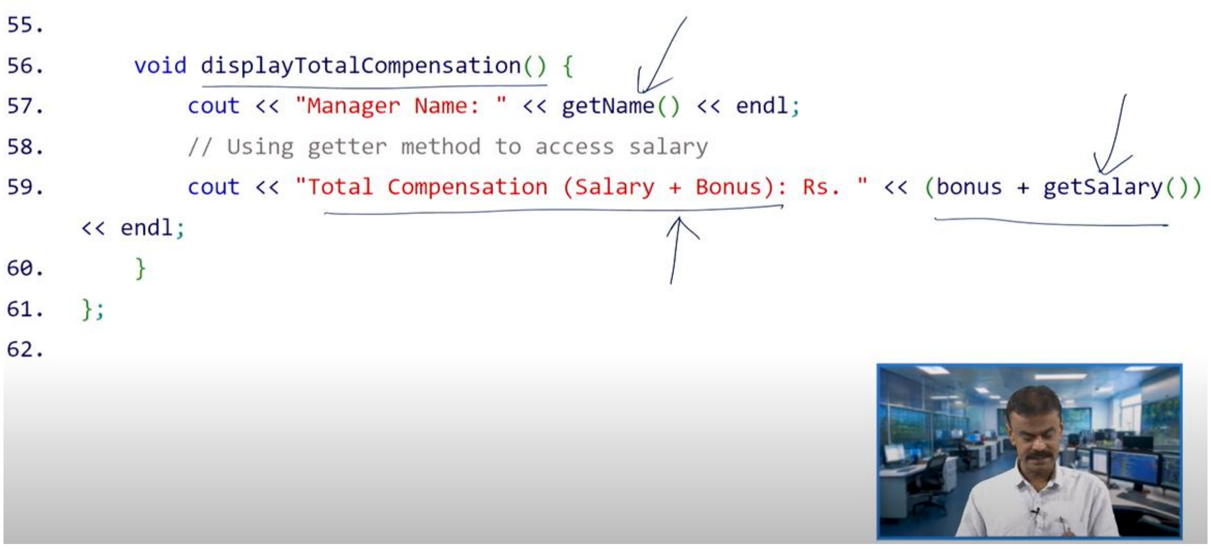
That is the meaning; we know this. And in line number 53, you have employee bonus, which is assigned to bonus. So here, you have void displayTotalCompensation. Right. So, you are displaying total compensation.

All right. So, cout manager name. So, you are getting it; it's nothing but the getName. And here, the total compensation, which is salary plus bonus. Right.

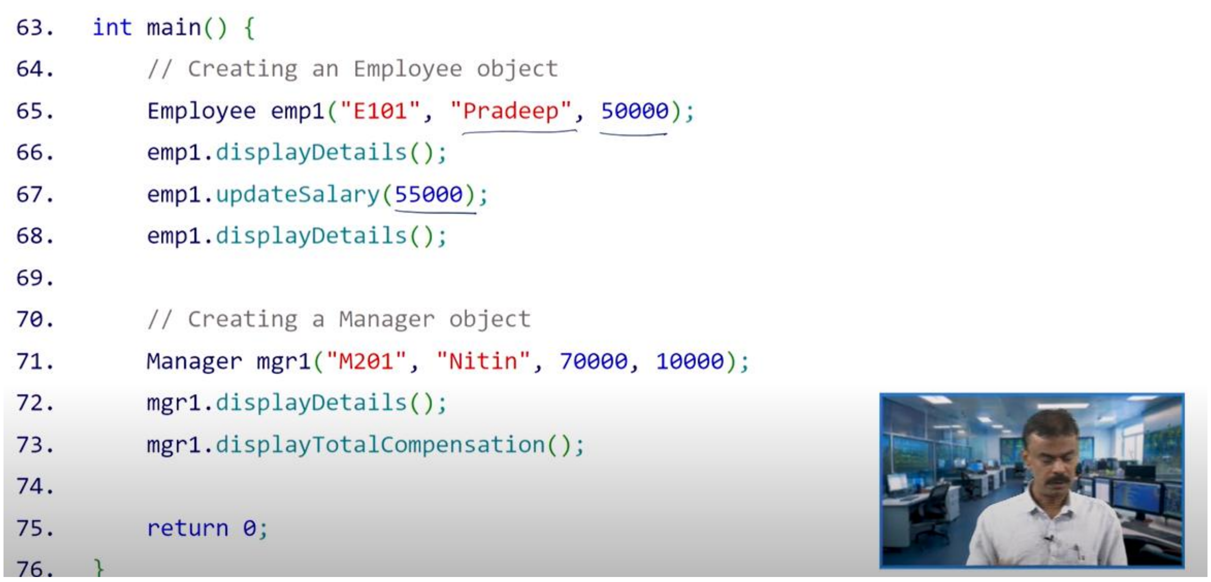
Which is salary plus bonus. That means you are right. Bonus and getSalary function will be called, and getSalary is nothing but it is returning salary. It is nothing but bonus plus salary. OK.



```
39.     // Public getter method to access salary
40.     double getSalary() {
41.         return salary;
42.     }
43. };
44.
45. // Derived class Manager, inheriting from Employee
46. class Manager : public Employee {
47. private:
48.     double bonus;
49.
50. public:
51.     Manager(string id, string empName, double empSalary, double empBonus)
52.         : Employee(id, empName, empSalary) {
53.         bonus = empBonus;
54.     }
```



```
55.
56.     void displayTotalCompensation() {
57.         cout << "Manager Name: " << getName() << endl;
58.         // Using getter method to access salary
59.         cout << "Total Compensation (Salary + Bonus): Rs. " << (bonus + getSalary())
    << endl;
60.     }
61. };
62.
```



```
int main() {
    // Creating an Employee object
    Employee emp1("E101", "Pradeep", 50000);
    emp1.displayDetails();
    emp1.updateSalary(55000);
    emp1.displayDetails();

    // Creating a Manager object
    Manager mgr1("M201", "Nitin", 70000, 10000);
    mgr1.displayDetails();
    mgr1.displayTotalCompensation();

    return 0;
}
```

So now I have. All right. Two employees, right? One is, let us say, employee emp1, right? So, emp1, which is E101, name is Pradeep, and the salary is 50,000, right?

Employee 1 dot display details, right? So, this is displaying the detail, right? So, this will display details. First one is emp1 display details, and emp1 is updating the salary, right? So, when you are updating the salary, it is, let us say, 55,000, that means 50,000.

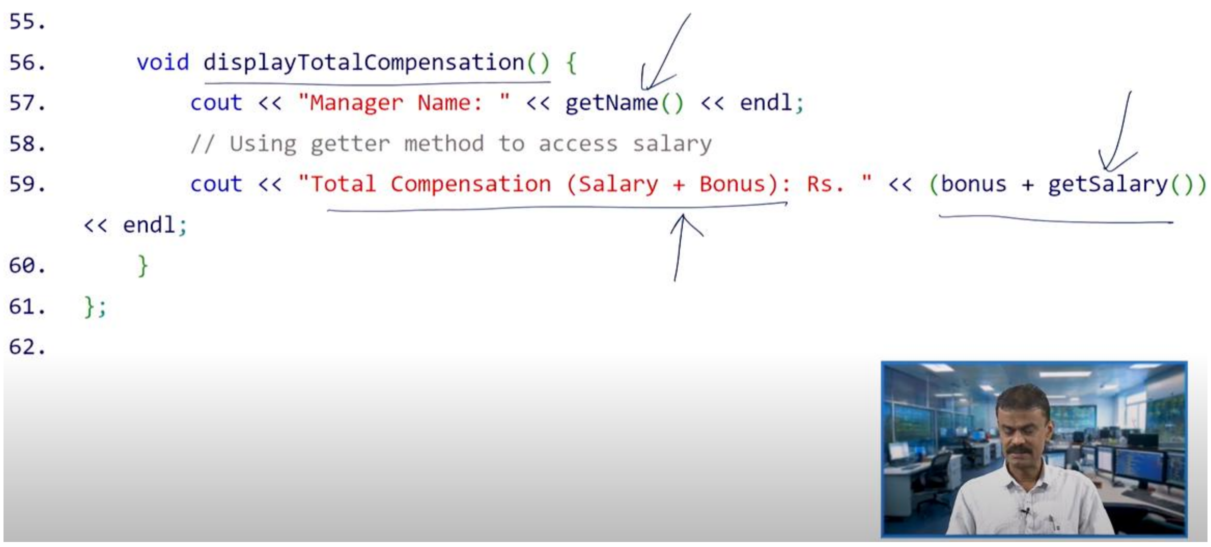
It will be 55,000. So, when emp1 is invoking displayDetails, right? When emp1 is invoking displayDetails, these will be printed, right? Employee ID, employee name, and salary. So, it is in fact in terms of rupees, right?

Anyway, I have taken care in the further case. So, right now, consider this as rupees, okay? Because whatever you print, it will be printed, right? Because in the main program, I have changed it to rupees. So, consider the salary in rupees, right?

So, these will be printed. And this is the update. So, the updated salary is 55,000 now. So, once you update the salary, it is calling updateSalary. Here you go.

So, when it is calling update salary, you are passing newSalary, right? So, when the newSalary is greater than zero, right? So, newSalary will be assigned to salary. So, 55,000 will be assigned to salary. Now,

Obviously, it has been updated. So, when you are printing again, assume that you are printing details again, now salary will be 55,000. And the second object I am creating under manager mgr1, right, under manager, passing M201 Nitin 70,000 and 10,000, right. So, what are all the things in the case of manager you



```

    void displayTotalCompensation() {
        cout << "Manager Name: " << getName() << endl;
        // Using getter method to access salary
        cout << "Total Compensation (Salary + Bonus): Rs. " << (bonus + getSalary())
  << endl;
    }
};

```

have, right. So, this ID, employee name, employee salary, and here you have bonus.

So, that means your employee details will be assigned to these values, right? ID, employee name, and employee salary. So, the first three, ID, employee name, and employee salary, right? And then here you go. So, it is mgr1 invoking displayDetails.

Yes, it is the derived class, right? So, what exactly do you want? displayDetails. So, displayDetails will print. Here you go, right?

So, displayDetails, whatever you have written, employee ID, name, and salary will be printed, all right? And then displayTotalCompensation. So, mgr1 is invoking displayTotalCompensation. So, when it is invoking displayTotalCompensation, with salary, manager, then constructor, yeah, displayTotalCompensation, it is printing, cout manager name. getName, right?

So, the getName function, here you go, it will return the name. The getName member function will return the name and total compensation, right? That means total, in fact, it is printing this total compensation, salary plus bonus, in terms of rupees. So, bonus plus getSalary, right? Bonus plus getSalary, bonus, right?

Whatever you have passed, 10,000, we have passed, right? Correct. Plus, getSalary. getSalary will return the salary. So, 70,000 plus 10,000, it will be 80,000.

Right. So, when I am trying to, right, call this. So, 70,000 plus 80,000. In fact, you are trying to print here. So, that will be printed.

And if you look at, right, so when I run the code, I have to get this. So, initially, right, so E101. The name of the employee is Pradeep. The salary is 50,000, right? And here you go.

Employee ID is E101. The employee name is Pradeep. And the salary is 55,000. We have updated 50,000 and then 5,000 after the increment, right? We have increased it.

In fact, we have updated the salary to 55,000. So we are getting the output as 55,000. And the next one is the employee ID of the manager. The employee name is Nitin, and initially, the salary was 70,000, right? And then the bonus is 10,000.

So when the bonus is 10,000, the name of the manager is Nitin, and the total compensation, salary plus bonus, is 80,000, right? So the main idea in this particular case study is that you have to go through it once. You can see that we have used all access modifiers. So, for the first time, we have used all access modifiers, private, right? So we have used the employee ID and salary.

So inside the constructor member function, they all can access this, right? So we carefully use this from outside the class. And similarly, the derived, right? So you have the derived class, how this can access these data, right? For example, the private member data, you cannot access directly, right?

So you may have the getter function and setter function. This we have studied already, right? And then here you have a protected string name. So this was also being accessed, right? And in fact, if you would have public, that will also be accessed, right?

So in fact, yeah. So we have not mentioned any public. Yes. Yeah, it's when I'm not putting anything. Yeah, here you go.

Public getSalary, right? So that means we have used All, in fact, in the case of a derived class, we use public. Here you have used private, right? So, all the access modifiers, right, or all the access specifiers like private, public, and protected.

So, we have used them, and now you know from outside the class how we can access. And when we run the code, it was successfully run, and then we got the output like this, okay? So in today's lecture, we talked about data abstraction in terms of data members and three access modifiers like public, private, and protected that we had seen. So with this, I am concluding today's lecture. Thank you all.