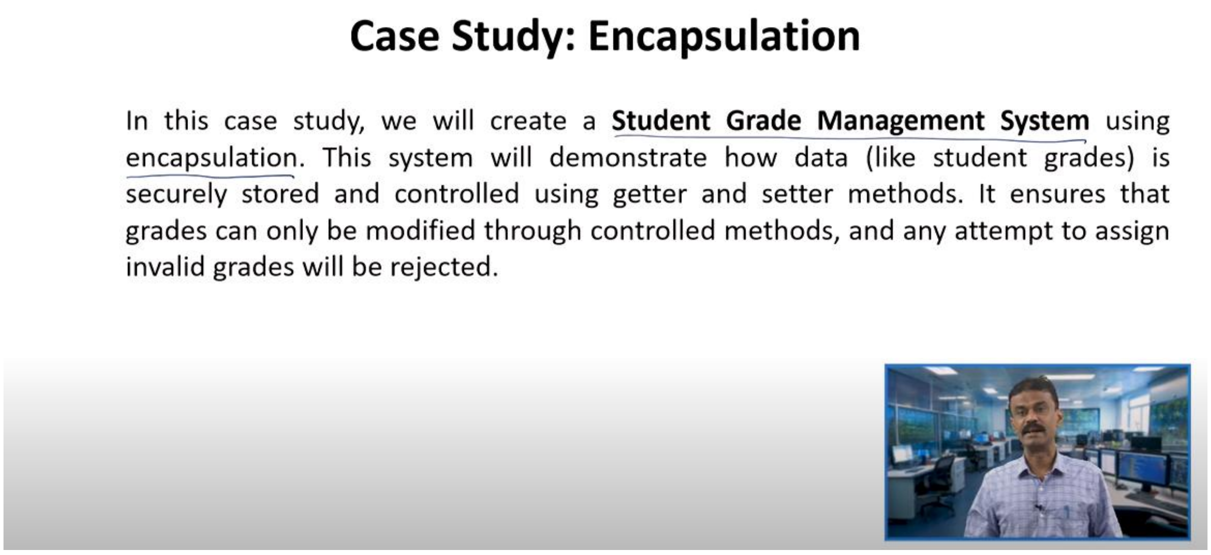
FUNDAMENTALS OF OBJECT ORIENTED PROGRAMMING

Lecture22

Lecture 22: Encapsulation II



Case Study: Encapsulation

In this case study, we will create a **Student Grade Management System** using encapsulation. This system will demonstrate how data (like student grades) is securely stored and controlled using getter and setter methods. It ensures that grades can only be modified through controlled methods, and any attempt to assign invalid grades will be rejected.

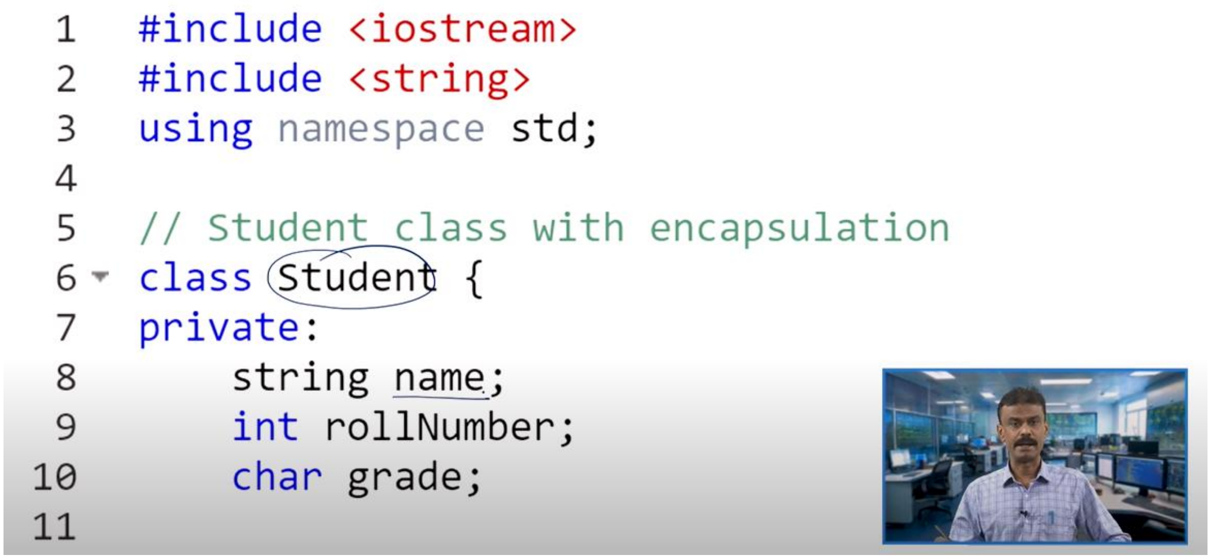
So, welcome to lecture 22. Encapsulation and abstraction. So, in the last class, we talked about encapsulation and data hiding. In fact, we had seen two programs in C++ and one program in Java. So, now we will see two case studies, one in C++ and another one in Java.

So, let us consider the student grade management system. In fact, we had seen this for different concepts. So, now with the help of encapsulation, how are we going to do this? So, this system will demonstrate how certain data, like let us say student grades, is securely stored. So, that means we are going to use it as private member data, and then it will be controlled by getter and setter methods or setter functions.

So, it also ensures that the grades can only be modified through control methods. So, you can change the grade. All right. Or you can say modify the grade or update the grade using the control methods. The control methods are nothing but getter methods and setter methods.

And any attempt to assign invalid grades will be rejected. Right. So this is a case study that we are going to see. Right. And then we can learn what you mean by encapsulation.

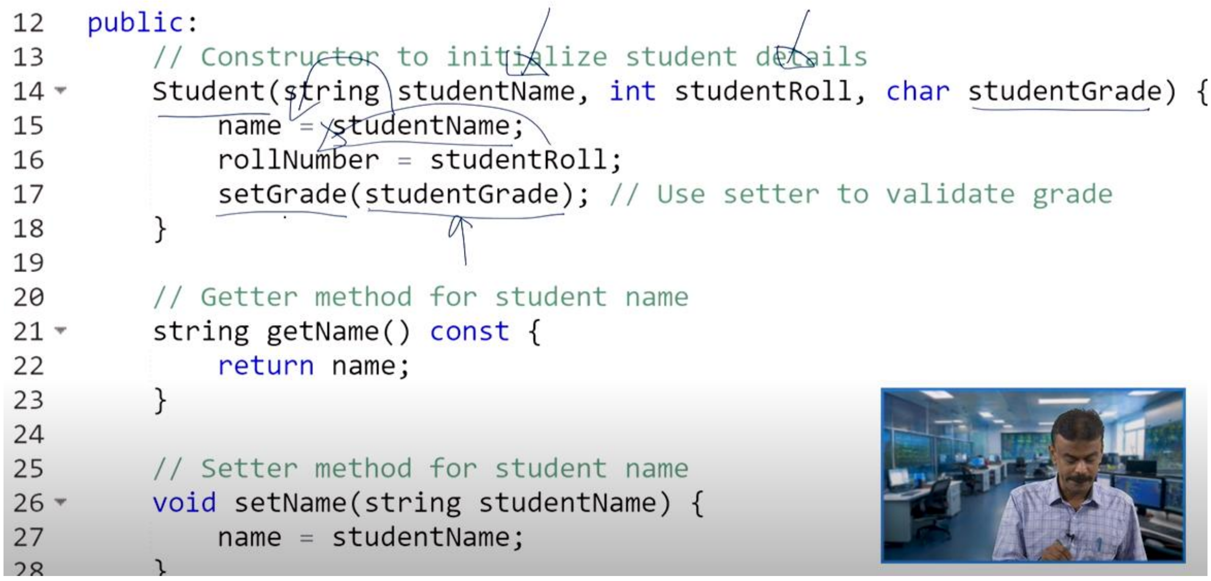
Right. With the case study. So now, as I said in the last class, I said. You should try to use more private member data. So that we are going to see now.



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

// Student class with encapsulation
class Student {
private:
    string name;
    int rollNumber;
    char grade;

```

All right. So let us assume that we have a class student. All right. So class student, you have the private member data. All right.

Private member data. String name. Right. Name of the string, int rollNumber, and the character grade. Right.



```cpp
public:
    // Constructor to initialize student details
    Student(string studentName, int studentRoll, char studentGrade) {
        name = studentName;
        rollNumber = studentRoll;
        setGrade(studentGrade); // Use setter to validate grade
    }

    // Getter method for student name
    string getName() const {
        return name;
    }

    // Setter method for student name
    void setName(string studentName) {
        name = studentName;
```

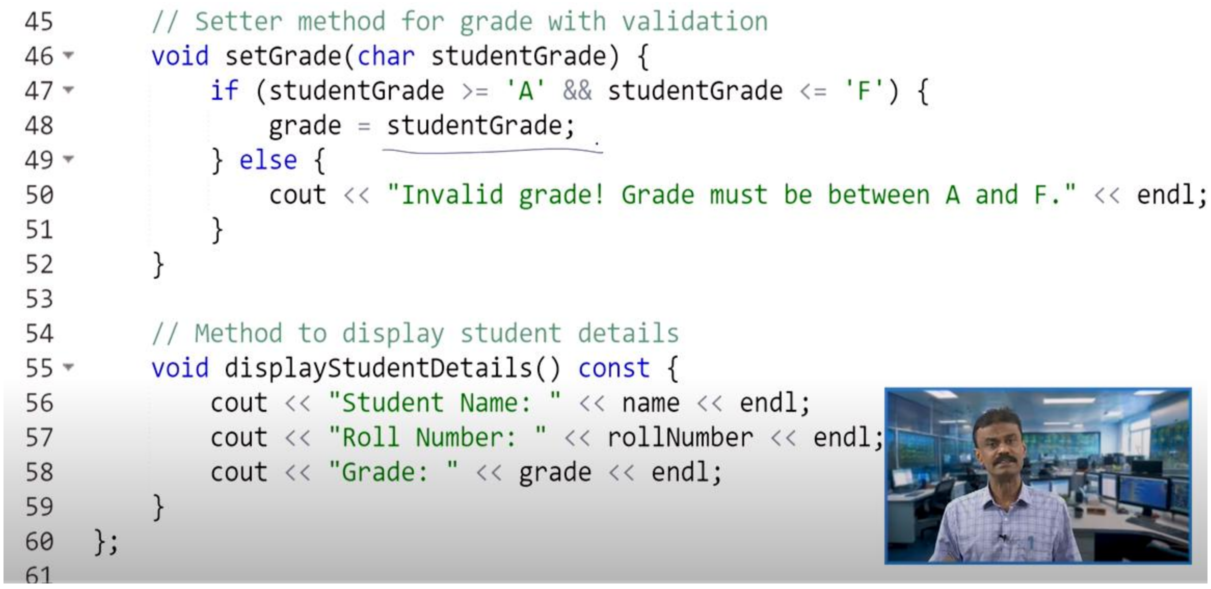
So three private member data we are using. And then. So here you have the constructor. The constructor will. Initialize.

So Student is a constructor. If you look at the name of the class, which is Student. And here you go. The name of the constructor is Student. Right.

So which has studentName, student roll number, and studentGrade. Right. So studentName. So that will be assigned to name. Right.

So student roll number that will be assigned to roll number. Right. And then you have a set grade. Right. Right.

So you are going to use setGrade. Right. So studentGrade that you are passing. So that means you're going to call this particular function setGrade. And in fact, this member function, which is a public member function, which is available over here, line number 46.



```
45      // Setter method for grade with validation
46      void setGrade(char studentGrade) {
47          if (studentGrade >= 'A' && studentGrade <= 'F') {
48              grade = studentGrade;
49          } else {
50              cout << "Invalid grade! Grade must be between A and F." << endl;
51          }
52      }
53
54      // Method to display student details
55      void displayStudentDetails() const {
56          cout << "Student Name: " << name << endl;
57          cout << "Roll Number: " << rollNumber << endl;
58          cout << "Grade: " << grade << endl;
59      }
60  };
61
```

Right. So the setGrade will be called. And if it is greater than or equal to A, Less than or equal to F. So there is. We will see.

We will see. All right. So that means what I mean to say. So this particular function will be called in the constructor. Right.

SetGrade is a member function. Right. And whatever we are passing, studentGrade. Right. So that will be passed in this particular member function.

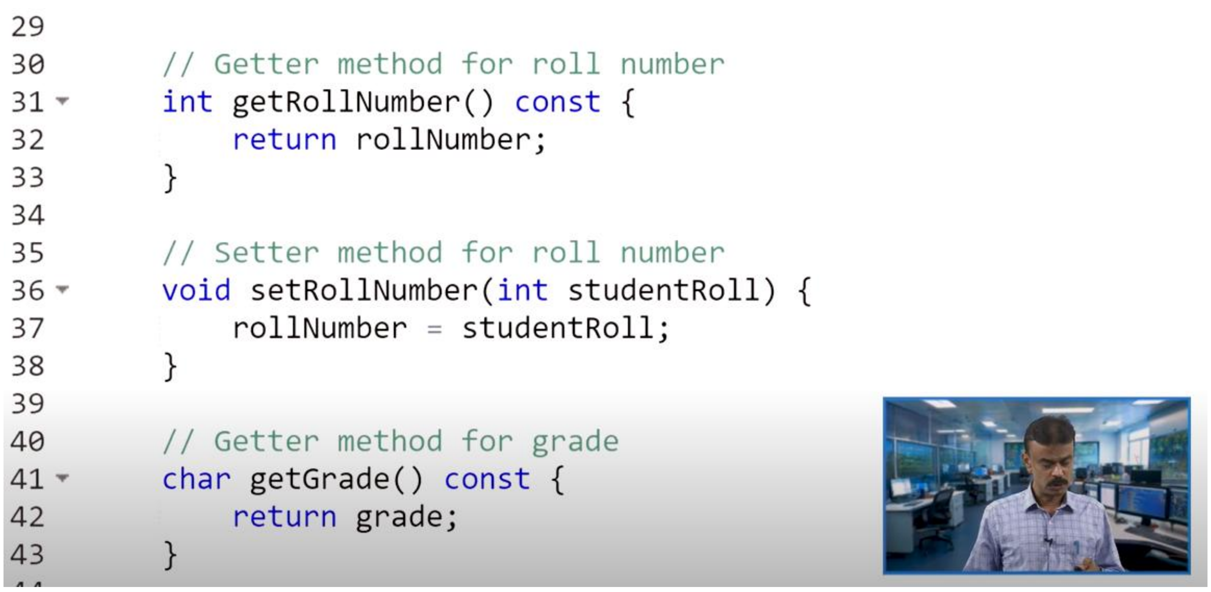
And this will be invoked because it is available in line number 46. Right. So that means it is available inside the class. So, inside the class, any other member function can access other member functions. Right.

So, that we have already studied. So, that we will see. Right. I hope you understand. So, here you have one member function in line number 17.

So, that we have defined in line number 46. So, this will be called. So, this is your constructor. And the next one is the getter method. So, this is your getter method.

Right. So, getName is the name of the member function. All right. And the return type is string. So this is returning the name.

Right. So, a getter function. Right. Another one is obviously a setter function. So you are passing studentName.



```
    // Getter method for roll number
    int getRollNumber() const {
        return rollNumber;
    }

    // Setter method for roll number
    void setRollNumber(int studentRoll) {
        rollNumber = studentRoll;
    }

    // Getter method for grade
    char getGrade() const {
        return grade;
    }
```

StudentName is under string. And StudentName is assigned to name. Right. StudentName is assigned to name. And then you have another getter method, which is getRollNumber.

All right. So, one more getter function. So, you can use as many functions as you want, right? So, you have a getter method for the rollNumber. So, getRollNumber.

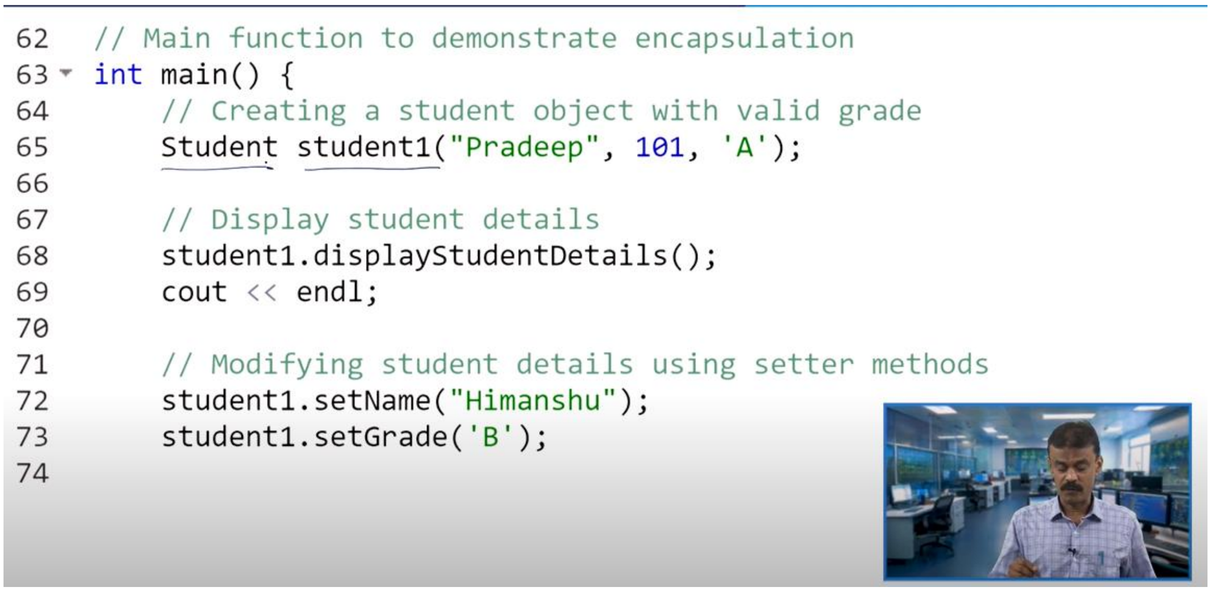
So, getRollNumber whose return type is integer and it will return rollNumber, right? And the next one is setRollNumber. You are passing the student's roll number, right? So, the student's roll number is assigned to rollNumber, all right? The student's roll number is assigned to rollNumber.

So, whose return type is void, right? Right. And you have one getter method for grade, a getter method for grade. So, getGrade. That is a getter method whose return type is character, which is returning grade.

Right. Which is returning a grade. So now you have a setter method. So you have set the grade. You're going to pass the grade in the main.

Right. So that means you have an argument called studentGrade. So, this method, line number 46, we have already seen in the constructor, it is being called, right. So, what exactly is the setGrade doing? So, you can see over here you have studentGrade, right.

So, which is a character. So, now there is some logical operation, right. So, studentGrade greater than or equal to A, the character that you are writing based on the ASCII Whether it is greater than or equal to A or less than or equal to F, right? So, that means you have to pass only A, B, C, D, E, and F, right?



```
62  // Main function to demonstrate encapsulation
63  int main() {
64      // Creating a student object with valid grade
65      Student student1("Pradeep", 101, 'A');
66  
67      // Display student details
68      student1.displayStudentDetails();
69      cout << endl;
70  
71      // Modifying student details using setter methods
72      student1.setName("Himanshu");
73      student1.setGrade('B');
74  
```

So, the grade, studentGrade will be assigned to grade, right? So, now you can see in all the cases, right? So, this particular function is accessing a private member data called grade, right? So, I will come to those two, all right? So, studentGrade is assigned to grade.

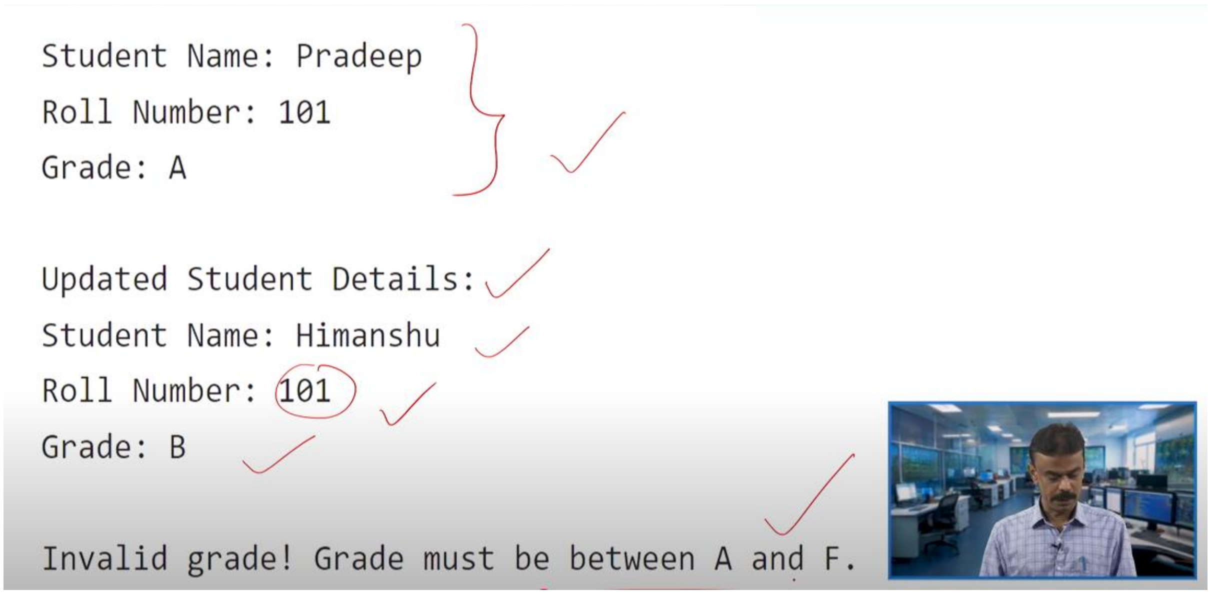
Else, right, it is an invalid grade. So, suppose you are entering G, H, Z, right. So, they are all invalid grades. So, the valid grade is between A and F, right. So, here this particular setter function can access the private member data called grade, right.

So, now you can see that your setter method is accessing rollNumber. getRollNumber, which is accessing rollNumber. And if you look, rollNumber is a private member data, right? And then here you have name, right? Your getName is accessing name, and setName is accessing name, right?

And name is the private member data. So, this is how the functions work, right? Getter and setter functions or the member functions, which are controlling this data. Private member data, right. So, this can be controlled only by these functions. I cannot directly access. Suppose I have, let us say, student S1, right? The object S1, and I am trying, let us say, S1 dot name. It will give an error, right? Or S1 dot roll number, error. S1 dot grade will be an error, right.

So, we have done up to line number 52. If it is not between A and F, it is going to give an error. So next one, line number 55, you have one method called displayStudentDetails. So the displayStudentDetails, which is a member function. So that member function, so this also can access.

So in fact, that is going to print name, rollNumber, and grade. So name, rollNumber, and grade will be printed. Yes, this also can access. So, now if you see, these are all the public member functions, the getter, setter, and display, right. So, these are all the public member functions, and they can access the private member data, right.



```
Student Name: Pradeep
Roll Number: 101
Grade: A

Updated Student Details:
Student Name: Himanshu
Roll Number: 101
Grade: B

Invalid grade! Grade must be between A and F.
```

This is exactly what happened, right. So, they can access data. The private member data. So now we will go to the main. We are entering the main program.

Right. So I have created an object called student1. Right. Whose class is student. So in fact, I am passing three parameters.

Pradeep, 101, and A. Right. So we know that this is a parameterized constructor, student one. When I am creating an object, this will invoke the parameterized constructor. So where it is invoking the parameterized constructor, you can see over here. All right.

I have passed the student name Pradeep. Right. I have a roll number, I believe, 101. And the grade is A. Right. So this is what is exactly happening.

Right. So this is what is exactly happening. Right. So I have passed the name. The name is Pradeep.

And the student roll number is 101. Right. And the student grade is A. So these are what we are passing. So now, the student name. Now this method can control the private member data.

So now, name is equal to Pradeep. And roll number is equal to 101. Right. So now, setGrade function. This is invoking the setGrade function.

Right. So, setGrade function. What is studentGrade? studentGrade is A. Correct. So, this I am passing.

A, I am passing. And it is invoking setGrade. Right. In fact, this particular constructor is invoking setGrade inside. All right.

So, what is setGrade? So, setGrade is here. If studentGrade is equal to A, yes, it is A. Right. So, then A will be assigned to grade. Now, this function is using grade.

Right. So, grade is A now. Right. So, grade is A now. So, now studentName is, in fact, name.

Pradeep, rollNumber is 101, and grade is A. Right. So, this is clear now. Right. So, this is clear now. So, now the student 1, the object which is invoking displayStudentDetails.

Right. So when it is invoking displayStudentDetails. So these are being printed. The name will be printed. Pradeep, roll number is 101, and grade is A. So these three will be printed when student1 is invoking display.

These will be printed. Right. And the next one. Again, student1. Right.

So, student1. Set name to Himanshu. Right. You are setting the name to Himanshu. There is no enrollment number.

So, you know that already. Student1. Right. The enrollment number is there. Right.

With the previous case. We have not deleted that copy. So, 101 is there. So, that will be printed. And Student1, which is invoking setGrade.

setGrade is B. So, now what is happening? The name of the student is Himanshu. Same enrollment number, 101, and the grade is B, right? So, when I am calling this particular function, when it is invoking, right? So, when you are invoking Student1, displayStudentDetails, right?

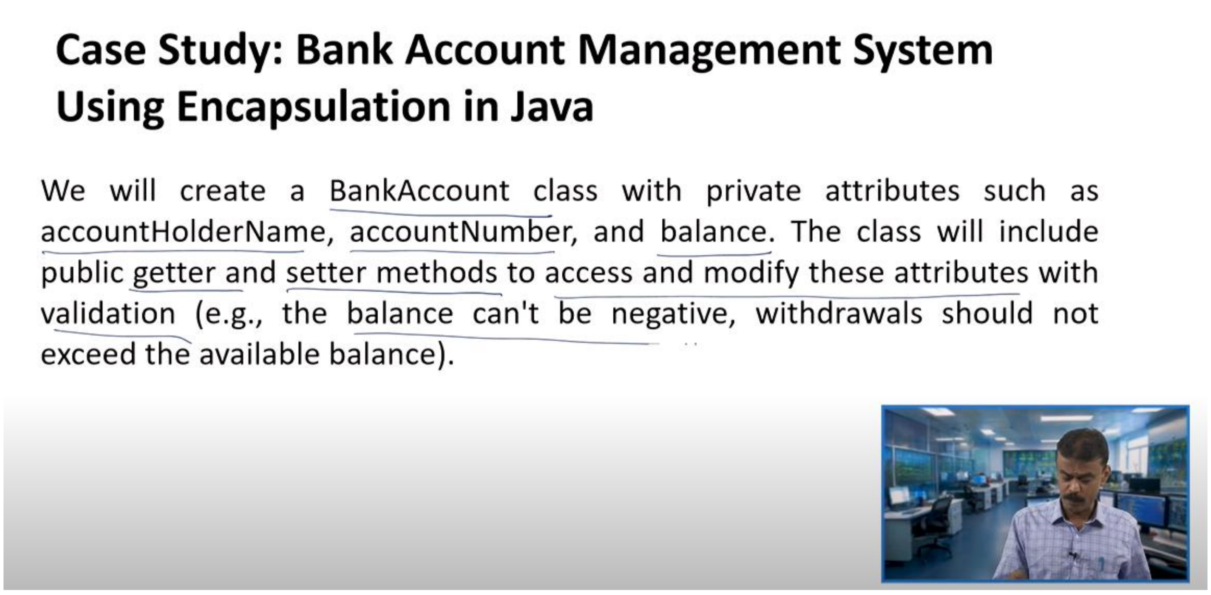
When the student1 object is invoked, displayStudentDetails. So, when you run the code, So, you will get this, right? So, initially, we displayed this information, right? So, when you look at this particular case, right?

So, we pass this and then, right? So, in line number 68, student1 is invoking displayStudentDetails, correct? So, it is printing Pradeep, 101, and A, right? And in the second case, right? We are printing.

Right. So, we are setting student1.setName to Himanshu and setGrade to B. Right. So now, this is changing. Right. So, setName is Himanshu.

SetGrade is B. Right. And there is no change in the roll number. Right. So, therefore, this will print updated student details. Right.

So that is what is happening over here. Updated student details. And then you have Himanshu's grade B. 101, there is no change. Correct.



Case Study: Bank Account Management System Using Encapsulation in Java

We will create a BankAccount class with private attributes such as accountHolderName, accountNumber, and balance. The class will include public getter and setter methods to access and modify these attributes with validation (e.g., the balance can't be negative, withdrawals should not exceed the available balance).

So this is what we are getting. And finally, when student 1 is invoking setGrade by passing Z, this character is not in the range. Right. So we know. The range is between A to F, and it is going beyond the range.

So, we have to get this as the input. Invalid grade; grade must be between A and F, right? So, that is what we are getting as an output, right? I hope it is clear, right? So, this is how it works in the case of C++.

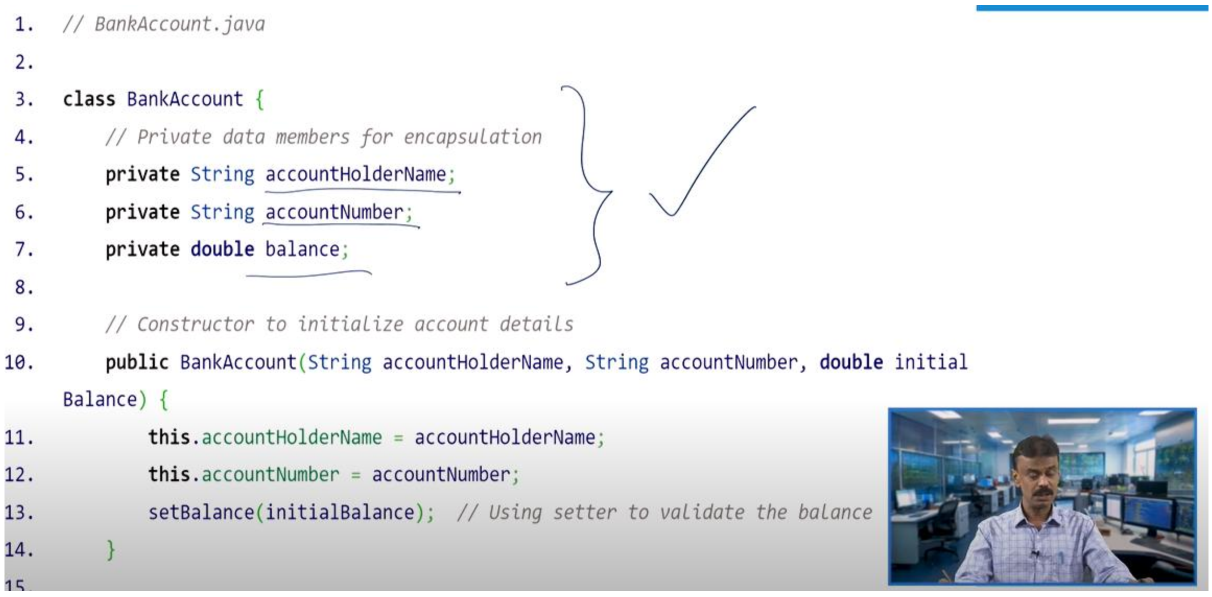
All right. It is working. Correct. In the case of C++, it is working like this. So now we will see one more case study with the help of Java.

So, what have we done in C++? So, we have, let us say, in the previous example, three member data. All right. And then the getter and setter functions. So those are all the control functions, particularly the display style as well.

Right. So, the display methods as well. All right. So, these functions can control. The private member data.

So, this is what exactly is happening, and then we can control with the help of public member functions, and we can access the private member data. We can see one more case study: bank account management system. The bank account management system using encapsulation in Java, right. So, let us create a class called BankAccount, all right, and then you have some private attributes, right? The private member data, such as accountHolderName, accountNumber, and balance, right? So, the class will include getter and setter methods, right? The class will include getter and setter methods to access and modify these attributes, to access and modify these attributes. All right. So, with validation.

All right. So, what is the validation? The bank balance cannot be negative, and withdrawal should not exceed the available balance. Right. So, this is the validation.



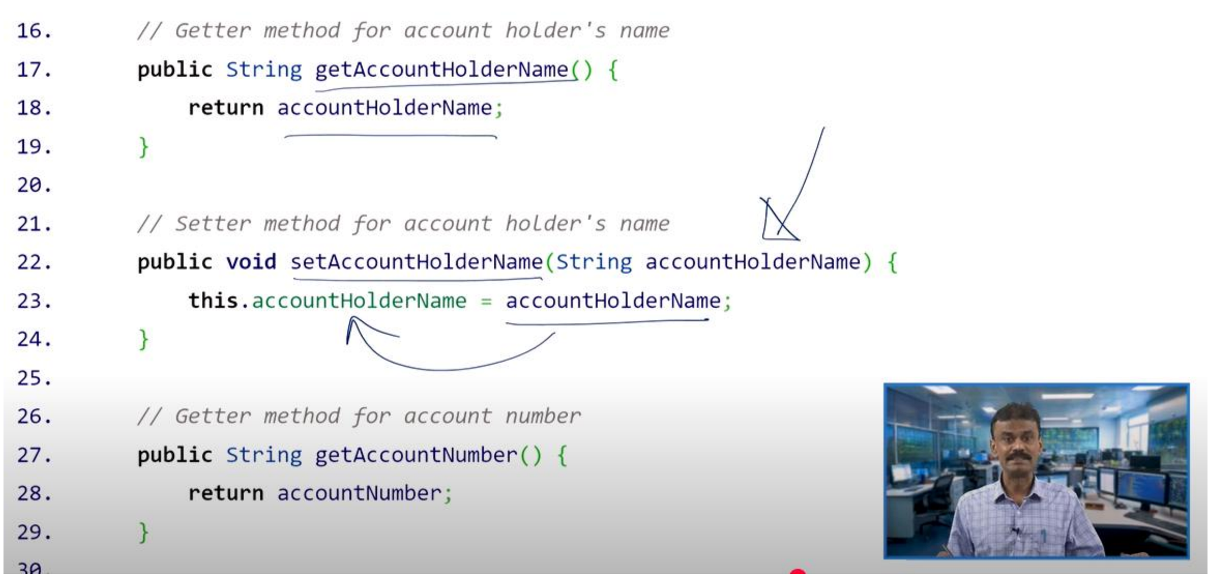
```java
// BankAccount.java

class BankAccount {
    // Private data members for encapsulation
    private String accountHolderName;
    private String accountNumber;
    private double balance;

    // Constructor to initialize account details
    public BankAccount(String accountHolderName, String accountNumber, double initial
Balance) {
        this.accountHolderName = accountHolderName;
        this.accountNumber = accountNumber;
        setBalance(initialBalance);  // Using setter to validate the balance
    }
```

You have to check this validation. So now, with the help of encapsulation, we are going to have private member data like account holder name, account number, and balance, right? And then we are going to have several getter and setter methods. So that we can access accountHolderName, accountNumber, and balance. So let us see.

So we have three private member data, right? So we have three private member data: accountHolderName, right? AccountNumber and balance, right? Right. So balance, whose data type is double; account number, whose data type is string; and account holder name, whose data type is string.

OK. So now let us have the constructor bankAccount. Right. So you have the constructor bankAccount, and we are going to pass accountHolderName, accountNumber, and initialBalance. So these are the three arguments.

That means three values you have to pass in the main. All right. And then they will all be assigned. accountHolderName will be assigned to this dot accountHolderName. So that means accountHolderName is getting the value initialized properly.



```
16.     // Getter method for account holder's name
17.     public String getAccountHolderName() {
18.         return accountHolderName;
19.     }
20.
21.     // Setter method for account holder's name
22.     public void setAccountHolderName(String accountHolderName) {
23.         this.accountHolderName = accountHolderName;
24.     }
25.
26.     // Getter method for account number
27.     public String getAccountNumber() {
28.         return accountNumber;
29.     }
30.
```

And accountNumber is assigned to this dot accountNumber. All right. And initialBalance. Right. That will be passed.

So whatever you are passing, that means it is going to call the function called setBalance. Exactly like the previous program. All right. So it is going to call accountHolderName. Let us say you have one getter method.

All right. So the getter method, you have getAccountHolderName. So the getter method, you have one getAccountHolderName. So that will return accountHolderName, which is the private member data. And you have another setter method.

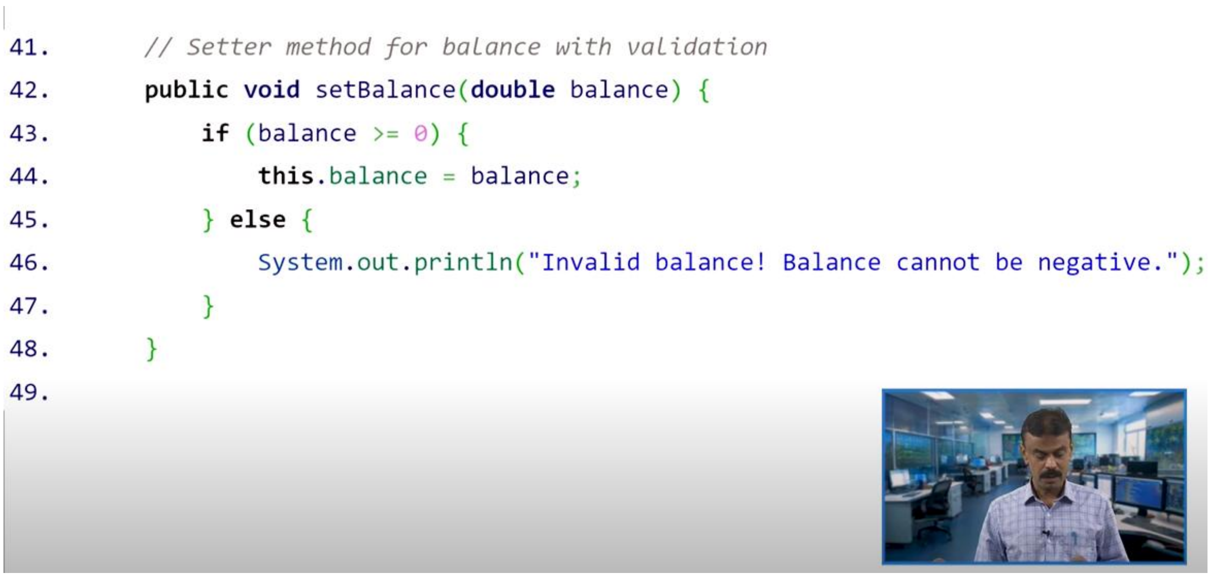
So you have a setter method. The setter method calls setAccountHolderName. You are passing accountHolderName. So the accountHolderName is assigned to this dot accountHolderName. All right.

So that will be assigned to this dot accountHolderName. And line number 26, you have 27, you have getAccountNumber method, which is a getter method. All right. So that getter method will return account number. All right.

So similarly, you will have a setter method. So the setter method for account means you are passing the accountNumber. All right. So the setAccountNumber, the account number will be assigned to this dot accountNumber so that the initialization will be proper. And you have one getter method for balance.

All right. So you have the function called method called getBalance whose return type is double, which is public and that will return balance. Right. So, that will return balance and here you have. So, this particular function, this particular method is already called by your constructor, right.

If you look at the constructor BankAccount. So, this particular method is being called, right. So, you call it as the initialBalance, correct. So, you are passing the initialBalance, set balance will be called that means you are passing from here, right. And when it is calling this particular method,



```
    // Setter method for balance with validation
    public void setBalance(double balance) {
        if (balance >= 0) {
            this.balance = balance;
        } else {
            System.out.println("Invalid balance! Balance cannot be negative.");
        }
    }

```

Correct? So, it is being called, right? You are passing, and this will be like this. If balance is greater than or equal to 0, balance will be this dot balance. So, if it is going negative, it is an invalid balance. Balance cannot be negative, all right?

So, if you are passing minus 5, correct? Minus 5.00, all right? So, then it is an invalid balance. Balance cannot be negative. And then you have one public method. The public method is deposit.

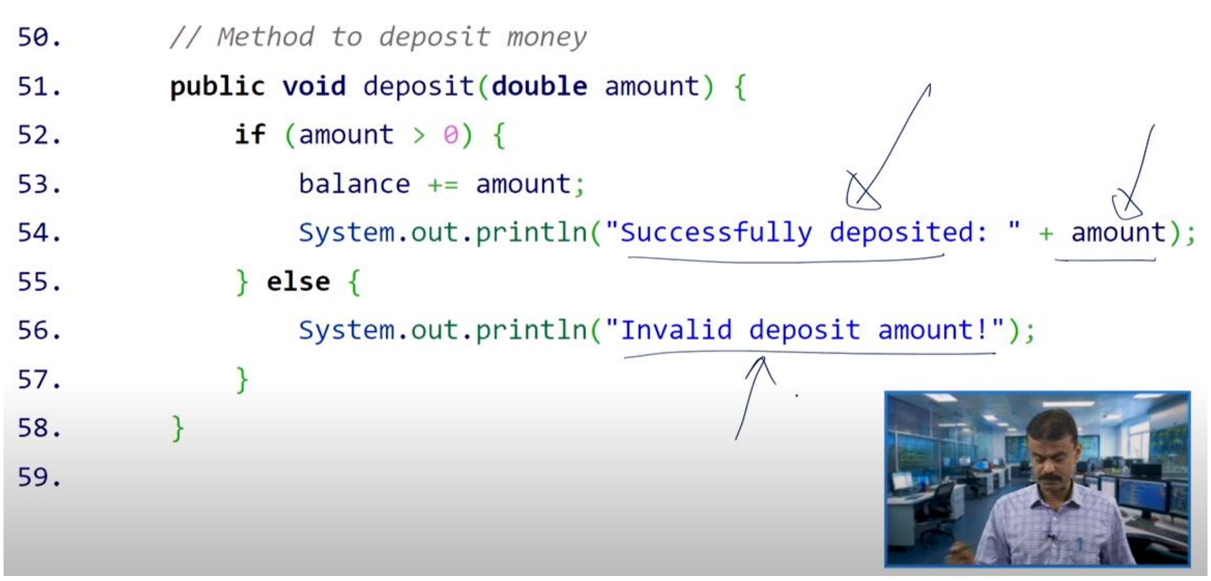
All right. If the amount is greater than 0, balance equals balance plus amount. All right. If the amount is greater than 0, balance equals balance plus amount. All right.

So, system dot output printer length. So, it will give successfully deposited. All right. So, that you have an amount. That you have an amount.

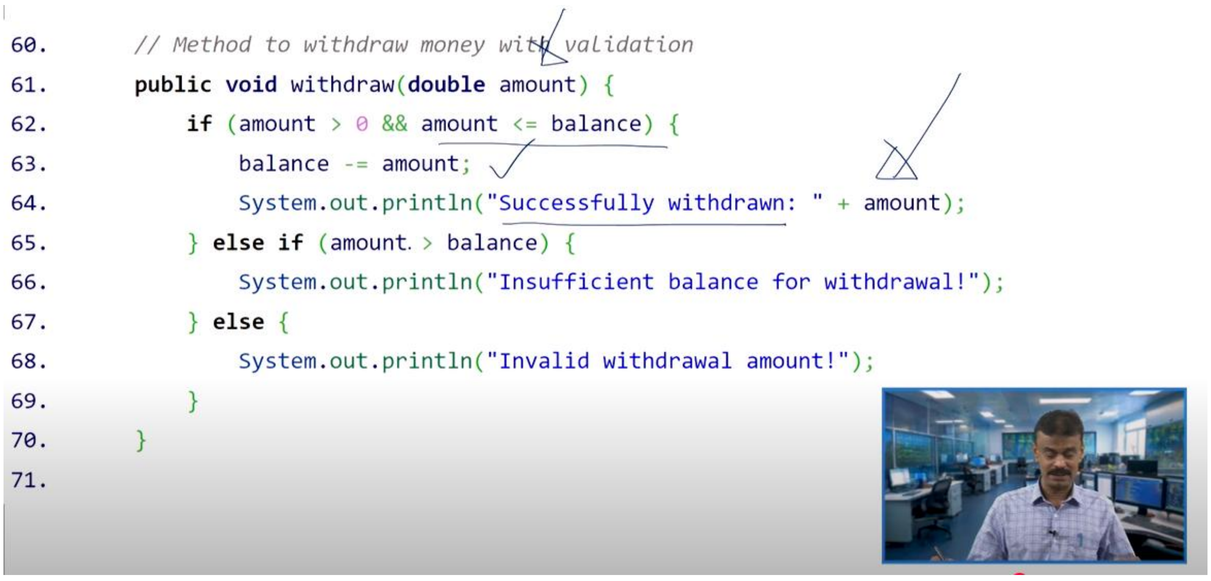
All right. Otherwise, invalid deposit amount. All right. So, that is an invalid deposit. Deposit amount.

If the amount is less than 0, alright, which is not possible. When you are depositing, you will not deposit a negative one, alright. So, invalid deposit. So, the balance will be incremented by whatever amount you are depositing, right. So, successfully you have deposited the amount.

So, that means the balance is incremented by whatever amount you have deposited. So, previous balance plus the amount, and you are displaying the amount. Right. So, the amount has been successfully deposited. So, now you have one more public method, withdraw, right?



```java
    // Method to deposit money
    public void deposit(double amount) {
        if (amount > 0) {
            balance += amount;
            System.out.println("Successfully deposited: " + amount);
        } else {
            System.out.println("Invalid deposit amount!");
        }
    }

```



```java
    // Method to withdraw money with validation
    public void withdraw(double amount) {
        if (amount > 0 && amount <= balance) {
            balance -= amount;
            System.out.println("Successfully withdrawn: " + amount);
        } else if (amount > balance) {
            System.out.println("Insufficient balance for withdrawal!");
        } else {
            System.out.println("Invalid withdrawal amount!");
        }
    }

```

So, in withdraw, you are passing the amount. So, if the amount is greater than 0 and less than or equal to the balance, there is a change in the balance. Balance equals balance minus the amount, right? And then you are printing: System.out.println, 'Successfully withdrawn whatever the amount,' successfully withdrawn whatever the amount. So, this is the amount that you are withdrawing. That means if it is less than or equal to the balance, you can withdraw it. Otherwise, if the amount is greater than the balance, right? Suppose you have a thousand rupees, and then you want to withdraw two thousand. The balance is a thousand, but you want to withdraw two thousand, right? So, it will print 'Insufficient balance for withdrawal.' It will print 'Insufficient balance for withdrawal.' Else

invalid withdrawal amount, right? So, minus 10, something like that, right? So, when you put it correctly, so if you are displaying, right? If you are, if you are, if you are passing the amount which is, let us say, less than 0 or greater than the balance, right? In fact, greater than the balance, it will be taken care of, right? Other than this range, in fact, less than 0, less than or equal to 0 All right. So, you are printing 'Invalid withdrawal amount.' In fact, if I put 0, all right, 0 can be passed. So, if it is passed, so it will be coming over here, and then, all right.

So, that will be printing an invalid withdrawal amount, correct. So, this is one of the methods, a public method withdraw, right. So, the last one is displaying all the details, right. So, the displayAccountDetails. So, you have

AccountHolderName, accountNumber, and balance. Right. So, what are all the methods we have? So, we have, let us say, one constructor. bankAccount is a constructor.

Yes, of course, the class name of the class is bankAccount. And here you have set. In fact, the get method for account holder. getAccountHolderName. And then the set method for setAccountHolderName.

And a getter method for getAccountNumber. Right. setAccountNumber. getBalance and setBalance. Right.

So these are all the getter and setter methods. And then you have deposit. Right. This is a method. Deposit is a method.

Withdrawal is a method. Right. And then finally you have displayAccountDetails. Correct. So here I have the public main.

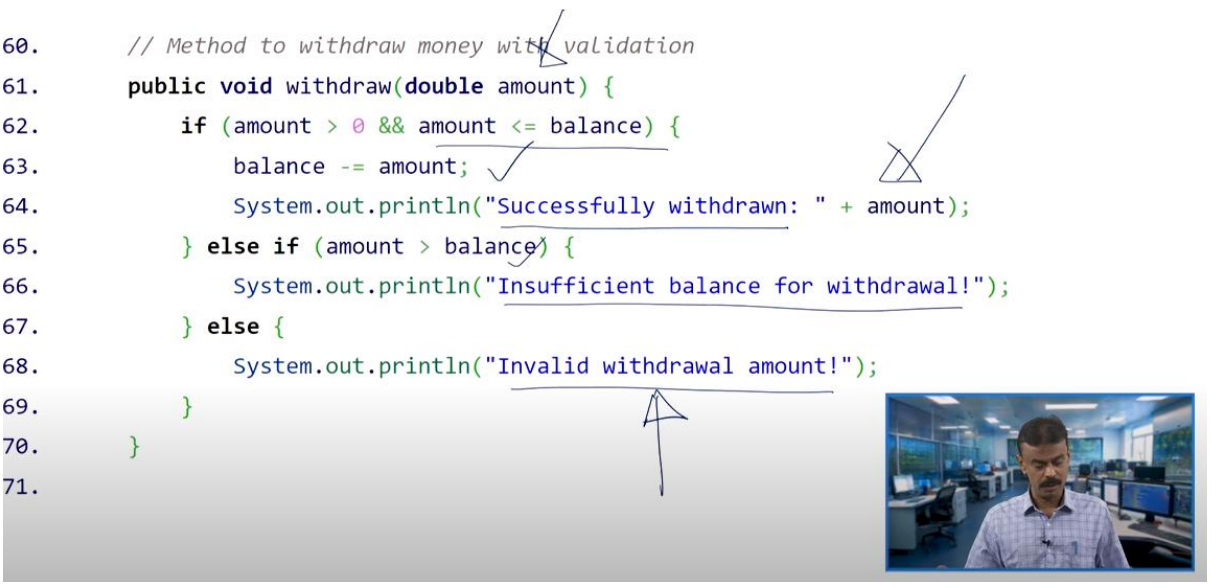
Right. So, you have a driver class, the main driver class. Right. So here, you have account1 as the object. And bankAccount is the class, right?

Account1 is the object, and bankAccount is the class. So, with the help of dynamic array, sorry, with the help of dynamic memory location. So, you have the constructor, right? Constructor, and you are passing three values: Tapan, this is the account number, and this is the amount. All right.

This is the amount. You are passing these three values. Right. So, the constructor. So, the constructor is being called.

Right. The name is Tapan. So the accountHolderName is Tapan. And then the accountNumber, we have put the A, Z, everything. And the setBalance function will be called.

Right. So the initialBalance. How much? 500 rupees. Right.



```java
    // Method to withdraw money with validation
    public void withdraw(double amount) {
        if (amount > 0 && amount <= balance) {
            balance -= amount;
            System.out.println("Successfully withdrawn: " + amount);
        } else if (amount > balance) {
            System.out.println("Insufficient balance for withdrawal!");
        } else {
            System.out.println("Invalid withdrawal amount!");
        }
    }
```

Or 500 dollars. Right. So this is being passed. So the setBalance function will be called. That means the setBalance method will be called.

Balance is greater than or equal to 0. Yes. True. So this dot balance will be equal to balance. Right.

So this dot balance will be equal to balance. So that means I have name, accountNumber, and balance being initialized. They are being initialized. Right. So next, what is happening?

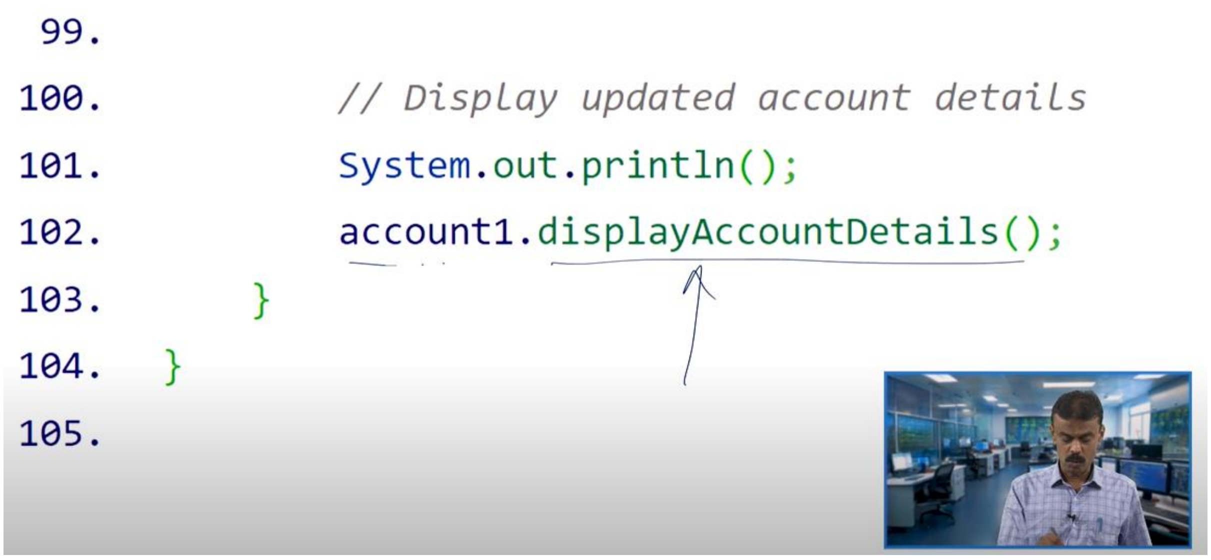
When you are creating an object, these three values are being initialized. So now you are displaying accountDetails. So these details will be displayed. System.out.println is empty. So these will be displayed.

That means when you call this particular method, accountHolderName, accountNumber, and balance will be displayed. The corresponding accountNumber, this is the accountNumber, and 500 will be printed. And now you are depositing 150. Right. account1, which is invoking the method called deposit.

Right. Which is invoking the method called deposit. So you are passing 150. So the deposit method will be invoked. Where is deposit?

Here. So the amount is 150, which is greater than 0. So balance is equal to balance plus amount. Right. So the balance will now be 650.

500 plus 150. 650. Right. Right, and then you have successfully deposited 150, which will be printed. That is the amount, so it will not go to the else part. Next, you are withdrawing 100, so 650 is there, right? When you are withdrawing 100, it should be minus 100, right? So, the resultant will be 550, right? So, you are



```

            // Display updated account details
            System.out.println();
            account1.displayAccountDetails();
        }
    }

```

trying to withdraw, which means this particular method will be called, right? How much are you trying to withdraw? 150, right? How much are you trying to withdraw? 100. Let us say 100, you are trying to withdraw.

So, 100 is between this, right? Between the amount greater than 0 and the balance. So, balance equals balance minus amount, right. So, the resultant will be 550. 650 was there, 100 you want to withdraw, 650 minus 100 is 550, right. So, this is what is happening.

So, you have successfully withdrawn whatever the amount that you have passed. So, you have passed 100. So, that 100 will be displayed. That means you want to withdraw 100.

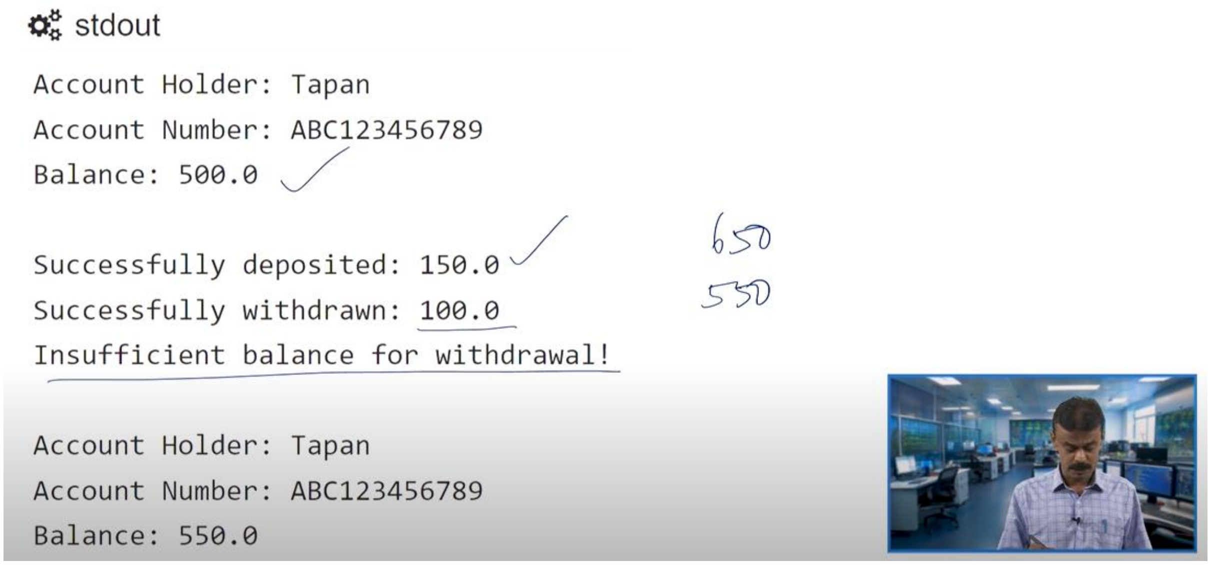
Right. So, what is the balance now? 550. So, when it is 550, you are trying to withdraw 600 now. So, that is more than the available balance.

So, I have a balance of 550. I am trying to withdraw 600. Right. So, I have to get the warning message. Right.

So, that means if the amount is greater than the balance, it has to give an insufficient balance for withdrawal. So, this has to be displayed. So, this will be displayed. And then, finally, the displayAccountDetails. So, when account1 is invoking displayAccountDetails, I will get the complete output, the accountNumber.

Name: Tapan. Corresponding account number, whatever the available balance is right now. All right. So those things will be displayed. So you can see that. Right.

So initially, I have 500. All right. So then you deposited 150. The balance will be 650. Right.



```
Account Holder: Tapan
Account Number: ABC123456789
Balance: 500.0

Successfully deposited: 150.0
Successfully withdrawn: 100.0
Insufficient balance for withdrawal!

Account Holder: Tapan
Account Number: ABC123456789
Balance: 550.0
```

And then you are trying to withdraw 100. Right. So when you try to withdraw 100, the balance should be 550. And then you are trying to withdraw 600. Right.

When you are trying to withdraw 600. Right. What is happening? Insufficient balance for withdrawal is being printed, right? And then finally, you are displaying.

When you are displaying, the name of the account holder is Tapan, and this is the account number, and as expected, 550 will be the remaining. We are getting 550, right? So, we have seen two different case studies. One is with respect to C++, right? Another one is

So, in all the cases, you have the private member data, and then you have control methods or control member functions called getter and setter or even another member function. So, they can access the private member data. So, this is called encapsulation or data hiding, and in the next class, we will see data abstraction, right? Thank you very much.