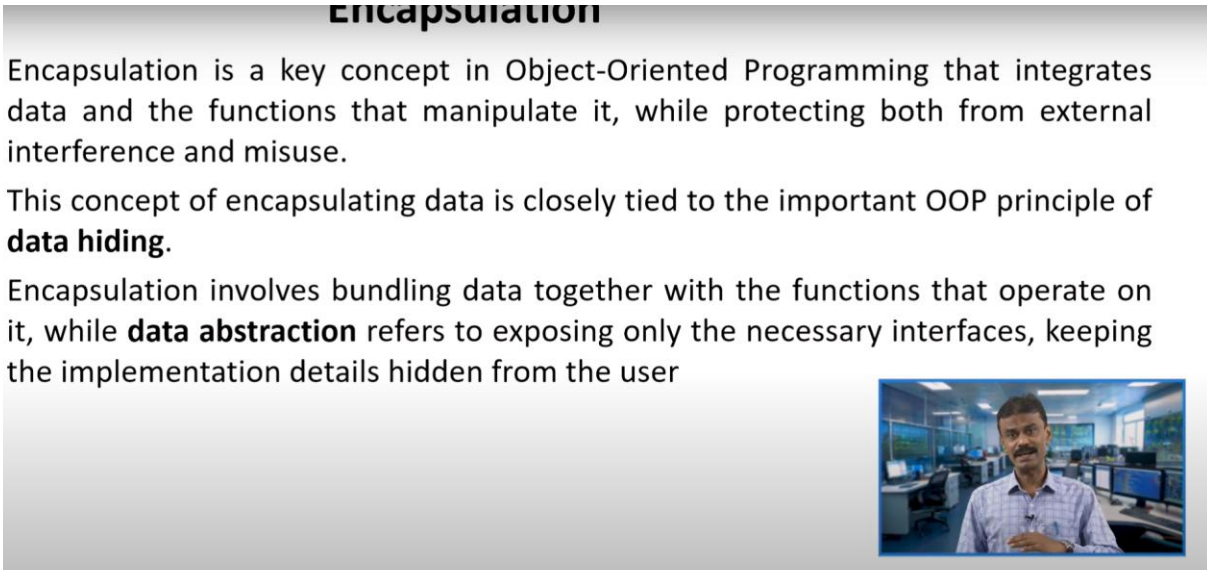
FUNDAMENTALS OF OBJECT ORIENTED PROGRAMMING

Lecture21
Lecture 21: Encapsulation I



Encapsulation

Encapsulation is a key concept in Object-Oriented Programming that integrates data and the functions that manipulate it, while protecting both from external interference and misuse.

This concept of encapsulating data is closely tied to the important OOP principle of **data hiding**.

Encapsulation involves bundling data together with the functions that operate on it, while **data abstraction** refers to exposing only the necessary interfaces, keeping the implementation details hidden from the user

Welcome to Lecture 21, Encapsulation and Abstraction. So, this is a new chapter in So, we will see what you mean by encapsulation first. So, we know when you are writing a class, the class contains several member data and member functions, right? In C++, or you call it a data member and methods in Java. So, when you are using these member data and member functions, how do you protect them from external interference and misuse, right?

Suppose I assume that I have declared a data member. Right, our data members, all right. So, let us consider I want to use it only inside that particular class, right. So, we know access specifiers, but is it possible to externally use, let us say, member data and member functions from outside, right? So, even though the protection is there, right, and how can I carefully handle it from the

So, at that point in time, when I am using those, there should not be any misuse or any interference from outside. All right. So, it's like a capsule. Right. So, the capsule, the medicine is tightly packed.

So, in a similar way. So, when I start the class, let us say, begin to end. All right. So, you are closely packing your member data and member functions. Right.

So, this is an important principle in object-oriented programming, and since we are protecting them from external interference and misuse, we also call this data hiding, right? So, that means data will be handled only inside the class, and for outside use, we are going to carefully use this. So, this involves, right, encapsulation involves You are keeping all the data together with the functions. That means the member data and member functions are together.

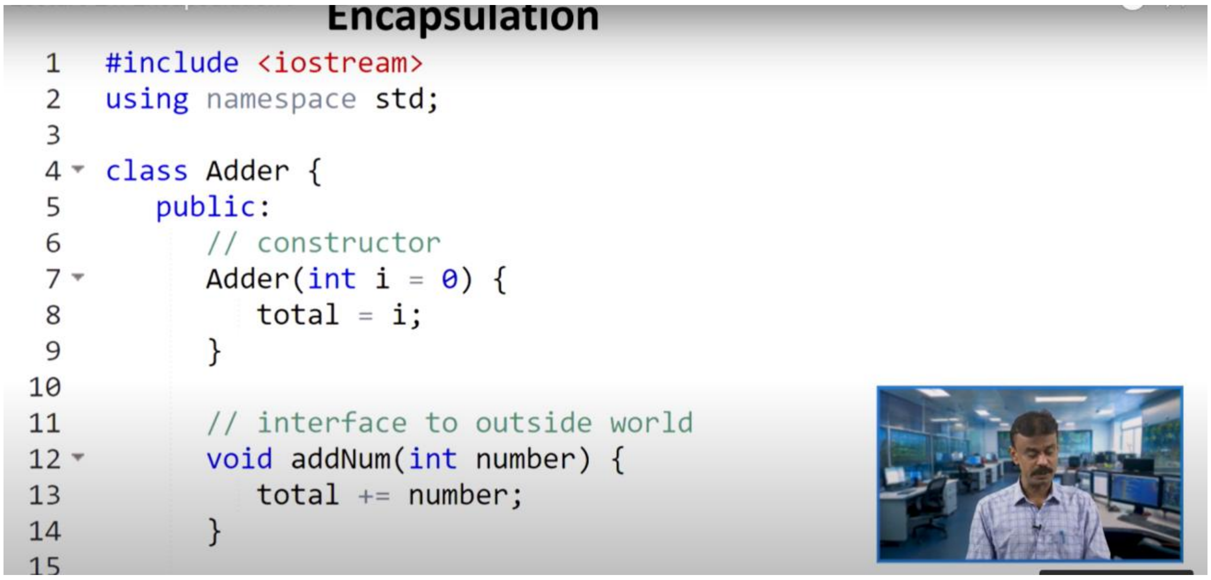
And when I am talking about data abstraction, right? So, in terms of functions and member data, we are going to explore, right? By having set functions, get functions, etc., right? So, from the outside, I can try to call the function, right? Let us say, publicly accessible functions.

And from those functions, you can access the data members and member functions, right? So even though they are private, all right? So we have the concept of data hiding and data abstraction. In fact, we are talking about data abstraction slightly later.

As I already said, it's like an abstract, right? So when you're writing a research paper abstract, you are not giving any details. For example, I want to write certain functions, right? I want to implement certain functions.

So I'm just giving the name. And then I will define it outside. So that is called data abstraction or simply abstraction or an abstract class. So how can we use this particular property, right, the encapsulation property, in C++? So this can be done.

So, C++ supports the properties of encapsulation and data hiding, all right. So, we create the user-defined function, right? We create the user-defined function, and this function assumes that there are some objects in the particular class, and that particular object will invoke these functions. So, these functions, let us say, are defined inside the classes, and then they can access the private member data, all right. The concept of encapsulation and data hiding, all right. So, we are going to, let us say, the set function, all right.



Encapsulation

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

class Adder {
   public:
      // constructor
      Adder(int i = 0) {
         total = i;
      }

      // interface to outside world
      void addNum(int number) {
         total += number;
      }

```

So, the set function. So, I can call it outside the class. In fact, from the main, the particular object will invoke it, all right. The set function will access, and then the get function will access the member data. So, in fact, we already talked about, if you recall, private, protected, and public, all right.

So, private, protected, and public, and the default means, if I am not mentioning anything, it is considered as private. So now we will see an example. All right. So, what do you mean by encapsulation? So, let us consider you have a class adder.

Right. So, class Adder, you have the constructor. Right. So, you have the constructor Adder(int i) is equal to 0. Right.

Total equal to i. So, add number. Right. So, another function you have is add number. So, that is also public. So, one is a constructor.

So, this one is a constructor, and you have a member function. Add number. So, int number. So, total equal to total plus number. This function is adding total with number.

You are passing a number, and then this will be added to the total. The next function is int getTotal. The next function is int getTotal. So that will return the total. So now you have a private member data.

So the concept of encapsulation starts from here. So you have the private member data called total. Correct. So how can the total be accessed? Right.

Indirectly, not directly. You cannot access it from outside. But how are we going to access this from the outside? So that is the concept of encapsulation. So let us assume that I have one object called a, right?

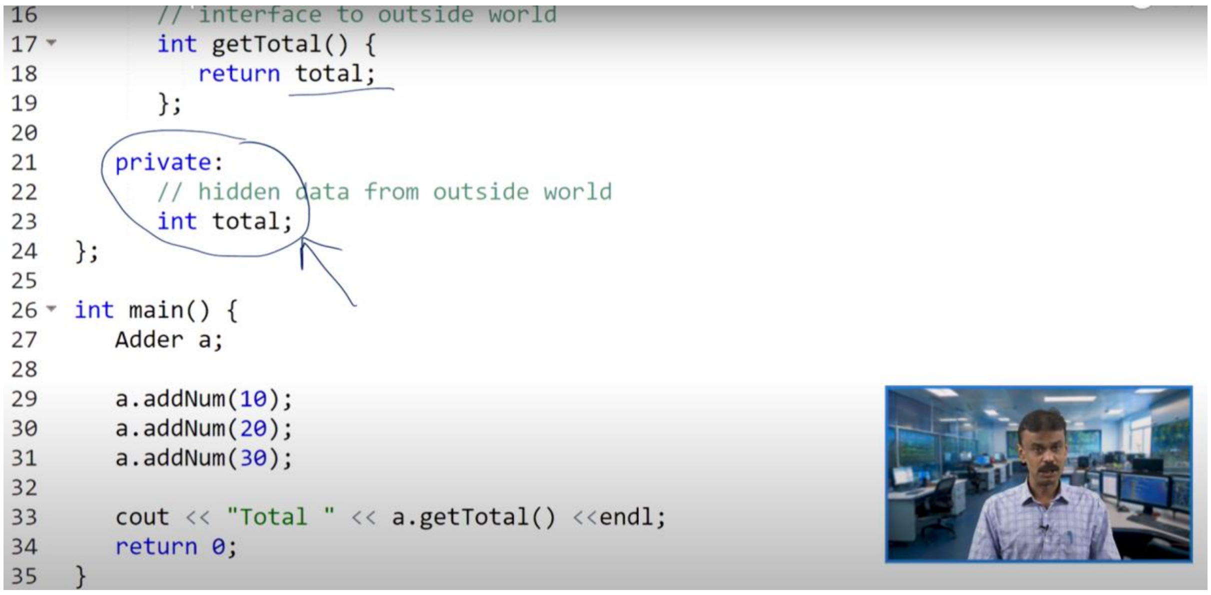
I have one object called 'a' whose class is Adder, right? And 'a' is invoking add_number, right? 'a' is invoking add_number, right? So you are passing 10, right? So, when you are creating this object, the total is initialized to 0, right?

Right. Because this will be calling this particular constructor, and 'i' will be 0, and total is equal to 'i'. Right. I hope it is clear. Total will be 'i' now. Right.

'i' is what? 0. So total is 0. So now, when you are calling by passing 10. So what is happening?

The add number function will be called. The add number member function will be called. Right. You are passing 10. Right.

Passing 10. So the number is 10 now. Total equals total plus number. So 0 plus 10. So after invoking the function, the function will have the right total 10. Right, the total is 10.



```
        // interface to outside world
        int getTotal() {
           return total;
        };

     private:
        // hidden data from outside world
        int total;
  };

  int main() {
     Adder a;

     a.addNum(10);
     a.addNum(20);
     a.addNum(30);

     cout << "Total " << a.getTotal() <<endl;
     return 0;
  }
```

So now, carefully look. This particular object, a, is invoking the add number member function, right? And I can access this total, right? So indirectly, we can access the total, right? So once you are going, once you are accessing, in fact, when you are calling, when you are creating this object, yes. The total is initialized to zero, right? You can access it from outside, but what we are doing safely is accessing it, right? So that is an important point of data hiding and encapsulation. We are hiding this total data from the outside world, right? We are hiding this data from the outside world, but how are we accessing it? So you have understood. So when I am creating an object, the total is zero when you are creating it, right? And then what is happening when this object is invoking add number? When you are passing 10, the total will be added with 10, right?

So, total plus number because you are passing 10 here. So, what is the total now? 10. The total is 10 now. So, now the object 'a' is invoking another add number, right?

The same function next time. Add number. Now you are passing 20, right? So, the total is already 10, right? So, this is your total.

Now you are adding with 20, right? So, 10 plus 20, now the total will be 30. The total is what? 30. So, again 'a' is invoking one more time at number 30, right?

So, already 30. So, 30 plus 30 will be 60. So, the total is 60 now, right? And then you are printing the total, right? By calling, a is calling the function getTotal.

a is calling the member function getTotal. So, getTotal is here. Again, I can access the total, right? So, 60. 60 will be printed, right?

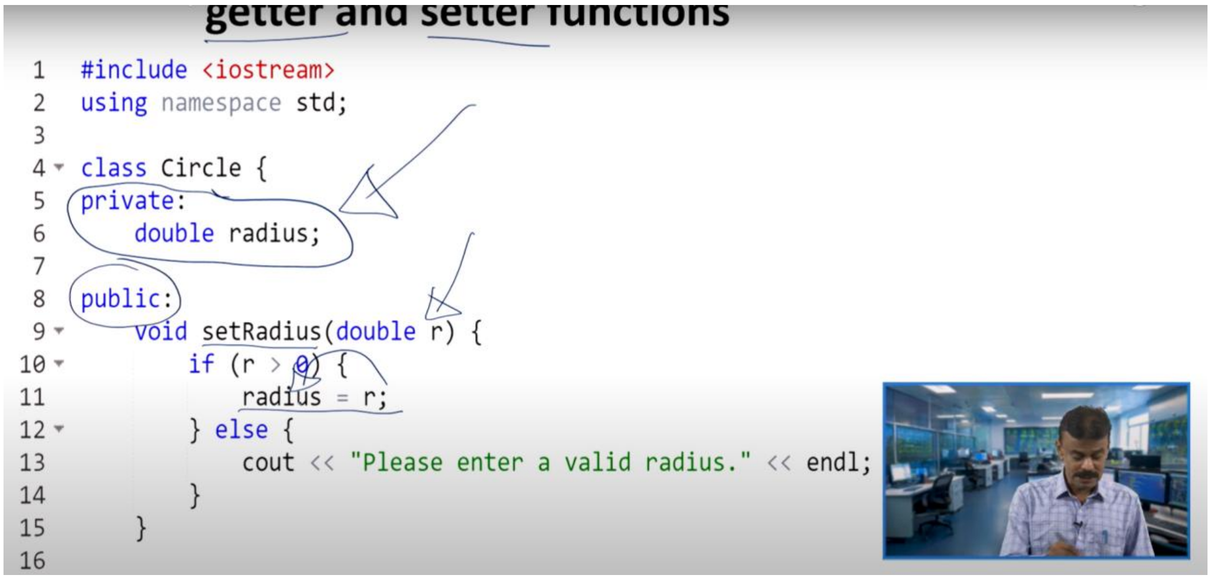
I have to get the output 60, correct? So, when I run the code, you can see the total is 60, correct? So, this is how I can explain the concept of encapsulation or even call it data hiding. So, which data are we hiding? We are hiding the private member data called total, right?

So, In fact, I mean if I write something over here, total, correct a, it is a private member data. Suppose I write a dot total outside, we know that it will give an error, right. So, if suppose I put cut a dot total, right, you can try. So, what you can do instead of calling this particular function, try a dot total, right.

So, you will get the, right. So, there will be an error, right, because you cannot access outside the class, right. And moreover, the access modifier or access specifier, you can see it is private. So, this concept is called data adding. Even though we are using, again and again we are using, three times we are using, right?

When you are calling only one time, when you are calling a function one time, add number function, when you are creating, in fact, when you are creating an object, the total will be assigned to 0. And then when it is calling add number of 10, right? When it is calling this particular function, right? So, the total, right, total equal to total plus number, we are accessing total. And finally, when you are printing a dot getTotal, again return total, right.

If you look at line number 17, the getTotal function, a is invoking getTotal, getTotal is being called, and it is returning total, right. So, when you are running the code, you are getting the output. So, basically, where we are hiding the data, line number 21 to 24, right, 21 to 23. So, data hiding, and you call this concept encapsulation. So, you have a private member data, and then, right?



getter and setter functions

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

class Circle {
private:
    double radius;

public:
    void setRadius(double r) {
        if (r > 0) {
            radius = r;
        } else {
            cout << "Please enter a valid radius." << endl;
        }
    }

```

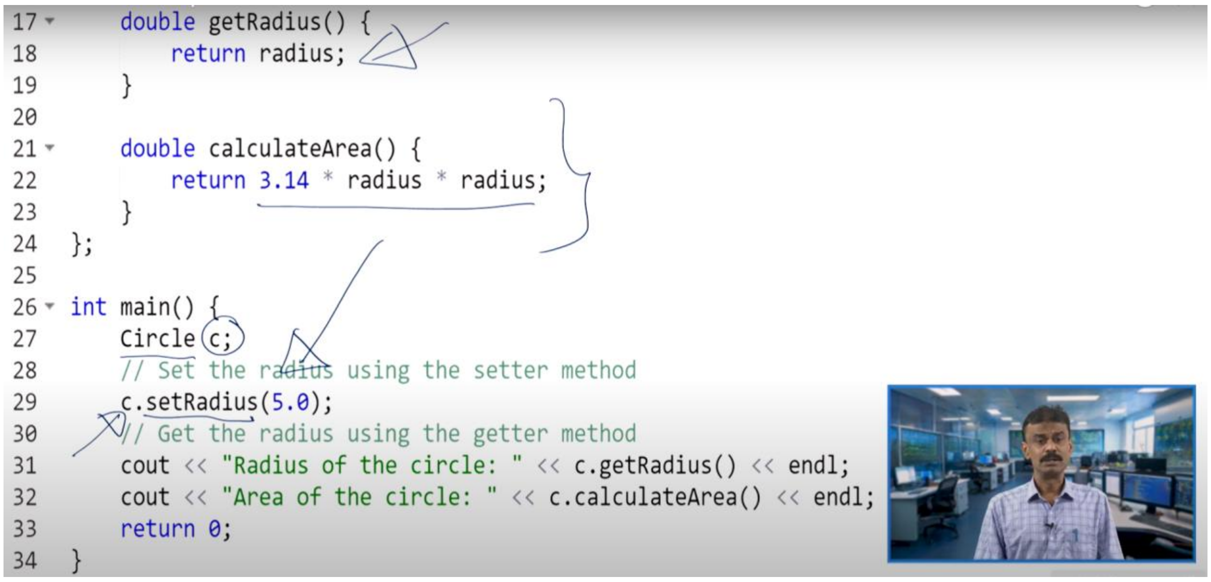
So, you have several public member functions. So, through public member functions, we are able to access this private data. So, this concept is called encapsulation, or you also call this data hiding, right? Another famous, right? The functions getter and setter, right?

So, once you are becoming an advanced... programmer in object-oriented programming, it is advisable to use these getter and setter functions, right. So, how do you get and set, right. So, the data hiding here is in your class Circle, you have the private member data called radius, right. So, this we are going to do the hiding, right, or this we are going to do the encapsulation.

So, I have one set radius function. I am passing r, right? If r is greater than 0, right? If r is greater than 0, radius is equal to r, right? That means I am assigning r to radius, right?

So, this we have done several times. In fact, when we wrote the code previously in all the previous four chapters, you can recall we have done this, right? But extensively, we do not know. This concept is data hiding or encapsulation, right? r is assigned to radius, right?

So, r you are passing, from outside, and this is a public member function. So, now this public member function can access radius. So, therefore, we put radius is equal to r, right? Otherwise, you are writing if r is less than 0, radius less than or equal to 0 is not possible. So, you are writing cout, please enter a valid radius, right now.



```
    double getRadius() {
        return radius;
    }

    double calculateArea() {
        return 3.14 * radius * radius;
    }
};

int main() {
    Circle c;
    // Set the radius using the setter method
    c.setRadius(5.0);
    // Get the radius using the getter method
    cout << "Radius of the circle: " << c.getRadius() << endl;
    cout << "Area of the circle: " << c.calculateArea() << endl;
    return 0;
}
```

So, this is a set function. And another get function returns the radius, right? Another get function returns the radius. Get radius will return the radius, okay? So, one more member function we have is calculateArea, right?

So, the calculateArea is returning pi r squared, 3.14 into radius into radius. So, now you are creating an object c, right? You are creating an object c whose class is Circle, whose class is Circle, right? So, now this object is calling the set function. So, let us see if any constructor is there.

I do not think any constructor is there. It is like, I mean, the default constructor. So, no problem. So, now c is invoking the setRadius function, a member function, and you are passing 5.0, right? So, you are passing 5.0, right?

So, when you are passing 5.0, right. Your radius, right, which will be getting 5.0. Your radius is 5.0 now, right. So, from outside, setRadius is a public member function, not a problem. And 5.0, I am passing.

So, when I am passing 5.0, here you can get it, right. So, that means, indirectly, I can access the radius, right. So, now the radius is equal to 5.0. I hope it is clear for everyone. So, now, let us say, cout, right.

So, what is the radius? Radius of the circle, right? c is invoking the getRadius function. c is invoking the getRadius function, right? Again, right?

So, get function. So, the get function will call this particular member function, returning the radius. What is the radius? 5.0. So, the first output I have to get is 5.0.

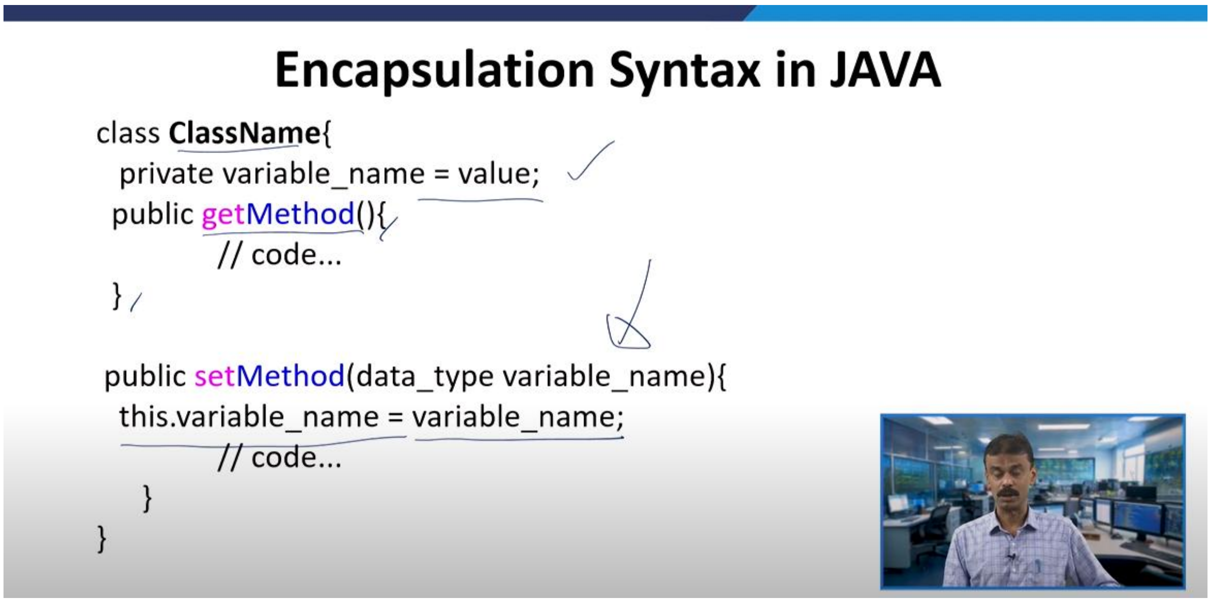
So, from outside, this is the way. So, you have several public member functions: getRadius, setRadius. So, all are public member functions. So, through public member functions, because they are inside the class, they can access your radius, right? So, this is what you are hiding, right? That is called data hiding.

So, the double radius is being hidden, right? And now, how do you access it with the help of public member functions, getter function, and setter function? So, c.getRadius, you are getting 5.0, right? And the area of the circle, c.calculateArea, right? So, calculateArea, when it is calling, 3.14 into radius into radius, pi r squared. So, 3.14 into 5 into 5. So, you are getting the output.

So, return 0. So, whatever the value is, multiply 3.14 by 5 by 5. So, you can use your calculator and see. So, we are going to get that particular result. So, when you are running the code, you have to get this output.

In fact, yeah, 78.5 you will get. So, one can see the radius of the circle is 5. So, this is what we are expecting, and we are getting it, sorry. So, we are getting this, which is what we are expecting, and 5 we are getting, and 3.14 multiplied by 5 multiplied by 5 is 78.5. Okay, you can check. So, these are all the outputs.

In both the programs, what we are doing is hiding, right? We are hiding the private member data. So, you can even have a private member function, right? Which cannot be accessed outside the class. So, you are hiding, right? This concept is called data hiding, and on the whole, this is called encapsulation, right? So, you are protecting the data. Where I mean, this cannot be unnecessarily or no one can misuse this radius. So here, in this program, the radius, and in the previous program, the total. Okay. So, similarly, the same concept.



Encapsulation Syntax in JAVA

```
class ClassName{
  private variable_name = value;
 public getMethod(){
          // code...
 }

 public setMethod(data_type variable_name){
  this.variable_name = variable_name;
          // code...
    }
}
```

Right. So we have it in Java. Same concept. Right. So here, the name of the class is, let us say, class name.

Right. And then you have a private member data. Right. So private variable name equals value private. So let us say these are all the syntax.

So no need to worry. We'll see a real example in the next slide. All right. One private member data, or you call it a data member in Java. And another one is a public method.

Right. You have a public method. Right. Public getMethod. So assume that you have returned that public getMethod.

You have returned from here to here. And then you have one more method called setMethod. Right. So setMethod, you are passing a variable name. So the variable name is assigned to this dot variable name.

Right. So we use this pointer for many cases. Just recall. Right. So this dot variable name equals variable name.

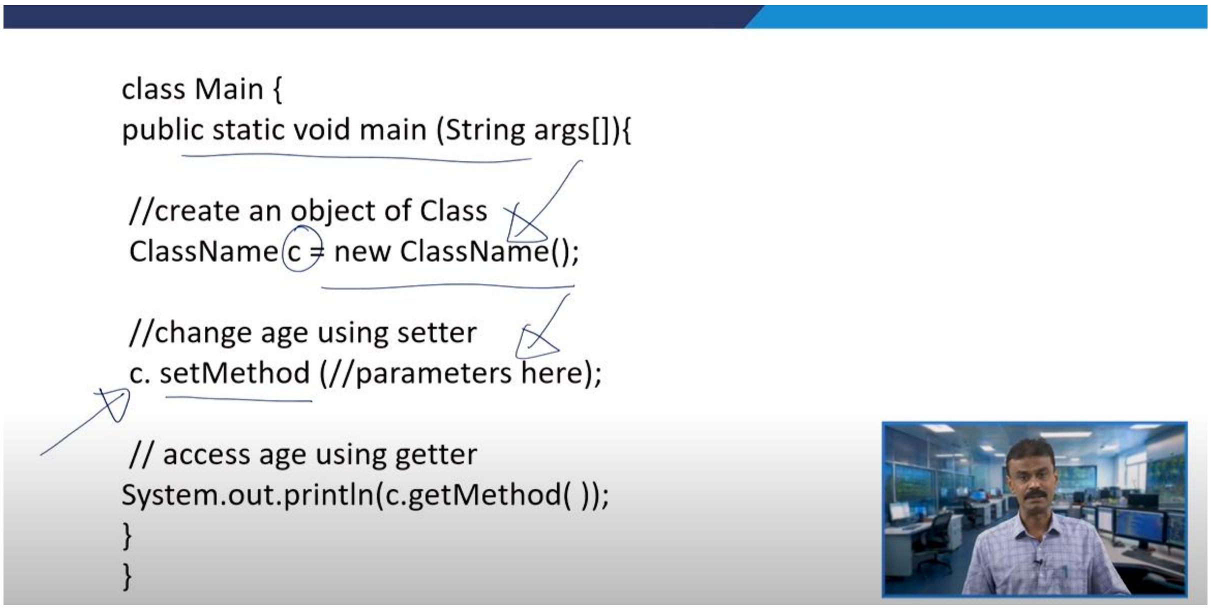
Right. So, these are all the member functions that can access the variable name. Correct. So, you are passing and then the variable name. Yes.

Fine. So, because of the private member data, this setMethod can access this, and the variable name is assigned to this dot variable name. So, this is your setMethod, and your class is over here. Your complete class is over here. So, now this is the syntax.

This is the exact syntax. So, I use the get method and set method. I use the get method, the getter and setter functions. So, whenever we talk about encapsulation, you have getter and setter methods or getter and setter member functions in C++. So, we will see.

One example we will see. All right. So, here you go. Fine. I am continuing the previous, right, the syntax.

So, here you go. In the main program, right, so this is your main in Java. I am creating an object C, right, under the class name, right. The name of the class is class name. All right.



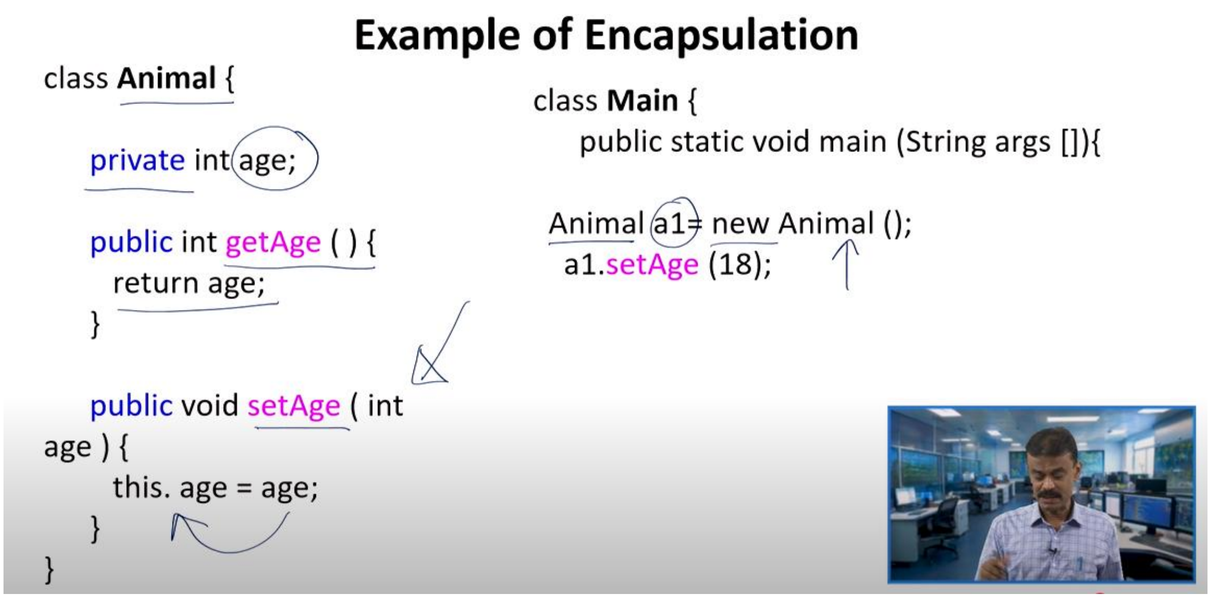
```
class Main {
public static void main (String args[]){

 //create an object of Class
 ClassName c = new ClassName();

 //change age using setter
 c. setMethod (//parameters here);

 // access age using getter
System.out.println(c.getMethod( ));
}
}
```

So, dynamically allocated memory and the constructor is class name. So, now this C is invoking setMethod, right? The object C is invoking setMethod, and you are passing the parameters, right? Whatever parameters you are passing, you have a setMethod over here, right? It goes under the variable name, and then you use this pointer. So, this dot variable name is whatever you are passing, correct? That means I can access the variable name. indirectly from outside the class, right? And then you are printing C dot getMethod. So, C is invoking getMethod. When getMethod is invoked, right? Here you go, this particular method will be called, right? So, this is how the exact syntax works in Java. We will see an example, right? So here you have class Animal, right?

Animal class. And you have one private member data called age, right? So you have one private member data called age. And here you have getAge, right? So here you have a getAge method, right?

So the getAge method is returning age. Fine, right? So it will return age. So it's a getter function. And then you have a setter function.

So, here you have a setter function setAge, you are passing int age, correct. So, you are passing all right, that is the argument, and age is assigned to this dot age. Age is assigned to this dot age. So, this is your get method and set method. So now let us go to the main class. So in the main class, right, so you have the main method, right.



Example of Encapsulation

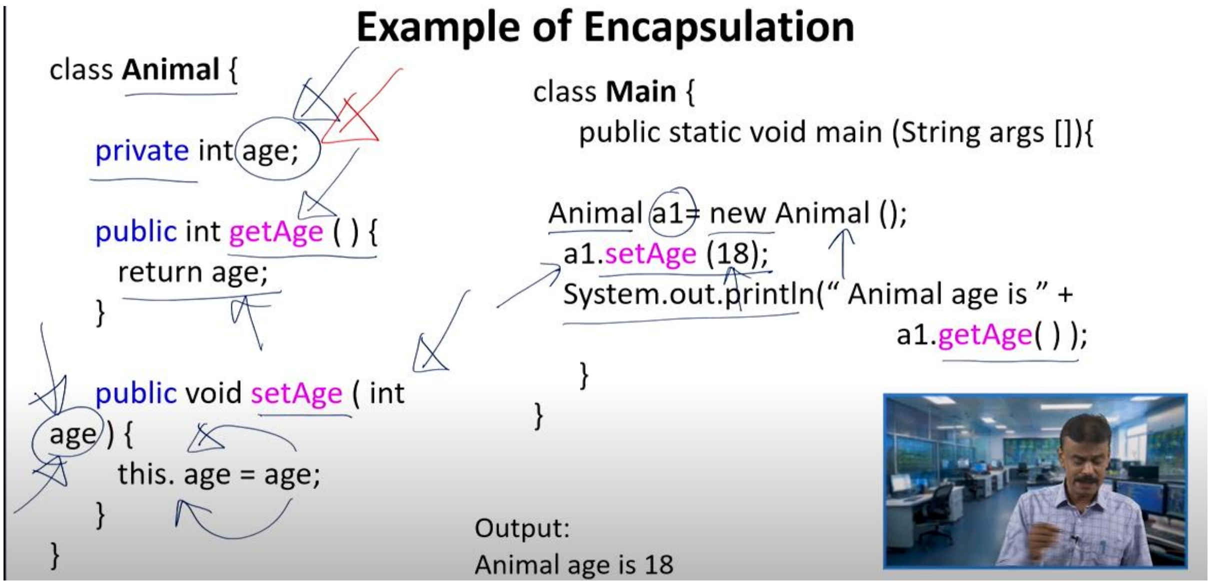
```
class Animal {

    private int age;

    public int getAge ( ) {
      return age;
    }

    public void setAge ( int
age ) {
      this. age = age;
    }
}
```

```
class Main {
    public static void main (String args []){

Animal a1= new Animal ();
 a1.setAge (18);
```

So in the main method, you are creating an object, right, a1 under the class Animal, dynamically allocating memory with a constructor Animal, okay. So you have created an object. So now this a1 object is invoking setAge. You are passing 18. setAge, when you are passing 18,

Right. I can access age, which is a private member data or data member. Right. So age is assigned to this dot age. Right.

So age, when it is having this dot age. So the meaning is your age will be 18. Right. So the private member data age is now getting the value 18 when the object is invoking the setAge method, and now in the system.out.println, you are going to print a1 is invoking getAge, right? a1 is invoking getAge, so when it is invoking getAge, alright? So when it is invoking getAge, what is happening? So getAge, when it is being invoked, it is returning age, alright? So returning age, so that means 18 should be printed.



Example of Encapsulation

```
class Animal {

    private int age;

    public int getAge ( ) {
      return age;
    }

    public void setAge ( int
age ) {
      this. age = age;
    }
}
```

```
class Main {
    public static void main (String args []){

  Animal a1= new Animal ();
   a1.setAge (18);
   System.out.println(" Animal age is " +
                                   a1.getAge( ) );

    }
}
```

```
Output:
Animal age is 18
```

So your main class is over here. That is the driver class. All right. So when you run the code, you have to get this particular output. So animal age is 18.

So if you see here, we are protecting this particular data from the outside world. Right. So this particular data from the outside world. All right. Age, which is a private member data.

So now assume that when your function, So that means the object a1 is invoking set age 18. That means 18 you are passing. So age. So here I can access age.

That means this particular function, setAge, is the public member function. In fact, the public method. I am talking in terms of Java. Public method. So, the public method here can access age now.

And then, age is assigned to this dot age. That means age is getting the value 18. And when I am printing, that means a1, the object which is invoking getAge. So, when it is invoking getAge, right?

So, return age, correct? So, return age. So, this will be returned. That means this will be printed. That means when I run the code,

So, I will get the output animal age as 18, right? So, this is how I carefully handle the data, right? So, two examples we have seen in C++ and one example in Java. So, the main idea is how we are protecting, how we are hiding the data from the external user, right? So, I cannot misuse this private member data called age.

So, I cannot access a1.age outside, right? So, we know very well previously, but this is the first time we are seeing it. So, this concept is called encapsulation, right? So, in all the cases, we are protecting this particular data from external use, right? And then in all the examples, if you look, we have the getter and setter functions or getter and setter methods.

So, the setMethod will set the value you are passing. So, that value when you are assigning in the class, right? When you are calling the member function inside the class, since that is a member function which can access that private member data, we are using it carefully, right? So, carefully we are using setAge in this particular example, a1, right? The object a1 is invoking setAge. And then what are you doing? You are passing the value 18, correct?

And then this 18, when it is calling this particular function, you have, right, int age. So, that time only, right? So, I can think about, in fact, the compiler can think about this particular data, right? This particular data member, right? And then age is assigned to this dot age.

So, age is getting the value 18, and you have another function called get function, right? Right. Get method. So, getter and setter method. So, a1 dot getAge.

So, getAge you can see it is returning age. So, therefore, we are getting the value 18. So, this is how we can access the private member data. So, the unnecessary use can be avoided. So, for example, when you may think, right?

So, what will happen if I write a1 dot age? You write it. You write a1 dot age and try to run the code, right? So, write a1 dot age and try to run the code. So, it will tell.

So, you are trying to access the private member data of the class animal. So, you will get an error. You should get an error, right? So, when I am talking about the concept of encapsulation or data hiding, it is nothing but I mean, no one should misuse the private member data outside the class.

All right. Or simply say, the data should not be used outside the class. And how carefully I can handle this particular data. So this is in my hand. Right.

So, what you can do is write some sample code. Right. Both in C++ and Java. Right. What you can do is create one class.

Try to use. Now, try to use more private member data. Anyway, we are going to see in the case study. So, what can you do? What do I suggest?

Try to use many private member data. All right. If you are using C++, use private member data. And then, try to think about the getter and setter functions. Get function and set function.

Correct? So, based on this get function and set function, you try to pass the value outside the class. Right? In the main. So, and then when you are passing the value, it will be assigned to your private member data.

So, now this function can access it, and then when you want to display, use your getter method or getter function. So, this is the concept called encapsulation that we had seen earlier. Two programs in C++ and one program in Java. So, now in the next class, what can we do? We can run some case studies.

Thank you.