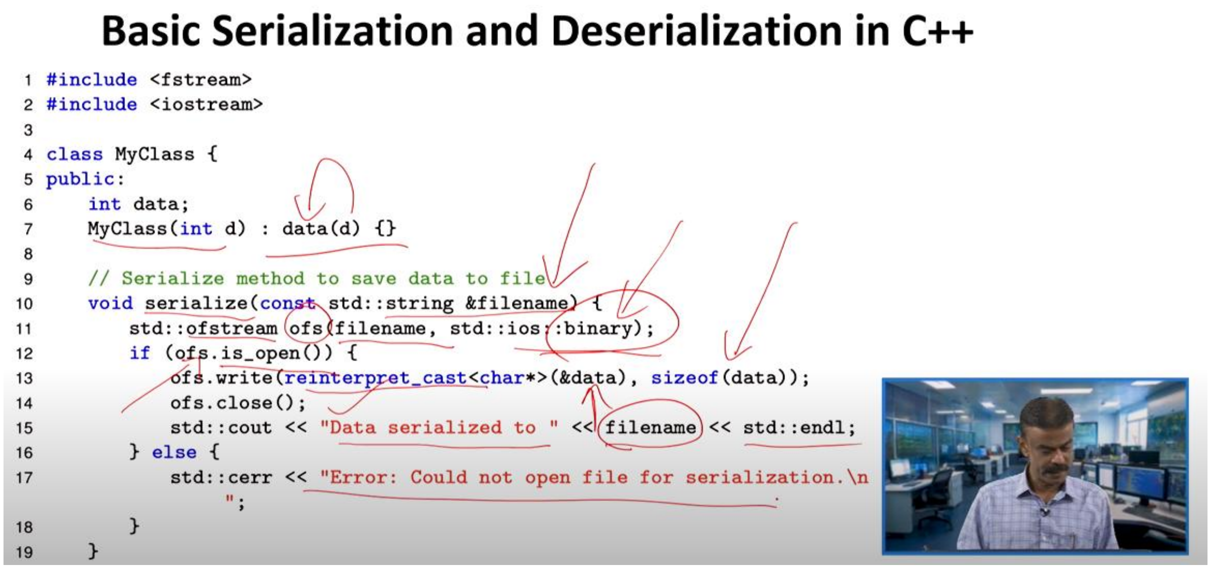
FUNDAMENTALS OF OBJECT ORIENTED PROGRAMMING

Lecture35

Lecture 35: Serialization and Deserialization



Basic Serialization and Deserialization in C++

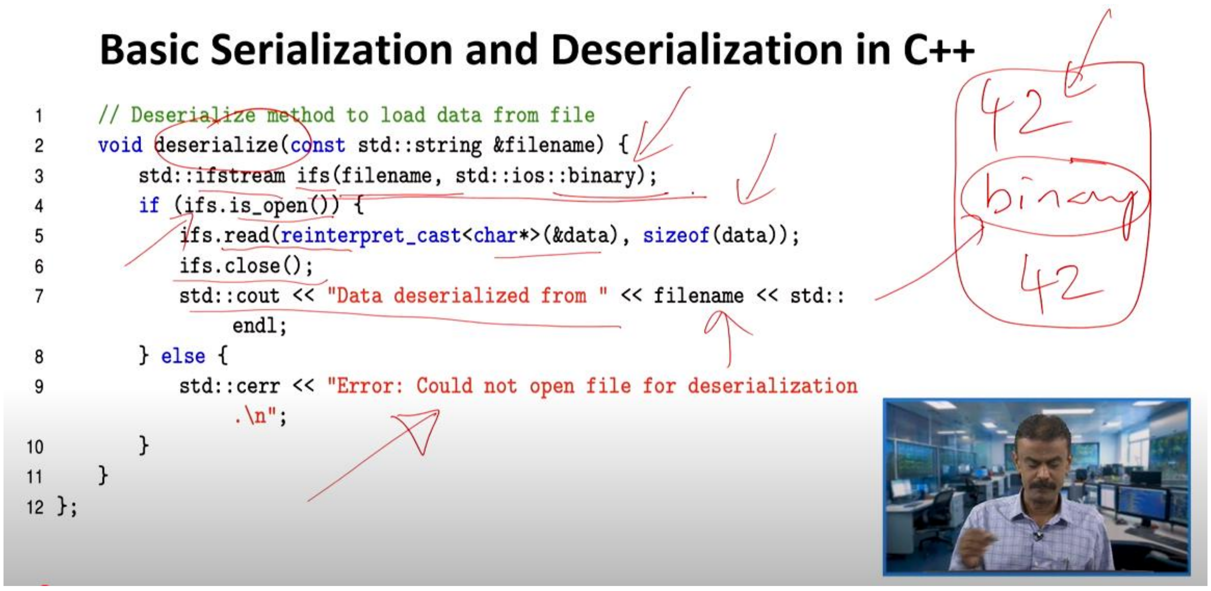
```cpp
#include <fstream>
#include <iostream>

class MyClass {
public:
    int data;
    MyClass(int d) : data(d) {}

    // Serialize method to save data to file
    void serialize(const std::string &filename) {
        std::ofstream ofs(filename, std::ios::binary);
        if (ofs.is_open()) {
            ofs.write(reinterpret_cast<char*>(&data), sizeof(data));
            ofs.close();
            std::cout << "Data serialized to " << filename << std::endl;
        } else {
            std::cerr << "Error: Could not open file for serialization.\n
                ";
        }
    }
```

Welcome to Lecture 35, File Handling. In the last lecture, we talked about serialization. Right. When you are converting the file into a byte stream and when you are reconstructing it back from a byte stream to the object or file, right. So, we call this serialization and deserialization, respectively. So, now we will see this particular example.

So, an interesting example in C++, right. So, let us have In a file, assume that we have some integer. We take some integer because we know how to convert that into binary, alright. So, we take this from the file, let us say input the value. Assume that in this particular code, I am using 42. So, that will be your input from the file. You know that because we have already seen it, and this will be converted into binary, which is nothing but serialization.

So, you are converting it into some byte stream data, alright. So, here in this case, binary And from binary, we are going to reconstruct the original value, right. So, how we are going to do it with the help of C++ is what we will see in this program. So, let us include the file stream, correct.

And then, if you look at the class, it is a MyClass. You have a member data, int data. And you have a MyClass, which is the constructor. And you are assigning to data, right. Whatever you are going to do, right.



Basic Serialization and Deserialization in C++

```
// Deserialize method to load data from file
void deserialize(const std::string &filename) {
    std::ifstream ifs(filename, std::ios::binary);
    if (ifs.is_open()) {
        ifs.read(reinterpret_cast<char*>(&data), sizeof(data));
        ifs.close();
        std::cout << "Data deserialized from " << filename << std::
            endl;
    } else {
        std::cerr << "Error: Could not open file for deserialization
            .\n";
    }
}
};
```

So, give it as an input. Or you are passing it as an input that will be copied onto data. So, now you can see the function serialize, it is having the filename, right? It is going to access the filename, serialize. We are going to access the filename, right? And if you look at ofs, which is the object under ofstream, which is converting the file, right? Whatever you are reading, right. So, assume that here we are reading some data, right.

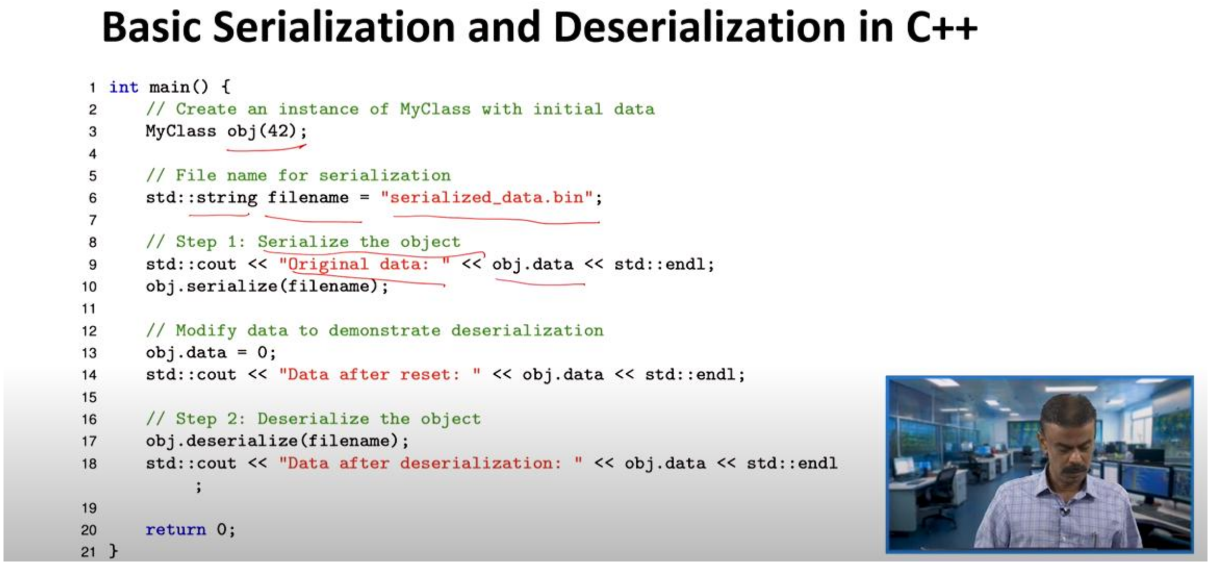
So, in this case, So, here you are having the input filename, right? Whose data type is string, the reference, right? And ofs, which is an object under the class ofstream, right. So, this filename will be converted into binary, right. So, this will be converted into binary. So, how are we going to do it?

If ofs is invoking the function to open, right? If you are able to open. So, here ofs is invoking, right? Right. If you are able to open the file. So, right.

And this is the pointer reinterpret cast. Right. So, which is nothing but the pointer of the data. Right. The pointer of the data.

Right. And then you also have the size of the data. So, it is writing. Right. Starting from the pointer, it is writing.

So, assume that if 42 is my input, the corresponding binary will be written. This is what I want to correct. This is all inbuilt std scope resolution operator ios, right? Scope resolution operator and then binary, right. So, once it is over, you are converted into binary. You are closing the file and saying that data is serialized to this particular filename, and you can see. Right. The endl. Right.



Basic Serialization and Deserialization in C++

```cpp
int main() {
    // Create an instance of MyClass with initial data
    MyClass obj(42);

    // File name for serialization
    std::string filename = "serialized_data.bin";

    // Step 1: Serialize the object
    std::cout << "Original data: " << obj.data << std::endl;
    obj.serialize(filename);

    // Modify data to demonstrate deserialization
    obj.data = 0;
    std::cout << "Data after reset: " << obj.data << std::endl;

    // Step 2: Deserialize the object
    obj.deserialize(filename);
    std::cout << "Data after deserialization: " << obj.data << std::endl
        ;

    return 0;
}
```

The new line. Right. So now, if suppose you are not able to open the file. Right. So that will throw an error message.

This will throw an error message. So this is about serialization. And now, deserialization. So exactly, you are going to reconstruct. So you are having the binary format.

Right. So, the binary format will be converted into the original value. Assume that my input is 42, and the corresponding binary will be there in the filename. And from here, the deserialization—the meaning of deserialization—I am converting back. So, again, I want to have 42, right?

So, here you have IFS, which is nothing but the object whose class is IF stream, right? So, you have a filename, and then you are going to read the binary. That is the meaning here. Exactly the opposite of what we have done in serialization, right? So, similarly, the ifs object, if it is able to open,

then it is reading. So, there we have done writing. Exactly, you are reading from the base pointer, all right, the pointer and then, what is the size of the data? All right, so this will be reading, and once it has read, it is closing and then it is printing out the data deserialized from this particular filename, all right. So, if you are not able to open, it will throw an error. So, a slightly advanced program, but once you work it out with the files—previous classes that we had seen how to use the file— So, then you can understand.

So, all are like inbuilt, right? The inbuilt modules that we are using. I hope you understood. So, one is, it is exactly going on like this. So, in the file, we have, suppose, one integer.

I take a simple example. You can play with the text, image, or something, right? So, that also you can do. In fact, one CSV file we had seen, right, how to read and write. So, you can take any input.

So, because of the easiness, we have taken 42. So, serialization, it will be converted into binary. And deserialization binary, we are giving as an input, right, file name and the binary, right. And this will give you the output 42, right. So, this is exactly what we are going to do.

We are creating an object, obj, passing 42. So, the data will be 42 now. Right. So, this is exactly what you are passing, and in the constructor, we know that the data will be 42 now. So, you have a file name under the string data type.

So, here you have the file name as serialized_data.bin, fine. So, here, first, initially serialize the object, cout original data, right. So, obj.data we know obj.data is nothing but 42, all right. So, now I am calling the function; this obj will be invoking serialize.

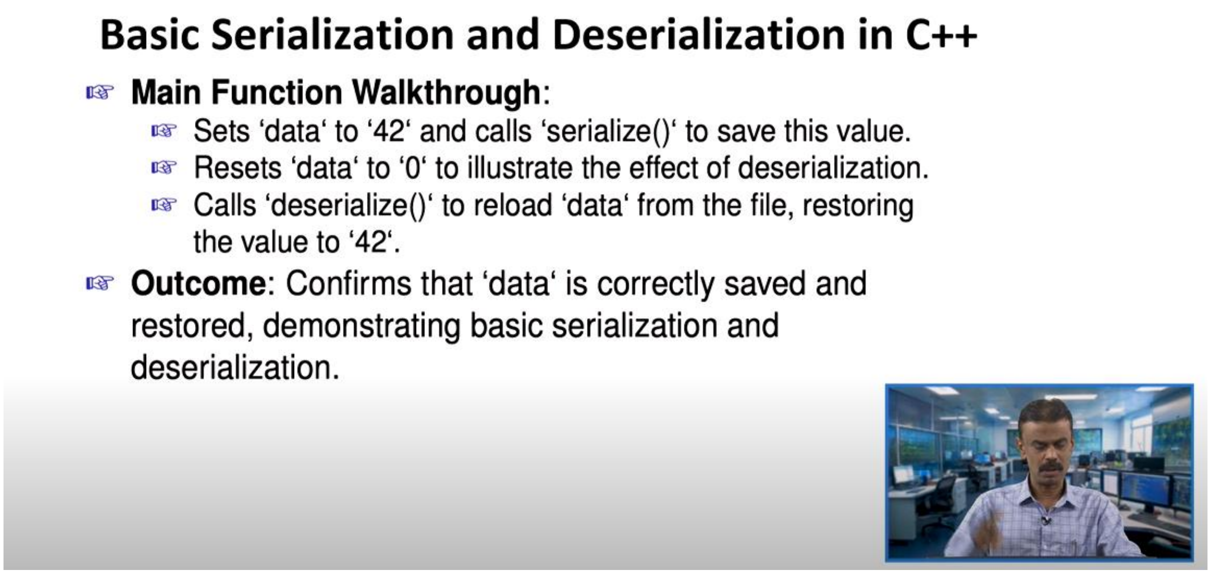
So, you are putting the filename, right? You are passing the filename, right? serialized_data.bin, you are passing, and we know that. So, I already explained over here. So, when you are passing the file name, So, this function will be executed. All right.

So the file name you have. So that will be converted into binary. That is what this particular constructor will do. Ofs inbuilt. See, ofs is under ofstream.

All right. Line number 11. So you have the file name. The bin file. And then.

So it will be converted into binary. Correct. So how exactly is it doing it? So here you go. So if you are able to open the file.

Assume that you have a file. With, let us say, 42. All right. If you are able to open the file. So, this is happening.



Basic Serialization and Deserialization in C++

- **Main Function Walkthrough**:
  - Sets ‘data‘ to ‘42‘ and calls ‘serialize()‘ to save this value.
  - Resets ‘data‘ to ‘0‘ to illustrate the effect of deserialization.
  - Calls ‘deserialize()‘ to reload ‘data‘ from the file, restoring the value to ‘42‘.
- **Outcome**: Confirms that ‘data‘ is correctly saved and restored, demonstrating basic serialization and deserialization.

It is going to write. By taking the base pointer, right. So, the character pointer will be converted into the character pointer, and one more parameter over here is the size of the data, correct? And then it is closing. So, that means the given input, 42, will be converted into binary, right. So, that means you are serializing the data into this file name, right.

So, what is the file name that we pass? It is serialize_data.bin. So, similarly, you are going to deserialize this. So, for deserializing, initially, what I do is I initialize the object obj.data equal to 0, right. So, then the data after reset, right.

So, I am resetting it to 0 initially; it will print 0, right? So, that we know that, then I am calling the function deserialize. So, that means you are passing the filename. So, deserialize. So, this will happen.

So, it is in a binary format; it is reading the binary and then converting it into the original format, reconstructing it into the original format. So, once it is done, I am printing the data after deserialization. So, now I am printing what the value is, right? It is reading the file from the binary. So, binary to original object.

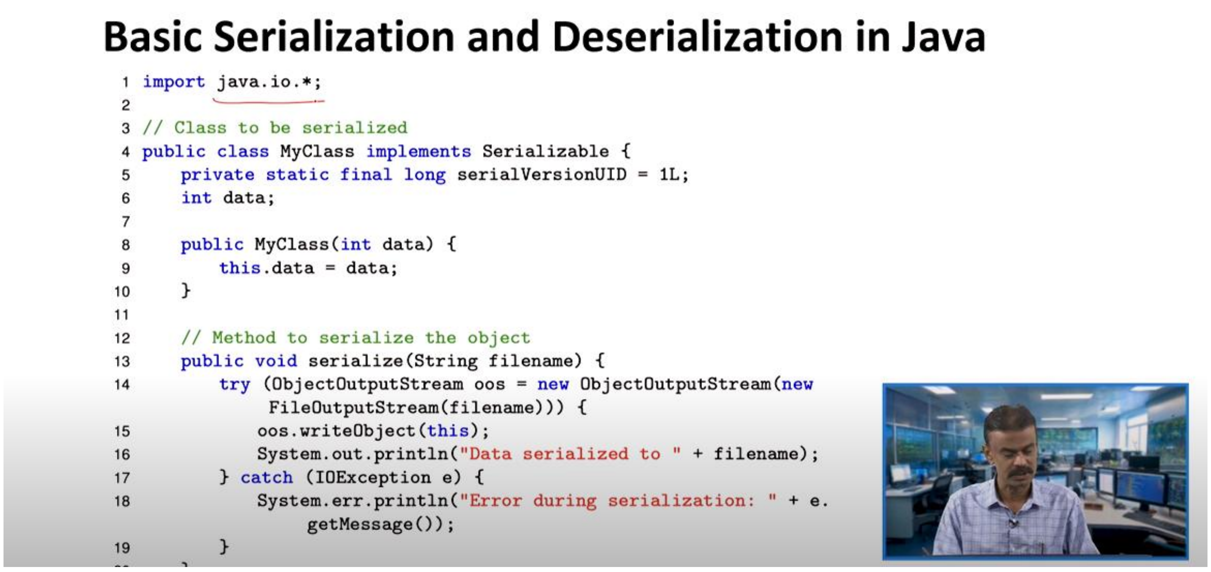
So, we know the definition, right, of serialization and deserialization. So, we have seen this is nothing but when you have, let us say, an object, the behavior of the object or the value of the object will be converted into some byte stream, right. This is nothing but serialization. And when you do vice versa, right, reconstruct from the byte stream to the nature of the object or the value of the object.

So, you can call this deserialization, right? So, let us run the code. So, this is the program that I explained. So, when you execute the code, compile and run this. So, in fact, you can see here, right?

So, this is the original data, which is 42, and the data is serialized. So, this is what you are printing, right? And that is what serialization is. Then, in deserialization, what you are doing is setting the object to 0, and then it is trying to read from the binary. Then, you can see that after deserialization, the value is 42, all right? And then you can see that bin file in that directory. So, what is the name of the file? We will just check in the program. The name of the file is serialized_data.bin.

So, you can see that, yes, serialized_data.bin, okay? You can see that fine. So, what you can do is try to open it once you are running the file. It will not be very difficult. Or you can open it using the Linux command. In fact, my advice is to run this code in Linux. And then you can easily open these sorts of bin files. So now, the idea is the number 42, which is converted into binary, and from binary, we are taking back the reconstructed original value.

So, this is all about serialization and deserialization in C++, right? So, this is the explanation. So, initially, we have a myclass and a serialization method. What we have done is we have the serialized function, which is saving the data to binary, right? And then, from binary, we are reconstructing it back. So, that is when you are converting data into binary, you call it serialization. And when you are reconstructing binary back to data, you call it the deserialization method, right?



Basic Serialization and Deserialization in Java

```java
import java.io.*;

// Class to be serialized
public class MyClass implements Serializable {
    private static final long serialVersionUID = 1L;
    int data;

    public MyClass(int data) {
        this.data = data;
    }

    // Method to serialize the object
    public void serialize(String filename) {
        try (ObjectOutputStream oos = new ObjectOutputStream(new
             FileOutputStream(filename))) {
            oos.writeObject(this);
            System.out.println("Data serialized to " + filename);
        } catch (IOException e) {
            System.err.println("Error during serialization: " + e.
                getMessage());
        }
```

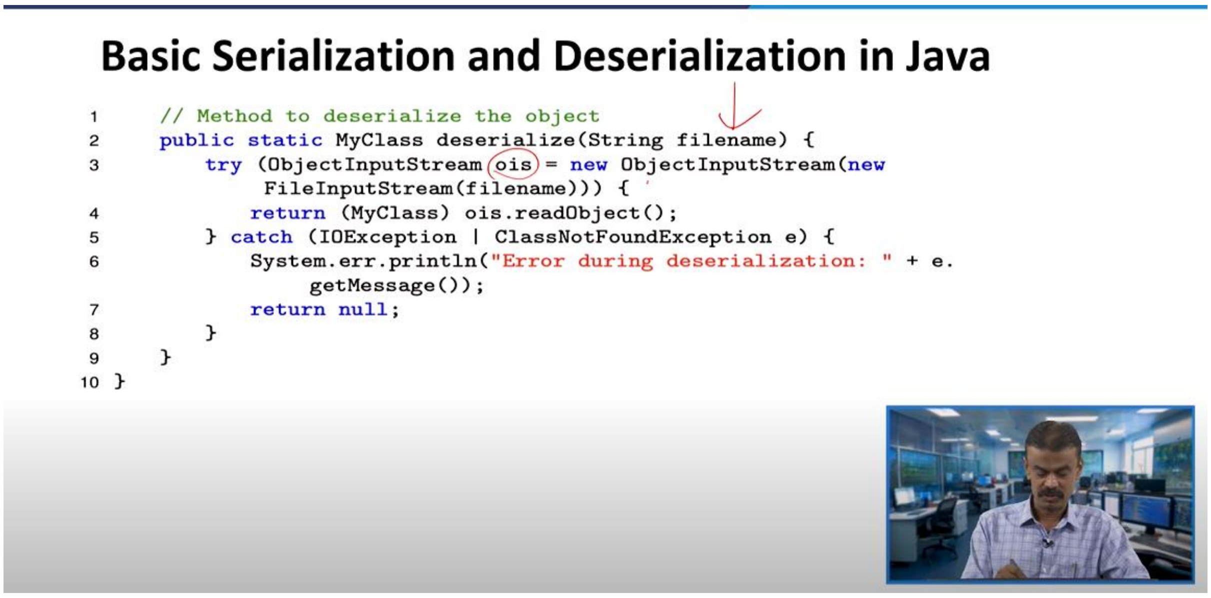
So, we have 42, and that will be saved as binary. And then what will happen? We are resetting the value to 0, and from the binary, we are converting that into the original value. So, what did we test here? So, we are converting data into some byte stream, right?

That is, in this case, binary. Now, assume that I am storing this data. I am storing this data, correct? Let us say you have several security concepts, right? So, data security, or many are talking about cybersecurity, right?

So, the security point here is information security. One particular data I am storing in a different form, right? So, you can modify this algorithm, right? So, modify this algorithm. You can save it in several forms, right?

And there is a lot of scope when we talk about security. And then what we can do is deserialization, we are able to reconstruct, all right. So, this is exactly what is going on in areas like encryption and decryption. Right, encryption and decryption, so you can, I mean, those who are really interested in this domain, you can see several research papers, right, on how and when we can encrypt the data. So, for this, this will give you some idea, right? So, once you know what you mean by serialization and deserialization, it will just open up, right?

So, you can, I mean, you are not exactly into either cybersecurity or information security right now, so this will open up some ideas. Where we can save the data in some other form, particularly the byte stream form, and from the byte stream form, how we are going to reconstruct. So, we call this serialization and deserialization in C++. So, the same concept we can do in Java. So, in the case of Java, it is completely inbuilt.



Basic Serialization and Deserialization in Java

```java
    // Method to deserialize the object
    public static MyClass deserialize(String filename) {
        try (ObjectInputStream ois = new ObjectInputStream(new
             FileInputStream(filename))) {
            return (MyClass) ois.readObject();
        } catch (IOException | ClassNotFoundException e) {
            System.err.println("Error during deserialization: " + e.
                 getMessage());
            return null;
        }
    }
}
```

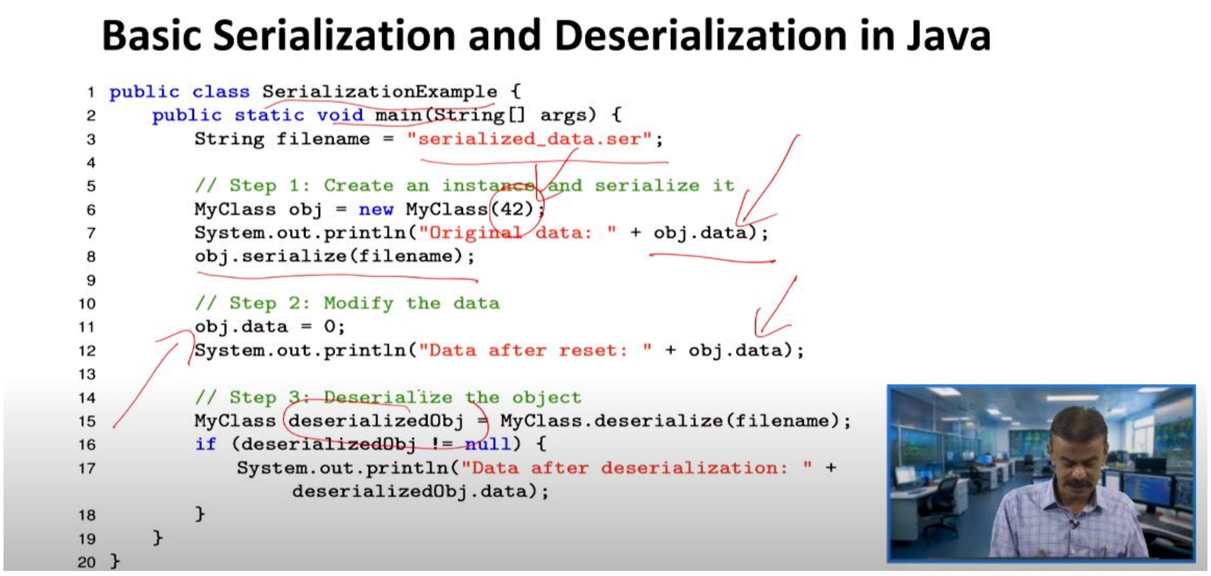
So there, I mean, we have almost written the code, right? Even though we have used some of the functionalities. So we have written the code. So here the task is slightly easier. So you can see that, right?

So I have my class. I have imported everything, java.io.star. And then you can see that my class, right? Which is implementing the already existing class or inbuilt class serializable, right? So here you have, right?

You can find. How you are writing serial version UID, you are using UID 1L, slightly the advanced program, right. But when you are comparing this with C++, it is almost similar, right. So, similar to the one we had seen, so here whatever data you are passing, you are copying into this.data, right. So, we are using the member data and this.data and then you are having.

Serializable function or you call it as a method in Java by passing the filename exactly what we have done in C++. And then you are trying out oos is an object under ObjectOutputStream, right? You are allocating memory. And then you have the constructor inside the constructor you have FileOutputStream. Which is handling the filename, right? Exactly similar to the bin file that we have used in C++, right? So now oos. The object which is invoking, right? Object, right.

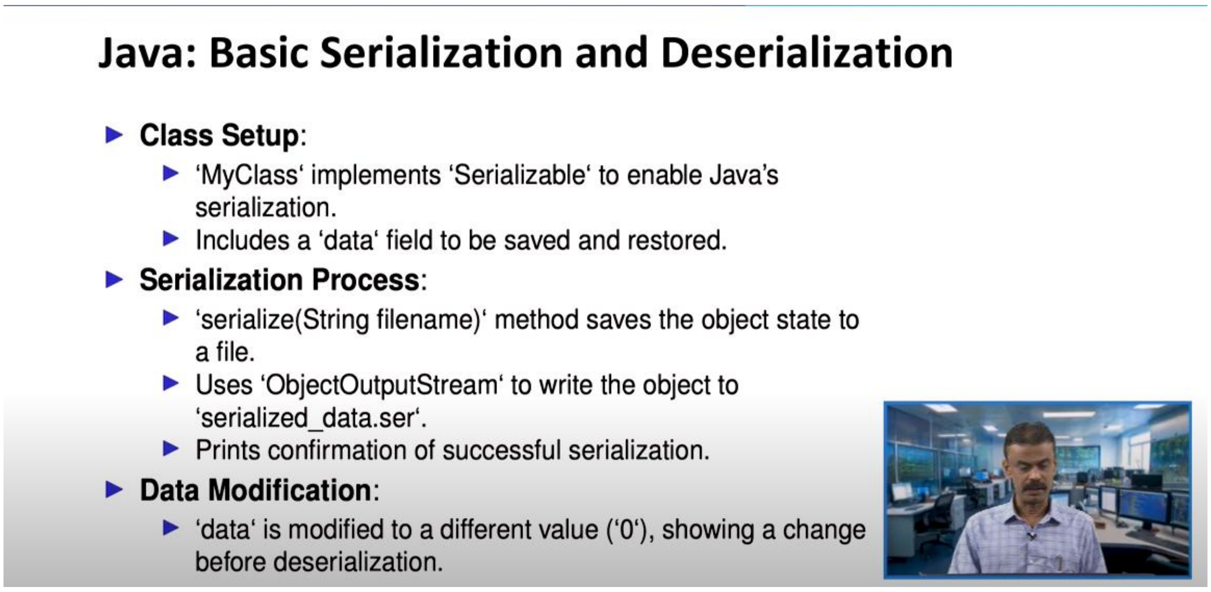
So here in this case, this in the sense I mean it will access the superclass of the object. Right, superclass of the object or you can call it as a constructor, right? So then what is happening? So you are doing everything with the help of the inbuilt function. So the filename which will be converted into some other form, so let us say it will be converted into binary form. So that is what you are writing over here, data serialized to the filename. So in fact, so when you are having this all right, writeObject this because oos you have defined in ObjectOutputStream class.



Basic Serialization and Deserialization in Java

```java
public class SerializationExample {
    public static void main(String[] args) {
        String filename = "serialized_data.ser";

        // Step 1: Create an instance and serialize it
        MyClass obj = new MyClass(42);
        System.out.println("Original data: " + obj.data);
        obj.serialize(filename);

        // Step 2: Modify the data
        obj.data = 0;
        System.out.println("Data after reset: " + obj.data);

        // Step 3: Deserialize the object
        MyClass deserializedObj = MyClass.deserialize(filename);
        if (deserializedObj != null) {
            System.out.println("Data after deserialization: " +
                    deserializedObj.data);
        }
    }
}
```

So, this is a built-in class and this will be taking care to convert into one form to another and similarly the other way around deserialization. So, you are having the filename you are passing and then so whatever we have done. So, we are going to have the reverse process create ois under ObjectInputStream constructor and FileInputStream is accessing the file and then right. So, this will help you. To read the object from the binary and it is converting right into the original data.

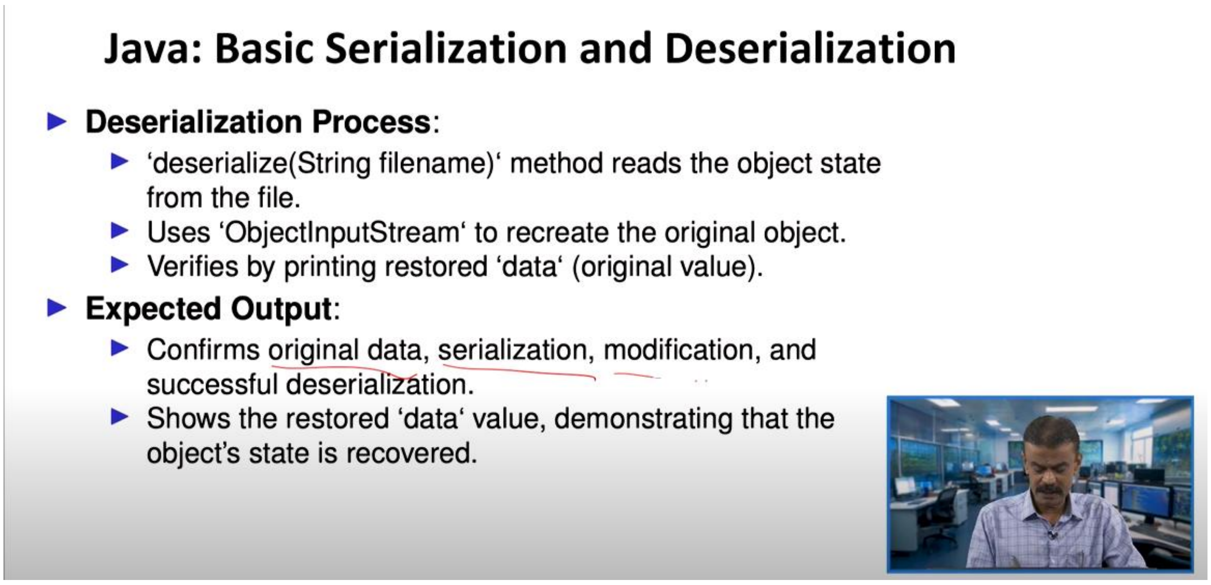


Java: Basic Serialization and Deserialization

- **Class Setup**:
  - 'MyClass' implements 'Serializable' to enable Java's serialization.
  - Includes a 'data' field to be saved and restored.
- **Serialization Process**:
  - 'serialize(String filename)' method saves the object state to a file.
  - Uses 'ObjectOutputStream' to write the object to 'serialized_data.ser'.
  - Prints confirmation of successful serialization.
- **Data Modification**:
  - 'data' is modified to a different value ('0'), showing a change before deserialization.

So if you are not able to open the file, then you will get the exception or you are not able to construct the object. You will get the exception and it will be caught, and in particular, when I am not able to open the file, see both are the same. When you are not able to open the file, you cannot create the object ois, and similarly here also. So, otherwise, you will catch the exception if you are not able to read the file. So, that means obviously you cannot create the object.

So, it will throw the exception, the exception will be caught here, and similarly, the exception will be caught here, all right. So, this is your complete MyClass, and then how you are doing in the main, all right. So, here you have a serialization example which is the driver class. Because you have the main program over here, and I have the file name serialize_data.ser, right. So, same 42, right.



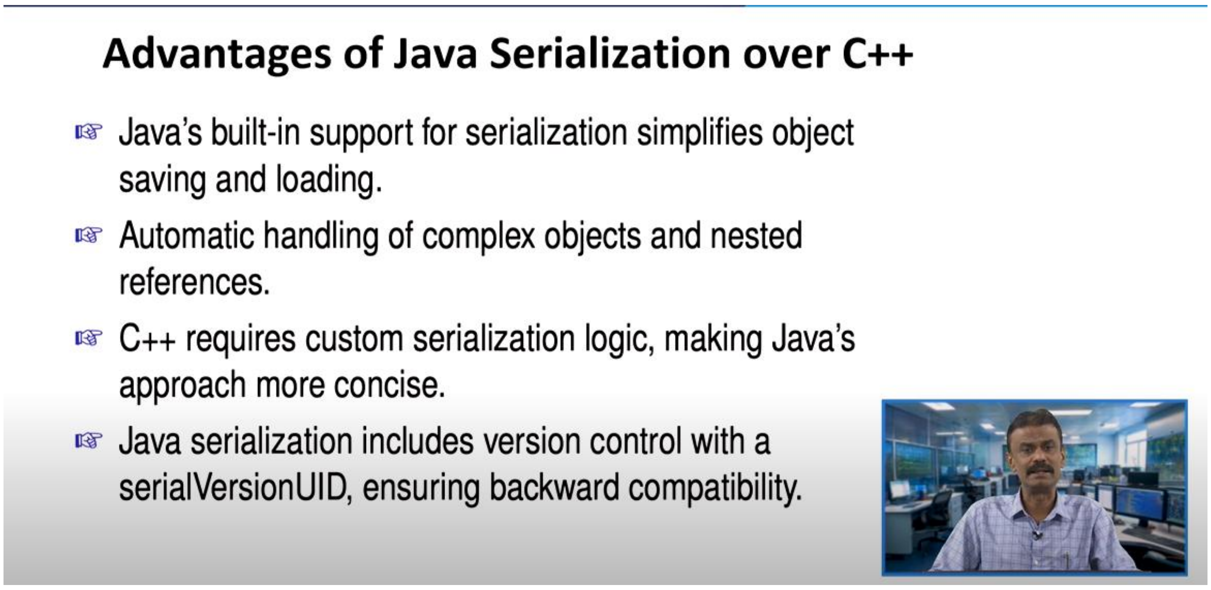
Java: Basic Serialization and Deserialization

- **Deserialization Process**:
  - ‘deserialize(String filename)‘ method reads the object state from the file.
  - Uses ‘ObjectInputStream‘ to recreate the original object.
  - Verifies by printing restored ‘data‘ (original value).
- **Expected Output**:
  - Confirms original data, serialization, modification, and successful deserialization.
  - Shows the restored ‘data‘ value, demonstrating that the object’s state is recovered.

So, I have an object obj under MyClass, you have a constructor passing 42. So, that is your original data you are printing, right. So, then you are calling the function serialize, exactly what we have done in C++, we are doing it in Java. Right, and then I am initializing obj.data which is equal to 0, right, so data after reset it will print 0, correct. And then I have an object deserialized_obj right under MyClass, right, so this MyClass is invoking the deserialized method.

You are passing the filename correctly, and here you go, right? So, this will deserialize, right? In fact, when you are calling deserialize. So, here you have the deserialize method, right. So, which will deserialize, that means initially 42 is converted into binary. So, you are reading binary here, and binary will be again converted back to 42. So, when MyClass is invoking deserialize filename.

So, this will be called, and then this is converting the binary into the original value, all right. So, this is what you are writing over here: data after deserialization, you are writing the value of the data. So, when we run the code in Java, right, so here you are getting, you can see the output. Yeah, I will run. Yeah.


Advantages of Java Serialization over C++

- Java's built-in support for serialization simplifies object saving and loading.
- Automatic handling of complex objects and nested references.
- C++ requires custom serialization logic, making Java's approach more concise.
- Java serialization includes version control with a serialVersionUID, ensuring backward compatibility.

Original data is 42. Data serialized to serialized data.serve. We will see the file name. Data after reset, yes, it is 0. Line number 12.

And then, after deserialization, the data becomes 42. Can we see the SCR file? Yes. Alright. So, you can see the SCR file.

So, this is the meaning of serialization and deserialization, and we have seen examples in Java as well as C++. Right. So, here it goes. So, you have MyClass. So, you have done for

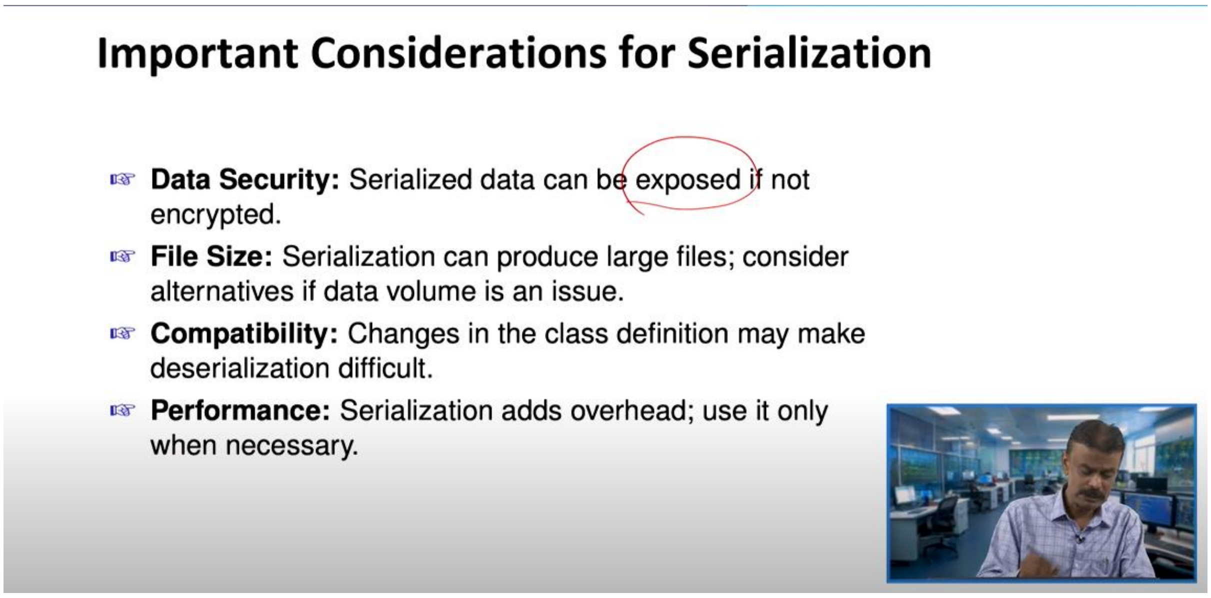
the serializable, all right. So, the data that you have in the filename, right under the file, is correct. So, that will be converted into a byte stream, all right, that will be converted into, let us say, binary, exactly what we have done in C++. So, here we are doing it in Java, all right. So, it is being converted into binary.

And then, right, in fact, it was writing this into the serialize_data.ser. All right. And in fact, when you want to open this again, my advice is to try to use Linux, right, the terminal. In the terminal, use the commands to compile and run it. And then see this particular file will be created, and you can view the file.

Correct. So in this particular file format, what it is doing is writing data into binary form. Okay. Or I can call it a byte stream. All right.

And then we are getting the confirmation that successful serialization has been done. All right. So now what do we do? We are going to reconstruct it back. So for that, what are we doing?

So, we are modifying the data. All right. So, when we are modifying the data, you are setting the value to 0. Data is equal to 0. And then, you are calling the deserialization function.



Important Considerations for Serialization

- **Data Security:** Serialized data can be exposed if not encrypted.
- **File Size:** Serialization can produce large files; consider alternatives if data volume is an issue.
- **Compatibility:** Changes in the class definition may make deserialization difficult.
- **Performance:** Serialization adds overhead; use it only when necessary.

Deserialization method. All right. So, when you are calling this method, This is using ObjectInputStream, right, to recreate the original object. So, that is exactly what is happening.

So, in the previous case, it is using ObjectOutputStream to write the object to serialize data dot ser, OK. So, here in this case, the deserialization case. So, you can see that, right. So, the deserialization case. So, here you can see ObjectInputStream, right.

So, that is exactly what we are saying over here. So, this is using an object input stream to recreate the original object, right? And then it is verifying by printing the restored data. That means the original value, in fact, we had seen when you look at the output we had seen, all right. So, it confirms we have the original data initially. So, then we have serialization, and then you are modifying the data to 0.

And then, successfully, it is deserializing. So, this is the meaning of serialization and deserialization in both C++ and Java. So, you are going to have the value. So, you are keeping this value in some other format, particularly byte stream format. It has got several applications.

Several applications in the context of signal processing, image processing, or computer vision. So, these kinds of advanced research areas, all right, particularly the concept of security, right? I want to keep this data in some other format. So, that the intruder, I mean, may not know even if somebody hacks, I mean, what this data is. Assume that I serialize and put it in some other format. So, here we have taken a simple example of binary, right? So, you can modify your algorithm, right? One image is given.

I mean, there are several algorithms. I mean, one cannot see what is going on, all right? In the encryption process, the given image will be converted completely into a different format, right? So, exactly here, I mean here, because we are at the initial level, we take some number, and the number is converted into binary. But when this is very well known, I mean, from the binary, there will be a guess, right? So, I am able to open these files and see the binary values. All right, I will try to convert that into an integer. All right. So, whereas your algorithm should be robust.

So, when you are converting into a byte stream, the intruder should not know what exactly it is. All right. So, you have several scopes. I mean, you can work with cryptography, or you can work with watermarking. All right.

So, like these advanced topics. So, this is going to be your right, the I am opening the front gate. Right. So, you can explore. So, what we have done is we have handled a simple example, right?

So, now you can expand your knowledge on this. So, from a language point of view, this is called serialization and deserialization. So, in fact, I mean we have seen this in C++ code. So, we have done it. So, what we have done there, I mean, you require manual conversion, right?

So, manual conversion of object data to bytes, whereas here you have used several inbuilt right, inbuilt methods, ObjectOutputStream, ObjectInputStream, right. So, these we have done in the case of Java. So, in fact, Java has an edge over C++ in the serialization concept, right. So, because here we are completely using the built-in support, in Java we are using the built-in support completely.

Whereas, in C++, what we have done, you have to make your own logic, which sometimes may not be correct, right. So, here we are taking all the inbuilt methods or inbuilt functionalities, and then we can do it elegantly and even better than C++. In Java, we have automatic handling of complex objects and also nested references. When I am talking about Java serialization, so this includes version control with the UID. In fact, we use UID in one of the lines, right? You can look at line number five, right. So, this we have not done in C++, right? So, the reason is it ensures backward compatibility. So, these are all the important considerations for the concept of serialization, so data security So, once you are converting from one form to another form and you are keeping it, that is what I talked about encryption.

So, in fact, one of the examples I mentioned. So, you are converting 42 into, let us say, the binary. So, the intruder can easily try to guess. So, the first thing I will do is try to find out what the decimal equivalent is. So, that is what can be exposed.

That is the meaning. Serialized data can be exposed. So, that is why I also talked about encryption algorithms, alright. So, if you do the encryption along with this. So, for example, I have converted it into binary, and from the binary, I am doing some algorithm. You have the Chinese remainder theorem, alright, and several other encryption algorithms. Fine, you use it.

And then you can still make the data complex or secure, alright. If you are converting, let us say, the data into binary, I mean, it can be easily exposed. And file size, serialization can produce large files. Yes, of course, 42, when I am converting it into binary, you know that, right. So, there itself, the size will be large, right.

So, what you can do, right. Suppose I want to make it equivalent to or less than the original data. So, for this, you have to explore further areas, particularly encryption, watermarking, or cryptography. So, but whereas right now, when you are converting. So, that is an example 42 when I am converting into, let us say, binary, and then the binary file you are storing.

So, we know that this binary, because you have ones and zeros, you are representing with the digits. So, this will have more. So, this volume will be more. So, this is a simple example we are taking, but still, when you are storing 42 or ones and zeros. So, you know that, right.

So, the file size. So, it will produce large files, compatibility, right. So, suppose you are making some changes in the class definition. So, this will create a problem when you have the deserialization, right. So, the compatibility and the performance.

So, serialization adds overhead, of course. In this example itself, 42 is being converted into binary or any data is being converted into a byte stream. So, whenever it is required, we have to use this. So, otherwise, I mean, you have now understood the concept of what is meant by serialization or what is meant by deserialization. So, with this, I am concluding the file handling chapter. Thank you very much.