

# FUNDAMENTALS OF OBJECT ORIENTED PROGRAMMING

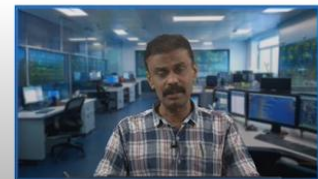
## Lecture34

### Lecture 34: File Handling - Character, Line, and CSV File Reading

#### Example: Writing Formatted Data in Java

- 👉 String.format() method is used to create a formatted string.
- 👉 It specifies that floating-point numbers should be formatted to two decimal places.

```
1 import java.io.FileWriter;
2 import java.io.IOException;
3
4 public class WriteFormattedData {
5     public static void main(String[] args) {
6         try {
7             FileWriter writer = new FileWriter(
8                 "data.txt");
9             writer.write(String.format(
10                 "Value: %.2f\n", 123.456));
11             writer.close();
12         } catch (IOException e) {
13             e.printStackTrace();
14         }
15     }
16 }
```



Welcome to Lecture 34: File Handling. So, in the last lecture, we saw writing formatted data in C++, right? So, here in Java, it is almost similar. So, only the syntax will be slightly different, right? So, here we use the String.format method.

So, this method is used to create a formatted string, right? So, in fact, it specifies the floating-point numbers that should be formatted to, let us say, in this particular example, two decimal places. So, this is what we have done in C++ also. So, here also, we do it, then we can change it, right? In the program, we can change it, as you have done in C++.

So, similarly, you change it. In fact, in line number 10, instead of 0.2f, you put 0.1f, right? Because it is 0.456, yeah, 0.1f. So, now we have a class, WriteFormattedData, which is a driver class, and you have void main. Correct, and you have a try-throw-catch. So, in try, you have a writer object under FileWriter, right? So, here you have java.io.FileWriter and java.io.IOException.

So, now you have allocated memory with a constructor, and then this will handle data.txt, right. So, it will try to open it. So, if it is not able to, it will throw an exception,

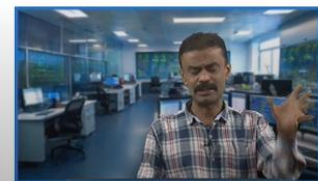
and it will be caught in line number 12. So, now the writer is invoking write, right. So, that string will invoke the dot format, which is what I said, right? String format.

So, it is used to create a formatted string, right. So, here string dot format, and then here you have the value. So, you are printing this, and then you know, percentage point to f in fact c. So, see, you can use the C syntax percentage 0.2 f, which means after the decimal, you are going to represent this in two precision, which is exactly what we have done in C++, right? Set precision. So, 123.456, all right.

## Java: Writing Formatted Data

**Task:** Write a list of names and scores to output.txt, aligning the data in columns.

```
1 import java.io.*;
2 import java.util.Formatter;
3
4 public class FormatWriter {
5     public static void main(String[] args) {
6         try (BufferedWriter writer = new BufferedWriter(new FileWriter("
            output.txt"))) {
7             Formatter formatter = new Formatter(writer);
8             formatter.format("%-20s %10s%n", "Name", "Score");
9             formatter.format("%-20s %10d%n", "Alice", 95);
10            formatter.format("%-20s %10d%n", "Bob", 87);
11            formatter.format("%-20s %10d%n", "Charlie", 78);
12            formatter.close();
13        } catch (IOException e) {
14            e.printStackTrace();
15        }
16    }
17 }
```



So, let us see how it is going to print, and then close the file. So, let us quickly go to the Java console, all right. So, here you have the format 123.456, let us run the code I explained already. So, if you run the code, yeah, it is successfully running.

And we will see which file we have to see. Just go to the Java code output.txt, right? So, we will just open output.txt here. You go, yes, the value is now 123.46, fine. So, similarly, right, set width 20, set width 10, right? The same program if you write in Java. So, I have java.io.\* and then Formatter, right? Utility Formatter. And then I have a class, right? This class is FormatWriterDriver class, the main method. Writer is the object under BufferedWriter.

As usual, we have used several times, and then you have a memory allocation with the constructor and FileWriter, which is handling output.txt. So, we are able to open fine. Line number 7 to 12 will be executed; otherwise, there will be an exception. It will be caught in line number 13. So, now When you look at, right, so Formatter, that is the object under capital Formatter, which is the class. You have dynamic memory

allocation with the constructor, and then you are having this object writer, correct? So, now the formatter, this object is invoking the format method.

So, here when you put percentage, right, dash 20s, right, and then percentage 10s, right. So, for name, it is allocating 20 characters, and for score, it is allocating 10 characters. So, that is the meaning, right. So, let us see when you are running the code. The rest is clear. Format.close, yes. We have the names: Alice, Bob, Charlie, and we quickly go to the Java console, right.

So, in the Java console, what we can do is we will run this particular code, right? So, here you go. When I run the code, it is successfully running, and then we have to open output.txt. So, in output.txt, you can see the output. All right. So, name allies score is 95, and so on.

All right. So, this is what we have to print. And then you can see in both the languages, C++ and Java, we can see how you can format the data. Right. Format the output file.

So, this is the brief explanation about the code. So, the output.txt. So, we need some alignment. Right. So, we are making two columns, and each will have some set width.

So, here we are using the C code percentage right dash 20s and similarly percentage 10s. So, one is for the name, we have 20 characters, and for the score, we have 10 characters. So, that 10 characters in this is up to 10 characters for the score and a name up to 20 characters. So, that is the meaning, right? And then we use the BufferedWriter, which is the class under this we have used the object. So, that means it will deal with output.txt.

So, if it is able to open this, not a problem. That means if it is able to deal with this output.txt, not a problem. The rest of the code will run; otherwise, it will throw the exception and go to the try block. And then we have used the class Formatter, under this we have an object small f, right? Formatter. So, then we know what formatter did. We have allocated the memory and then, all right. So, when it is invoking dot format. So, we have 20 characters for the name and 10 characters for the score, all right? And then we have used the new line, etcetera, right.

Here you can see percentage n, all right? Percentage 10s, and then since there is an integer, we use percentage 10d. So, 10 characters, right. So, 10 spaces, right. So, this worked, in fact. We have got so every line we have the name, score, first line, and then, in fact, we have got the output like this, right. So, name Alice score 95,

right? Name Bob score 87, and so on, right? We have got all these outputs, so this is how we can write the formatted data.

Into the file, right? So this is applicable both in C++ and in Java, right? So far, we have read, let us say, in the file you have a list of characters. So we have done line by line, right? Every line we read. So now you may ask the question: Is it possible to read character by character? Yes, you can do that, right? So let us consider `myfile`, which is an object under input file stream. So, we are including `fstream`, that means both input and output, and `iostream` and `string`, the usual one, right. So, first it is printing: reading character by character.

So, that means `myfile`, initially line number 6, `myfile` is handling `example.txt`, fine. So, now we have one variable `c` under data type character, ok. If `myfile` is invoking `open`, if you are able to open this `example.txt`, and while `myfile.get(c)`, right? `myfile` is invoking `.get` character, right? `c` is a character.

So it is going. If I am able to open and when it is reading character by character, right? So it goes, right? So `myfile.get(c)`, which is true. Right.

## Java: Writing Formatted Data to File

- 📖 **Purpose:** This code writes names and scores to 'output.txt', aligning them in two columns.
- 📖 **BufferedWriter writer:** Opens 'output.txt' in write mode with buffering for efficient writing.
  - 📖 The file is automatically closed at the end of the 'try' block.
- 📖 **Formatter formatter:** Wraps 'writer' in a 'Formatter' for formatted output.
  - 📖 'Formatter' allows for structured data alignment similar to printf-style formatting.
- 📖 **formatter.format:** Formats the output using specified width and alignment:
  - 📖 `"%-20s"`: Left-aligns the string in a 20-character width for the "Name" column.
  - 📖 `"%10d"`: Right-aligns the integer in a 10-character width for the "Score" column.
  - 📖 `"%n"`: Inserts a newline after each formatted line.



So, then if `c` is equal to a new line. So, we have to print here. We are printing just an example. We are printing the new line, right? Else, if `c` is equal to a blank space, okay? Else, if `c` is equal to a blank space, we are printing what? We are printing. So, in that case, we are printing one underscore, right? So, what we are doing, whenever suppose I love India.

Right. So, I blank space love blank space India. If it is there. So, we are going to have the output I underscore love underscore India. Right.

So, that is the case for the blank space. Otherwise, the character will be printed. cout C. So, myfile dot close. It is not a very difficult program. So, you can see that we will go to the


C++ console, and we can see example.txt, yes. So, we have several here: hello world, this is an example. In fact, we have used this, and you can see a new appended line, right? So, now we will see how we are going to print this. So, now let us go to the C++ program.

So, we opened example.txt. And we had seen, right? So, the same code I explained. So, it is continuing after line number 22; it is continuing. So, cout is reading line by line, right?

So here, one case is reading character by character. Another one is reading line by line. So, in that case, it is a usual one. This we had already seen, right? So, you know that we have already done this.

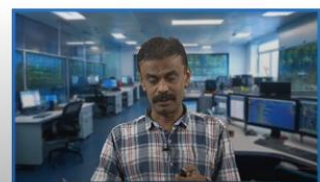
So myfile will invoke open. So example.txt is what we are handling. So, if myfile is open, if you are able to do that. So, put a line variable under string initially, and put the integer variable line number equal to 1. So, while your getline myfile, that means you are handling it right; myfile is opening, myfile.open is handling example.txt, right?

## Java: Writing Formatted Data to File

 `formatter.close()`: Closes the formatter and flushes data to the file.

 **Result:** The file will contain a formatted list:

Name	Score
Alice	95
Bob	87
Charlie	78



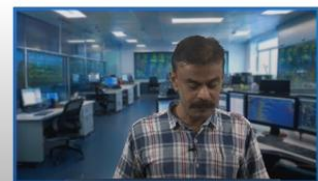
And then, for example, the first line I am reading, right? You are putting it in that line. First line, whatever, let us say, I love India. So now line is equal to I love India. You know that we have seen in the last class, right? So now we are printing the line.

It will give line number one as what? Whichever is the line number one will be printed. So one is character by character. Another one is line by line. So let us see how this works.

In fact, we are printing, right? I mean, cout one is one. It will print. Right. Another cout statement also.

## Example: Reading Character by Character in C++

```
1 #include <fstream>
2 #include <iostream>
3 #include <string>
4
5 int main() {
6     std::ifstream myfile("example.txt");
7
8     // Character-by-character reading
9     std::cout << "Reading character by character:\n";
10    char c;
11    if (myfile.is_open()) {
12        while (myfile.get(c)) {
13            if (c == '\n') {
14                std::cout << "\\n"; // Indicate newline characters
15                explicitly
16            } else if (c == ' ') {
17                std::cout << "_"; // Replace spaces with underscores for
18                visibility
19            } else {
20                std::cout << c;
21            }
22        }
23        myfile.close();
24    }
```



So that means you have several cout statements. Line numbers 14, 16, 18, and then 32. So when we run the code, one is character by character. Another one is line by line. Let us run the code.

All right. So when you run the code, you can see that. All right. So initially, you can see everything with an underscore. First, writing character by character.

And then you can see hello, underscore world. Because of the blank space. So, all the blank spaces we have replaced with underscores. Correct? So, that means we have read the file.

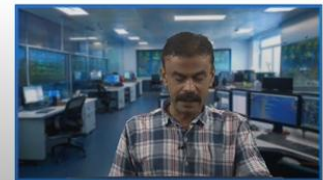
Right? We have read the file and then we are manipulating it. So, when you are going character by character, replace. New line should be. Right?



You are writing slash new line and then you have underscore. If you have a big blank space, you can see one particular case underscore is also big. And the second case, line by line. So, what is line 1, line 2, line 3? So, this we are printing.

## Example: Reading Character by Character in C++

```
1  std::cout << "\n\nReading line by line:\n";
2
3  // Reopen file for line-by-line reading
4  myfile.open("example.txt");
5  if (myfile.is_open()) {
6      std::string line;
7      int lineNumber = 1;
8      while (std::getline(myfile, line)) {
9          std::cout << "Line " << lineNumber++ << ": " << line << std::endl;
10     }
11     myfile.close();
12 }
13
14 return 0;
15 }
```



So, we can manipulate both character by character as well as line by line. I hope it is clear for everyone. So, the character by character, this I already explained. So, when you are using the while loop character by character. So, here in this case, whenever it is true.

So, get character sub character occurs, and then the loop is executed. And then you are printing with an underscore as well as the new line point of view. All right. And in the second case, it is reading line by line. So this is the second loop: while getline myfile, line.

## Example: Reading Character by Character vs Line by Line in Java

```
1 import java.io.*;
2
3 public class FileCharacterReader {
4     public static void main(String[] args) {
5         try {
6             // Character-by-character reading
7             System.out.println("Reading character by character:");
8             try (FileReader fr = new FileReader("example.txt")) {
9                 int i;
10                while ((i = fr.read()) != -1) {
11                    char c = (char) i;
12                    if (c == '\n') {
13                        System.out.print("\n");
14                    } else if (c == ' ') {
15                        System.out.print("_");
16                    } else {
17                        System.out.print(c);
18                    }
19                }
20            }
21        }
22    }
23 }
```



So that means it is reading line by line. So, character by character and line by line. So the same code when we deal with in Java. Right. We have seen C++.

Inside the case of Java. So, you have the object fr under file reader. All right. So, you have allocated the memory, and then you have a constructor. So, it is handling example.txt.

So, while i is equal to fr.read, that is not equal to minus 1, all right. So, whenever you have come to the end of the file, all right, the end of the file, the last character. So, the object fr, which is invoking read, right, assigning to i, and if it is not equal to minus 1, right. So, suppose I have the same example: i blank space love blank space India, correct.

So, suppose it is reading the first character i, right, reading the first character i. So, it is converting into the integer ASCII value of that capital I, right. What is the ASCII value of capital I? Definitely not equal to minus 1, right. So, if it is not equal to minus 1, it is continuing. This is the idea.

If it is reaching the end of the file, right. So, then it is a null character. So, it is in Java. So, it is assigned to minus 1, right. So, you cannot use anything else.

Even minus 2 does not work, right. So, in Java, by inbuilt. So, they have used minus 1. So, that means you are at the end of the file. So, it is equal to minus 1.

So, the while loop will not be executed; you have to terminate, right. So, that is a termination case. Otherwise, the rest of them are clear. So, the ASCII values are converted into characters, and if a character is a new line. Put the new line.



## Example: Reading Character by Character vs Line by Line in Java

```
1 // Line-by-line reading
2 System.out.println("\n\nReading line by line:");
3 try (BufferedReader br = new BufferedReader(new FileReader("
4     example.txt"))) {
5     String line;
6     int lineNumber = 1;
7     while ((line = br.readLine()) != null) {
8         System.out.println("Line " + lineNumber++ + ": " +
9             line);
10    }
11 } catch (IOException e) {
12     e.printStackTrace();
13 }
14 }
```



System.out.println new line. Otherwise, System.out.print underscore. Whenever you have a blank space. It is underscore. Otherwise, print the character.

Right. The same thing. Or line by line. So you have br. Which is an object.

The BufferedReader. Under the class BufferedReader. Allocating memory with a constructor. And then the FileReader. Right.

So that will handle example.txt. So if you are able to open. Well and good. You are able to access, well and good. So, otherwise, it will throw an exception.

So, here you have a line variable under string and, as usual, line number you put 1, which is the integer variable. So, now while br.readLine, right? So, this we have done. So, br is invoking readLine, put it in the line. So, complete line.

So, put it in the line, right? For example, the first line I and I blank space love blank space India. So, line is now I love India, all right. So, which is not null?

True. So, that is printing, right? So, the line will be printed with the blank space and whatever the line number is, first, then only it will be incremented, line number 1, all right. And then you are printing what is that line? The same, whatever we had seen in C++.

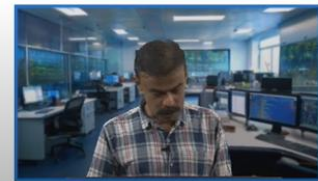
So, that is in Java, right? So, this is what I explained. Correct. So, character by character and line by line. So, maybe we will run the code.

So, we will go to the Java compiler. So, you can see all the output, right? So, reading character by character, we are getting an underscore, right? And whenever you have a new line, it is indicating a slash n, correct? And next, reading line by line, you can

see line 1: hello world, line 2: this is an example file, right? So, like this, we have the output. So, this is how we can read.

## C++: Reading from a CSV File

```
1 #include <fstream>
2 #include <iostream>
3 #include <sstream>
4 #include <string>
5
6 int main() {
7     std::ifstream csvFile("data.csv");
8     std::string line, cell;
9
10    if (csvFile.is_open()) {
11        while (std::getline(csvFile, line)) {
12            std::stringstream lineStream(line);
13            while (std::getline(lineStream, cell, ',')) {
14                std::cout << cell << "\t";
15            }
16            std::cout << std::endl;
17        }
18        csvFile.close();
19    } else {
20        std::cerr << "Unable to open CSV file";
21    }
22    return 0;
23 }
```



Character by character, as well as line by line. I hope it is clear for everyone. So, let us consider one more example where we are reading from a CSV file. Right. So, comma-separated values file, CSV stands for.

So, I include fstream, file stream, input-output stream, and string stream. sstream stands for string stream, and then the usual string. Right. So now we have csvFile as an object. So, which is dealing with data.csv under the class ifstream, right?

I mean to say, input file stream. So, we have line and cell as a string, the usual string data type. So now, if csvFile, which is an object, is able to open, right, this data.csv, it is going into the while block, right? So, while you have getline. So, getline, I mean, you may have, let us say, 10 lines.

We have seen several programs. So, the first line will be copied into the line, that is the meaning, right? So, the first line will be copied into the line. So, CSV file. So, we know that line number 7 is an object, and the line getline, the first line will be copied into the line, that is the meaning.

## Java: Reading from a CSV File

```
1 import java.io.*;
2
3 public class CSVReader {
4     public static void main(String[] args) {
5         try (BufferedReader reader = new BufferedReader(new FileReader("
6             data.csv"))) {
7             String line;
8             while ((line = reader.readLine()) != null) {
9                 String[] cells = line.split(",");
10                for (String cell : cells) {
11                    System.out.print(cell + "\t");
12                }
13                System.out.println();
14            } catch (IOException e) {
15                e.printStackTrace();
16            }
17        }
18    }
```



So, now we have an object line stream under string stream, we have included line number 3, sstream, all right. The line. Let us assume the first line. V is a while loop. It will go to the first line.

So now, line. Assume that the first line is something. Right. With a string. So that particular variable will be stored under line.

Right. Those particular values are stored under the variable line. Right. And lineStream is an object of the class string stream. Right.

We have already included sstream. Okay. So now you have one more while statement. So this while statement you can see that getline. So getline you have an object lineStream here and then you have a string cell which is separated by a comma delimiter.

So that means it will split the current line into an array of cells using the comma delimiter. That is the meaning. And then what you are doing is you are cout cell. which are with a tab, all right, cout cell with a tab. So, once you are coming out of the while loop, it will keep on doing for all the lines, all right.

So, it will be split into cells with a comma delimiter, and then it is printing the cell, right. So, you just keep on doing that, and you have a cout statement, all right, endl, cout endl. All right, that means the next line, so your while loop is ending over here, begin and end, all right. And then you are in this while loop for that cout one new line, all right. And then your while loop, line number 11, while loop is entered in line number 17.

So now you close the csvFile, invoking close, and then if at all you are not able to open, all right, that csv file data.csv. Then you are having the error message, unable to open CSV file. So, let us consider this interesting program in the C++ console, yeah. So, whatever I explain, it is under data or CSV. So, now if you run the code, all right.

So, you can see the output, this is objects, you can see the tab. So, this is an object, and in fact, this is an object, and then the tab, maybe the next line is going object-oriented programming, right. So, this is object-oriented programming, you can see two lines. So, in fact, if possible, we will open data.csv, yes, this yeah, that I said your cell which is right splits the Current line.

So your current line is this. Comma is a comma object. So that means. It splits the current line. Into an array of cells.

So that is what you are getting here. And then what you are printing. Printing with a tab. So that is why you are getting this. Tab is a tab object.

So an array of cells. Using the comma delimiter. So when I have the comma delimiter. So it is making it into. The words with the tab.

So, that is why if you look at the output, we will go to the C++ maybe one more time, and we will run it all right. So, if you look at the output, you are getting something like this. There is no command now. This tab is tab right, object-oriented right, and then tab programming right, ok. Similarly, we can do it in Java as well, right. So, almost similar. So, you have class CSVReader, and this is your driver class because of main, and you have a reader object under BufferedReader memory allocation with a constructor, right.

So, FileReader is handling data dot CSV. So, you have a line under string. So, while line is equal to reader dot readLine, that means yes, every line, right. So, the reader is invoking readLine, and you are putting it in line, right. Till it is not equal to null.

So, if it is null, over the while is at false, all right, it will come out, and then you have cells. So, unlike in C++, we have written it on our own. So, here in this case, you can always call the method, all right. So, cells is the array of string, all right, and then line is invoking split. So, whenever you have to, you will have a comma.

In a text, you will have, you are having a comma, let us say, in the line right, this comma is a comma object, comma-oriented like this. So, we have to split this with the help of this comma delimiter and then put the cells, all right. So, in this case, you

are calling the method split, like what we have done in C++. So, here you are calling the method and then for string. All right.

## Introduction to Serialization and Deserialization

- 👉 **Serialization:** The process of converting an object's state to a byte stream, allowing it to be saved to a file or transmitted over a network.
- 👉 **Deserialization:** The reverse process of reconstructing an object from a byte stream.
- 👉 Serialization makes it possible to persist objects for future use or to send them to another system or application.



So, cell which is equal to cells. Correct. So, you have an array of cells. All right. So, here you are using this for.

All right. The case of cell, and then you are printing this cell. So, that means every word will now become the string of cells. Right. So, that you are copying into cell.

So I am printing this cell with the tab. Exactly what we have done in C++. Similar. Right. And then you are gathering the new line.

Right. Every time you are going to have the new line. So like exactly the same as what we had seen in Java. Sorry. Exactly what we had seen in C++.

So here we are going to run the code in Java. Yeah. Anyway, I explained this. Yeah. So you can see this.

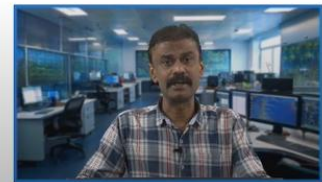
And then we are having a tab, and we are having a tab object. Right. And then the new line-oriented tab programming. OK. So this is how we can handle or read the CSV file.

Comma-separated values, CSV stands for comma-separated values. Fine, so this is how we can read the file, write the file, all right. And then you append, you can have the format of how you can do the CSV file, how you can read the CSV file, right? So all this we have seen. So now, how is this happening? For this, you should know the concept of serialization and deserialization. So, for example, when I am talking about

serialization, it is nothing but the process of converting an object state to a byte stream. So, the byte stream, in the sense, you can save it as a file, or you can put it in a database, or you can transmit it to a network.

## Why Use Serialization?

- 👉 **Persistence:** Save the state of an object to a file for future use.
- 👉 **Data Transfer:** Transfer objects over a network (e.g., for client-server applications).
- 👉 **Interoperability:** Share objects between applications written in different languages (with formats like JSON or XML).
- 👉 **Deep Copying:** Create a full copy of an object, even if it contains references to other objects.



So that is what you call serialization. And another one is deserialization. That means exactly the reverse process. So that means I am going to reconstruct an object. Where will you reconstruct?

Where will you reconstruct from? You will reconstruct from the byte stream. Right. So when the object state is converted into a byte stream. Correct.

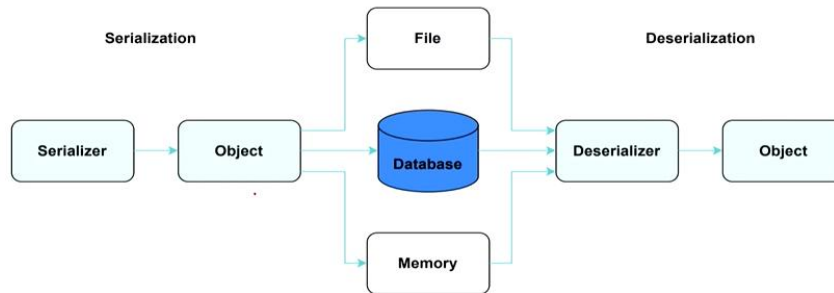
You call that serialization. That means I am saving this object state as a file. You call it a byte stream. Or put it in any database. Or put it in any memory.

Or transmit it over a network. So this is called serialization. When you do the other way around, right, the reconstruction from a byte stream to, let us say, the object, you call it deserialization. So why is it required? The reason is serialization makes it possible to persist objects.

So, these objects you can use for future use, or you can even send them to another system or application. So, the persistence of objects is what we use serialization for. In fact, this slide will tell you why we are using serialization. So, as I already said, the persistence. So, you can save the state of an object to a file for future use.



## Serialization and Deserialization Workflow



- 🔗 **Serialize:** Convert object to byte stream.
- 🔗 **Store/Send:** Save byte stream to file or send over network.
- 🔗 **Deserialize:** Recreate the original object from the byte stream.



So that is called persistence and data transfer. So you can transfer the data, right? In fact, you can transfer the objects over a network. So in networking, I mean, some of you might have studied client-server applications, right? So which you call it data transfer.

And the next one is interoperability, right? So I want to share objects between applications. So you can use several applications. You are using, let us say, different languages, right? And then in one language, I have to use the .obj, that is objects, right.

So, in that case, when I want to share an object between applications written in different languages, serialization is very useful. Also, the deep copy, right. So, you can have a full copy of an object, right, even if it contains a reference to other objects, the address of other objects, right. So, you can have a full copy of an object. Object.

So, this is what it means. So, when you understand the meaning of serialization, right, serialization. So, you have an object. So, the object that you are storing as a file, a database, or in memory, right. So, this is nothing but serialization, which means you are storing, right.

## Basic Serialization and Deserialization in C++

- ☞ C++ does not have built-in serialization like Java, but serialization can be implemented by manually writing data to a binary file.
- ☞ Use `ofstream` and `ifstream` for writing and reading binary data.
- ☞ Structures or objects need to be saved by writing each data member to the file.



So, storing as a byte stream. So, the byte stream may be a file, a database, or memory. Correct. And deserialization. So here, in the byte stream format, you are reconstructing the object.

So, for that, the concept is nothing but deserialization. So, when we talk about C++, it does not have built-in serialization. Whereas, in languages like Java, you can have the concept of built-in serialization. So, here, what we can do is write our own code, right? So, we can write the code, all right, in the case of C++, all right.

Also, the use of streams, the meaning of output file stream and input file stream, for reading and writing binary data, right. So, for writing, in the case of output file stream, writing, and input file stream, reading. So, all right, we can use these for writing binary data, and finally, the structures or objects that need to be saved by writing each data member to the file, right. So, for this, in fact, in the next class, what we can do is talk about basic serialization. And deserialization in C++ as well as Java.

Alright. So, with this, I am completing today's class. Thank you all.