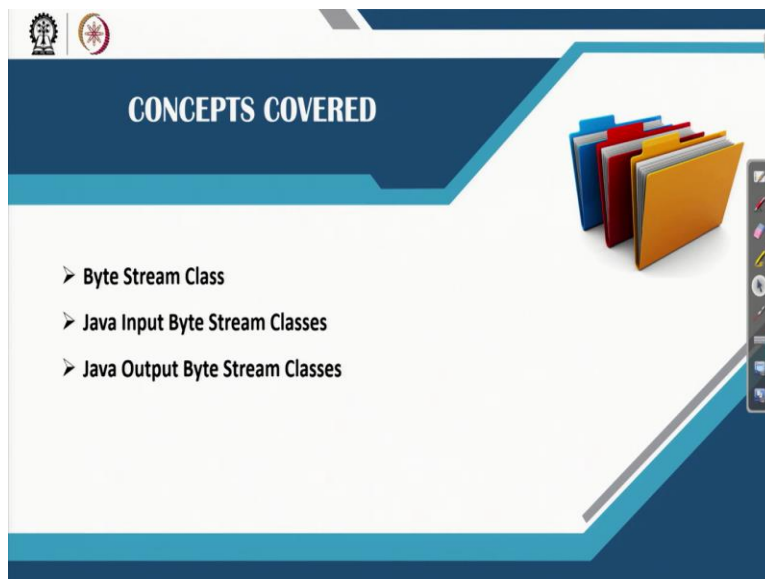


**Data Structures and Algorithms Using JAVA**  
**Professor Debasis Samanta**  
**Department of Computer Science and Engineering**  
**Indian Institute of Technology, Kharagpur**  
**Lecture 43**  
**IO with Byte Streams**

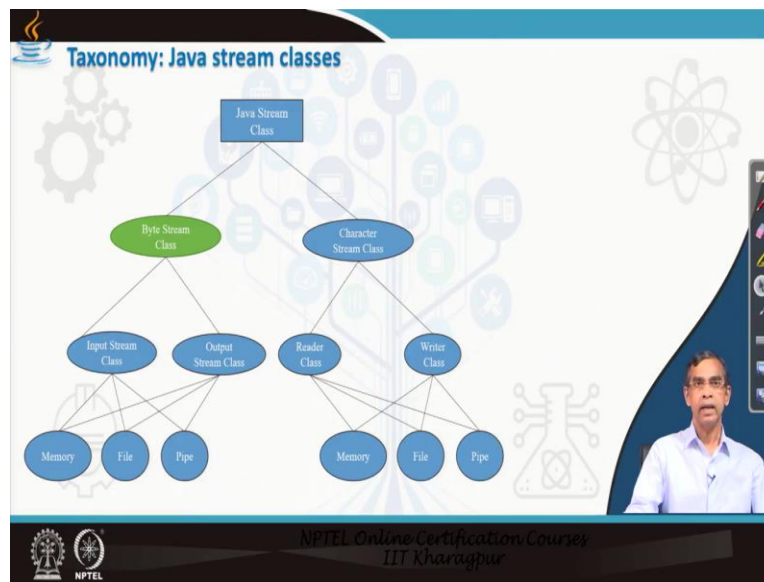
In this video, we will learn Java's most significant one features dealing with input/output. It is the bytes stream, byte stream for input/output.

(Refer Slide Time: 0:48)



So, today we shall learn about the facilities regarding this byte stream input/output. We have already discussed about there are several classes dealing with input byte stream byte stream mechanism. So, we will discussed about these mechanism related to input and output. So, let us first discuss about the different classes that those are available to support the input/output mechanism, in this category; there is a byte stream class.

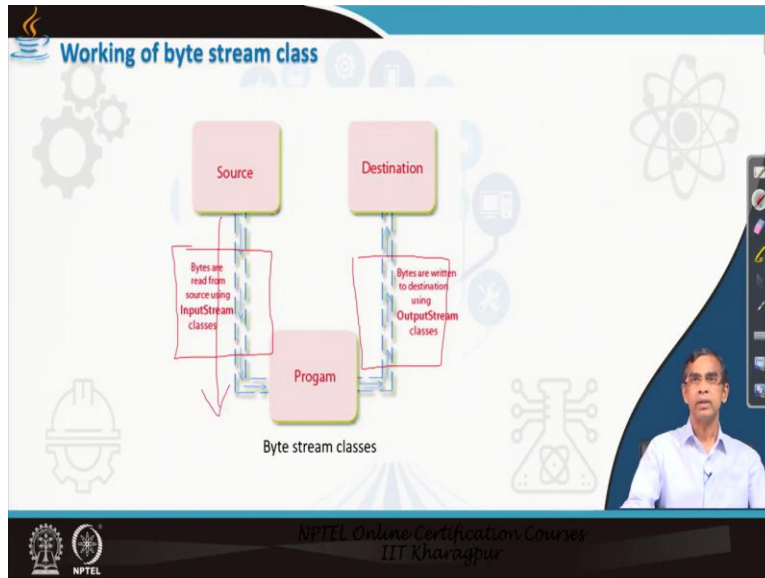
(Refer Slide Time: 1:30)



So, there are input stream and output stream, these are the basically abstract class, which is defined in Java dot io package. Input stream this class is basically support any sort of input using byte stream mechanism. On the other hand, output stream is basically another abstract class, which will basically to support output byte stream.

(Refer Slide Time: 2:11)

The slide, titled "Byte stream classes", contains the following text: "Byte Stream Classes are used to read bytes from an input stream and write bytes to an output stream." The slide is presented in a video lecture format with a presenter's video feed in the bottom right corner and NPTEL logos at the bottom.



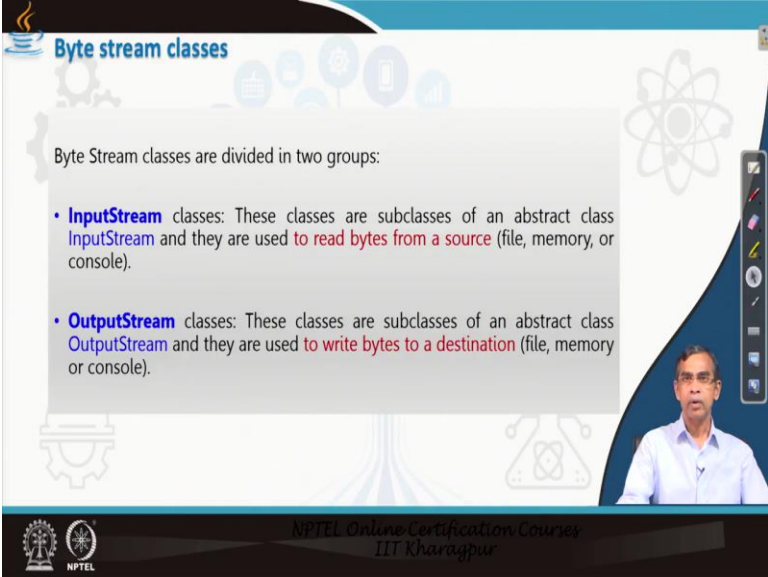
Now, so for the input byte stream class is concerned, there are many what is called the classes are available. And so here the main concept that is there in bytes stream class is basically the source, which will basically available to a program. And then interface between the source and the program is basically input stream arrays from program to a destination; it is called output stream.

Now, in this picture as you see a program can read data from a source as input, and there will be the flow of input data in the form of raw bytes. Whereas, the flow of data that occurs from program to output is again in the form of bytes. So, there is called bytes stream, so you can say that raw data; so, here the flow that will occur from this source to this target. So, this flow is basically is supported by some what is called the interface program.

Similarly, here the flow of data from program to output destination is supported by some interface mechanism. This interface is nothing but some programs, so here the programs are already been developed by Java developer. We want to use this program to handle both input stream as well as output stream. Now, so far the input stream is concerned there are several classes, as well as output stream is concerned there are again several classes which are the different classes those belongs to input stream and output stream; we have certain familiarity which we have covered in the last video.

So, today we will see the programs those are there to support reading and writing in details; and will discuss all those things with examples.

(Refer Slide Time: 5:18)



The slide is titled "Byte stream classes" and features a presenter's video feed in the bottom right corner. The text on the slide is as follows:

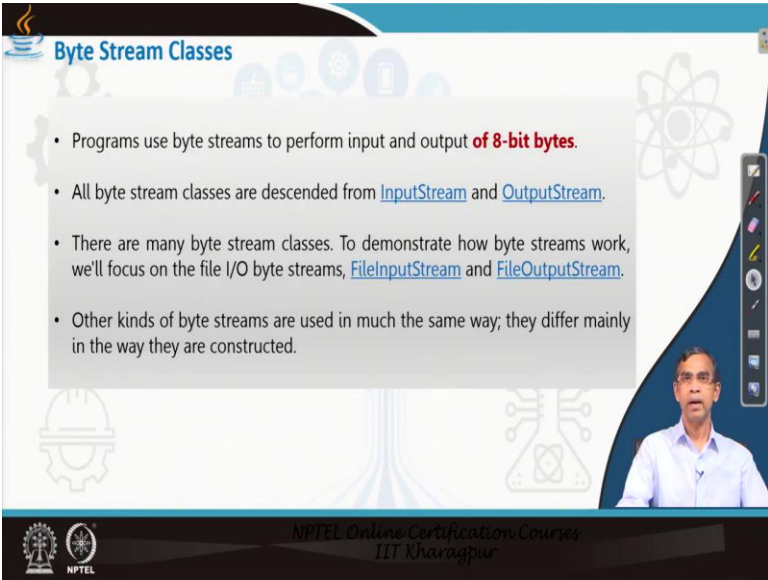
Byte Stream classes are divided in two groups:

- **InputStream** classes: These classes are subclasses of an abstract class `InputStream` and they are used to **read bytes from a source** (file, memory, or console).
- **OutputStream** classes: These classes are subclasses of an abstract class `OutputStream` and they are used to **write bytes to a destination** (file, memory or console).

At the bottom of the slide, it says "NPTEL Online Certification Courses IIT Kharagpur".

So, we have already mentioned input stream to read data from the input source to program. The output stream classes, there are many classes like such belongs to these output stream; they are basically mean for writing bytes from program to output destination.

(Refer Slide Time: 5:48)



The slide is titled "Byte Stream Classes" and features a presenter's video feed in the bottom right corner. The text on the slide is as follows:

- Programs use byte streams to perform input and output of **8-bit bytes**.
- All byte stream classes are descended from `InputStream` and `OutputStream`.
- There are many byte stream classes. To demonstrate how byte streams work, we'll focus on the file I/O byte streams, `FileInputStream` and `FileOutputStream`.
- Other kinds of byte streams are used in much the same way; they differ mainly in the way they are constructed.

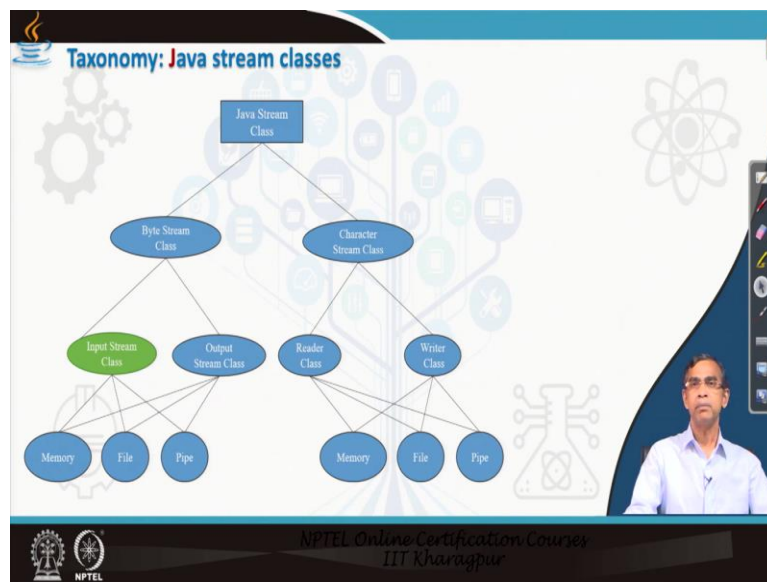
At the bottom of the slide, it says "NPTEL Online Certification Courses IIT Kharagpur".

Now, let us learn about the different things those are here. There are many but it is not possible to cover so many classes; but their concept basically same will emphasize two such interface mechanism. One is called the file input stream, this is for input; input from data source we have programmed.

And here data source is basically a file, the file contains data; program wants to read data from the file and that reading is basically as a byte stream. Similarly, file output stream class is there, here the program wants to write something into this destination; that means output destination which is basically a file. So, we will try to understand about this file input stream and file output stream in details and other input stream classes and output stream classes which are there; they are basically more or less similar.

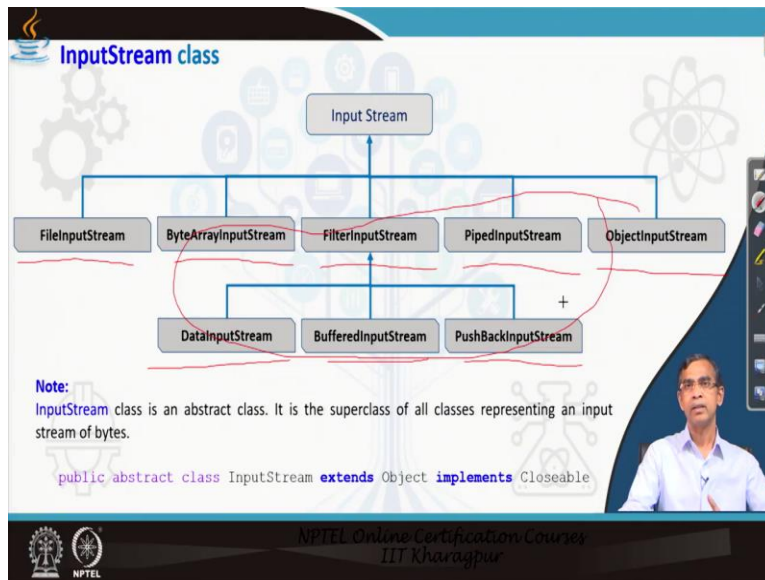
So, those things I will give you the materials, the link to the materials; so that you can go through and you can study but, if you understand this file input stream and file output stream; then other things easy to study because they are basically same, only the thing is that we have to make a connection to that particular resource; basically which represents different type of source or targets for the data.

(Refer Slide Time: 7:41)



So, let us discuss about Java input stream classes, and this input stream classes can help you to read data from different type of source. Different type of source as I mentioned you file is one important source; other than file you can read from any other buffer. You can read as an input source as a standard input device like keyboard; that is also an input source of data. Maybe another program or it can read from a from an array which stores data; this is called the byte array or you can read something from a pipe; pipe basically contains some information, so you can read like this one. So, let us see about different type of classes those are there.

(Refer Slide Time: 8:36)



So, I told you that input stream class is basically the abstract class, and all other all other classes are the subclasses of this input stream class. Here, I have listed those are the different subclasses, so the first is file input stream; here the input source is basically a file. The byte array input stream, here the input source is an array of bytes, where data is stored in an array, array is of types bytes; you know array means is a collection maybe also.

Then filter input stream is one mechanism, it basically helps you to data converting from byte to a primitive type data. Now, piped input stream is basically useful for threading; where multiple threads can run together where thread can communicate, that means one thread can produce data that is other thread can consumes that data, so the data which produce is basically output stream piped output stream.

And another thread which consumes is basically called piped input stream; so, there are two streams, it is input stream and piped input stream. Now, object input stream is very useful, when you want to store directly an object to a through it's; when you want to read an object from a source. Maybe objects are already stored in a file and you want to read the object. So, how the data reading that means reading objects and flow of bytes corresponding to the data relevant to objects. This is basically handled by object input stream.

Now, there is data input stream, this class is useful for reading data from standard input device mainly but here actually main concept is that with this input stream class, you can read data;

although it is stored in a byte but, to you it will store in the form of a primitive data. So using this class, you can read as an integer, read as a long, read as a stream like this one.

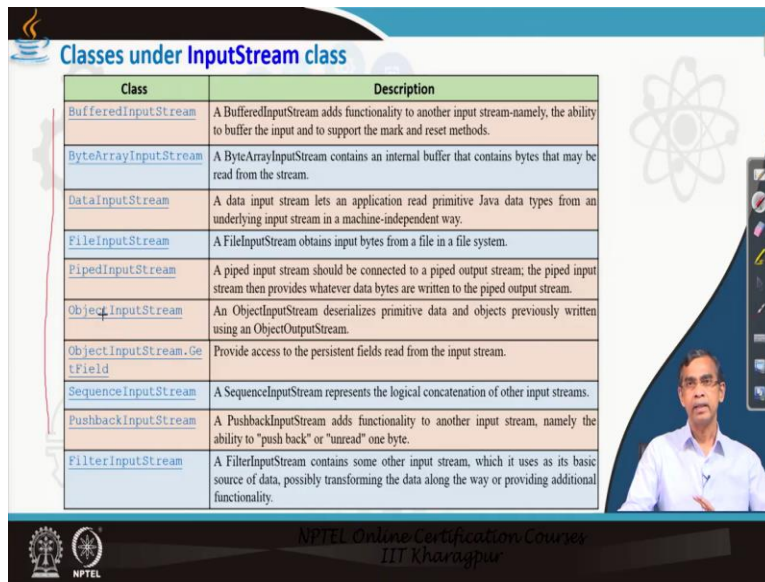
Now, buffer input stream is basically one mechanism, which makes the input mechanism input reading faster, that means it basically stores data from an input source, but is a buffered actually. So, data will be from input source, but it will store as a buffer; from that buffer the program will read.

This is because the data source and then program may not of the same speed; so if we do the buffering, it will basically make faster. Now, pushback input stream is basically sometimes it is required that in all these classes, actually if you read data from an input source; then it will automatically skip the data to the next available data. But, in case of pushback input stream, it gives you an idea about after reading data; you can return the data into the source. So, that you can read again, there are many situations when the reading data and returning data to the input source is required.

So, if you want to read then the pushback input stream is useful for that purpose. Now, so these are the subclasses or the class input stream; input stream class is basically is a is an abstract class, which is defined in Java dot io package. Now, being an abstract class, this class defines many methods; now those methods are ultimately implemented in all these subclasses. So, this means that the all the methods name or the argument types same; those are there in different classes like these are. They are basically same but their operations they are different; those operations are implemented by the Java system.

So, you can just use it, not necessary to bother about how those things are implemented. This is up to the implementation details and that is at the system level. Now, so this basically the input stream class; now let us discuss about the different classes in details that we have discussed.

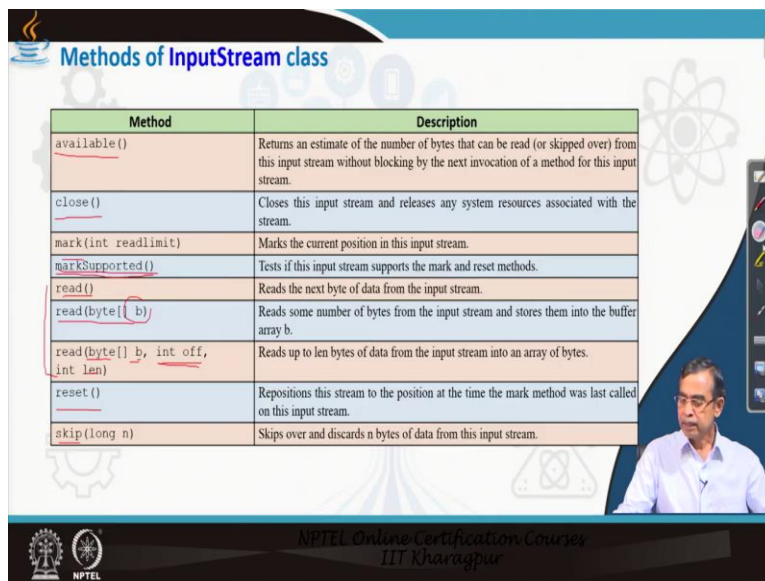
(Refer Slide Time: 13:35)



Class	Description
<a href="#">BufferedInputStream</a>	A <code>BufferedInputStream</code> adds functionality to another input stream—namely, the ability to buffer the input and to support the mark and reset methods.
<a href="#">ByteArrayInputStream</a>	A <code>ByteArrayInputStream</code> contains an internal buffer that contains bytes that may be read from the stream.
<a href="#">DataInputStream</a>	A data input stream lets an application read primitive Java data types from an underlying input stream in a machine-independent way.
<a href="#">FileInputStream</a>	A <code>FileInputStream</code> obtains input bytes from a file in a file system.
<a href="#">PipedInputStream</a>	A piped input stream should be connected to a piped output stream; the piped input stream then provides whatever data bytes are written to the piped output stream.
<a href="#">ObjectInputStream</a>	An <code>ObjectInputStream</code> deserializes primitive data and objects previously written using an <code>ObjectOutputStream</code> .
<a href="#">ObjectInputStream.GetField</a>	Provide access to the persistent fields read from the input stream.
<a href="#">SequenceInputStream</a>	A <code>SequenceInputStream</code> represents the logical concatenation of other input streams.
<a href="#">PushbackInputStream</a>	A <code>PushbackInputStream</code> adds functionality to another input stream, namely the ability to “push back” or “unread” one byte.
<a href="#">FilterInputStream</a>	A <code>FilterInputStream</code> contains some other input stream, which it uses as its basic source of data, possibly transforming the data along the way or providing additional functionality.

So, as I have already told you these are the different classes, which are basically subclasses of the input stream class. And each class basically treat the input source in different way; and these are the different classes meant for different type of input source.

(Refer Slide Time: 14:06)



Method	Description
<a href="#">available()</a>	Returns an estimate of the number of bytes that can be read (or skipped over) from this input stream without blocking by the next invocation of a method for this input stream.
<a href="#">close()</a>	Closes this input stream and releases any system resources associated with the stream.
<a href="#">mark(int readlimit)</a>	Marks the current position in this input stream.
<a href="#">markSupported()</a>	Tests if this input stream supports the mark and reset methods.
<a href="#">read()</a>	Reads the next byte of data from the input stream.
<a href="#">read(byte[] b)</a>	Reads some number of bytes from the input stream and stores them into the buffer array b.
<a href="#">read(byte[] b, int off, int len)</a>	Reads up to len bytes of data from the input stream into an array of bytes.
<a href="#">reset()</a>	Repositions this stream to the position at the time the mark method was last called on this input stream.
<a href="#">skip(long n)</a>	Skips over and discards n bytes of data from this input stream.

Now, I told you the different methods, so these are the different methods, which are declared in the input stream class.



This basically the method available, this means you are reading a data from an input stream; that is our input source. So, how many bytes are remaining or available that can be learn calling this method available. For example, file is an input source, so what is the number of bytes available for further reading; so this available. And close method this basically used to close the stream; then mark, you can mark the data file you are reading that basically helps you comeback to the same position if required; so that mark.

Now, all stream may not supported the mark marking procedure; for example, buffer input stream may not support you. So, in that case we have to check whether this input stream can support the mark or not. If it is not there, then you will not be able to have these facilities; so this method can return you whether a particular stream can help you mark supported or not. Now, these are the read method, three read method; this will read one byte at a time this read basically array of bytes, so this is bulk reading. This read also bulk reading, but starting from a particular position to a particular.

So, depending on you can specify that how many bytes you want to read, and then you can mention it so that it will read in this byte array with a specific part of the byte array can be read; and that can be stored into this bit. It basically return byte array, this also returns a byte array; this return a byte. Now, reset as you see this is mark, if you sometime want to reset the mark; so it can reset to the very beginning of the of the reading. Then the reset method can be called and the skip method you know; so if you want to skip few bytes to read, then you can use it skip.

So, these are the different methods those are there; these methods are declared in input stream class. And all other classes that we have mentioned they are basically subclass of the input stream class. This means that all these methods are inherited to the subclasses; so, this means all those methods are there. So, whether it is reading a file or reading a buffer or reading an object stream; so no problem only read method only for you. Now, so these are the methods, those are there.

(Refer Slide Time: 17:22)

**Java input stream classes**

**InputStream** classes is used to read 8-bit bytes and supports a number of input-related methods

- Reading bytes
- Closing streams
- Marking positions in streams
- Skipping ahead in streams
- Finding the number of bytes in stream
- and many more...

NPTEL Online Certification Courses  
IIT Kharagpur

Now, as I told you this input stream classes basically use to read bytes; so it is basically one byte is 8-bit information. And using those classes for a given input source; you will be able to read byte-by-byte. A stream can be closed particular position of a stream can be mark; we can skip some bytes to read. You can find how many bytes are available and so many; those are basically as mentioned in the method declaration.

(Refer Slide Time: 18:02)

**Some input stream methods**

**DataInputStream**

readShort( )	readDouble( )
readInt( )	readLine( )
readLong( )	readChar( )
readFloat( )	readBoolean( )
readUTF( )	

NPTEL Online Certification Courses  
IIT Kharagpur

Now, let us discuss the different ways all those methods can be used; now again I just want to mention there are certain subclasses like say data input stream. It is some special class actually;

this class as I told you it can read byte-by-byte also. In addition to this it can read bytes; but returning these bytes to a program in the form of a primitive data.

That means it can read bytes, but it can return as you say a short integer; it can read an integer which is 4-bytes reading. So, it will read automatically 4-bytes at one go, and return the integer value which these 4-bytes represents. So, likewise the different concepts are there; so read UTF this method you can use to read a stream or a text that is stored in a file. It will read byte-by-byte, but ultimately it will return maybe a line of text like this. So, there are different what is called the method to read primitive data; by name itself you can understand.

So, read Boolean means it read 1-byte, which basically represent either true or false it is like this. Read Char it basically 2-bytes because in Java it store as a 2 uni-code, which is 16 bits. readLine basically is a text until the carries return or next line is encounter. So, it basically read a text, which is basically store in a buffer, or in a file, or in an array like. So, these are the different methods if a particularly it is a data input stream; we use data input stream when we want to get the data as a primitive data type.

Now, first will discuss about very simple example and this is let see reading from keyboard; where keyboard is a input source or you can say it is an input stream. Now, so this input stream we want to read, and which class we can think for reading the data from an input stream, like keyboard will be discussed in detail.

Now, also keyboard as you know unlike the other any other stream like file or memory or network or pipe; it is basically called a standard input. So, this standard input is defined in a in a class that is called the system dot in. So, system is a class which is a static class, declared in Java dot lang package.

And what this lang that class, there is a field; this field is called in. in basically represents standard input that is the keyboard. So, system dot in basically regarding the standard input keyboard. So, there are many methods like system dot in dot read; this means that if we call this read for this right, then it will read a byte.

(Refer Slide Time: 21:26)

The slide displays a Java program named 'KeyboardReading'. The code imports 'java.io.\*' and defines a 'main' method that reads three inputs from the user: a string, an integer, and a double. It then compares the integer and double values. The code is as follows:

```
import java.io.*;

public class KeyboardReading {
    public static void main(String args[]) throws IOException {
        DataInputStream dis = new DataInputStream(System.in);
        System.out.println("Enter a String: ");
        String str1 = dis.readLine();
        System.out.println("Entered String value is: " + str1);
        System.out.println("Enter a whole number: ");
        String str2 = dis.readLine();
        int x = Integer.parseInt(str2);
        System.out.println("Enter a Double value: ");
        String str3 = dis.readLine();
        double y = Double.parseDouble(str3);
        if (x > y)
            System.out.println("First number " + x + " is greater than second number " + y);
        else
            System.out.println("First number " + x + " is less than second number " + y);
        dis.close();
    }
}
```

The slide also features a video feed of a presenter in the bottom right corner and the NPTEL logo in the bottom left corner.

Now, let us first start about this concept, and whether that we can discuss on program; and I will try to explain this program so that you can understand about it. So, we are using Java dot IO specialty, so for each program related to this IO handling. And we should input java dot io; so this is must.

Now, this program explain how the keyboard reading is possible; and to read the keyboard our objective is to read as a primitive data. So, that we read some integer keyboard, so user will type something from the keyboard; and your program will read that data which user has typed and then do certain printing or processing whatever it is there; now, let us proceed with this program. So, this is a main method and the main class is Keyboard Reading; so here we first create an object of type data input stream.

Because we want to read data as an input stream, so this object is dis; and this basically of type data input stream. So, there is a constructor which we can use it, and this is the top source from where we want to read. So, system dot in basically indicates the form, which input we want to read data; as I told system dot in basically keyboard. Now, here the source can be anything which will discuss about a source can be a file; it can be memory location, or something else. So, in that case accordingly that name can be (())(23:06).

But, we want to read keyboard data; so system dot in is the input source in this case. Now, here just you give a prompt from the program, Enter a string; now this basically read line. Now, here

actually this read line is the method, which is used to read anything the user has typed; and then press Enter. So, if it is there then it will read and then this read line method will return as a string; so, this string will be stored as a str1. So, in this case you keep a prompt to the user; user typed something and then return and then you read.

Now here, then after reading system, it basically print; so you enter string is string 1. So, what whatever you have typed, it will print; now, let us read some other type of data here. Enter a whole number, you ask the user to type something, so integer. Now, here first you read as a text because in this input stream whatever user will type; will be written as a string. And there is a conversion from string to some type line. So, here first read as a string whatever user type; even if user type say 1, 2, 3; it read as a string and it return to here.

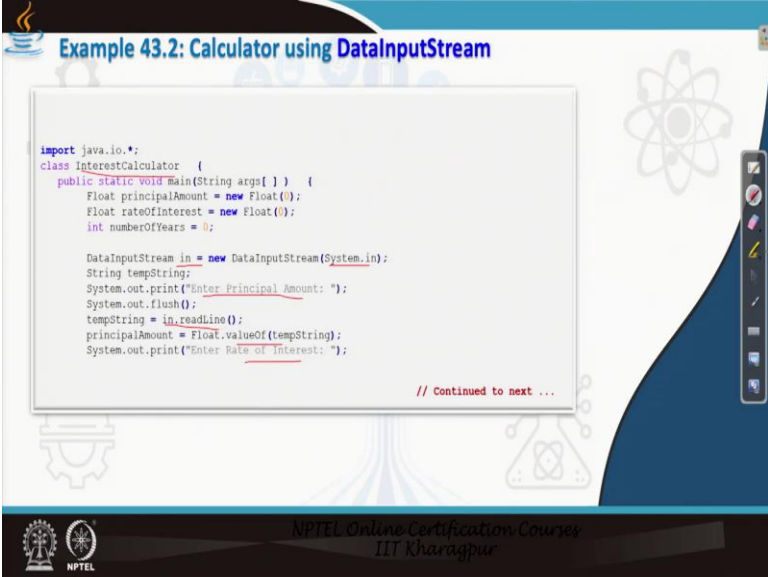
And then this string is towards to needs to be passed to get the integer value; so this method is Integer dot parseInt. So, it is basically convert from a string to an integer, so x will now store whatever the integer user has typed. Now, next ask the double value, read as a string; and then here the Double dot parseDouble.

So, that the double value can be obtained from the string type; so this y is a double value. Now, you do the comparison to read the speed, and finally you just close the input string that you have opened. So, this ends the program and this program should be under try-catch block; so you can write here try and then catch.

Otherwise you can write throws IOException, so try-catch block is must, without this throw statement here or try catch block including this; it can leads to the program can give compile time error because program may not overlies you to compile the program without try catch. So, try catch is mandatory code for any output, input/output handling.

So, this program shows how some data can be read from the keyboard using the data input stream class.

(Refer Slide Time: 26:06)



```
import java.io.*;
class InterestCalculator {
    public static void main(String args[] ) {
        Float principalAmount = new Float(0);
        Float rateOfInterest = new Float(0);
        int numberOfYears = 0;

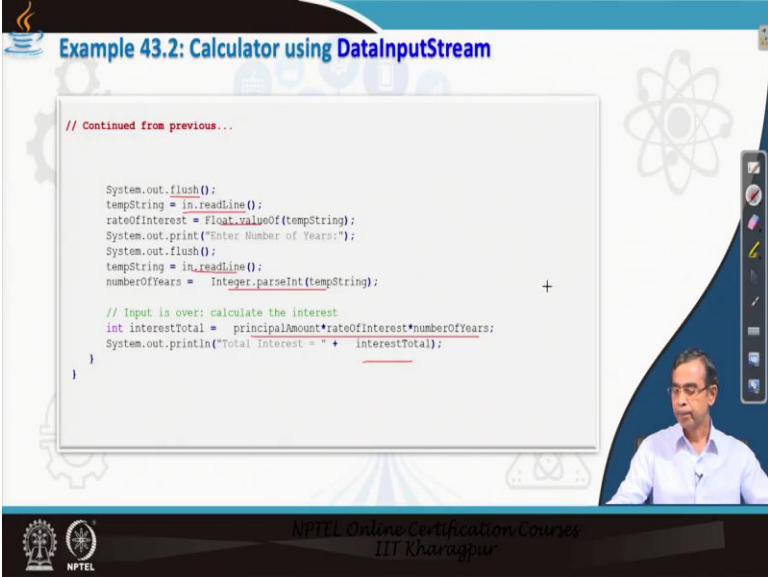
        DataInputStream in = new DataInputStream(System.in);
        String tempString;
        System.out.print("Enter Principal Amount: ");
        System.out.flush();
        tempString = in.readLine();
        principalAmount = Float.valueOf(tempString);
        System.out.print("Enter Rate of Interest: ");

        // Continued to next ...
    }
}
```

Now, this is another extension of the same program, here the same program of course. This is a interest calculator is basically we want to calculate the data; but reading the data from the keyboard.

So, here this is the input string that we have created; again from the keyboard. So, here you read the value, then convert this value and then write; then then then process it, and it will continue like this. So, this basically reads certain data as a primitive data, now let us continue this program again.

(Refer Slide Time: 26:48)



The slide displays a code editor window with the following Java code:

```
// Continued from previous...

System.out.flush();
tempString = in.readLine();
rateOfInterest = Float.valueOf(tempString);
System.out.print("Enter Number of Years:");
System.out.flush();
tempString = in.readLine();
numberOfYears = Integer.parseInt(tempString);

// Input is over: calculate the interest
int interestTotal = principalAmount*rateOfInterest*numberOfYears;
System.out.println("Total Interest = " + interestTotal);
}
}
```

The slide also features a small video inset of a man in a light blue shirt, the NPTEL logo, and the text "NPTEL Online Certification Courses IIT Kharagpur" at the bottom.

Here we read few more here again read from the keyboard as a Float; and then read as an integer from the string and you store. And then finally you can use this data for your calculation. So, like you read principal, you read interest, you read number of years; and then finally calculate your program can calculate, so reading from the keyboard.

Now, here you see a flush, flush is basically see if you are reading the data; so flush is basically one sort of cleaning the keyboard buffer actually, so it clean the buffer. So, whenever you execute a flush command, all data which is stored in the input stream will be flushed. That means cleaned, it somehow goes to some program or somewhere right; so, if you type something it will basically read that part only. Another thing is that suppose you want to read an integer, but you type say suppose abc; and then you call this method integer dot parseInt from the string.

Then it will basically gives an IO exception; it will basically invalid type exception sort of things are there. So, you have to be careful about it, whenever you type say integers; so you just type integer value from your side only. And then program will manage or your data input stream class will manage to get back the data; as a converting from byte to your primitive data type. So, this is the concept that is that is the important to remember. Now, so this is the input stream class, now let us see a reading a file; how we can read a file from this is file is a input stream class here.

(Refer Slide Time: 28:36)

**Example 43.3: Reading bytes from a file and display data**

```
// Read the file address from the Command Line
import java.io.*;
class ReadBytesDemo {
    public static void main (String args[]) {
        FileInputStream infile = null; // Create an input file stream
        int b;
        try {
            infile = new FileInputStream(args[0]);
            // Connect infile stream to the required file
            while((b = infile.read()) != -1) {
                System.out.print((char)b); // Read and display data
            }
            infile.close();
        }
        catch(IOException ioe) {
            System.out.println(ioe);
        }
    }
}
```

NPTEL Online Certification Courses  
IIT Kharagpur

This is one simple program that is you can consider. So, here read bytes demo is basically this is a input stream and then byte stream; so, it will read byte-by-byte, a very simple program it is. So, first of all we have to have the file input stream, because it is the input stream in our case. So, we declare a a declare an object called infile as a file input stream; initially it is null. Now, here file input stream args 0, what is the problem is that whenever you run this program; read by demo you have to give the file name first. So, you want to read something from the file, say abc dot data or abc dot txt.

So, you just give the so this program is say read by test demo abc dot txt. Then args 0 will be abc dot txt, because it is a common line input; so this basically serve as a file name, from which you want to read the data. Then file input stream is basically make a connection from your program to this file; file is stored suppose in your current directory from where you are running the program. Now, this is actually you see read method we are calling for this file input stream class; that is read method, it written 1-byte at a time, so it will written as a b; and this this is under while step, so it basically byte-by-byte.

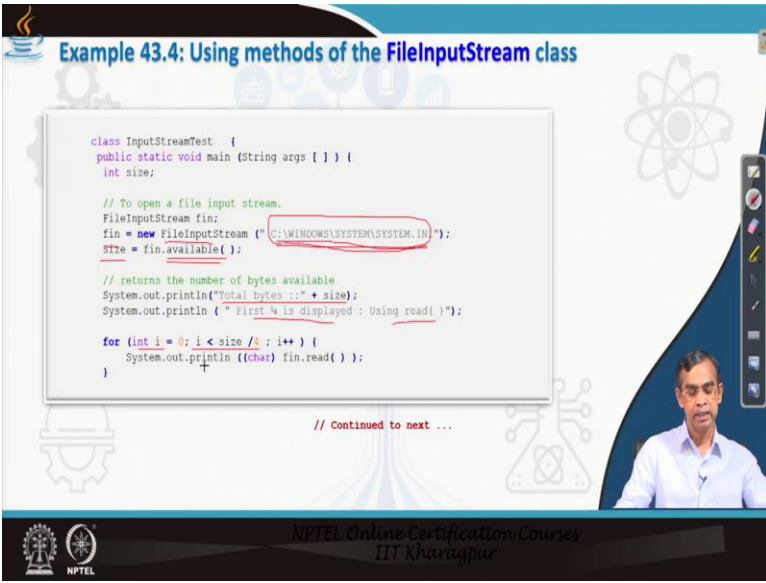
And then it convert into the character and then it basically display; so this basically read the entire file byte-by-byte, that means one character at a time and then display all bytes; and here you see this is a byte is an ASCII code. So, Java is basically compatible to all, so ASCII values those are there; they are although it is Unicode standard all the characters but, ASCII value Java



can recognize and then can be represent. So, whatever thus text you type, character you type from the keyboard, whatever the symbol it is there; they are basically standard within the ASCII range actually.

So, it will display on the screen and finally, and you see we have included within the try-catch; this is very important and the user syntax of try-catch construct, so this ultimately scan the entire file that you passed as an common line input, and it will read the entire file. So, this is basically example how it can read byte-by-byte, from a file as an input source.

(Refer Slide Time: 31:30)



The slide displays a Java code snippet for reading a file using the `FileInputStream` class. The code is as follows:

```
class InputStreamTest {
    public static void main (String args [] ) {
        int size;

        // To open a file input stream.
        FileInputStream fin;
        fin = new FileInputStream (" C:\WINDOWS\SYSTEM\SYSTEM.IN ");
        size = fin.available();

        // returns the number of bytes available
        System.out.println("Total bytes ::" + size);
        System.out.println ( " First 4 is displayed : Using read()");

        for (int i = 0; i < size / 4 ; i++) {
            System.out.println ((char) fin.read() );
        }
    }
}
```

The code is annotated with red boxes and lines. A red box highlights the file path `" C:\WINDOWS\SYSTEM\SYSTEM.IN "`. A red line underlines the `available()` method call. Another red line underlines the `read()` method call inside the loop. Below the code, the text `// Continued to next ...` is visible. The slide also features the NPTEL logo and the text "NPTEL Online Certification Courses IIT Kharagpur" at the bottom.

Now, let us consider another example which basically read the bytes; but using the different methods those are there, declare in the input stream class. That is also available as a file input string class, you just note this program. Again the similar kind of program, we create basically a file input stream; let this is the input file we want to read.

We have given the direct path that also you can specify; that is where it is in the C drive or whatever it is there. So, this fin is basically is a connection from your program to the input; what is called the source of the file namely this one. And here you call the available method then how many bytes are available there; that means yet to read basically it will be return there.

So, the in this case it will be return the total size of the file it is here; so this is the size, then you can ping this one. Now, here first one fourth of the file let us read it, how you can read it; you see

what you are doing for integer I equals to 0, I less than size by 4. So, that means it will basically read first 25 percent of the file, it will read as a character in the string method that we have discussed. Now, we can continue the reading because we have read only one fourth of the part.

(Refer Slide Time: 32:49)

**Example 43.4: Reading a file using FileInputStream**

```
// Continued from previous...  
System.out.println (" Remaining bytes : " + fin.available() );  
System.out.println ("Next % is displayed : Using read (b)");  
byte b[] = new byte[size/4];  
if (fin.read (b) != b.length )  
    System.err.println ("File reading error : ");  
else {  
    String temp = new String (b, 0, b, b.length );  
    // Convert the byte[] into string  
    System.out.println (temp) ;  
    // display text string.  
    System.out.println (" Still available:" + fin.available() );  
    System.out.println (" skipping % : Using skip ( )");  
    fin.skip(size/4);  
    System.out.println (" File remaining for read : " + fin.available() );  
}  
fin.close (); // Close the input stream  
}
```

NPTEL Online Certification Courses  
IIT Kharagpur

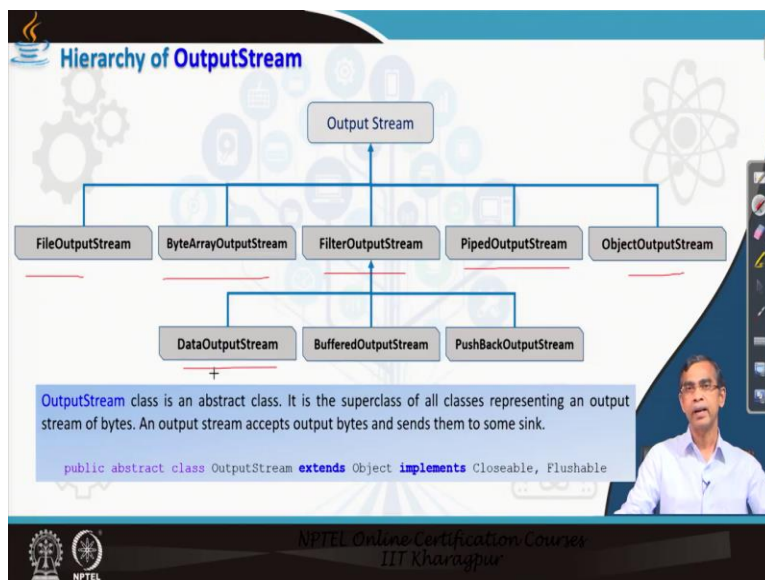
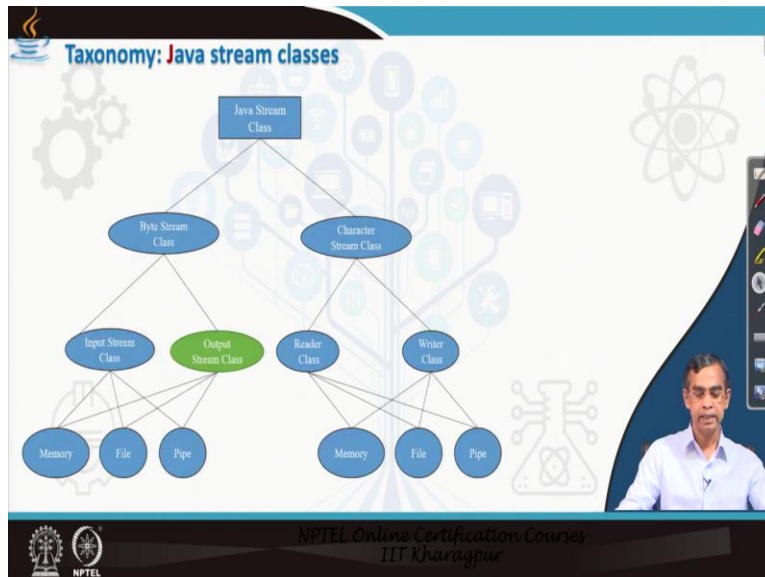
In the next part of the program as you see remaining bytes after reading one fourth; it will give the remaining. So, it means basically say that 75 percent is available, and then again you are reading the next one fourth of the file using byte buffer. So, here basically we have array of buffer and then you can do this.

So, we declare an array of buffer is a array of bytes actually, the total size of the array as declare size by 4; because we want to read next one fourth of the data. So, here you see read method we call as b, so this basically it will read the whole junk of bytes and stored into the bytes; that will be printed here. So, this basically store of the byte, can be stored as a string; and this is the string basically the byte that you have read.

It is stored as a form of a string, and then that string can be printed here. Then you can have the total bytes that is available, you can skip like one fourth; this is the skip is the that you can remove. And then you can again read available, so this way. So, this indicates that how you can scan the entire input string according to your own control, so that you can read the file one by one. So, this is very helpful so that because we have a good control over the byte-by-byte

reading; so this is a good advantage that is basically available from this one. Now, let us discuss about byte output stream class.

(Refer Slide Time: 34:19)

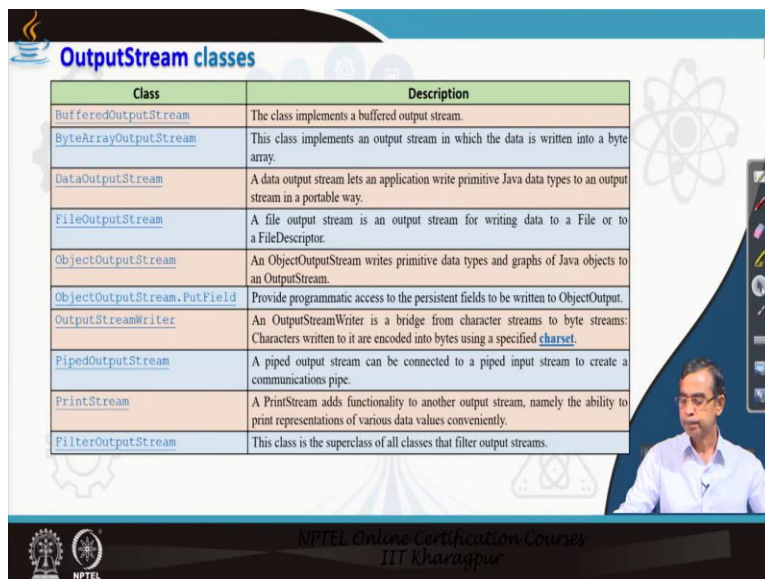


There are many okay we have learned about input stream; now let us see output that means you can write something to the target destination. Like input strings there are many many such destination target; where you can write, we can write into a file, we can write into an array, we can write into a filter output stream; that means converting data into primitive type and then it can store. We can write as a piped that means multifaceted program where it can work as a

destination pipe. We can write as an object output into some target; it can be maybe control or it can be file.

File data output stream we are already familiar to data input stream; it is a just opposite to; that means if we want to write back into the form of a primitive data. So, you can from your program write as a primitive data, automatically save as a byte. Buffer is basically is a faster access, pushback data output stream is a after reading, you can comeback and then this one. So, these are the different methods are there, which you can consider in order to I mean store data into an output stream.

(Refer Slide Time: 35:29)



Class	Description
<a href="#">BufferedOutputStream</a>	The class implements a buffered output stream.
<a href="#">ByteArrayOutputStream</a>	This class implements an output stream in which the data is written into a byte array.
<a href="#">DataOutputStream</a>	A data output stream lets an application write primitive Java data types to an output stream in a portable way.
<a href="#">FileOutputStream</a>	A file output stream is an output stream for writing data to a File or to a FileDescriptor.
<a href="#">ObjectOutputStream</a>	An ObjectOutputStream writes primitive data types and graphs of Java objects to an OutputStream.
<a href="#">ObjectOutputStream.PutField</a>	Provide programmatic access to the persistent fields to be written to ObjectOutput.
<a href="#">OutputStreamWriter</a>	An OutputStreamWriter is a bridge from character streams to byte streams: Characters written to it are encoded into bytes using a specified <a href="#">charset</a> .
<a href="#">PipedOutputStream</a>	A piped output stream can be connected to a piped input stream to create a communications pipe.
<a href="#">PrintStream</a>	A PrintStream adds functionality to another output stream, namely the ability to print representations of various data values conveniently.
<a href="#">FilterOutputStream</a>	This class is the superclass of all classes that filter output streams.

Now, these are the different method, these are the different classes which we have already discussed about. It is listed in a tabular form and you can take your time to study about; these are the different things, different way, the different type of destination output can be controlled.

(Refer Slide Time: 35:47)

### Use of class OutputStream

- Writing bytes
  - void write (byte b)
  - void write (byte b[])
  - void write (byte b[], int off, int len)
- Closing a stream
  - void close()
- Clearing a buffer
  - void flush()

NPTEL Online Certification Courses  
IIT Kharagpur

### Methods of OutputStream class

Method	Description
close ()	Closes this output stream and releases any system resources associated with this stream.
flush ()	Flushes this output stream and forces any buffered output bytes to be written out.
write (byte [] b)	Writes b.length bytes from the specified byte array to this output stream.
write (byte [] b, int off, int len)	Writes len bytes from the specified byte array starting at offset off to this output stream.
write (int b)	Writes the specified byte to this output stream.

OutputStream classes is used to write 8-bit bytes and supports a number of input-related methods

- Writing bytes
- Closing streams
- Flushing streams
- etc.

NPTEL Online Certification Courses  
IIT Kharagpur

And here the different write method is available, because it is writing purpose. You can write as a single byte, you can write array of bytes, you can write byte is junk of bytes also. That is there possible and then close and then flush; because you have to after writing, you can flush in consent all the all the all the data that is stored. Either in your buffer or in your input repository; so that it to the output destination.

(Refer Slide Time: 36:14)

**Some methods in output stream classes**

**DataOutputStream**

writeShort( )	writeDouble( )
writeInt( )	writeLine( )
writeLong( )	writeChar( )
writeFloat( )	writeBoolean( )
writeUTF( )	

NPTEL Online Certification Course  
IIT Kharagpur

So, let us see this is the data output stream is more interesting, because here from the programs program size; you can treat the data as a primitive data. And whenever the data goes to the destination, it will basically store as a byte. So, for this writing primitive data, these are different methods like writeLong, writeShort, writeInt; as the name implies you can write as a primitive data.

(Refer Slide Time: 36:43)

**Example 43.5: Storing data into a File**

```
import java.io.*;

class ReadWritePrimitive{
    public static void main (String args[]) throws IOException{

        File primitive = new File("prim.dat");
        FileOutputStream fos = new FileOutputStream(primitive);
        DataOutputStream dos = new DataOutputStream(fos);

        //Write primitive data to the "prim.dat"file
        dos.writeInt(1999);
        dos.writeDouble(375.85);
        dos.writeBoolean(false);
        dos.writeChar('X');

        // Continued to next ...
    }
}
```

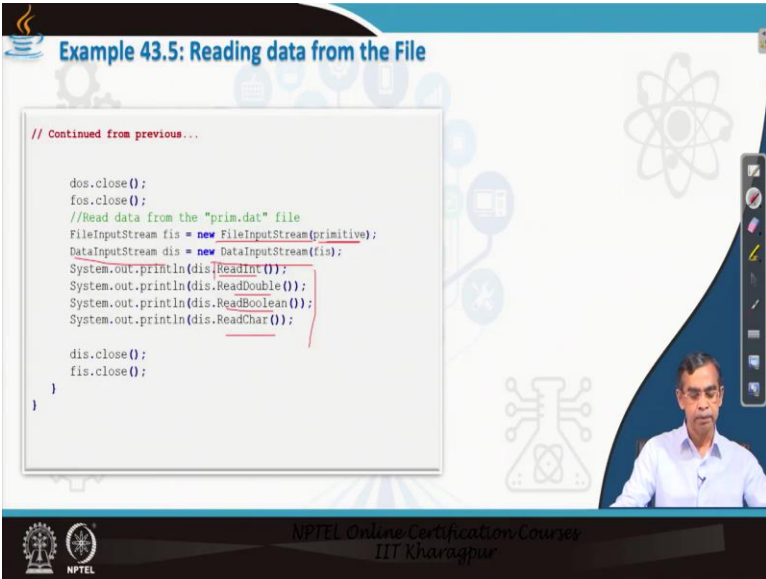
NPTEL Online Certification Course  
IIT Kharagpur

Now, let us have some discussion about how writing the byte into a file; and this is one program is a read/write primitive data. We can read from an input file and write into an output file; this is the idea that to actually this program.

So, for this purpose we have to create the file output stream object for a given file; so primitive is the name from the file. So, we create a file object here, this is the name of the file from which we want to create; and you can note as the how file object can be created. So, one argument can be given as a file and where you want to and then data output stream is basically is a basically connection from your program to this interface. So, here interface is a data output stream, which take the data from your program; and finally this program this data from the program will be sent to the destination namely the primitive; which is basically is a file primitive data.

Now here very simple, there are very few statements that we have; you can understand WriteInt means you want to write this as an integer, false is a Boolean is a character. So, all these things go goes all these things go to the file primitive data.

(Refer Slide Time: 37:54)



**Example 43.5: Reading data from the File**

```
// Continued from previous...

dos.close();
fos.close();
//Read data from the "prim.dat" file
FileInputStream fis = new FileInputStream(primitive);
DataInputStream dis = new DataInputStream(fis);
System.out.println(dis.readInt());
System.out.println(dis.readDouble());
System.out.println(dis.readBoolean());
System.out.println(dis.readChar());

dis.close();
fis.close();
}
}
```

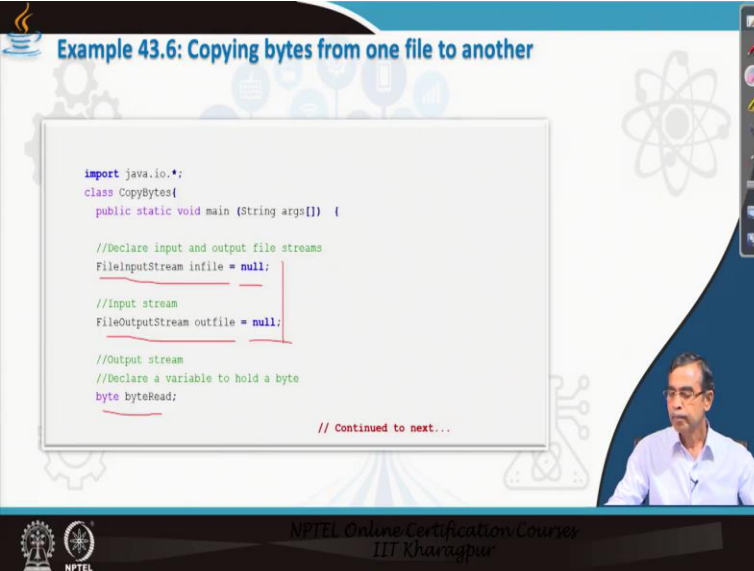
NPTEL Online Certification Course  
IIT Kharagpur

Now, let us extend this program as a continuation; we can read again. So, for this read we have to create the input stream; so file input stream, again for the same data file can be consider the input now. Data input stream object can be created here, this is an interface from your program to this one; because your program will read from the data input stream which intern read from this file.

And these are the few statements where the readInt, readDouble, readBoolean, readCharacter whatever the data you have stored it will read automatically. And here one thing again very important, so in the order you have write in the same order you read, then it is okay but for example if you readDouble as a readInt, then all things will be (())(38:35, so that will read an IO exception.

So, this program should be enclosed within the try-catch block; or throws IO exception should be declare, when the main method is there. And finally you have to close the input/output stream both. So, this is a program shows about tell about, how the reading is possible from there.

(Refer Slide Time: 38:58)



The slide displays a Java code snippet for copying bytes from one file to another. The code is as follows:

```
import java.io.*;
class CopyBytes{
    public static void main (String args[]) {

        //Declare input and output file streams
        FileInputStream infile = null;

        //input stream
        FileOutputStream outfile = null;

        //Output stream
        //Declare a variable to hold a byte
        byte byteRead;

        // Continued to next...
```

The slide also features a small video inset of a man in a light blue shirt in the bottom right corner. At the bottom of the slide, there are logos for NPTEL and IIT Kharagpur, along with the text 'NPTEL Online Certification Course IIT Kharagpur'.

Now, copying a file into another file, we just try to give one more example. This is very popular one what is called the program called the copy byte program, so byte-by-byte. The same thing earlier we have just used the data input stream as a byte-by-byte; but here will use as a copy byte-by-byte. You see what is the mechanism that we can read it, so first there are two inputs stream input stream. First two stream, one is input stream, another output stream; initially these streams are created but they are not connected to any device, so they are null.

Then we just create a byte, where the temporary byte will be stored; so, now let us continue this program.



(Refer Slide Time: 39:55)

**Example 43.6: Copying bytes from one file to another**

```
// Continued from previous...

try {
    //Connect infile to in.dat
    infile = new FileInputStream("in.dat");

    //Connect outfile to out.dat
    outfile = new FileOutputStream("out.dat");

    //Reading bytes from in.dat and writing to out.dat
    do {
        byteRead = (byte) infile.read();
        outfile.write(byteRead & 0xFF);
    }
    while(byteRead != -1);
}

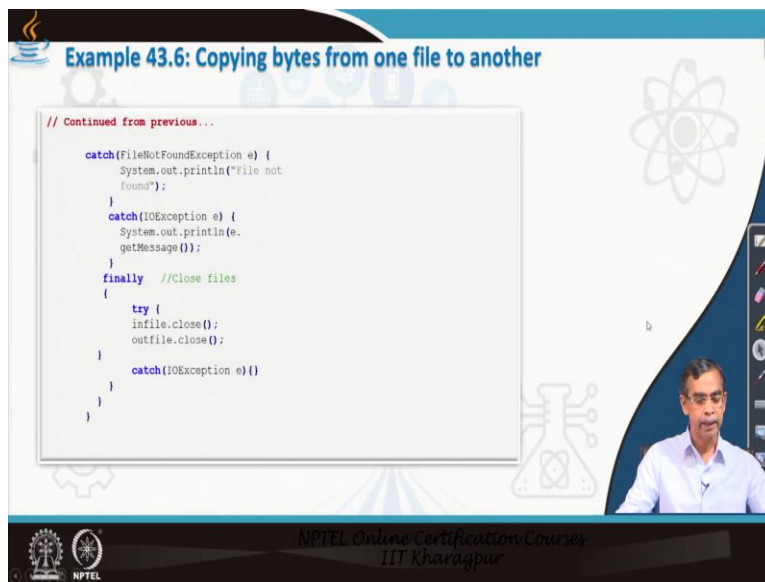
// Continued to next...
```

NPTEL Online Certification Courses  
IIT Kharagpur

Now, here you see we just create a connection that is the int dot dat, assuming that this file is already available to you. If it is not exist then it will throw an error; so int dot data is a already input file where some data is stored. And out data is basically target file; if it is not available then it will not create a problem. But, it will automatically create a file and store the data there. And if this data this file contains some initial data; then it will overwrite actually. Now, here you see within a do-while loop, what we are doing? We just read the file one byte-by-byte; and we write the same thing into our output file.

So, it basically it will scan starting from beginning of the input file in that; will read byte-by-byte and write the byte into this one. So, it will basically copy byte information, and this byte will continue until the end of file. End of file is basically when the, this file will return minus 1. So, end of file means minus 1, minus 1 is basically is a byte concept there is data available from the input source. So, this program is very important regarding this reading from a file byte-by-byte.

(Refer Slide Time: 41:06)



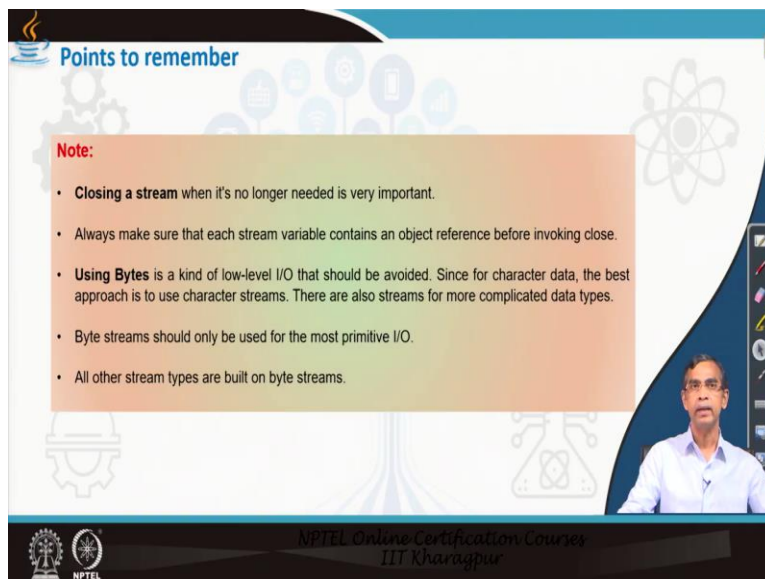
**Example 43.6: Copying bytes from one file to another**

```
// Continued from previous...  
  
catch(FileNotFoundException e) {  
    System.out.println("File not  
    found");  
}  
catch(IOException e) {  
    System.out.println(e.  
    getMessage());  
}  
finally //Close files  
{  
    try {  
        infile.close();  
        outfile.close();  
    }  
    catch(IOException e){}  
}  
}
```

NPTEL Online Certification Courses  
IIT Kharagpur

And there is a try-catch block should be enclosed to make the program reliable.

(Refer Slide Time: 41:12)

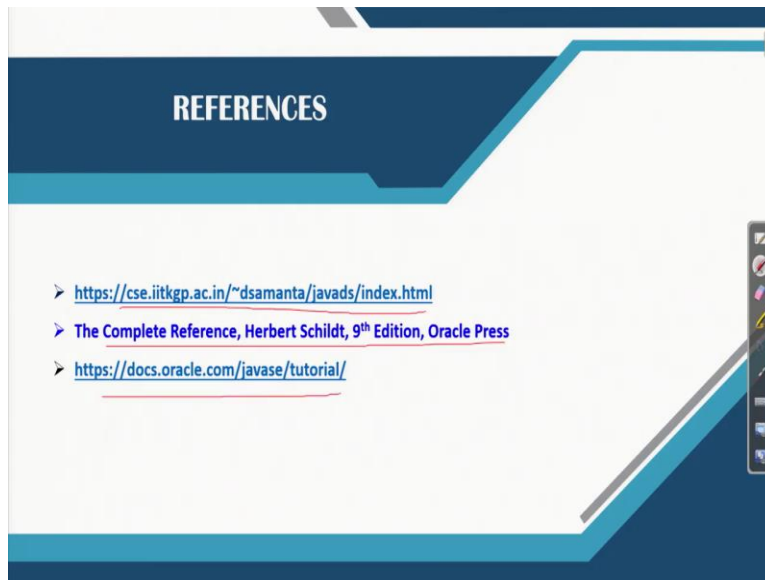


**Points to remember**

**Note:**

- **Closing a stream** when it's no longer needed is very important.
- Always make sure that each stream variable contains an object reference before invoking close.
- **Using Bytes** is a kind of low-level I/O that should be avoided. Since for character data, the best approach is to use character streams. There are also streams for more complicated data types.
- Byte streams should only be used for the most primitive I/O.
- All other stream types are built on byte streams.

NPTEL Online Certification Courses  
IIT Kharagpur



So, we have planned about how the input stream and output stream as a byte can be used; so that can be read. And for many more discussion you can link, you can cancel this book; this book is very has the many discussion there. And also this document also, this link webpage is there; you can find many information from there also. And this is the good tutorial from where you can learn many more things; so I should advice you to check beyond this discussion which I have made here.

Otherwise for your extra learning that is the supplementary material. So this is the concept that I want to convey it as an input or byte stream class actually. And there are many more things about one; this is a one part of the Java dot IO package; there are many more things. Mainly there is another ways that data can be dealt with the; it is called the character byte stream class. So, this class will discuss in our next next video class. Thank you, thank you very much.