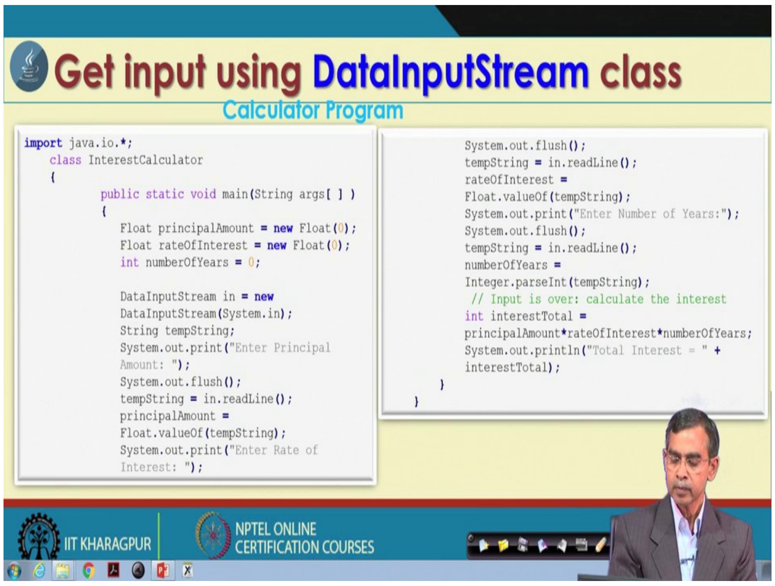
# Programming in Java
**Prof. Debasis Samanta**
**Department of Computer Science and Engineering**
**Indian Institute of Technology, Kharagpur**

## Lecture – 31
## I – O Stream – II

We have learned about that I-O stream classes and java.io package. Now let us see how the different classes in these packages can be utilize to meant to process the input output mechanism in java program. So, basically this module includes the concept of the application of I-O stream classes into the program.

(Refer Slide Time: 00:45)



### Get input using DataInputStream class

Calculator Program

```
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: ");
               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);
            }
    }
```

Now, let us first start with a very simple one which you have already familiar to. So, suppose the calculator program to calculate something, which will read some value from the input say may be standard input. So, the program we have already familiar to this kind of program it is there as we see. We have given the name of the classes input InterestCalculator as we see here this is the name of the class and this is the method and in this method as this actually purpose of this program is to read something from a keyboard. Now keyboard it is identified by a System dot in. So, here is basically you see System.in so it is a System dot in.

Now first you see what how we can connect a connect a link from your program to this standard input namely the keyboard in this case. Now to do this things, we have to create an object of the class DataInputStream. So, we create the object and in let in with the name of the object and this is the syntax for creating an object of the class DataInputStream as you see DataInputStream = new DataInputStream (System.in).

So, this basically create a connection from your program to the keyboard. Keyboard here being a input stream source. Now once you create this objects, then we can read the data from this. Now to read it here we see, I use in dot readLine. So, in.readLine the readLine is a method which is define their in java.io package for the DataInputStream classes. This basically read whatever the number of entity.

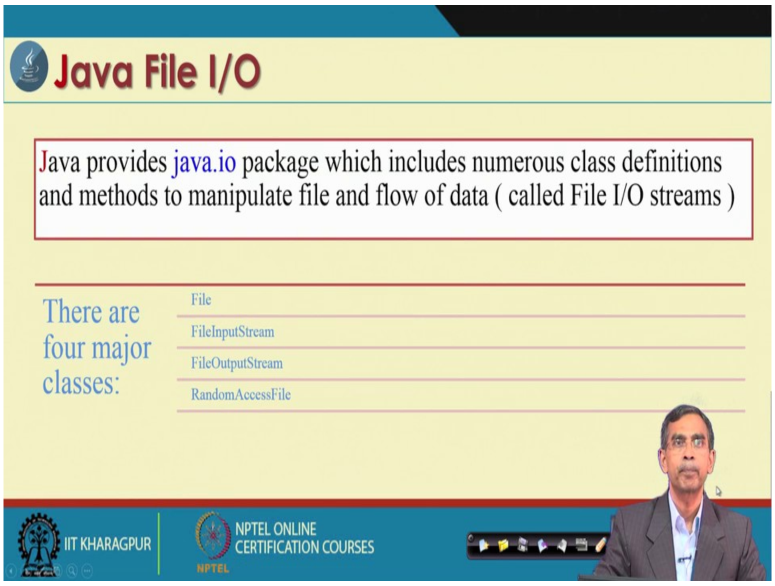
For example, if we type say 2, 4, 5. So, read line is basically read 3, 4, 5 as a whole together if I write say 2, 4, 2 4 5.6 7. So, it will read as a whole thing. If we enter some string, it will read whole the string as a whole together so; that means, the idea about the readLine. Now for the object in, we call the method readLine which basically read the entire thing that is presently available in the buffer and read it. And then so here basically we see we can read here for example, this one basically reading the principal amount in order to calculate the interest here for which we need the input from the keyboard and this will be stored in a principal amount, rate of interest, number of years.

So, three readings are here involved and in the first reading as we see, we can read for the principal amount and then in the second reading, we can read for the rate of interest and we can read here for the number of years. Now after reading, we can see we read all this things in the form of a byte. So, this byte is store here in the form of a this buffer tempString and it is basically this byte is store as a I mean considered as a string actually and then this method you see this is interest important to note that this method basically convert the string because you rate as a string and this string needs to be converted into the desire number. So, it is basically float.

So, this method is basically read a string from the by data DataInputStream and convert into a float value. So, this is a value of method which is define in java dot lang package for the class float will convert the stream into the double or float value. Likewise this is also as we see, it change from the stream and reading and change into the float value.

Similarly, here this basically reading the number of years as an integer and we see we call this method which is define in java dot lang package to convert into the integer. So, Integer dot parseInt (tempString) like. We are already familiar to in several our discuss problem program we have use this kind of concept here anyway. So, this basically shows an idea about how the DataInputStream class is basically used particularly here in this particular example to reach something from the keyboard.

(Refer Slide Time: 05:51)



## Java File I/O

Java provides java.io package which includes numerous class definitions and methods to manipulate file and flow of data ( called File I/O streams )

There are four major classes:

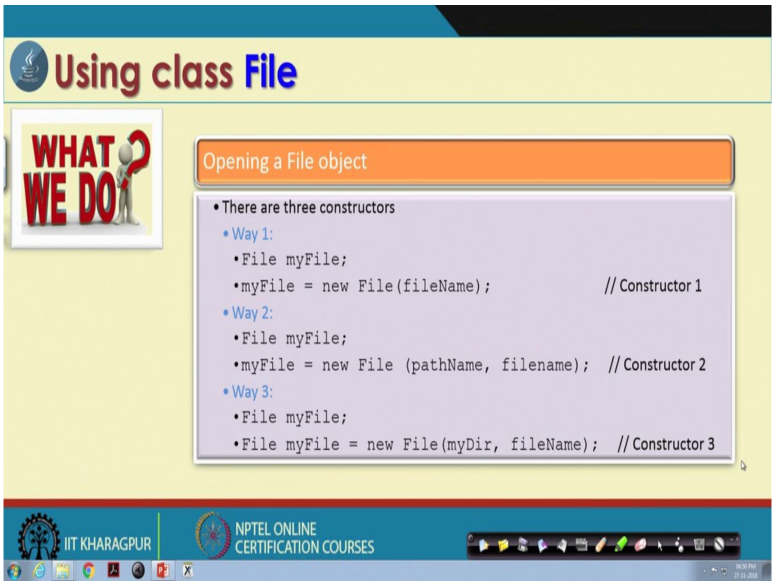
- File
- FileInputStream
- FileOutputStream
- RandomAccessFile

Now so, this is just an example and one more important example that is dealing with files. So, how we can read some data from the file? How we can write into file? What are the different way the file can be read. For example, using byte stream classes or how a file can be handle using characteristic classes etcetera. So, our next few slide includes this kind of definition and mechanism rather discussion. So, this file handling right.

So, as I already mention that a java provides the package call the java.io package which includes number of class declaration and the methods in its class and one important methods one important class out of the so many classes is called the File class. So, it is basically called File IO stream more precisely. So, there are 4 major impact classes belongs to file IO stream is are there.

For example, the class File so, it is a general file and then class FileInputStream, this is regarding how we can read from a file and then FileOutputStream - how we can write something into a file and then another is that RandomAccessFile. So, all the File, FileInputStream, FileOutputStream their sequential that mean reading a one by one where as RandomAccessFile allows one to read and from any random position. So, these are the four different classes rather we can say four different mechanisms to deal with the input output stream related to the file.

(Refer Slide Time: 07:31)



## Using class File

Opening a File object

- There are three constructors
  - Way 1:
    - File myFile;
    - myFile = new File(fileName); // Constructor 1
  - Way 2:
    - File myFile;
    - myFile = new File (pathName, filename); // Constructor 2
  - Way 3:
    - File myFile;
    - File myFile = new File(myDir, fileName); // Constructor 3

Now, let us see what we can do. Now so, for the file is concerned first task is that how we can open a file object; that means, file is basically in a object oriented concept everything is an object. So, file is an object. how we can create a file object? Now this is basically as I told you there is a class called File. So, this basically this declaration; if you declare, then myFile is an object of the file and then it so my file can be open using this statement new File fileName. Here basically every file that is there in your memory or in your system, they are basically have a unique name. So, you have to pass the fileName.

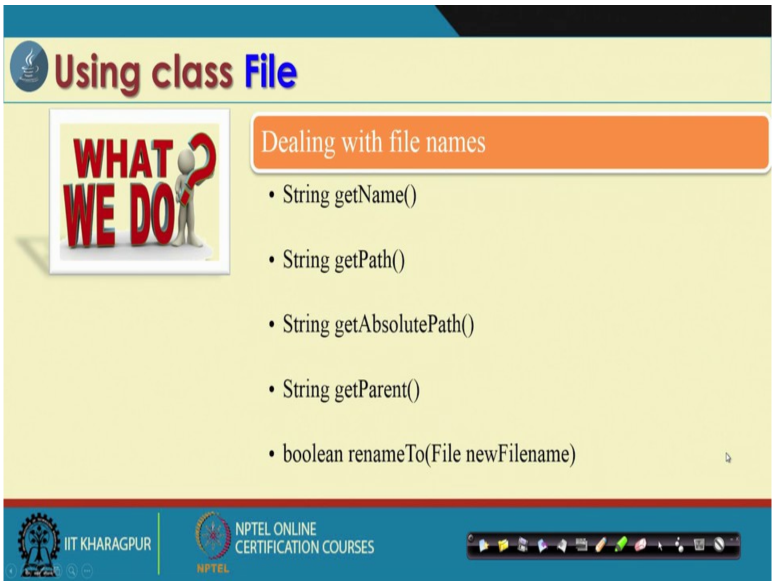
So, if you this is basically constructor. So, if we passed a fileName and then we can create a file object. So, creating this file object means that from your program, you connect one connection to that fileName file. So, this is a one way and these the first constructor that you can use it. There is another constructor file that also you can mention the pathName and then fileName.

Now, the difference between this constructor and this constructor is that, it assume that your file is belongs to the same directory from where your working that is from your working directory and the file is store in the same working directory whereas, if we want to do access a file which is not currently in your working directory that you can specify pathName and filename.

So, these the second constructor and this is also similar; third way very similar to this one here basically in the same working directory, but in the different directory. So, you can specify which is the directory under your working directory and what is the file name. So, these are the three constructor by which we can create a file object. In any way there are actually the one main thing that is there file names should be mentioned uniquely, if do not mention the right fileName, but you try to open it then it will written an error.

So, error is basically handle that way. So, this is the try catch block can be there. So, it can to check the exception if it is occur and then it is. So, sometimes that file is existing, but the file is damaged or that file is existing, but you do not have any permission to read or write. So, in that case also it will throw an exception and then exception can be handle accordingly to make the program robust.

(Refer Slide Time: 10:21)



## Using class File

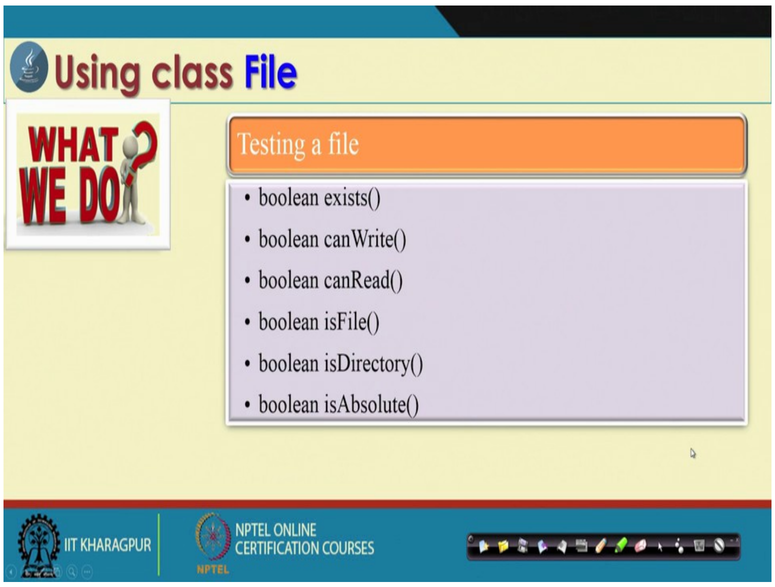
Dealing with file names

- String getName()
- String getPath()
- String getAbsolutePath()
- String getParent()
- boolean renameTo(File newFilename)

WHAT WE DO?
IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
NPTEL

So, this is the different way the file objects can be created and once the file object is created, then we will be able to have the different information through this file object. For example, there are many methods which are related to this file class say getName- it basically written the full name of the file that you have currently connected to, the getPath - it will say the what is the path, gateAbsolutePath- it is basically from any directory it will find and then path will be there, getParent-what is the directory above this the file where it is present and then whether renameTo- if want to rewrite the file into a some other name also it is possible using this file object. So, they are the those are the many utility methods we can say available related to the file.

(Refer Slide Time: 11:07)



## Using class File

WHAT WE DO?

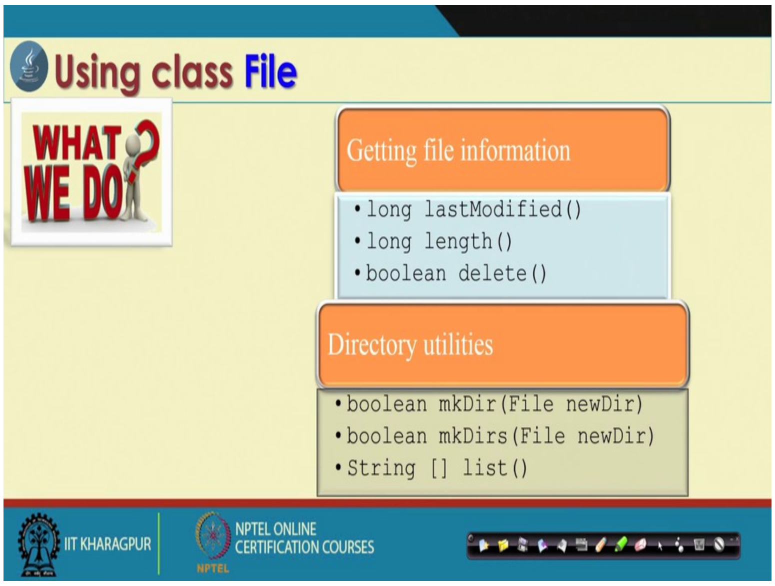
### Testing a file

- boolean exists()
- boolean canWrite()
- boolean canRead()
- boolean isFile()
- boolean isDirectory()
- boolean isAbsolute()

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
NPTEL

And there are many other things also you can test a file that ok. There is some methods like whether with the file exists or not, you can just do this using the exists method canWrite; that means, whether the file is writable, canRead whether it permission is there whether it is a file or it is something then say other form, is a directory or is a absolute; that means, it is a standard file or not. So, those are the things that you can do using this file object.

(Refer Slide Time: 11:43)



## Using class File

WHAT WE DO?

Getting file information

- long lastModified()
- long length()
- boolean delete()

Directory utilities

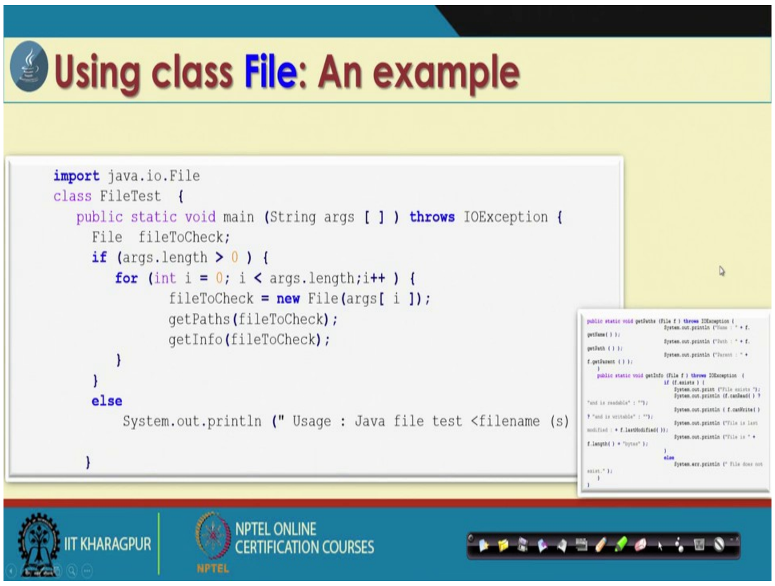
- boolean mkDir(File newDir)
- boolean mkDirs(File newDir)
- String [] list()

IIT KHARAGPUR
NPTEL ONLINE CERTIFICATION COURSES
NPTEL

Now so, regarding this file information few more things also we can have just lastModified; that means, it will tell tell you the date and time when the file has been correct I mean use last time and it will also has the method length. So, that it will written the length of the file. If you want to delete a file that also you can do using delete method and there are some directory till it is; that means, using a file we can create a directory and then in the directory the file can be replace there and make directories more than one directory also can be created and list of files that is store in a particular directory also can be accessed using method list method it is there.

So, there are many utilities methods related to file which basically helps the program are to handle the files in a in their own ways in the different ways that they want.

(Refer Slide Time: 12:41)



## Using class File: An example

```
import java.io.File
class FileTest  {
   public static void main (String args [ ] ) throws IOException {
     File  fileToCheck;
     if (args.length > 0 ) {
        for (int i = 0; i < args.length;i++ ) {
              fileToCheck = new File(args[ i ]);
              getPaths(fileToCheck);
              getInfo(fileToCheck);
        }
     }
     else
         System.out.println (" Usage : Java file test <filename (s)

    }
```

```
public static void getPaths (File f ) throws IOException {
                              System.out.println ("Name : " + f.
getName( ) );
                              System.out.println ("Path : " + f.
getPath ( ) );
                              System.out.println ("Parent : " +
f.getParent ( ) );
    }
    public static void getInfo (File f ) throws IOException  {
                              if (f.exists ) {
                                  System.out.print ("File exists ");
                                  System.out.println (f.canRead( ) ?
"and is readable" : "");
                                  System.out.println ( f.canWrite( )
? "and is writable" : "");
                                  System.out.println ("File is last
modified : + f.lastModified( ));
                                  System.out.println ("File is " +
f.length( ) + "bytes" );
                              }
                              else
                                  System.err.println (" File does not
exist." );
    }
}
```

Now, let us have a simple example about we have learned about the different method those are there in file and some methods related to the checking status of a file; whether file exists, file available, what is the name of the file, what is the path and everything like like. So, this example will help you to understand the idea about status of the file.

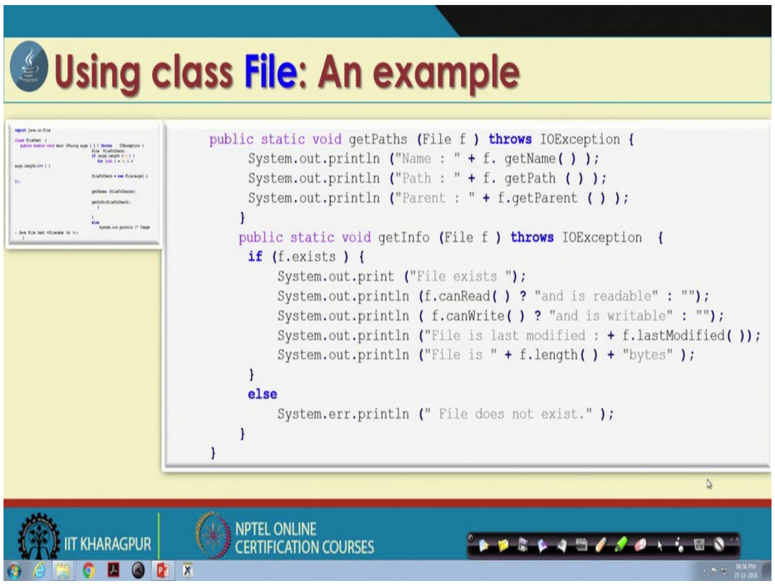
Now, let us have a quick look about this program as we see this is the class that we have been given and we import these the java.io.File because we want to have the methods those are there. Now as you see here so this is the main method and; obviously, throw and try catch blocks should be included here in order to make the program exception; handle the exception more reliable rather. So, here we see first what we do we create an object call fileToCheck. This is the name of the object of type of class File.

So, this is the object that we want to do. Now in this program we assume that user will send the name of zero or more files for which you wants to have the information about those files. Now here if user does not give any file name whenever he runs program, then is basically args length equal to 0. In that case it will say that java FileTest, this is the name of the program and file name you should give at least one or more file names there.

Now, suppose if you if user gives at least one or more one or more file name, then it will basically enter into this loop. Then for each file each input into the common line i = 0; i< args dot length, it basically create the method say the file to check newFile (args dot length); that means, it crate an connection from the program to the file which is basically in the input. Now so, for this file will just find this method getPaths and getInfo these are the two methods which is basically under this method we define this two method here.

Now so, let us see what is the getPaths method and getInfo user define method. Now here is the idea about here is the getPath method.

(Refer Slide Time: 15:13)



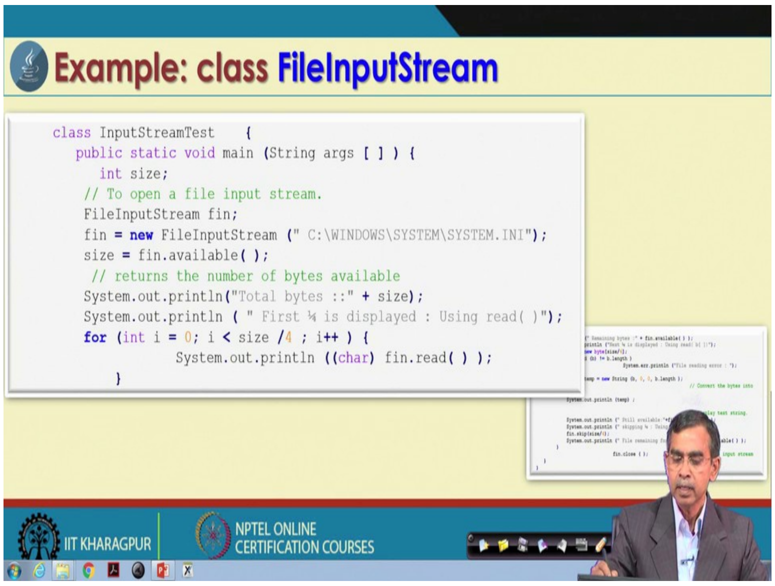
## Using class File: An example

```
public static void getPaths (File f ) throws IOException {
     System.out.println ("Name : " + f. getName( ) );
     System.out.println ("Path : " + f. getPath ( ) );
     System.out.println ("Parent : " + f.getParent ( ) );
    }
    public static void getInfo (File f ) throws IOException  {
     if (f.exists ) {
         System.out.print ("File exists ");
         System.out.println (f.canRead( ) ? "and is readable" : "");
         System.out.println ( f.canWrite( ) ? "and is writable" : "");
         System.out.println ("File is last modified : + f.lastModified( ));
         System.out.println ("File is " + f.length( ) + "bytes" );
     }
     else
         System.err.println (" File does not exist." );
    }
}
```

As we see so, getPath method which is define under this class. These the FileTest class we can say under this method, this is the getPath. Now if you see getPath method f dot getName. So, this is basically will pass this function here basing a fileToCheck like and then f.getName, getName is a built in method which is defining in the class file.

Similarly, getPath the method will written about the file object, the path and getParent written the parent of the file directories of this one and then it will written it. So, it will print after getting about the file object path name and everything it will written, it will displayed on the input screen and similarly the getInfo is also user defined one method and it basically has that f dot canRead; that means, readable or writable read write permission it is there or what are the date or time that it has been modified, what is the length of the file like this one. So, these are the different way the status of a file can be learned using the file object and the different methods belongs to this class File.

(Refer Slide Time: 16:37)



## Example: class FileInputStream

```
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.INI");
    size = fin.available( );
     // returns the number of bytes available
    System.out.println("Total bytes ::" + size);
    System.out.println ( " First ¼ is displayed : Using read( )");
    for (int i = 0; i < size /4 ; i++ ) {
                System.out.println ((char) fin.read( ) );
        }
```

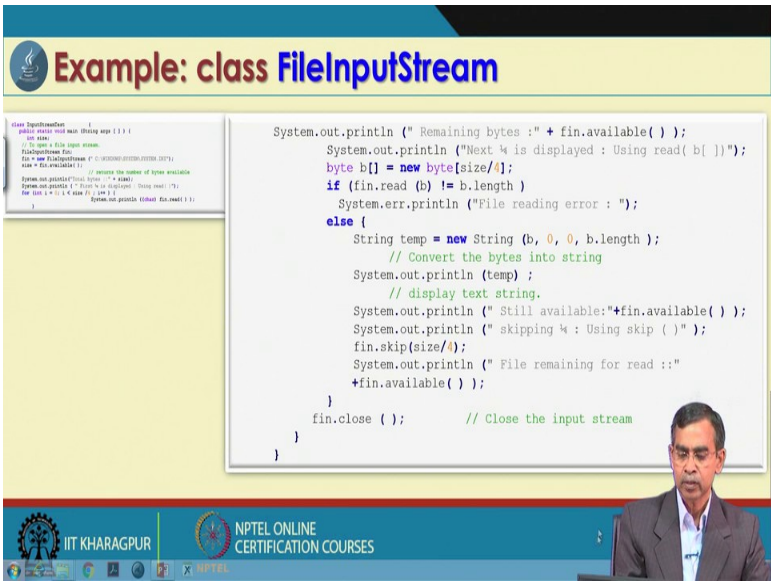
Now so, we learn about how to open a file object. Now let us have an idea about how some contents which is already there in a file can be read from this file to the program. So, it is basically reading a file. Now we have small program to illustrate this concepts. Now let us check this program here. We define one class, this basically to check that how we can read a file. The name of the program here is file input InputStreamTest.

Now, in this program as you see the, this is the declaration of main method and here again you see we create FileInputSteam. So, FileInputSteam fin as you know this FileInputSteam is basically belongs to the byte Stream classes. So, it will read as a byte form like. Now here the method that is already define in the files input stream classes call the available. So, it basically if we call this method available for this object FileInputSteam object, then it will written what is the total bytes that is there in the file. So, it basically written the number of bytes and it will print the total bytes is this one.

Now, here is the suppose how we can read first one fourth portion of the file. So, it is basically using this for loop for (i = 0; i<size/4; it will read fin.  fin read basically the read method that is there in the FileInputSteam class and then it read the byte and then it basically convert this byte into characters. So, is basically read the byte may the ASCII value and then convert in the character that character will be display on the screen.

So, whatever the contain of the file, it will be there and then first one-fourth position of the byte will be read and it will display on the screen. The same program can be extended to read part by part reading. Here is the next part of the program with basically tell you how the other portion of the file can be read.

(Refer Slide Time: 18:53)



## Example: class FileInputStream

```
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.INI");
    size = fin.available( );
                              // returns the number of bytes available
    System.out.println("Total bytes ::" + size);
    System.out.println ( " First ¼ is displayed : Using read( )");
    for (int i = 0; i < size /4 ; i++ ) {
                               System.out.println ((char) fin.read( ) );
       }
```

```
System.out.println (" Remaining bytes :" + fin.available( ) );
        System.out.println ("Next ¼ 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, 0, b.length );
                  // Convert the bytes into string
            System.out.println (temp) ;
                  // display text string.
            System.out.println (" Still available:"+fin.available( ) );
            System.out.println (" skipping ¼ : Using skip ( )" );
            fin.skip(size/4);
            System.out.println (" File remaining for read ::"
            +fin.available( ) );
        }
      fin.close ( );         // Close the input stream
    }
}
```

Now, continuing this discussion you see this basically remaining bytes; that means, it basically; so after reading one fourth byte it will tell that, how many bytes yet to read it. So, it basically give the print statement this is the file that is still remaining and then for the remaining byte again we can write here the different way just explaining that new byte size 4.

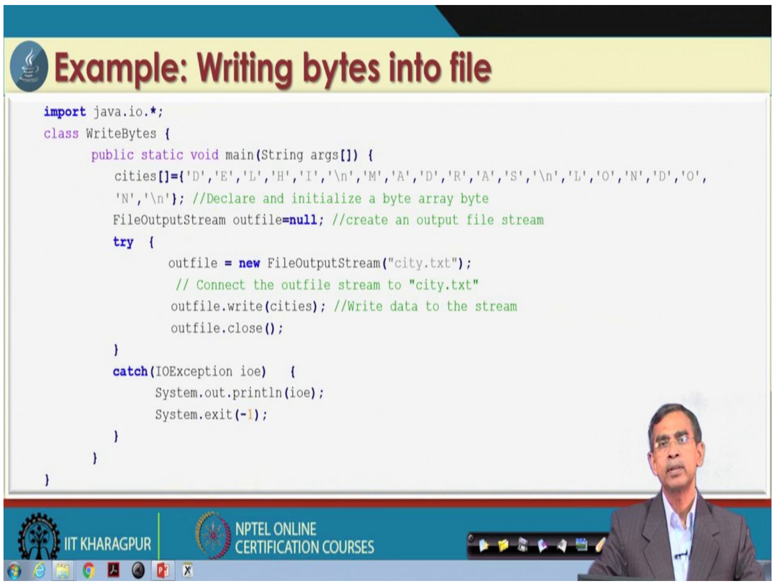
So, this basically my plan is to read the next one-fourth part of the byte from the current location and those byte will be store into this array temporary buffer we can say. So, we declare a byte array of byte array namely the b here of size by 4. So, it will be declare so, that will we can store it. Now here read b the another way the method can be called. It basically read the buffer I mean byte input stream file here and store the result into the b array and if (fin dot read not equal b dot length); that means, it is not the capsized, then defiantly it is an file error and if it is not there. So, if there total of bytes is successful read actually then it will basically call this part here.

So, we just string (b, 0, 0 ,b dot length) is basically converting the string and then we store into the byte that we have written and store into the temp this string. So, is basically the idea about how we can convert a bytes into a string that is basically more convenient. So, that if the byte stream is covert into string and then we can process the string.

Now, here we can see once the whole byte array is converted the String in the last example, we are storing that one particular byte can be converted the character, but here the entire bytes stream can be converted in to the string and then this string can be printed on the screen. So, this is the statement and likewise we can again still available this one is basically after reading first one-fourth part of the file, then next-one fourth of the part of the file then the remaining file.

Now it basically says that this file is available and here you can see there is again the methods skip is basically skip the byte number of bytes that is there. So, is basically keep size by 4. So, this way up to this we can cover the three-fourth part of the file into the input stream and then file remaining for read is basically a fin available. So, it is this one. So, this basically shows that, how sequentially we can process we can cover we can traverse an input file.

(Refer Slide Time: 21:53)



## Example: Writing bytes into file

```
import java.io.*;
class WriteBytes {
      public static void main(String args[]) {
         cities[]={'D','E','L','H','I','\n','M','A','D','R','A','S','\n','L','O','N','D','O',
         'N','\n'}; //Declare and initialize a byte array byte
         FileOutputStream outfile=null; //create an output file stream
         try  {
                outfile = new FileOutputStream("city.txt");
                 // Connect the outfile stream to "city.txt"
                outfile.write(cities); //Write data to the stream
                outfile.close();
         }
         catch(IOException ioe)   {
               System.out.println(ioe);
               System.exit(-1);
         }
      }
}
```

Now so, this is about how we can read. Here we have use the byte input stream classes one mechanism reading in the form of a byte. Now will discuss about the just opposite one process call writing into a file; how a byte can be write into a file or a character or a string can be write into a file. Now here is a small program again to illustrate the concept of writing bytes into file. Now here is a method class that we have discussed here namely, the write bytes. Basically writing bytes into file and this program you see this is basically the character string which basically show that this is the args values of D args value of E basically bytes.

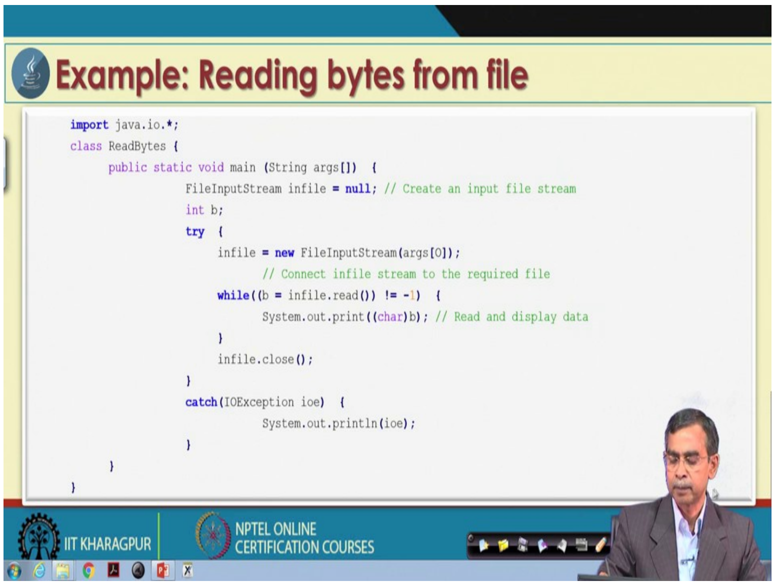
So, it is basically stream of bytes we first store into an array call the cities. So, it basically store all the information in the form of a byte it is there and then we want to write it into a file, for this things here we see we create an object call out file of type FileOutputStream. So, if you want to read some file, then we use FileInputStream. On the other hand, if we want to write something into a file, we should create an object of class FileOutputStream and this output file is basically the object that we connect your program to a file where you want to write something. Now so, it is basically we put this code under try catch block to make the program robust. If there in exception occurs so, that it can be caught here and here you see we first created we just declare an object of out file of type FileOutputStream.

Now, create an object, in order to do this things we first pass a file name where you want to store it so, city dot txt. If it is already exists, then fine I will come to this discussion later on. So, if the city dot txt is the file name by where the it will make an it will establish a connection from your program to that file and this is the connection basically objects is basically established here.

Now, here out file write is the method. If we pass this array so, this is the array of bytes actually if pass there the entire array of bytes will rise into this files city.txt. Now here again one important thing is that, if already there exists a file names city.txt and if we write it, then the previous content will be over written; that means, it will erased only then new content that you want to do write it will be stored there.

Now so, this example explain you how a large shrunk of data can be stored into a file and then finally, the out file dot close; that means, you have to close the file it is always customize that once you open a file you should close a file. So, here is the basically opening a file and here is the closing file after your operation is there. So, this program explain nicely that how we can write something into a file.

(Refer Slide Time: 25:13)



## Example: Reading bytes from file

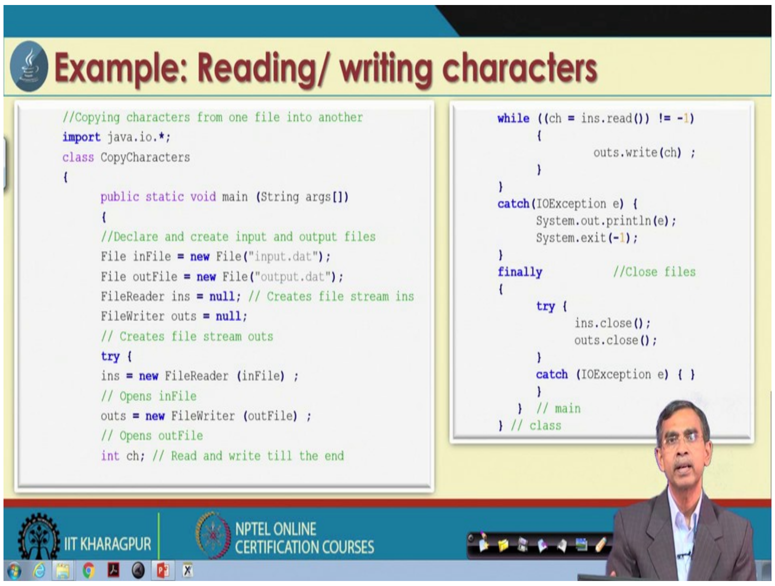
```
import java.io.*;
class ReadBytes {
    public static void main (String args[])  {
            FileInputStream infile = null; // Create an input file stream
            int b;
            try  {
                infile = new FileInputStream(args[0]);
                    // Connect infile stream to 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);
            }
    }
}
```

Now so, so we have learn about reading; reading from a file and writing into a file using the different method data input stream and data output stream classes. So, there are many other things way also a file can be read. Now I am to discuss the other mechanism how the file can be read.

Now so, in the last example we have been discussed about a FileInputStream and FileOutputStream. So, that we can read form a file or we can write it from a file here. So, here again another example which explain how the reading in the form of a bytes from a file is possible. So, again same thing the FileInputStream, this is the class for which an object in file to be created and then again we just make a connection and then read method is basically read. Now here this read method read one byte at a time.

So, this is the read method and here is basically this basically is a end of file; that means, it will continue read until the end of file occurs. Now you read one byte store this in a temporary b here and then this b can be converted into character and then the same can be printed that mean it will read a file and then content will be despite on the standard input device there is a System.out.print. And finally, once the entire file is read one byte by one another, then it will close this is very simple way, it is most similar the file output stream classes that we have consider there or like. So, it is the idea actually the sequentially one byte at a time and then the inter file can be diverse or can be read.

(Refer Slide Time: 27:11)



## Example: Reading/ writing characters

```
//Copying characters from one file into another
import java.io.*;
class CopyCharacters
{
    public static void main (String args[])
    {
    //Declare and create input and output files
    File inFile = new File("input.dat");
    File outFile = new File("output.dat");
    FileReader ins = null; // Creates file stream ins
    FileWriter outs = null;
    // Creates file stream outs
    try {
    ins = new FileReader (inFile) ;
    // Opens inFile
    outs = new FileWriter (outFile) ;
    // Opens outFile
    int ch; // Read and write till the end
```

```
    while ((ch = ins.read()) != -1)
        {
                outs.write(ch) ;
        }
    }
    catch(IOException e) {
        System.out.println(e);
        System.exit(-1);
    }
    finally          //Close files
    {
        try {
            ins.close();
            outs.close();
        }
        catch (IOException e) { }
        }
      } // main
    } // class
```

Now, let us ok. So, we have been discuss so far using the byte stream classes. Now it is necessary to discuss about again doing the same thing, but using CharacterStream class. So, here is the one simple program which includes how a file can be read and or it can be write in the form of a characters is very similar to previous one only the method that whether byte or character that is all.

Now here you can see this is the method class we have declared and in method here we can see what we have done inFile new File, this is the name of the input file from where you want to read and these are the name of the output file where you want to write into. So, these are two object inFile and outFiles for the input file and then the output file is created and then here we use FileReader is basically to create I mean to read a file and FileWriter to read write a file.

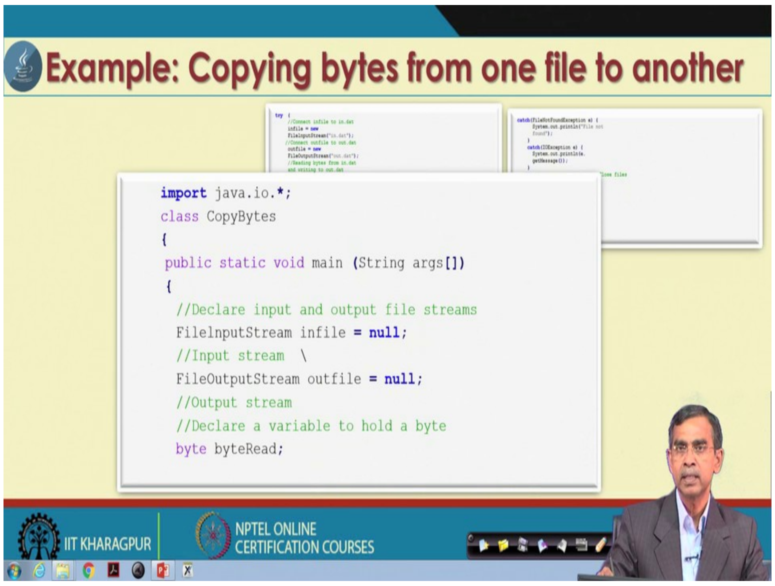
So, the two other objects are created. Those are will be responsible for reading a file and outputFile and this basically once the connection is established. Now here this ins basically is now object is declared here and now object is created; that means, ins is now connect to this using this FileReader constructor giving the input file; that means, this objects. So, the inFile input data is a input file objects output data is output file objects and for this using ins method and outs method we make a connection between the two objects input data output data from the program and now this is the integer character is a temporary variable and here we can see ins read; that means, from the input object the read method will read one character at a time and it is basically read until the end of file.

So, this basically end of file and then it read a character from a file and outs write that mean we write the same character into the output file named here  output dot dat. So, it basically read one character from a input file call input.dat and then write a same character input and output file call output dot dat. Again then once the whole the entire file is read and write it is over, then it basically close the two objects here using the close method for the two objects.

So, this way you can see how reading and writing is there and here we can see all those exception handling mechanism is there. In case suppose we are not able to open the file or not able to write the file or file is corrupted or whatever the file permission is not there so many reasons and then and then output dot file as it there as it is a override so always in override. So, if the contain that is not an appending; that means, what are the contents are there it will go on adding is not like that, but it can be done also if the different way.

Now, here basically we see that if the file already exists, then it will start writing from the very beginning. So, the previous content will be erased. So, this is the way that using character stream class how the copying of a file is possible.

(Refer Slide Time: 30:25)



## Example: Copying bytes from one file to another

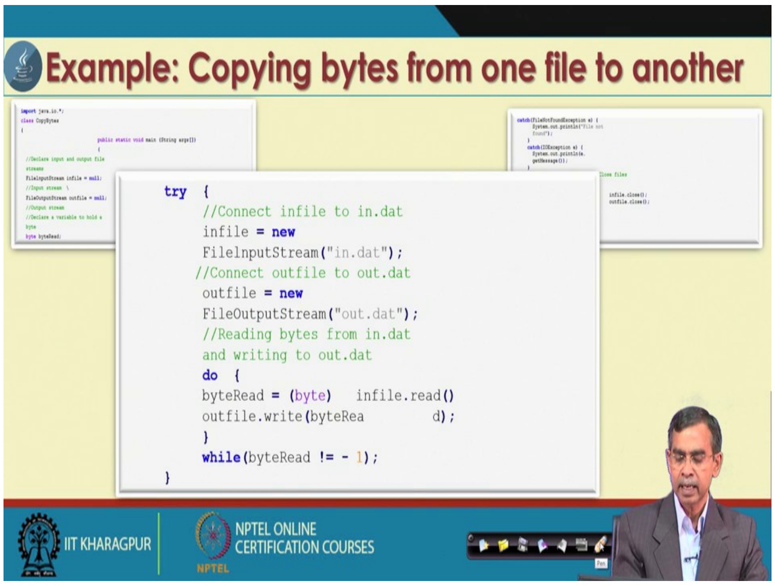
```
try {
    //Connect infile to in.dat
    infile = new
    FileInputStream("in.dat");
   //Connect outfile to out.dat
    outfile = new
    FileOutputStream("out.dat");
    //Reading bytes from in.dat
    and writing to out.dat
```

```
catch(FileNotFoundException e) {
    System.out.println("File not
    found");
  }
  catch(IOException e) {
    System.out.println(e.
    getMessage());
  }
```

```
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;
```

Again again using the same, but using byte stream classes also how the copying can be possible. I will just tell you the mechanism for this. Here again so we can create two stream FileInputStream and FileOutputStream the two object in file and out file for the input streaming and for the output streaming. And here as we want to use the byte stream class say so instead of character, we just declare a temporary variable call the byteRead. So, where byteRead that means, where the buffer byte will be temporary read and store there.

(Refer Slide Time: 31:09)



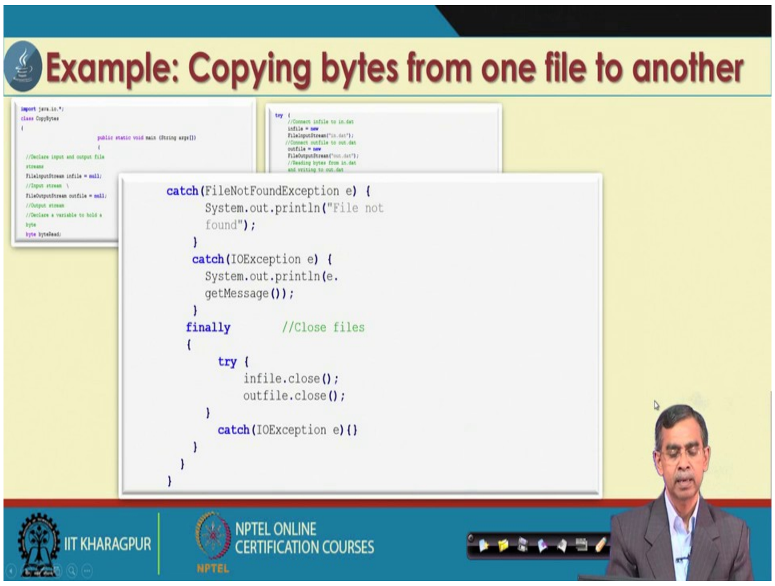
## Example: Copying bytes from one file to another

```
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(byteRea        d);
    }
    while(byteRead != - 1);
}
```

Now, the next is that we have to connect the file. So, here for this connect we create an object in file new using this input file name and similarly output file name in discuss in data and then output dot dat. Now so, this way we can connect create an connection from the program to the input source and then output store source and then it will basically start reading until the in file is completes.

So, it is basically byte read byte in file read out file write byteRead. So, the same will read and write and it is basically while the end of file is there. The concept more or less same in each cases only the mode the way we can read actually and then copy into there. Here in this example as we see in the form of a byte and then finally, the method will be here ok.

(Refer Slide Time: 32:05)



## Example: Copying bytes from one file to another

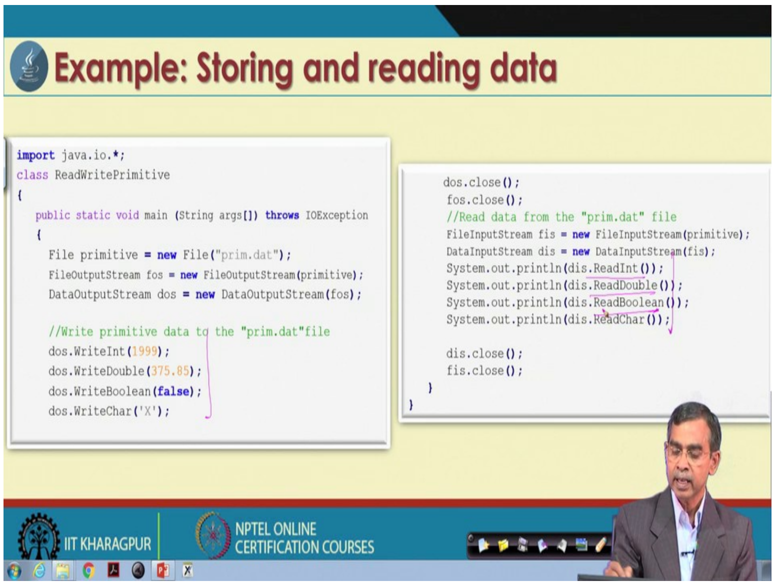
```
import java.io.*;
class CopyBytes
{
                public static void main (String args[])
                {
 //Declare input and output file
 streams
 FileInputStream infile = null;
 //Input stream  \
 FileOutputStream outfile = null;
 //Output stream
 //Declare a variable to hold a
 byte
 byte byteRead;
```

```
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
```

```
    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){}
     }
   }
}
```

We have to have some closing mechanism that this basically closing.

(Refer Slide Time: 32:11)



## Example: Storing and reading data

```
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');
```

```
     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();
   }
}
```

Now so, we learn about reading, writing copying and then storing data into a file. Now here again see from the program how the data can be store interactive manner rather in one by one like. So, this example is basically explain you how the data can be post into some file. So, here basically read write primitive it will read from file and write it another file, but storing some data from the program generated like example.

So, first we create a File type object that is the name of the file where you want to store it, this basically the name of the file and then we create a File object primitive. So, this is the File object once the file object is created, then it basically FileOutputStream is there; that means, connection. So, FileOutputStream class connection to this primitive means this file. So, fos and DataOutputStream is basically another stream for the data out prim, then this is the objectives is created; that means, here if you see this is the file and fos for FileOutputStream and dos for the DataOutputStream.

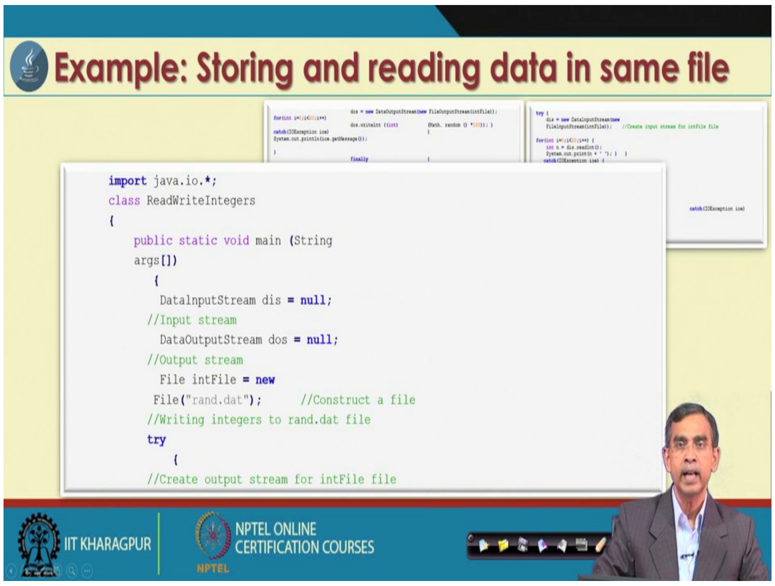
So, through the data output, we have to connect to the file output and then it basically connect to the file for storing some data into here. Now here again you see for the DataOutputStream WriteInt the method is there write int; that means, it knows how to write an integer in to this data input DataOutputStream. Similarly there is WriteDouble method there in the DataOutputStream classes which basically it helps how to write a floating point number in to the object and Boolean also how and then characters.

So, here we can see the four different type of data which can be store into a DataOutputStream and then this basically store it. Once the DataOutputStream is ready, then the same data can be post in to this FileOutputStream. Now this is the method and the rest of the part here as we see ok.

So, we just write we just write in all this things is basically create the of these are the value to the, this file output stream and finally, we have to close it. Now again the object that we have created now that can be read also the opposite mechanism here; here we can see FileInputStream and DataInputStream the same can be read from the other version also. So, here basically we write it here and here will read the same content using DataInputStream and FileInputStream mechanism. So, both way the things it is there.

So, this example will tell you how we can store the data from program. These are the data from the program into some File object and then same data which is their how we can read it here, but while you are reading you have to mention that which type of data you want to read it. If you know that this structure if you read it, then it will what find; if you do not read in that order. So, first it is ReadDouble and this one then some error you encounter. So, we have to be careful about that in the way the data is stored the same way it can be. So, this way any object having their values can be store into the array.

(Refer Slide Time: 35:33)

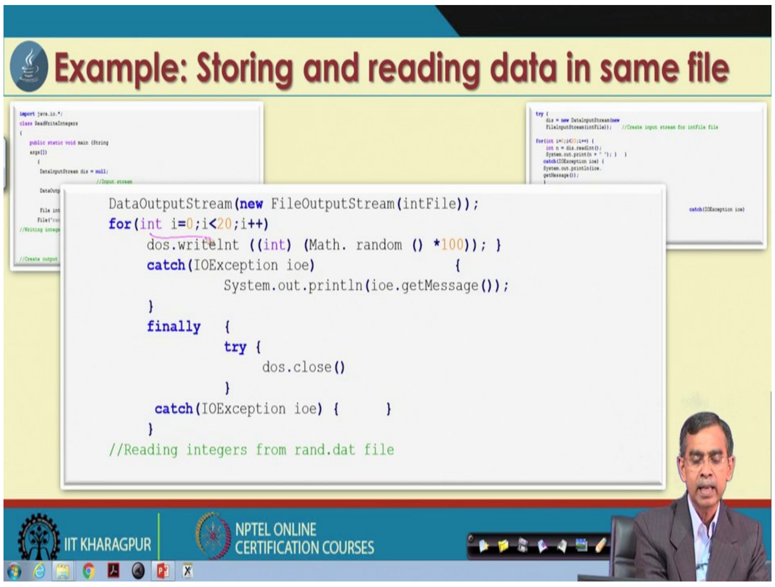


## Example: Storing and reading data in same file

```
import java.io.*;
class ReadWriteIntegers
{
    public static void main (String
    args[])
      {
       DataInputStream dis = null;
     //Input stream
       DataOutputStream dos = null;
     //Output stream
       File intFile = new
      File("rand.dat");      //Construct a file
     //Writing integers to rand.dat file
     try
         {
     //Create output stream for intFile file
```

Now so, storing and reading into the same file. Here we can see we have store ones and then read another, but at the same time both things can be done. So, this is an example to explain it here and here the read write integers and here we can see this is the create objects of DataInputStream dis and this is also output stream objects. So, two objects are created of type input stream and output stream. A file is write declared here the, this is the random data file here and here you can see how we have developed the program. So, it basically generate a random number and store into the program here.

(Refer Slide Time: 36:13)

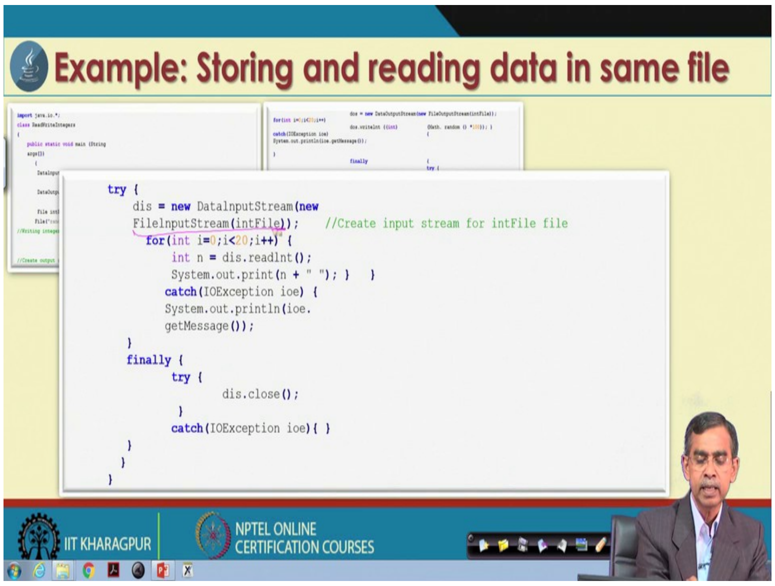


## Example: Storing and reading data in same file

```
DataOutputStream(new FileOutputStream(intFile));
for(int i=0;i<20;i++)
     dos.writeInt ((int) (Math. random () *100)); }
     catch(IOException ioe)          {
               System.out.println(ioe.getMessage());
     }
     finally  {
               try {
                    dos.close()
               }
      catch(IOException ioe) {       }
     }
//Reading integers from rand.dat file
```

Now, here is basically for i equals to 0 total 20 random number will be created and then they will be generated using random function which is there in java dot line and then write into that stream. So, this way it basically write into the file. Once the file writing is over we can write into the something into file into the file here.

(Refer Slide Time: 36:41)



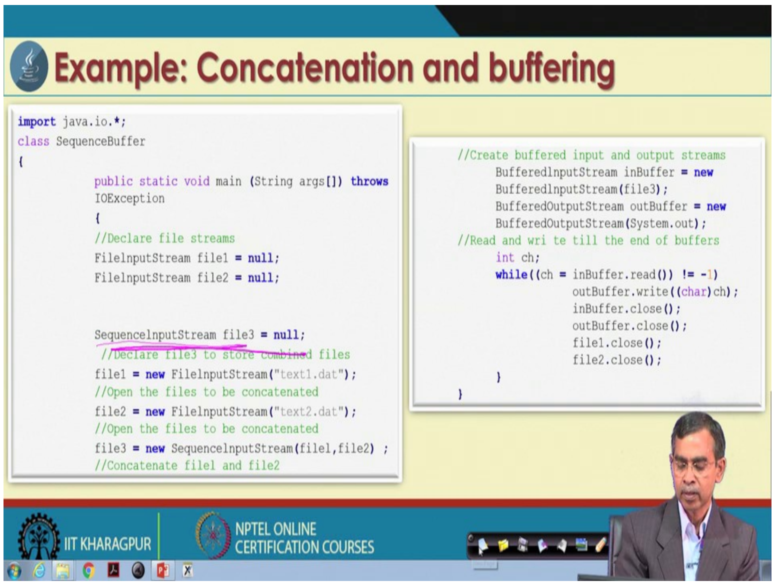
## Example: Storing and reading data in same file

```
try {
    dis = new DataInputStream(new
    FileInputStream(intFile));    //Create input stream for intFile file
     for(int i=0;i<20;i++) {
         int n = dis.readInt();
         System.out.print(n + " "); }   }
        catch(IOException ioe) {
        System.out.println(ioe.
        getMessage());
  }
  finally {
         try {
                 dis.close();
          }
         catch(IOException ioe){ }
  }
 }
}
```

As here is a file input stream the intFile again these are the values that is there and readInt all the; that means, whatever the file that we have, written they are we can read we can read in this form and then finally, it is close.

So, this basically the idea that tell you how a random data which is generated in your program can be propagate can be post to the file in a memory ok.

(Refer Slide Time: 37:09)



## Example: Concatenation and buffering

```
import java.io.*;
class SequenceBuffer
{
            public static void main (String args[]) throws
            IOException
            {
            //Declare file streams
            FileInputStream file1 = null;
            FileInputStream file2 = null;


            SequenceInputStream file3 = null;
             //Declare file3 to store combined files
            file1 = new FileInputStream("text1.dat");
            //Open the files to be concatenated
            file2 = new FileInputStream("text2.dat");
            //Open the files to be concatenated
            file3 = new SequenceInputStream(file1,file2) ;
            //Concatenate file1 and file2
```

```
        //Create buffered input and output streams
              BufferedInputStream inBuffer = new
              BufferedInputStream(file3);
              BufferedOutputStream outBuffer = new
              BufferedOutputStream(System.out);
        //Read and wri te till the end of buffers
              int ch;
              while((ch = inBuffer.read()) != -1)
                          outBuffer.write((char)ch);
                          inBuffer.close();
                          outBuffer.close();
                          file1.close();
                          file2.close();
              }
        }
```

Now, next is merging two files. It is basically the file buffer input stream mechanism and this program is very simple here you can see we first create two FileInputStream file one and file two and there is a sequence stream SequenceInputStream file buffer. This basically take the two files and then convert into one file like. So, here is idea about file 1 the first object, file 2 is the second object that we have created. These are the name of the file suppose and then file 3 is basically the concatenation or merging of the two file.

So, we can create an object SequenceInputStream file 1 file 2. So, this basically concatenate the two files text 1 data, text 2 data into one file and the file can be store there. Now for storing, here is the idea about the BufferedInputDtream in buffer and this is the file 3 and then we can create it and then the same thing can be stored and then here is basically the content which is there can be read and then display it.

So, this way, this is the this is an example of SequenceInputStream class which basically concatenate two files into one and then same file can be displayed or can be managed or can be processed ok. So, these are the different way the file can be handle as we have discussed here and in our next module, we will discussed about the random access mechanism to handle the file.

So thank you; thank you for your attention.