# Lab IO1: Byte Streams

## Objectives

In this lab you will be copying a file byte by byte using FileInputStream and FileOutputStream

## Lab Setup

1. Create a Java Project with a package called iolab
2. Copy the sample data file from the lab resources into the root of the project directory. If you put it into any other location, your code will not be able to find it and will throw an IOException when an attempt is made to open it.
3. You can create your own file to use as sample input but you should ensure that it is a UTF-8 file if you want to use it in the next lab
4. Note: that the output file will not be visible in the project explorer since it is not added to the list of files tracked by the project manifest when it is created. To see your output file, you have to look at the project directory with the Windows file explore

## Coding the Java Components

### Step 1: Create the lab class

1. Create a class with a main method. In the example that follows the class is named ByteCopy.
2. Graphical user interface, text, application

   Description automatically generatedAdd the notation that the main class throws an IOException. This is not really a good practice – to have an unhandled IOException, but this little shortcut will make coding the lab simpler.

### Step 2: Set up the try catch block

1. Create a try-catch-finally block with the catch clause catching an IOException
2. Text

   Description automatically generatedIgnore the error saying that the exception is never thrown, this is because we don’t have any code in the try block.

### Step 3: Add the input file operations

1. Good programming practice is to close every file that is opened.
2. Add the open file code in the try block and the close file operation in the finally block – the reason for placing this in the finally block will be apparent soon.
3. Add the sample file, here named “SampleText.txt” into the main project directory. This file is UTF-8 sample file provided by W3C org for testing purposes.
4. You can copy the file from wherever it is then paste it into the project directly in Eclipse. It should look like this. Note that the file is not in the “src” directory bit in the main root project directory.

Graphical user interface, application

Description automatically generated

1. Add the statements to open and close the input file as well as an output statement in the catch block that will advise us if the input file cannot be opened. If your code throws this exception at this point the lab, that means that you have put the input file in the wrong place
2. Add a temporary output statement at the end of the code so we can confirm the code executed.
3. Run the code as a Java application to check that it works. The listing is on the next page.

Text

Description automatically generated with medium confidence

### Step 4: Add the output file code

1. Text

   Description automatically generatedThis is shown in the code below.
2. Run the code and then check to see that the output file is created. Note that you cannot see the file in Eclipse so use the Windows file explorer like the following:

Graphical user interface, application

Description automatically generated

### Step 5: Add the code that reads and writes the bytes

1. Add an integer variable byteCount that will count the number of bytes copied.
2. Add a byte type variable as well to receive the byte input. Notice we have to cast this to a byte since the read() method returns and int and assigning it to a byte sized variable is a narrowing conversion. The conversion works correctly because the byte read is stored in the lowest byte of the int.
3. As an experiment, you can use an int instead of a byte and confirm that code still executes correctly.

Text

Description automatically generated

### Step 6: Confirm the code worked

1. Run the code
2. Open the output file in the file explorer to ensure that the copy worked correctly.
3. Don’t delete your project. You can modify it to do the next lab.

Text

Description automatically generated with medium confidence