Purpose	Learn how to read and write files using Java I/O classes and catch exceptions.
What to Submit	Commit your solution to Github as a project named **`fileio`**. Use the package **`ku.util`** for Java classes in this lab. Include a **`README.md`** file containing a table of run-times for the different parts, and explanation of the results.
Files and Sample Code	Text file is in the "`docs`" directory of course Bitbucket repository. Starter copy for problem 2 is in `week6/FileCopyTask.java` on Bitbucket.

The Sample File

The sample file for this lab contains multiple copies of the story *Alice in Wonderland.* The file is approximately 2,101,560 bytes.

https://bitbucket.org/skeoop/oop/raw/master/docs/Big-Alice-in-Wonderland.txt

Copy this file to your computer and put it someplace where you can easily refer it.

You can put the file in your Eclipse project inside the src/ directory (make sure its in src/ and not the project top-level directory). See below for how to refer to a file on the project classpath.

1. Write a FileUtil class with 3 copy methods

`static void copy(InputStream in, OutputStream out )`	Copy the InputStream to the OutputStream one byte at a time. Close the InputStream and OutputStream when finsished.
`static void copy(InputStream in, OutputStream out, int blocksize )`	Copy the InputStream to the OutputStream using a byte array of size blocksize. Close the InputStream and OutputStream when finsished.
`static void bcopy(InputStream in, OutputStream out)`	Copy the InputStream to the OutputStream using a BufferedReader to read the InputStream and PrintWriter to write the OutputStream. Read and write one line at a time. If you are curious, also try reading into a character array, which should be faster since it does not check for end-of-line. Close the input and output when done.

To make it easy to use these methods, *catch* all exceptions and wrap them in a RuntimeException. Then throw the RuntimeException.

For the second method, the InputStream class has a method to read data to an array of bytes:

```
int blocksize = 1024;  // for example
byte[] buffer = new byte[blocksize];
// reads data into buffer, returns number of bytes read.
// When there is no more data, it will return count = -1 (or some value < 0)
int count = in.read( buffer );
```

2. Test and compare speeds of the FileUtil methods. Use TaskTimer from the stopwatch lab.

2.1 Write Runnable task files to copy a long test file using each of the methods in FileUtil (one task object per method to test). Use the TaskTimer and Stopwatch from the stopwatch lab to run these tasks and print the elapsed time. The file to copy is **`Big-Alice-in-Wonderland.txt`**.

Create a Runnable task for each of the copy tests listed below. Execute the tasks and record the elapsed times in README.md, using a table (as in Stopwatch lab).

)1 copy the file one byte at a time

)2 copy the file using a byte array of size 1KB (1024)

)3 copy the file using a byte array of size 4KB (4*1024)

)4 copy the file using a byte array of size 64KB (64*1024)

)5 copy the file using BufferedReader and PrintWriter to copy lines of text (the bcopy method)

)6 [optional] read and write using BufferedReader and BufferedWriter with an array of char (should be faster than copying lines of text)

The syntax for writing tables in Markdown syntax is explained here (its easy):
`https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet#tables`

2.2 After each test, compare the input and output files. They should be identical.

2.3 Explain why some copy methods are faster or slower than others.

2.4 To compute elapsed time, please reuse the `Stopwatch` and `TaskTimer` that you wrote in Lab 2. **Don't write a new timer code!**

There are two easy ways to add your stopwatch in this project.

Method I: Add Stopwatch Project to "Build Path" of this Project

(1) Open both projects (stopwatch and this project, fileio). Select the project for this lab (*not* stopwatch).

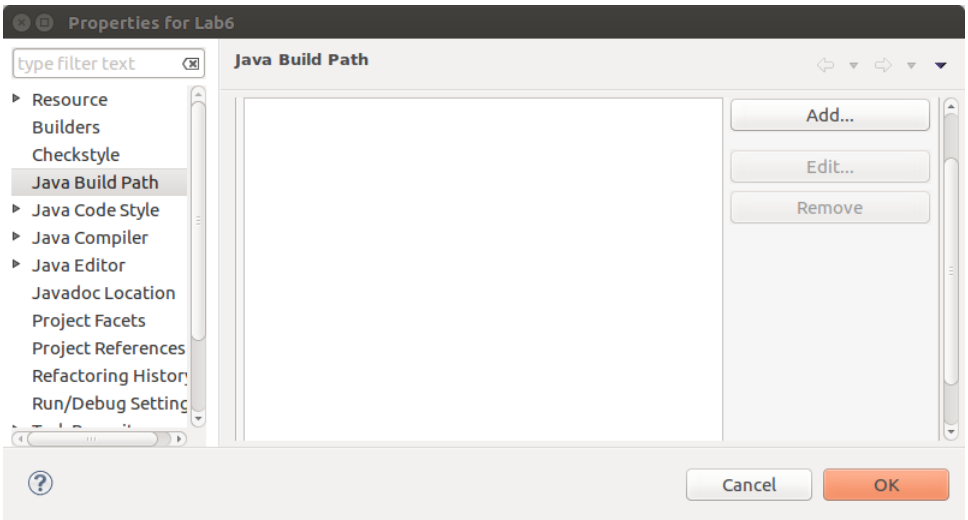
(2) In Eclipse, open the project settings: Project -> Properties.

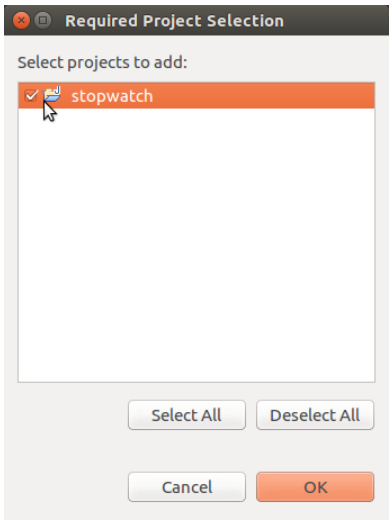
(3) Select "Java Build Path" (there is a shortcut to get here: right-click on the project name in Navigator)

(4) Click the "Projects" tab of the Java Build Path dialog. Click the "Add..." button.

(5) Select (check) the stopwatch project and click OK.

Now you can refer to classes from stopwatch in the code for this project!





Method II: Create a JAR file from stopwatch and add the JAR file to this project.

(1) To create a JAR file, open the stopwatch project and select "Export..." from the File menu. Select "JAR file" to export. In the dialog, click "Browse..." and go to the folder (directory) of your fileio project. Name the JAR file "stopwatch.jar".

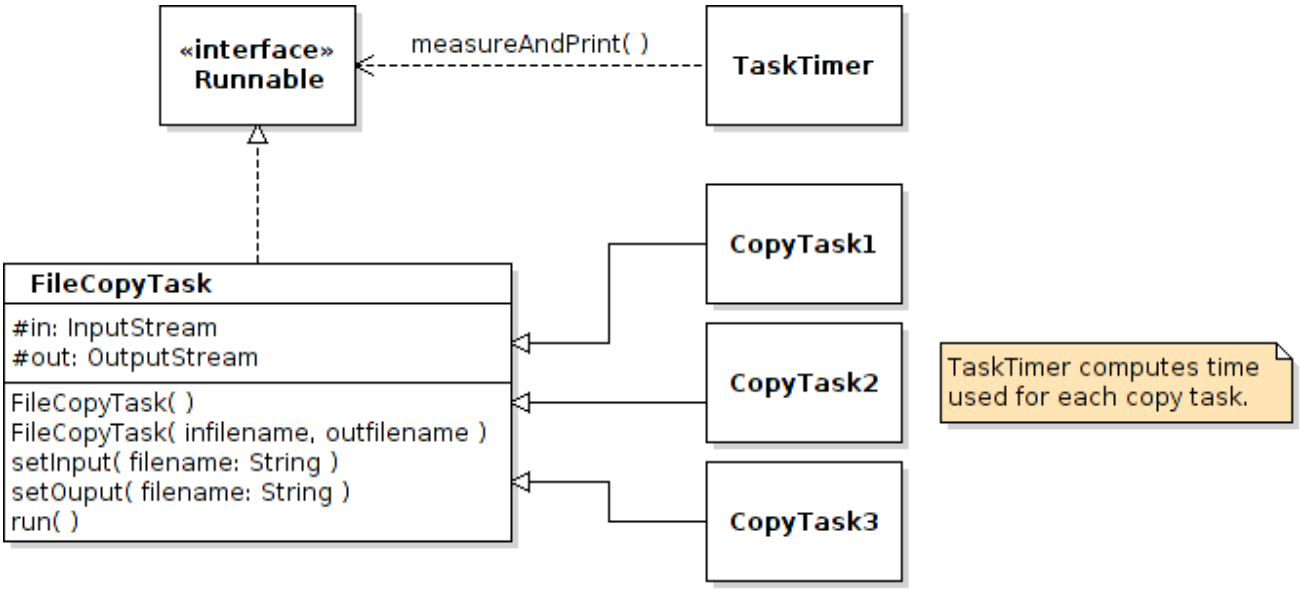
(2) Now add the JAR file to the Build Path of the fileio project. Choose Project -> Properties and select "Java Build Path".

(3) Click the "Add Jars..." button and add stopwatch.jar. If stopwatch.jar is not in the same directory as this project then select "Add External Jars" button. Add stopwatch.jar.

(4) Now you can use the Stopwatch classes in this project.

How To Write the File Copy Tasks

We want to create one Runnable "task" for each file copy test. All the tests require a file for input (the InputStream) and another file for output (the OutputStream). So, define a base class that has all this code. Then you can write a very simple subclass for each test you want to run. Use TaskTimer to run the tasks.



You can write the task classes as subclasses of FileCopyTask:

```
class ByteCopyTask extends FileCopyTask {
    public ByteCopyTask( String infilename, String outfilename ) ...
    /** Perform the copy */
    public void run() {
        FileUtil.copy(in, out);
    }
    /** Describe the task. */
    public toString() { return "Copy file one byte at a time."; }
}
```

The classes are so short that you may want to write them as anonymous classes.

How to Open a File that is on the project's Classpath

Many projects contain text files, images, and other files as part of the project. The files are usually in a subdirectory of `src`. You can open these files using a path *relative to your own project code* using the ClassLoader. This makes your project portable. The code can be run on another machine (in a different directory) or run as a JAR file. Here is an example of how to do this:

```
// in this example the file location is: src/data/example.txt
String filename = "data/example.txt";
// Get the ClassLoader than loaded this class.
ClassLoader loader = this.getClass().getClassLoader();
InputStream in = loader.getResourceAsStream(filename);
if (in == null) /* could not find the file */;
```