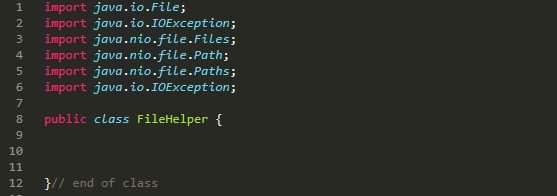
**File Sample App**

In this activity we’ll create a crude and simple app for working with files. You should have a folder called **file-app** in order to do this project. The folder has a sample text file that we will use (called test.txt). And we’ll be adding a few .java files to it. As you are writing the code for this assignment, pay close attention to the indentation of each line of code.

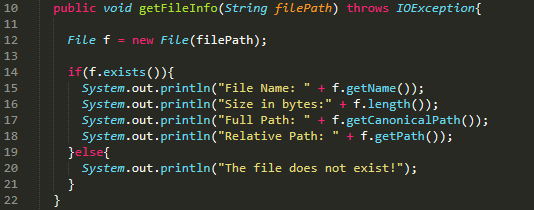
**Part 1**

1. Create a file named **FileHelper.java** in the file-app folder. We will write code in this file that defines a Java class named FileHelper.
2. Add the following code to declare a class named FileHelper:

Lines 1-6 is a list of classes that the FileHelper class will need to use. All of these classes are built into the Java language (we do not need to create them). When you need to use another class, you have to **import** it (this is similar to linking a web page to a .js file). Lines 1-6 are known as **import statements**. We will not actually be using all of the imported classes until we get to part 2 of this assignment.

Lines 8-12 declare that we are creating a class named FileHelper. All of the code for this class must go in between the pair of curly braces on lines 8 and 12. Note that all the code that gets put inside the body of the class (lines 8 – 12) should be indented.

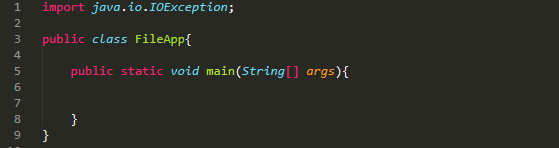
1. Now we’ll add a method to this class. Add this code in between the curly braces that enclose the class (start typing on line 10):

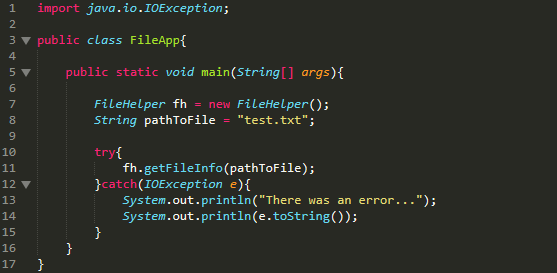


You’ll have to ignore some of the code for now (we’ll get to it eventually), but you should understand that we are adding a method to this class named getFileInfo(). It takes a parameter called filePath (which must be a string that represents a path to the file we wish to analyze). Inside the body of the method we make use of the first class that we imported. Java comes with a built-in class called File, which allows you to work with files and folders on a computer. On line 12 we are invoking the constructor function of the File class, and note that it includes a string parameter which specifies the path to the file on the computer. The **new** keyword must always be used before invoking a constructor function. This particular constructor function returns an object that represents a file on the computer. So we are storing a File object in a variable named **f**. The object has various methods that we can invoke, which is exactly what we are doing with the code in the rest of the screen shot. If you want to learn more about the File class in Java, here’s a link to the official documentation:

<https://docs.oracle.com/javase/7/docs/api/java/io/File.html>

It’s pretty overwhelming to look all the info on this page, but you’ll start to get used to it. The documentation will tell you all the properties and methods of a given class (and other things about the class too).

1. Now that we have defined a class, let’s put it to use by building a simple application that uses it. Create a file in the file-app folder named **FileApp.java**. Then put this code in it:

This class is what we call an **app driver** class, because we use it to launch our program. Note that it has a method named main(). When we run this program, the JVM (java.exe) will look for the main() method and invoke it automatically. So let’s add some code inside the main method now. Put this code in between the curly braces that enclose the body of the main method (on lines 5 and 8 in the screen shot):

On line 7 we create an instance of the FileHelper class by calling the constructor function. We are storing the FileHelper object in a variable named **fh**. On line 11 we are invoking the method that we added to the FileHelper class. Notice that we are passing in a string variable which specifies the path to the file that we want to analyze (in this case it’s the sample txt file that was provided with this assignment). You don’t have to worry about the rest of the code for now (just copy it correctly). But if you are curious about the try/catch stuff, note that lots of things can go wrong when a program runs. For example what would happen if we pass in the wrong parameter when we invoke the getFileInfo() method (what if we misspelled the name of the test file)? We use try/catch blocks to handle things that can go wrong in a program. We’ll more about them later in the course.

1. Now let’s compile and run the app! You can use GitBash or the Windows command line to do this. Navigate into the file-app folder and then run these commands:

**javac \*.java**

This will compile all .java files in the folder. Now run the app by entering this command. Now run the app by entering this command:

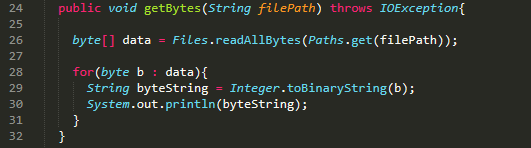
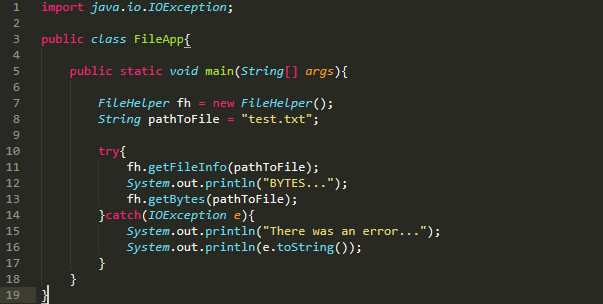
**java FileApp**

The output should look like this:

1. Now commit and push your changes.

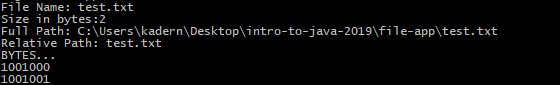
**Part 2**

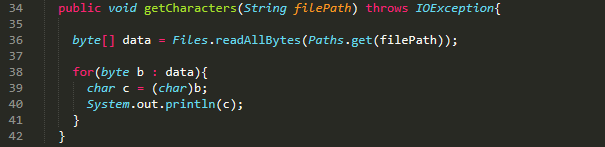
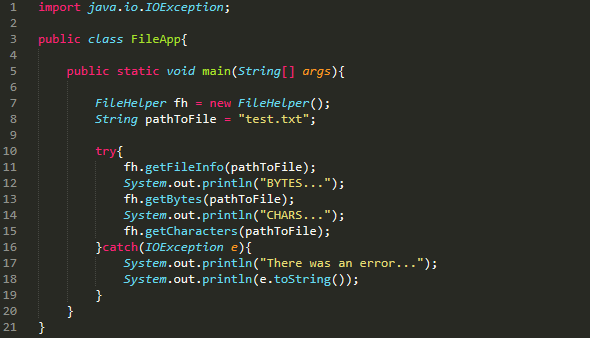
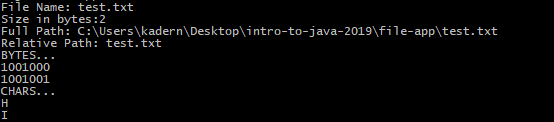
In this step, we’ll add a few more methods to the FileHelper class. The methods will allow us to read the contents of a file.

1. Add this method inside the body of FileHelper class (it will read the bytes in a file):
2. Now we’ll invoke the method in the app driver class. This is what the FileApp class should look like. Notice that the only lines you need to add are 12 and 13 in the screen shot below:
3. Compile and run the program by running these two commands:

**javac \*.java**

**java FileApp**

Here is what the output should look like:

1. Now we’ll add a method that reads the bytes in a file and then converts each one to a text character. Add this method to the FileHelper class:
2. Let’s now invoke this method in the app driver class. Update the FileApp class to look like this (lines 14 and 15 have been added in the screen shot below).
3. Finally, compile and run the program. The output should look like this:

COMMIT AND PUSH YOUR CHANGES:

**git add .**

**git commit –m “Added file app sample code”**

**git push origin master**