CT326 Lab 2:

(Note that this is a lab exercise and in not graded)

1. You are provided with a CSV file called "lab2.csv". The file contains real government data about land sales in Ireland in 2019. Each line contains a datapoint which is formatted as

   ```
   [land type],[region],[value of land sold in Euro]
   ```

   An example line from the file is

   ```
   Permanent Grassland,Mid-East,26616120.7
   ```

   Your task is to read the contents of this file using a character stream, and write the contents into multiple other files (one for each region) using a character stream so that a region's file contains data that is only relevant to that region. For example, the Mid-East only has two relevant lines and so the file called `Mideast.txt` should have the following contents:

   Arable Land 5103038.000000
   Permanent Grassland 26616120.700000

   Note the output file is a .txt file and the region is not included in the file's contents. It is also no longer comma-separated.

   Some directions are as follows:
   - Create an enum called `Region` with constants for each region contained in the CSV file. `Region` will need one `String` instance variable to represent the printed name of the region, i.e. the name that appears in the CSV file as these contain hyphens and lower-case letters, whereas the constant names should be uppercase by convention and cannot contain hyphens. Use `Region` in a switch statement to determine which file should be written to.
   - There's two delimiters that are important: the "," separating the values, but also the newline characters separating the lines. Use a `BufferedReader` to read the lines, and use a `Scanner` to read the tokens on each line separated by ",". The `Scanner` class has a `useDelimiter` method for setting the delimiter. The default delimiter is whitespace.
   - You can use a `PrintWriter` to write to the individual files. As the region data aren't sorted in the CSV file you'll need to open and close these for each line you wish to write to them. And you'll need to append to the files rather than overwriting them. Remember `FileWriter` has a `boolean` parameter for indicating whether to append or not.
   - You should write *formatted* strings to the files.
   - Don't forget to close your streams.
   - The reading functionality can be put in a `main` method in a `DataReader` class. The writing functionality should be put in a `DataWriter` class that has a single static method with the following signature:

     ```
     public static void writeLandDataToFile(String filename,
     String landType, double landValue)
     ```

Use a test-driven development approach to create the `DataWriter` class testing that the contents of the file that is written to are as you would expect.

- Exceptions should be caught but you don’t need to do anything specific to handle them – printing out the stack trace will suffice.
- Remember to place your code in appropriate packages.
- Remember to use Javadoc to fully document your code.