CSE 382
Fall 2022
Project #1 - US Locations

You will create a MS VS solution named `USLocations` that has the following 3 projects:

USLocations (library). For this project, you create software to support a software suite that will be used to answer questions about locations in the US (e.g., population, zip codes, etc). At the core of this work will be a class that stores information about locations in the US - `USLocations`. This class will provide basic support for locations in the US: What is the distance between two zip codes? What are the common names for two particular states? The contents of this library will be one class definition and no Main method.

Distance (console application). This console app will operate as follows: The user can type in two legal zip codes and determine the distance between the two locations. Or, the user can exit the program using “exit”. In the example, the user input is shown in bold, output is shown in italics. Use and do some research on the Haversine formula for computing the distance between two points on earth (given latitude and longitude).

```
distance> 45056 45003
The distance between 45056 and 45003 is 5.58 miles (8.98 km)

distance> 45003 45056
The distance between 45003 and 45056 is 5.58 miles (8.98 km)

distance> exit
Goodbye!
```

Common (console application). This console app will operate as follows: The user can type in two legal states and the output will consist of all city names that are located in both states in alphabetical order. Or, the user can exit the program using “exit”. In the example, the user input is shown in bold, output is shown in italics.

```
commons> MI OH
ADA
...
OXFORD
...
YALE

commons> exit
Goodbye!
```

Class definition details. Inside the USLocations shared library, you will define a class named `USLocations`. At a minimum, your class `USLocations` must define the following:

```
public class USLocations {
      private Task inputTask;             // Task for reading file
      private List<Location> locations;   // Read locations info here
      // Location is meant to be a custom class that stores, zipcode, state, city, etc.
      // This constructor will initiate the loading of the TSV file.
      // The constructor must return very quickly, perhaps before all
      // of the zip code information has been completely loaded. Tasks
      // will be needed to do this.
      public USLocations(string fileName) {
          inputTask = call async method for reading file
      }

      private async Task ReadFile …     // Asynchronous method for reading file

      /**
       * Returns the city names that appear in both of the given states.
       * "OH" and "MI" would yield {OXFORD, FRANKLIN, ... }
       */
      public ISet<string> GetCommonCityNames(string state1, string state2) {
          // Cycle through locations to find cities common to both states.
          // Before doing this, you may need to wait for the reading to complete.
      }
      /**
       * Returns all zipcodes that are within a specified distance from a
       * particular zipcode.
       */
      // Do this by researching the "Haversine" formula to do this one.
      // The formula for converting from degrees to radians is:
      //     radians = degrees * Math.PI / 180.0;
      public double Distance(int zip1, int zip2) {
          // Use Haversine to compute distance between two locations.
          // Before doing this, you may need to wait for the reading to complete.
      }
}
```

Notes.

- A data file containing information of all US zipcodes (`zipcodes.tsv`) has been provided to you and is located in the Google Docs folder. Be aware that the testing of your code may involve a different, and/or bigger, file than the one provided to you. You may find it helpful to put artificial zip codes into data file to increase the size of the file. The contents of the data file are described here.
- Before dealing with asynchronous processing, get the program to work using standard, synchronous, IO. Before answering questions on asynchronous IO, I will request that you have a working synchronous version.
- Both console applications should be very responsive to user input. The dictionary should be loaded first and the prompt be given to the user before the file has completed loading. This will allow the user to start typing immediately.
- In order to get the data file to be in the proper position during program execution.
  - Copy the data file into each of the console projects.
  - In the solution explorer, right click on the data file. In the properties window (advanced section):
    - Copy to Output Directory = "Copy if newer"

Scoring.

- **(85)** `USLocations` class implementation
  - **(25)** `Tasks` correctly used to load the dictionary in the background, for improved responsiveness.
  - **(60)** Aspects unrelated to `Tasks` are implemented correctly.
- **(15)** The stand-alone console apps
  - **(15)** Projects correctly set up as two console projects and one shared project
- Deductions may be taken for poor programming style.

What to submit.

- Source code. Your submission will be done using Git. Make sure that you have checked in your code (deductions will be taken if Debug, bin, obj, exe files are put into your repo), committed, and pushed your code before the due date/time.
- Report. You must write a short report that explicitly lists the aspects of your code that work and those that do not. This will generally be a bullet list and could look as simple as: "I have successfully completed all requirements to this project." This report must be submitted as a comment with your video (see next item) to Canvas. You do need to address how you implemented your program nor do you need to address difficulties that you encountered.
- Video. In addition, you must submit a video of you running your two console apps. Here is what your video, maximum of 4 minutes, should contain:
  - Announce your name
  - Announce what aspects of your program are correct and what is incorrect.
  - Use the file manager to show the directories of your solution and three projects.
  - Scroll through the source code that is directly relevant reading the input asynchronously. Spend a few seconds describing the changes you made or the code you wrote. Talk about the code in sections; do not analyze each line of code.
  - Use a font size large enough for the viewer to read.
  - Start your console apps and exercise your program's functionality, while announcing what you are doing. It is your responsibility to illustrate the required features outlined in the project's original write-up.
  - For the lookup project, use the following zip code pairs:
    - `distance> 45056 45003`
    - `distance> 45056 10001`
    - `distance> 96795 33040`
  - For the word formation project, use the following:
    - `commons> MI OH`
    - `commons> OH DE`

- `commons> AK FL`
- `commons> exit`