Program:	**CPA2-5**
Course:	**INFO-5101, *Section 1***
Professor:	**Tony Haworth**
Project:	**Canadian Cities**
Due Date:	**Monday, Feb 13th, 2023 by 11:59 pm**
Last Update:	**Jan 25th @ 7 p.m.**
This project must be done in groups of three.	
Sign-up for a project group in FOL. All students in a group must be in section 1.	

How Will my project will be marked?

This project counts for 15% of your final mark and will be graded using the following grid:

Task	Mark
Right coding style including proper indentation and use of variable and class naming conventions and suitable comments	2
Parse JSON, XML, and SCV input to C# Dictionary generic class (The *DataModeler* class)	3
Create the *CityInfo* class for holding cities information	2
Create the *Statistics* class for manipulating Dictionary data	4
Create a class called *CityPopulationChangeEvent* for notifying the user of a completed change made to a city's population	1
Creative but functional UI displaying the results from the *Statistics* class	2
The app runs with proper error management	1
Proper submission	1
Submit video describing the project with project run simulation	4
Add the *CalculateDistanceToCapital* method to the *Statistics* class. Modify the UI to use this method and include this feature in your video.	*1 bonus*
Total	20 marks

Project Description

One of the current trends in modern software applications is to query different data sources with various data formats such as JSON, XML, and CSV. These formats are widely used in several business domains, including distributed systems and data analytics, to foster data retrieval and analysis.

In this project, we want to work on several data formats directly in C#. As such, we'll need to find ways to convert between JSON, XML, and CSV and C# generics classes to work with the data effectively.

The data we're going to use in this project was extracted from the World Cities Database (https://simplemaps.com/data/world-cities). There are several specific goals that we're trying to achieve in working with the Cities data:

1. Parse either the JSON, XML, or CSV file to a Dictionary generic type based on the user's choice when the app starts. You can write customized code for the parsing process or use the Net Core libraries, such as JSON.NET.
2. Build a Dictionary generic type in which its key equals the City name, and the value is an object of the city information. You need to handle the duplicate of the city name! (e.g. Windsor, Ontario and Windsor, Nova Scotia)
3. Create a statistics class that will enable the user to retrieve all information about the stored cities in the Dictionary generic type.
4. Create a creative and user-friendly console or web interface to allow the end-user to access these statistics by prompting him/her to select the city or province and display all required information. The interface must have all the following features:
    a. It should have clear and concise statements that guide the end-user to choose among the developed statistics operations and describe how to type the requested inputs.
    b. The interface should never exit or end the application unless the end-user is selected to do so. In the case of creating a console interface, the application should allow the end-user to switch between the various file types (e.g., XML or JSON) at any point of running the application.
    c. The interface should also display an error message when the end-user tries to input an invalid input (e.g., entering a wrong number or letter).
5. Statistics are meaningless unless we compare the cities and provinces. The application interface should enable the user to reach the cities or provinces based on particular criteria such as population and location (using latitude and longitude).

Functional Requirements:

1. Create a non-generic class called CityInfo that will hold information about the city. It will have the following:
    a. Properties: CityID, CityName, CityAscii, Population, Province, Latitude, Longitude. (Note: *CityAscii* is an ascii-only representation of the city name. For example *CityName* is "Montr**é**al" and *CityAscii* is "Montr**e**al".)

   b. Methods: *GetProvince, GetPopulation, GetLocation*. *GetProvince* returns the Province the city belongs to. *GetPopulation* returns the population of the city. *GetLocation* returns the *Latitude* and *Longitude* of the city.
   c. Add the necessary constructors according to your business and technical logic.
2. Create a non-generic class called DataModeler that will parse the data files. It will contain the followings:
   a. One parsing method for each file. They will be called: *ParseXML, ParseJSON, and ParseCSV*. All of these three methods must have the same signature. Each parsing method accepts only one input parameter (the file name) and returns void.
   b. A customized delegate matches the signature of the three previous parsing methods.
   c. Another method called ParseFile will invoke one of the parsing methods in 2.a using the declared delegate in 2.b, according to the type of the parsed file. The method will accept the file name and type as input parameters. This method will also return the value of the generic type dictionary. The key of that dictionary parameter is the city name, where its value will be the city's information.
3. Create a class called Statistics that would include the followings:
   a. Property: a variable of the Dictionary generic type called CityCatalogue holds cities' information returned from the DataModeler class. The variable key is the city name itself, and the value is an object of the CityInfo class.
   b. Constructor (file name, file type). The user must specify the file name, “Canadacities”, and then determine the file type or extension to be JSON, XML, or CSV. You may get the value of the CityCatalogue here in this constructor by calling the DataModeler.Parse method.
   c. City’s methods:
      i. *DisplayCityInformation*: This method will take a CityName parameter and return all the city stored information in the CityCatalogue dictionary variable. The application should show all cities sharing the same name or be creative and ask the user which city should be displayed.
      ii. *DisplayLargestPopulationCity*: It will return the largest population city in a province.
      iii. *DisplaySmallestPopulationCity*: It will return the smallest population city in a province.
      iv. *CompareCitiesPopluation*: This method will take two parameters; each represents one city. It will return the city with a larger population and the population number of each city.

v. *ShowCityOnMap*: Use the name of the city and province to mark a city on the map. You may come up with a solution similar to what is shown in this link (https://www.latlong.net/).

vi. *CalculateDistanceBetweenCities*: This method calculates the distance between any two cities using the latitude and longitude of the input cities stored in the CityCatalogue dictionary variable. In order to calculate the distance, you can use one of the **free** Google services such as Google GeoCoding API. You may try other alternative APIs (if Google API fails) such as https://msdn.microsoft.com/en-us/library/ff701705.aspx or code your own function to calculate this.

vii. *BONUS: CalculateDistanceToCapital*: This method calculates the distance between a city selected by the user and the provincial capital for the same province. For example, it should calculate the distance between London and Toronto if the user selects London, Ontario. You can use the same technique to calculate the distance as was used in the *CalculateDistanceBetweenCities method.*

d. Province's methods

i. *DisplayProvincePopulation*: This method will take a province name parameter and return the sum of all populations of its cities saved in the CityCatalogue dictionary variable.

ii. *DisplayProvinceCities*: This method will take the province name parameter and return a list of all cities of that province from the CityCatalogue dictionary variable.

iii. *RankProvincesByPopulation*: It will sort all provinces by population. The order has to be ascending. The display should show the province and population number.

iv. *RankProvincesByCities*: it will sort all provinces by the number of cities in each. The order has to be ascending. The display should show the province and number of included cities.

v. *GetCapital*: It shows the capital of a province with its latitude and longitude.

4. Create a class called "CityPopulationChangeEvent" that will send a notification containing a message to be displayed to the end-user confirming an update to a city's population in a data file that he/she inputted via the UI. Use the C# Event concept discussed in class. The end-user has to select which file (e.g., XML or JSON) will be updated, and the notification message should show the old and new population numbers.

Other Requirements and Notes:

The following is a list of requirements and constraints for your application:

1. Visual Studio and .NET have tools that can be used with this project.
2. Include detailed comments in your code describing the critical aspects of your program.
3. Each class must be defined in a single separate cs file.
4. You can include additional types and methods you require for your solution.
5. All input parameters to the Statistics methods have to be selected by the end-user of your application.
6. You are **not** allowed to edit the three source data files.
7. Create a "Data" folder in your project to store the three source data files.
8. Include a short video demonstrating your project and all its functionality.

How should I submit my project?

Electronic Submission:

Please submit the entire solution as a single zipped file to the submission folder Project 1 on FOL before the due date **Monday, Feb 13, 11:59 PM**. Also include your video (preferably as a link). Only one-day last submission is allowed and will be penalized with 20% of the full project mark. Wrong submissions will not be marked! Please name the zipped file: **Project1_Group_X** (where X is the group number). Any copied work will not be marked, and Fanshawe regulations in this situation will be applied.