[1360] JSON.net

JSON.net is a open source library for parsing JSON into objects (called "deserialization") and for converting objects into JSON (called "serialization").

Documentation, examples and source code for JSON.net can be found on **http://json.net (http://json.net)**

Installing JSON.net

To use JSON.net, you first need to use NuGet to add it to your project. As you'll see, it is a very popular package with over 123 million downloads!

For Mac Users:
Right-click on the "Dependencies" folder inside your C# Project folder and select "Add Packages..." Next, type "Json.net" into the "Search" box and wait until the "Newtonsoft.Json" package appears. Select the Newtonsoft.Json package and click "Add Package" to install it.

For Windows Users:
Right-click on the "Dependencies" folder instide your C# Project folder and select "Manage NuGet Packages..." Next, click on "Browse" at the top of the NuGet window, and type "Newtonsoft.Json" into the Search box. When the package appears, click on it, then click on the "Install" button located on the right side of the window.

For Everyone:
Add "using Newtonsoft.Json;" and "using Newtonsoft.Json.Linq;" to your C# file. You'll also need "using System.Linq;" if you don't already have it.

Using JSON.net to Deserialize JSON

There are many, many, many (many!) ways to use JSON.net. It is extremely flexible. For this week's assignment, we'll just be using it's Parse() method to convert a JSON string into a JObject which, in turn, we can break apart into its various components. You can use "[fieldName]" to extract named fields from a JObject. You can also use Children() to dig down into lower level arrays and objects.

Note: It is CRUCIAL that you understand the format of the JSON data you are trying to parse with JSON.net. If you don't, your program will not work at all. To learn more about a JSON data feed, paste its URL into http://jsonlint.com

Here's our EarthquakeHistogram program using JSON.net to parse the data:

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Net;
```

```
using System.Linq;
using Newtonsoft.Json;
using Newtonsoft.Json.Linq;

namespace LWTech.QuakeHistogramWithJSON
{
  class QuakeProperties
  {
    public double? mag { get; set; }
  }

  class Program
  {
    static void Main(string[] args)
    {
      Console.WriteLine("Earthquakes by Magnitude (w/JSON.net):");
      Console.WriteLine("=====================================");

      string jsonText = "";
      WebClient client = new WebClient();

      try
      {
        Stream stream = client.OpenRead("https://earthquake.usgs.gov/earthquakes/feed/v1.0/summary/all_day.geojson");
        using (StreamReader reader = new StreamReader(stream))
        {
          jsonText = reader.ReadToEnd();
        }
      }
      catch (IOException ex)
      {
        Console.WriteLine("A network error occurred.  " + ex.Message);
        Console.WriteLine("Unable to continue.");
        return;
      }

      JObject quakesJO = JObject.Parse(jsonText);
      List<JToken> quakesJA = quakesJO["features"].Children().ToList();

      List<string> quakeMagnitudes = new List<string>();
      foreach (JToken quake in quakesJA)
      {
        QuakeProperties quakeProperties = quake["properties"].ToObject<QuakeProperties>();
        if (quakeProperties.mag != null)
          quakeMagnitudes.Add("" + (int)quakeProperties.mag);
      }

      Histogram quakeLocationHistogram = new Histogram(quakeMagnitudes, width: 100, maxLabelWidth: 5);
      quakeLocationHistogram.Sort((x, y) => x.Key.CompareTo(y.Key));
      Console.WriteLine(quakeLocationHistogram);
    }
  }

  class Histogram
  {
    private int width;
```

```
private int maxBarWidth;
private int maxLabelWidth;
private int minValue;
private List<KeyValuePair<string, int>> bars;

public Histogram(List<string> data, int width = 80, int maxLabelWidth = 10, int minValue = 0)
{
   this.width = width;
   this.maxLabelWidth = maxLabelWidth;
   this.minValue = minValue;
   this.maxBarWidth = width - maxLabelWidth - 2;   // -2 for the space and pipe separator

   var barCounts = new Dictionary<string, int>();

   foreach (string item in data)
   {
      if (barCounts.ContainsKey(item))
         barCounts[item]++;
      else
         barCounts.Add(item, 1);
   }

   this.bars = new List<KeyValuePair<string, int>>(barCounts);

}

public void Sort(Comparison<KeyValuePair<string, int>> f)
{
   bars.Sort(f);
}

public override string ToString()
{
   string s = "";
   string blankLabel = "".PadRight(maxLabelWidth);

   int maxValue = 0;
   foreach (KeyValuePair<string, int> bar in bars)
   {
      if (bar.Value > maxValue)
         maxValue = bar.Value;
   }

   foreach (KeyValuePair<string, int> bar in bars)
   {
      string key = bar.Key;
      int value = bar.Value;

      if (value >= minValue)
      {
         string label;
         if (key.Length < maxLabelWidth)
            label = key.PadLeft(maxLabelWidth);
         else
            label = key.Substring(0, maxLabelWidth);
```

```
                int barSize = (int)(((double)value / maxValue) * maxBarWidth);
                string barStars = "".PadRight(barSize, '*');

                s += label + " |" + barStars + " " + value + "\n";
            }
        }

        string axis = blankLabel + " +".PadRight(maxBarWidth + 2, '-') + "\n";    //TODO: Why is +2 is needed?
        s += axis;

        return s;
    }
  }
}
```

Note: To handle the possibility of "mag" being set to null, the mag field in QuakeProperties is a nullable "Double?" datatype.