PROGRAMMING ASSIGNMENT #8 w/ JSON.net

Program.cs

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Net;
using Newtonsoft.Json;
using Newtonsoft.Json.Linq;

namespace HistogramExample
{
    class Airplane
    {
        public string Icao { get; set; }
        public string Reg { get; set; }
        public string Engines { get; set; }
        public bool Mil { get; set; }
        public string Op { get; set; }
        public string Type { get; set; }
        public string Mdl { get; set; }
        public int Alt { get; set; }
        public string Cou { get; set; }
        public string Year { get; set; }

        public override string ToString()
        {
            string s = "";

            s += $"{Cou,-20} | {Reg,-10} - {Type,4} ({Year,4}): {Mdl,40} | {Op}";

            return s;
        }
    }


    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Currently Flying Boeing/Airbus Airplanes (w/JSON):");
            Console.WriteLine("================================================");

            string jsonText = "";

            WebClient client = new WebClient();
            try
```

```
            {
                Stream stream = client.OpenRead("https://stockcharts.com/dev/chipa/airplanes.json");
                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 airplaneJO = JObject.Parse(jsonText);
            var airplanesJA = airplaneJO["acList"].Children().ToList();

            var airplanes = new List<Airplane>();
            foreach (var item in airplanesJA)
            {
                airplanes.Add(item.ToObject<Airplane>());
            }

            var airplaneTypes = new List<string>();
            foreach (Airplane airplane in airplanes)
            {
                if (airplane.Type == null) continue;
                if (airplane.Type.StartsWith("B7", StringComparison.Ordinal))
                {
                    airplaneTypes.Add(airplane.Type.Substring(0, 3) + "7");
                }
                else if (airplane.Type.StartsWith("A3", StringComparison.Ordinal))
                {
                    airplaneTypes.Add(airplane.Type.Substring(0, 3) + "0");
                }
            }

            Histogram airplaneTypeHistogram = new Histogram(airplaneTypes, width: 100, maxLabelWidth: 5, minV
alue: 0);
            airplaneTypeHistogram.Sort((x, y) => y.Value.CompareTo(x.Value));       // Reverse sort order
            Console.WriteLine(airplaneTypeHistogram);

            Console.ReadLine();
        }

    }

    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";
            s += axis;

            return s;
        }
    }

}
```