PROGRAMMING ASSIGNMENT #8 REVIEW

Program.cs

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

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

         var typeAirplanes = new List<string>();

         string json = "";
         try
         {
            WebClient client = new WebClient();
            Stream stream = client.OpenRead("https://stockcharts.com/dev/chipa/airplanes.json");
            using (StreamReader reader = new StreamReader(stream))
            {
               json = reader.ReadToEnd();
            }
         }
         catch (IOException ex)
         {
            Console.WriteLine("A network error occurred.  " + ex.Message);
            Console.WriteLine("Unable to continue.");
            return;
         }

         string[] records = json.Split("{\"Id\"");
```

```
        foreach (string record in records)
        {
            int start = record.IndexOf("\"Type\":", StringComparison.Ordinal) + 8;
            string type = record.Substring(start, 3);

            if (type.StartsWith("B7", StringComparison.Ordinal))
                typeAirplanes.Add(type + "7");
            else if (type.StartsWith("A3", StringComparison.Ordinal))
                typeAirplanes.Add(type + "0");
        }

        Histogram airplaneTypeHistogram = new Histogram(typeAirplanes, width: 100, maxLabelWidth: 5, minVal
ue: 0);
        airplaneTypeHistogram.Sort((x, y) => y.Value.CompareTo(x.Value));       // Reverse sort order
        Console.WriteLine(airplaneTypeHistogram);
    }
}

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;
        }
    }
}
```