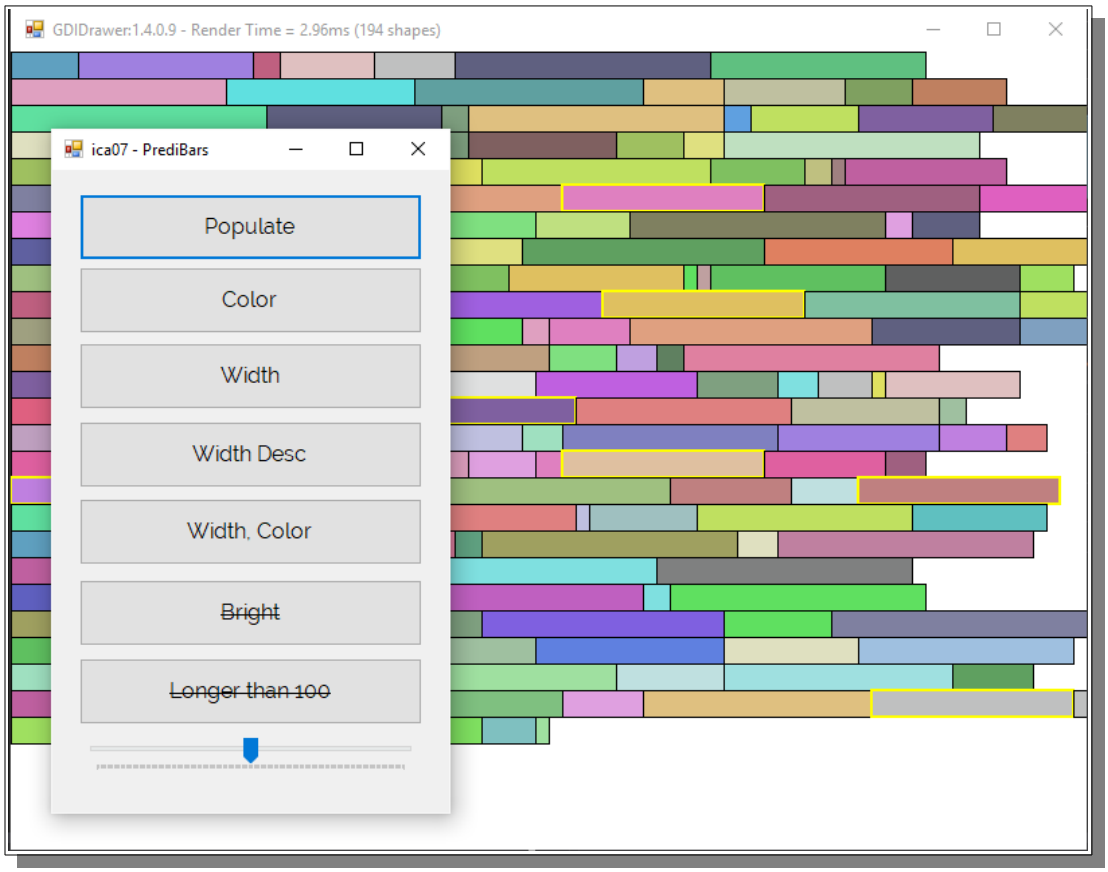
CMPE2300 - ICA #07 - Predi-Bars

In this simple ICA, you will exercise comparision, predicates and lambdas on a `List` collection.



Create your **Bar** class

- public automatic **static** properties : CDrawer with hidden set, int Height with hidden set
- private static field of Random, seed with 1
- public automatic instance property int Width, hidden set
- public automatic instance property bool HighLight
- private instance field of Color

In your **static** constructor, initialize Height to 20, allocate your CDrawer and set the background to white, ContinuousUpdate off.

In your instance default constructor, initialize the width to a random value between 10 and 190 (in increments of tens, ie. 10, 20, ... 180, 190 - hm..), set HighLight to false, and your Bar color to

```
Color.FromArgb(_rnd.Next(3, 8) * 32 - 1, _rnd.Next(3, 8) * 32, _rnd.Next(3, 8) * 32); // ?? WTH
```

Override Equals, return true if the width and color are the same for both Bars

Add a ShowBar() public method, accepts a Point and returns nothing. Add the Bar at the point supplied with a border of 1 in black. If the HighLight flag is true use a border width of 2 in Yellow

Main Form

In your main form class, add a list of Bar, a trackbar (Min=10, Max=190, Scroll updates the Longer button).

Add a rendering method called **ShowBars**(), clear the drawer, and iterate through the list invoking ShowBar for each Bar found. The Point will be determined by ensuring the fill starts in the top left, and fills across until there isn't enough room for the next Bar to fit, move to the next "row". Remember to Render() the drawer when done.

To the '**Populate**' button event handler, clear the list. You need to keep a running sum of the widths of the Bars added. While your running sum is less than 80% of the (scaled Area / Bar Height), create a Bar and if it doesn't currently exist in the list add it and update your running sum. Now ShowBars().

For the '**Color**' button, sort your list. Provide no arguments to `Sort`. Modify your Bar class to make this operation work correctly. Now ShowBars().

For the '**Width**' button, sort your list. This time, provide an explicit `System.Comparison<Bar>` compliant function to order the Bar objects in ascending order of Width. Place the static implementation of this function in your Bar class. Now ShowBars()

For the '**Width Desc**' button, sort your list. This time, provide a `System.Comparison<Bar>` compliant lambda function to order the Bar objects in descending order of Width. Remember the extra brackets for your lambda expression arguments. Now ShowBars()

For the '**Width, Color** ' button, call `Sort` in your list. This time, provide an explicit `System.Comparison<Bar>` compliant function to order the Bar objects in ascending order of Width then Color (ie. It will look like Ascending Width, except Widths that are the same will be in Ascending Color). Place the static implementation of this function in your Bar class. Now ShowBars()

For the '~~**Bright**~~' button, call `RemoveAll` in your list. This time, provide a `System.Predicate<Bar>` compliant function to indicate Bar objects that have a color Brightness property greater than 0.6. Update the form title text to indicate the number of Bars removed, Now ShowBars()

For the '~~**Longer**~~' button, call `RemoveAll` in your list. This time, provide a `System.Predicate<Bar>` compliant lambda function to indicate Bar objects that have a width greater than the current trackbar value. Update the form title text to indicate the number of Bars removed, Now ShowBars()

Now add a **MouseMove** handler to your form. This is designed to highlight Bars that are close to the X position indicated by the event argument. To pull this off, iterate through your list using a list.ForEach() and a `System.Action<Bar>` compliant lambda expression to mark all Bar objects as not highlighted. Call `FindAll` in your list, providing a `System.Predicate<Bar>` compliant lambda function to indicate Bar objects that have a Width that is **within** 10 units of the HighLightWidth, chain this to a ForEach() and a `System.Action<Bar>` lambda expression to set each Bar object as highlighted. ShowBars(). Yes! this method is 3 lines long → ForEach(), FindAll(), ShowBars().

Finally, the **trackbar** needs some tweaking. It should always have a Min and Max that matches the

current list values. Use extension method Min() and Max() and a `System.Func<Bar>` compliant lambda expression to get the current min and max values of the Bars (use width), set the min and max values of the trackbar appropriately, also setting the current value to the mid-point. These steps should occur any time the list contents change...

Partial Signoff is available for this ICA, but each must be fully compliant to attain the marks :

- Populate / ShowBars [20 Marks]
- Color [+10]
- Width [+10]
- Width Desc [+10]
- Width, Color [+10]
- Bright [+10]
- Longer [+10]
- MouseMove [+10]
- Trackbar [+10]