Versuchen Sie immer, Ihren Code zu kommentieren!

Aufgabe 1

Erstellen Sie ein Verzeichnis mit dem Namen „Texte“ auf dem Laufwerk D: oder C:, sofern noch nicht vorhanden. Kopieren Sie einige kleine Text-Dateien in das Verzeichnis. In einzelnen, unabhängigen Tasks soll die Buchstabenhäufigkeit für jede Datei ermittelt werden.

Erstellen Sie dazu eine Methode die für einen gegebenen Dateipfad folgendes leistet:

1. Der Inhalt der Datei mit dem angegebenen Pfad wird ausgelesen und die Häufigkeit der Buchstaben in der Datei bestimmt.
2. Die Häufigkeitstabelle für die Datei wird auf der Konsole ausgegeben.

Erstellen Sie im Hauptprogramm eine Schleife, die für jede der Dateien in dem Verzeichnis die Berechnung der Häufigkeit in einem eigenen Task durchführt.

Erweitern Sie die Anwendung so, dass die Häufigkeit nicht nur auf die Konsole geschrieben wird, sondern auch in eine Datei. Für jede Eingabedatei, soll eine Ausgabedatei mit der Endung „.freq“ (frequency) erstellt werden.

Aufgabe 2

Überwachen Sie mittels eines FileSystemWatcher einen Ordner und wandeln Sie jede Grafikdatei, welche in diesen Ordner kopiert wird, in Graustufen um. Speichern Sie die Schwarzweißbilder in einem anderen Ordner. Verwenden Sie dazu die unten angegebene Methode.

Starten Sie für jedes Bild einen Task zum Umwandeln in Graustufen.

Für die Klasse SKBitmap und die Struktur SKColor muss das NuGet-Paket SkiaSharp installiert werden. Der statischen Methode Decode von SKBitmap kann ein Pfad zu einer zu öffnenden Bild-Datei übergeben werden.

Hinweis: Kopieren Sie größere Bilder in den überwachten Ordner, wird der Event des FileSystemWatcher ausgelöst bevor der Kopiervorgang beendet wurde!

```
private static void ColorToGray(SKBitmap image, string grayFilename)
{
    for (int x = 0; x < image.Width; x++)
    {
        for (int y = 0; y < image.Height; y++)
        {
            SKColor color = image.GetPixel(x, y);
            byte gray = (byte)(0.2126f * color.Red + 0.7152f * color.Green +
0.0722f * color.Blue);
            SKColor sKColor = new SKColor(gray, gray, gray, color.Alpha);
            image.SetPixel(x, y, sKColor);
        }
```

```
    }

    try
    {
        using FileStream grayFileStream = File.Create(grayFilename);
        image.Encode(grayFileStream, SKEncodedImageFormat.Jpeg, 100);
    }
    catch (Exception e)
    {
        Console.WriteLine(e.Message);
    }

    image.Dispose();
}
```

Aufgabe 3

Erstellen Sie ein Programm, welches mittels Tasks Fibonacci-Zahlen berechnet. Verwenden Sie dazu eine rekursive Methode, welcher Sie die Stelle der zu berechnenden Fibonacci Zahl übergeben.

Beispiel:

```
public static long Fib(long x)
{
    return x <= 2 ? 1 : Fib(x - 1) + Fib(x - 2);
}
```

Berechnen Sie z.B. 15 Zahlen ab der 30. Zahl, damit die Berechnungen eine längere Zeitspanne dauern. Legen Sie pro Fibonacci-Zahl einen Task mit einem Lambda Ausdruck an und übergeben Sie die Stelle.

Warten Sie auf die Beendigung aller Tasks um sicherzustellen, dass zunächst alle Berechnungen durchgeführt werden und erst am Ende alle Ergebnisse gleichzeitig angezeigt werden.

Stellen Sie während der Berechnungsdauer z.B. Punkte auf dem Bildschirm dar, um dem Anwender zu signalisieren, dass Ihre Anwendung arbeitet. Wenn die Zahlen ausgegeben wurden, sollen keine Punkte mehr ausgegeben werden.

Nach der Ausgabe aller Zahlen, soll die Anwendung zur Beendigung auf einen Tastendruck warten. Dies soll sicherstellen, dass das Ausgeben der Punkte auch tatsächlich endet.

Die Ausgabe könnte z.B. wie folgt aussehen:

```
      832.040
    1.346.269
    2.178.309
    3.524.578
    5.702.887
    9.227.465
   14.930.352
   24.157.817
   39.088.169
   63.245.986
  102.334.155
  165.580.141
  267.914.296
  433.494.437
  701.408.733
1.134.903.170
```