Tentu, berikut adalah dokumentasi untuk kedua file C# dan Brightness.cs yang Anda berikan.

Dokumentasi: Brightness.cs

Ringkasan File

- **Namespace:** MiniPhotoshop.Logic.ImageProcessing
- **Kelas:** Brightness
- **Tujuan:** Kelas ini adalah *utility class* statis yang menyediakan satu fungsi: AdjustBrightness. Fungsi ini mengubah kecerahan (brightness) sebuah gambar dengan menambahkan atau mengurangi nilai adjustment ke setiap channel warna (R, G, B) dari setiap piksel.
- **Fitur Utama:** Kode ini menggunakan metode LockBits untuk pemrosesan gambar, yang **jauh lebih cepat** daripada GetPixel/SetPixel yang digunakan di ArithmeticOperations.cs.

Alasan Utama: Mengapa Menggunakan LockBits?

Anda akan melihat kode ini jauh lebih rumit daripada ArithmeticOperations.cs. Alasannya adalah **kecepatan**.

1. **Metode Lambat (GetPixel/SetPixel):** Metode ini mudah tetapi lambat. Untuk gambar 1920x1080 (2 juta piksel), Anda akan memanggil fungsi GetPixel dan SetPixel 2 juta kali, yang bisa memakan waktu beberapa detik.
2. **Metode Cepat (LockBits):** Metode ini "mengunci" memori gambar, menyalin *seluruh* data piksel ke dalam byte[] array C# dalam satu operasi (Marshal.Copy). Kita memanipulasi array ini secara langsung (sangat cepat), lalu menyalinnya kembali ke gambar dalam satu operasi. Ini mengurangi waktu proses dari "detik" menjadi "milidetik".

Analisis Kode per Blok

Blok 1: using Statements

C#

```
using System.Drawing.Drawing2D;
using System.Drawing.Imaging;
using System.Runtime.InteropServices;
```

- **using System.Drawing.Imaging;**
  - **Penjelasan:** Mengimpor *namespace* pencitraan GDI+ yang canggih.
  - **Fungsi:** Diperlukan untuk BitmapData, ImageLockMode, dan PixelFormat. Ini adalah inti dari fungsionalitas LockBits.
- **using System.Runtime.InteropServices;**
  - **Penjelasan:** Mengimpor layanan "Interop".
  - **Fungsi:** Diperlukan untuk kelas Marshal, yang digunakan untuk Marshal.Copy (menyalin data dari memori *unmanaged* GDI+ ke array byte[] *managed* C#).

Blok 2: Metode AdjustBrightness (Bagian 1: Setup & Lock)

C#

```
public static class Brightness
{
    public static Bitmap AdjustBrightness(Bitmap originalImage, int adjustment)
    {
        Bitmap resultImage = new Bitmap(originalImage);

        Rectangle rect = new Rectangle(0, 0, resultImage.Width, resultImage.Height);
        BitmapData bmpData = resultImage.LockBits(rect,
ImageLockMode.ReadWrite,resultImage.PixelFormat);

        IntPtr ptr = bmpData.Scan0;
```

- **public static Bitmap AdjustBrightness(...)**
  - **Penjelasan:** Metode publik yang mengambil gambar asli dan nilai adjustment. Nilai ini bisa positif (mencerahkan) atau negatif (menggelapkan).
- **Bitmap resultImage = new Bitmap(originalImage);**
  - **Penjelasan:** Membuat **salinan** dari gambar asli.
  - **Alasan:** Ini adalah praktik *non-destructive editing*. Kita tidak mengubah gambar aslinya, tapi mengembalikan salinan yang sudah dimodifikasi.
- **Rectangle rect = new Rectangle(...)**
  - **Penjelasan:** Membuat objek Rectangle yang mencakup *seluruh* area gambar.
  - **Alasan:** LockBits perlu tahu area memori mana yang ingin kita kunci.
- **BitmapData bmpData = resultImage.LockBits(...)**
  - **Penjelasan:** Ini adalah perintah inti untuk "mengunci" memori.
  - **Fungsi:** ImageLockMode.ReadWrite memberi tahu GDI+ bahwa kita berniat untuk **membaca** dan **menulis** ke memori ini.
- **IntPtr ptr = bmpData.Scan0;**
  - **Penjelasan:** Scan0 adalah *pointer* (penunjuk memori) ke **byte pertama** dari data piksel gambar yang dikunci.

Blok 3: AdjustBrightness (Bagian 2: Salin ke Array)

C#

```
        int bytes = Math.Abs(bmpData.Stride) * resultImage.Height;
        byte[] rgbValues = new byte[bytes];

        Marshal.Copy(ptr, rgbValues, 0, bytes);
```

- **int bytes = Math.Abs(bmpData.Stride) * resultImage.Height;**
  - **Penjelasan:** Menghitung jumlah total *byte* dalam data gambar.
  - **bmpData.Stride**: Ini adalah panjang satu baris gambar *dalam byte*. Ini **tidak** selalu lebar * 3 karena GDI+ terkadang menambahkan "bantalan" (padding) di akhir baris untuk efisiensi memori. Math.Abs digunakan karena *stride* bisa negatif jika gambar disimpan terbalik.
- **byte[] rgbValues = new byte[bytes];**
  - **Penjelasan:** Membuat array byte C# yang ukurannya sama persis dengan data gambar.

- **Marshal.Copy(ptr, rgbValues, 0, bytes);**
  - **Penjelasan:** Operasi penyalinan berkecepatan tinggi.
  - **Fungsi:** Menyalin bytes data, dimulai dari alamat memori ptr (gambar), ke dalam array rgbValues. Sekarang, semua data piksel ada di dalam rgbValues dan siap dimanipulasi.

Blok 4: AdjustBrightness (Bagian 3: Algoritma Inti)

C#

```
        int bytesPerPixel = Image.GetPixelFormatSize(resultImage.PixelFormat) / 8;
        int stride = bmpData.Stride;

        for (int y = 0; y < resultImage.Height; y++)
        {
            int rowOffset = y * stride;
            for (int x = 0; x < resultImage.Width; x++)
            {
                int i = rowOffset + (x * bytesPerPixel);

                for (int c = 0; c < 3; c++)
                {

                    int newValue = rgbValues[i + c] + adjustment;

                    if (newValue > 255) newValue = 255;
                    if (newValue < 0) newValue = 0;

                    rgbValues[i + c] = (byte)newValue;
                }
            }
        }
```

- **int bytesPerPixel = ... / int stride = ...**: Variabel pembantu untuk navigasi array.
- **for (int y = ...) / for (int x = ...)**: Perulangan standar untuk mengunjungi setiap piksel.
- **int rowOffset = y * stride;**: Menghitung indeks *byte* awal untuk baris y saat ini.
- **int i = rowOffset + (x * bytesPerPixel);**: Menghitung indeks *byte* awal untuk piksel (x, y)

saat ini. Dalam format BGR, i menunjuk ke channel **Biru**.

- **for (int c = 0; c < 3; c++)**
  - **Penjelasan:** Ini adalah inti dari algoritma. Loop kecil ini berjalan 3 kali untuk setiap piksel.
  - **Fungsi:**
    - c = 0: Memproses rgbValues[i + 0] (Biru)
    - c = 1: Memproses rgbValues[i + 1] (Hijau)
    - c = 2: Memproses rgbValues[i + 2] (Merah)
  - **Alasan:** Ini adalah cara yang efisien untuk menerapkan operasi yang sama persis ke ketiga channel warna (B, G, R) tanpa mengulangi kode. (Catatan: Ini mengabaikan *alpha/transparansi* jika ada, yang biasanya merupakan perilaku yang diinginkan untuk penyesuaian kecerahan).
- **int newValue = rgbValues[i + c] + adjustment;**
  - **Penjelasan: Ini adalah logika kecerahan.**
  - **Fungsi:** Mengambil nilai channel saat ini (misal, Biru) dan menambahkan nilai adjustment padanya. Jika adjustment adalah +50, nilai Biru bertambah 50. Jika adjustment adalah -30, nilai Biru berkurang 30.
- **if (newValue > 255) ... / if (newValue < 0) ...**
  - **Penjelasan:** Ini adalah implementasi "inline" dari fungsi Clamp.
  - **Alasan:** Memastikan nilai baru tidak pernah melampaui 255 (terlalu terang) atau kurang dari 0 (terlalu gelap), yang akan menyebabkan *error*.
- **rgbValues[i + c] = (byte)newValue;**
  - **Penjelasan:** Menyimpan nilai yang sudah disesuaikan dan di-*clamp* kembali ke dalam array.

Blok 5: AdjustBrightness (Bagian 4: Salin Kembali & Selesai)

C#

```
        Marshal.Copy(rgbValues, 0, ptr, bytes);
        resultImage.UnlockBits(bmpData);

        return resultImage;
    }
  }
}
```

- **Marshal.Copy(rgbValues, 0, ptr, bytes);**
  - **Penjelasan:** Kebalikan dari Marshal.Copy sebelumnya.
  - **Fungsi:** Menyalin bytes data dari array rgbValues (yang sudah dimodifikasi) kembali ke alamat memori ptr (gambar).
- **resultImage.UnlockBits(bmpData);**
  - **Penjelasan:** Perintah untuk "membuka kunci" memori gambar.
  - **Alasan:** Ini **WAJIB** dilakukan. Lupa memanggil ini akan menyebabkan kebocoran memori (*memory leak*) yang parah karena memori gambar tidak pernah dilepaskan.
- **return resultImage;**
  - **Penjelasan:** Mengembalikan Bitmap yang sekarang berisi data piksel yang sudah lebih cerah atau lebih gelap.