Tentu, berikut adalah dokumentasi lengkap dari kode ImageNegation.cs, yang menjelaskan fungsionalitas dan alasan desainnya blok per blok.

Dokumentasi: ImageNegation.cs

Ringkasan File

- **Namespace:** MiniPhotoshop.Logic.ImageProcessing
- **Kelas:** ImageNegation
- **Tujuan:** Kelas ini adalah *utility class* (kelas pembantu) statis yang menyediakan satu fungsi: Apply. Fungsi ini mengambil sebuah gambar dan mengembalikan versi "negatif" (klise) dari gambar tersebut.
- **Fitur Utama:** Kode ini, sama seperti Brightness.cs dan BlackWhite.cs, menggunakan metode LockBits untuk pemrosesan gambar. Ini adalah teknik yang sangat dioptimalkan untuk performa tinggi.

Alasan Utama: Mengapa Menggunakan LockBits?

Sebelum masuk ke detail baris per baris, penting untuk memahami *mengapa* kode ini ditulis dengan cara yang terlihat kompleks.

1. **Metode Lambat (GetPixel/SetPixel):** Metode seperti img.GetPixel(x, y) (yang digunakan di ArithmeticOperations.cs) sangat mudah digunakan, tetapi **sangat lambat**. Setiap panggilan adalah *function call* terpisah yang memiliki banyak *overhead*.
2. **Metode Cepat (LockBits):** Metode LockBits bekerja secara fundamental berbeda:
    - Ia "mengunci" seluruh data gambar di memori.
    - Ia menyalin *seluruh blok* data piksel dari memori *unmanaged* (GDI+) ke dalam byte[] array C# (*managed*) dalam satu operasi cepat (Marshal.Copy).
    - Kita memanipulasi byte[] array secara langsung, yang secepat kilat.
    - Kita menyalin byte[] array yang sudah diubah kembali ke memori gambar dalam satu operasi cepat.
    - Ia "membuka kunci" memori.

Kesimpulan: Penggunaan LockBits mengubah operasi yang bisa memakan waktu beberapa

detik (pada GetPixel) menjadi operasi yang hampir instan (hanya beberapa milidetik), yang sangat penting untuk aplikasi pengeditan gambar yang responsif.

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 kelas-kelas inti LockBits seperti BitmapData, ImageLockMode, dan PixelFormat.
  - **Alasan:** Tanpa ini, BitmapData dan ImageLockMode tidak akan dikenali.
- **using System.Runtime.InteropServices;**
  - **Penjelasan:** Mengimpor layanan "Interop".
  - **Fungsi:** Diperlukan untuk kelas Marshal.
  - **Alasan:** Kita membutuhkan Marshal.Copy untuk menyalin data antara memori *unmanaged* GDI+ dan array byte[] C# *managed*.

Blok 2: Definisi Kelas dan Metode

C#

```
namespace MiniPhotoshop.Logic.ImageProcessing
{
    public static class ImageNegation
    {
        public static Bitmap Apply(Bitmap currentImage)
        {
```

- **public static class ImageNegation**
  - **Penjelasan:** Mendeklarasikan kelas *utility* yang bersifat public dan static.
  - **Fungsi:** static berarti kelas ini tidak dapat diinstansiasi (tidak bisa di-new). Metodenya dipanggil langsung: ImageNegation.Apply(...).
  - **Alasan:** Ini adalah *helper class* standar yang hanya berisi fungsi murni.
- **public static Bitmap Apply(Bitmap currentImage)**
  - **Penjelasan:** Metode publik Apply yang menjadi satu-satunya *entry point* untuk kelas ini.
  - **Fungsi:** Menerima satu Bitmap sebagai input dan mengembalikan Bitmap baru sebagai hasil.

Blok 3: Setup dan Penguncian Memori (Locking)

C#

```
            Bitmap resultImage = new Bitmap(currentImage);
            Rectangle rect = new Rectangle(0, 0, resultImage.Width, resultImage.Height);

            BitmapData bmpData = resultImage.LockBits(rect,
ImageLockMode.ReadWrite,resultImage.PixelFormat);

            IntPtr ptr = bmpData.Scan0;
```

- **Bitmap resultImage = new Bitmap(currentImage);**
  - **Penjelasan:** Membuat **salinan** dari gambar asli (currentImage).
  - **Alasan:** Ini adalah praktik *non-destructive editing*. Kita tidak memodifikasi gambar aslinya, tetapi membuat salinan untuk dimodifikasi dan dikembalikan.
- **Rectangle rect = new Rectangle(0, 0, ...)**
  - **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 gambar.
    - **Fungsi:** Mode ImageLockMode.ReadWrite memberi tahu GDI+ bahwa kita berniat untuk **membaca** data piksel dan **menulis** data baru ke dalamnya.
- **IntPtr ptr = bmpData.Scan0;**
    - **Penjelasan:** Scan0 adalah *pointer* (penunjuk memori).
    - **Fungsi:** ptr sekarang menyimpan alamat memori dari **byte pertama** data piksel gambar yang dikunci.

Blok 4: Salin Data (Unmanaged ke Managed)

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*. Nilainya **tidak** selalu lebar * 3 karena GDI+ terkadang menambahkan "bantalan" (padding) di akhir setiap baris untuk efisiensi.
    - **Math.Abs(...)**: Stride bisa jadi negatif jika gambar disimpan terbalik (bottom-up), jadi Math.Abs digunakan untuk memastikan kita mendapatkan nilai absolut untuk perhitungan ukuran.
- **byte[] rgbValues = new byte[bytes];**
    - **Penjelasan:** Membuat array byte C# (managed) yang ukurannya sama persis dengan total data gambar.
- **Marshal.Copy(ptr, rgbValues, 0, bytes);**
    - **Penjelasan:** Ini adalah operasi penyalinan berkecepatan tinggi.
    - **Fungsi:** Menyalin bytes data, dimulai dari alamat memori ptr (gambar), ke dalam array rgbValues, dimulai dari indeks 0.
    - **Alasan:** Sekarang semua data piksel ada di dalam rgbValues, kita bisa memprosesnya dengan aman dan cepat.

Blok 5: Persiapan Looping dan 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);

                rgbValues[i] = (byte)(255 - rgbValues[i]);
                rgbValues[i + 1] = (byte)(255 - rgbValues[i + 1]);
                rgbValues[i + 2] = (byte)(255 - rgbValues[i + 2]);
            }
        }
```

- **`int bytesPerPixel = ... / 8;`**: Menghitung berapa byte yang digunakan per piksel (misalnya, 3 untuk 24bpp, 4 untuk 32bpp). / 8 karena GetPixelFormatSize mengembalikan *bits*.
- **`int stride = bmpData.Stride;`**: Mendapatkan nilai *stride* (panjang baris dalam byte) untuk navigasi.
- **`for (int y = ...) / for (int x = ...)`**: Perulangan for ganda 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 (umumnya), i menunjuk ke channel **Biru**.
- **`rgbValues[i] = (byte)(255 - rgbValues[i]);`** (dan i+1, i+2)
  - **Penjelasan:** Ini adalah **inti dari algoritma negasi (klise)**.
  - **Fungsi:** Mengambil nilai channel saat ini (misal, Biru di rgbValues[i]) dan menguranginya dari 255.
  - **Alasan:** Ini membalikkan nilai warna.
    - Hitam (0) menjadi 255 - 0 = 255 (Putih).
    - Putih (255) menjadi 255 - 255 = 0 (Hitam).

        - Warna tengah (misal, abu-abu 128) menjadi 255 - 128 = 127 (nyaris sama).
        - Biru terang (200) menjadi 255 - 200 = 55 (Biru gelap).
    - Ini diterapkan ke channel **Biru** (i), **Hijau** (i+1), dan **Merah** (i+2). Channel Alpha (transparansi, di i+3 jika ada) diabaikan, yang merupakan perilaku normal untuk negasi.

Blok 6: Salin Kembali dan Buka Kunci (Unlock)

C#

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

        return resultImage;
    }
  }
}
```

- **Marshal.Copy(rgbValues, 0, ptr, bytes);**
    - **Penjelasan:** Ini adalah kebalikan dari Marshal.Copy sebelumnya.
    - **Fungsi:** Menyalin bytes data, dimulai dari array rgbValues (yang sudah dimodifikasi), kembali ke alamat memori ptr (gambar).
    - **Alasan:** Ini adalah operasi yang "menerapkan" semua perubahan yang kita buat di rgbValues kembali ke Bitmap yang sebenarnya.
- **resultImage.UnlockBits(bmpData);**
    - **Penjelasan:** Perintah untuk "membuka kunci" memori gambar.
    - **Alasan:** Ini **WAJIB** dilakukan. Jika Anda lupa memanggil UnlockBits, memori gambar akan tetap terkunci selamanya, yang akan menyebabkan kebocoran memori (*memory leak*) besar dan aplikasi Anda akan *crash*.
- **return resultImage;**
    - **Penjelasan:** Mengembalikan Bitmap (klise) yang sudah dimodifikasi.