Tentu, berikut adalah dokumentasi lengkap dari kode ConstantOperations.cs, yang menjelaskan fungsionalitas dan alasan desainnya baris per baris (atau lebih tepatnya, blok per blok).

Dokumentasi: ConstantOperations.cs

Ringkasan File

- **Namespace:** MiniPhotoshop.Logic.ImageProcessing
- **Kelas:** ConstantOperations
- **Tujuan:** Kelas ini adalah *utility class* (kelas pembantu) statis yang menyediakan metode untuk memodifikasi kecerahan gambar dengan mengalikan (Multiply) atau membagi (Divide) nilai setiap piksel dengan sebuah nilai konstan (sebuah angka, double).
- **Catatan Performa:** Tidak seperti BlackWhite.cs atau Brightness.cs, kelas ini menggunakan metode GetPixel dan SetPixel. Metode ini jauh **lebih mudah dibaca dan ditulis** tetapi secara signifikan **lebih lambat** daripada LockBits pada gambar berukuran besar.

Analisis Kode per Blok

Blok 1: Definisi Kelas dan Namespace

C#

```
using System;
using System.Drawing;
```

```
namespace MiniPhotoshop.Logic.ImageProcessing
{
    public static class ConstantOperations
    {
```

- **using System; / using System.Drawing;**
  - **Penjelasan:** Mengimpor *namespace* dasar dari .NET.
  - **Fungsi:** System diperlukan untuk tipe data dasar (meskipun tidak banyak digunakan di sini). System.Drawing sangat penting karena berisi definisi untuk Bitmap dan Color.
  - **Alasan:** Tanpa using System.Drawing;, kompiler tidak akan mengenali apa itu Bitmap atau Color.
- **public static class ConstantOperations**
  - **Penjelasan:** Mendeklarasikan sebuah kelas *utility* yang bersifat public dan static.
  - **Fungsi:**
    - public: Membuat kelas ini dapat diakses dari bagian lain aplikasi Anda.
    - static: Menandakan bahwa kelas ini tidak dapat diinstansiasi (Anda tidak bisa membuat new ConstantOperations()). Semua metodenya harus dipanggil langsung dari nama kelas (misalnya, ConstantOperations.Multiply(...)).
  - **Alasan:** Desain ini ideal untuk *helper class* yang hanya berisi sekumpulan fungsi dan tidak perlu menyimpan data atau status (state) internal.

Blok 2: Metode Helper Clamp

C#

```
    // Helper Clamp
    private static int Clamp(int value)
    {
        if (value < 0) return 0;
        if (value > 255) return 255;
        return value;
    }
```

- **private static int Clamp(int value)**
  - **Penjelasan:** Ini adalah metode pembantu internal yang identik dengan yang ada di ArithmeticOperations.cs.
  - **Fungsi:** "Menjepit" atau membatasi sebuah nilai integer agar selalu berada dalam

rentang 0 hingga 255.

- **Alasan:** *Channel* warna (Red, Green, Blue) disimpan sebagai byte (8-bit), yang memiliki rentang 0-255. Operasi matematika seperti c.R * valueK (misal, 150 * 2.0 = 300) akan menghasilkan nilai di luar rentang ini. Clamp akan memaksa 300 menjadi 255 (atau nilai negatif menjadi 0), mencegah *error* saat membuat Color baru.

Blok 3: Metode Publik Multiply

C#

```
    // --- KALI KONSTANTA ---
    public static Bitmap Multiply(Bitmap img, double valueK)
    {
        Bitmap resultBmp = new Bitmap(img.Width, img.Height);
        for (int y = 0; y < img.Height; y++)
        {
            for (int x = 0; x < img.Width; x++)
            {
                Color c = img.GetPixel(x, y);
                int newR = Clamp((int)(c.R * valueK));
                int newG = Clamp((int)(c.G * valueK));
                int newB = Clamp((int)(c.B * valueK));
                resultBmp.SetPixel(x, y, Color.FromArgb(newR, newG, newB));
            }
        }
        return resultBmp;
    }
```

- **public static Bitmap Multiply(Bitmap img, double valueK)**
  - **Penjelasan:** Metode publik yang menerima satu gambar (img) dan satu angka (double valueK).
  - **Fungsi:** Mengalikan setiap channel warna (R, G, B) dari setiap piksel di gambar dengan valueK. Jika valueK > 1.0, gambar akan menjadi lebih terang (kontras bertambah). Jika valueK < 1.0, gambar akan menjadi lebih gelap.
- **Bitmap resultBmp = new Bitmap(img.Width, img.Height);**
  - **Penjelasan:** Membuat Bitmap baru yang kosong dengan ukuran yang sama persis dengan gambar asli.

    - **Alasan:** Ini adalah praktik *non-destructive editing*. Kita tidak mengubah gambar img yang asli, tetapi membuat salinan yang dimodifikasi.
- **for (int y = ...) / for (int x = ...)**
    - **Penjelasan:** Perulangan for ganda standar untuk mengunjungi setiap piksel dalam gambar, baris demi baris, dari kiri ke kanan.
- **Color c = img.GetPixel(x, y);**
    - **Penjelasan:** Mendapatkan objek Color (yang berisi nilai R, G, B) dari piksel di koordinat (x, y) pada gambar *asli*.
- **int newR = Clamp((int)(c.R * valueK));** (dan newG, newB)
    - **Penjelasan:** Ini adalah inti dari logika.
    - **Fungsi:**
        1. c.R * valueK: Mengambil nilai Merah (misal, 100) dan mengalikannya dengan valueK (misal, 1.5). Hasilnya adalah double (misal, 150.0).
        2. (int)(...): Mengubah (cast) hasil double tersebut kembali menjadi int (misal, 150).
        3. Clamp(...): Memastikan nilai int tersebut aman (tetap di antara 0-255).
- **resultBmp.SetPixel(x, y, Color.FromArgb(newR, newG, newB));**
    - **Penjelasan:** Membuat Color baru dari nilai newR, newG, dan newB yang sudah dihitung, lalu menetapkannya ke piksel di koordinat (x, y) pada gambar *hasil* (resultBmp).

Blok 4: Metode Publik Divide

C#

```
    // --- BAGI KONSTANTA ---
    public static Bitmap Divide(Bitmap img, double valueK)
    {
        // Hindari pembagian dengan nol
        if (valueK == 0) valueK = 1;
        // Bagi sama dengan mengalikan dengan 1/K
        return Multiply(img, 1.0 / valueK);
    }
```

- **public static Bitmap Divide(Bitmap img, double valueK)**
    - **Penjelasan:** Metode publik untuk membagi setiap piksel dengan valueK.
    - **Fungsi:** Jika valueK > 1.0, gambar akan menjadi lebih gelap.
- **if (valueK == 0) valueK = 1;**

    - **Penjelasan:** Ini adalah cek keamanan (safety check) yang sangat penting.
    - **Fungsi:** Memeriksa apakah valueK adalah nol.
    - **Alasan:** Pembagian dengan nol (DivideByZeroException) adalah *error* fatal dalam matematika dan pemrograman. Jika pengguna secara tidak sengaja memberikan nilai 0, baris ini akan mengubahnya menjadi 1. Membagi dengan 1 tidak mengubah gambar, yang merupakan perilaku aman dan lebih baik daripada membuat aplikasi *crash*.
- **return Multiply(img, 1.0 / valueK);**
    - **Penjelasan:** Ini adalah cara yang cerdas dan efisien untuk melakukan pembagian.
    - **Fungsi:** Daripada menulis ulang seluruh perulangan for x dan y hanya untuk mengubah * menjadi /, metode ini menggunakan identitas matematika: **Membagi dengan K sama dengan Mengalikan dengan (1/K)**.
    - **Alasan:** Ini adalah contoh bagus dari prinsip **DRY (Don't Repeat Yourself)**. Kode ini *menggunakan kembali* logika Multiply yang sudah ada. Jika Anda perlu memperbaiki bug di Multiply, perbaikan itu akan otomatis berlaku untuk Divide juga. Ini membuat kode lebih bersih dan mudah dipelihara.