

**LAPORAN JURNAL
MODUL 14**



Disusun Oleh :

Zaenarif Putra 'Ainurdin – 2311104049

Kelas :

SE-07-02

Dosen :

Yudha Islami Sulistya

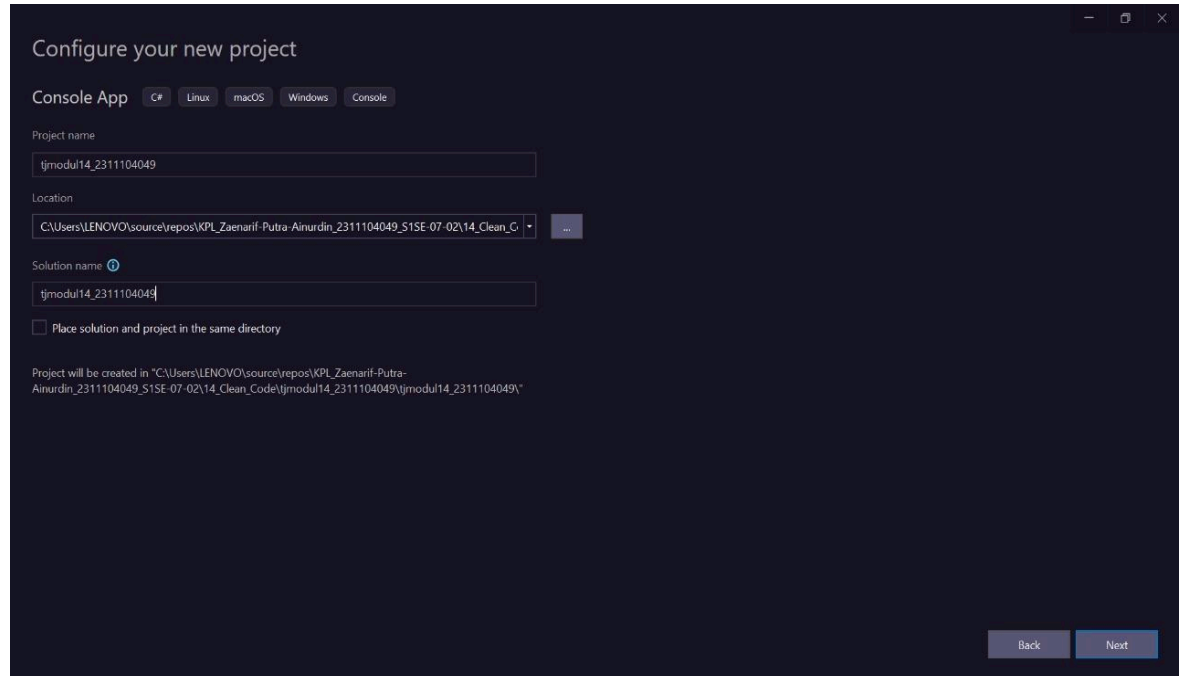
**PROGRAM STUDI SOFTWARE ENGINEERING
DIREKTORAT KAMPUS PURWOKERTO
TELKOM UNIVERSITY
PURWOKERTO
2025**

I. Link Github

https://github.com/zaenarifputra/KPL_Zaenarif-Putra-Ainurdin_2311104049_S1SE-07-02/tree/a4783b1e7d4c322eeae9315602032eb03fc66452/14_Clean_Code/TJ

II. Alur Pengerjaan

1. Dari modul14 berikut saya akan clean code pada modul 5 yang dimana, saya refactor dan membuat baru dengan menggunakan consoleapp seperti berikut:



2. Kemudian selanjutnya kita akan konfigurasi kekurangan dari modul 5 apa saja yang bisa di evaluasi dan di kembangkan lagi agar bisa lebih baik dan sudah bisa menerapkan konsep clean code pada modul 14:
 - a. kekurangan syntax sebelumnya : Syntax belum sepenuhnya mengikuti standar naming convention C# seperti penggunaan nama kelas dan method yang masih menggunakan bahasa Indonesia serta belum adanya komentar atau dokumentasi yang menjelaskan fungsi dari setiap bagian kode, sehingga mengurangi keterbacaan dan maintainability program.
 - b. Hal yang bisa diperbaiki : dari syntax pada modul 5 hal yang bisa diperbaiki Nama kelas dan method sebaiknya diubah menggunakan bahasa Inggris dan mengikuti format PascalCase, seperti mengganti Penjumlahan menjadi MathOperations dan JumlahTigaAngka menjadi AddThreeNumbers, serta menambahkan komentar XML dan inline comment pada setiap class dan method untuk menjelaskan fungsinya, serta merapikan whitespace dan indentasi agar kode lebih rapi dan profesional sesuai standar modul 14.

3. Kemudian setelah mengetahui apa saja yang bisa diperbaiki pada modul 5 dengan menerapkan konsep modul 14 berikut syntax dan penjelasan agar lebih mudah dipahami, yang dimana menggunakan consoleapp dan classnya diberi nama Program.cs seperti berikut :

```
using System;
using System.Collections.Generic;

namespace cleancodemd5
{
    /// <summary>
    /// Generic class digunakan sebagai tempat menyimpan data beserta cap waktu input.
    /// </summary>
    public class SimpleDatabase<T>
    {
        private readonly List<T> _storedData;
        private readonly List<DateTime> _inputDates;

        public SimpleDatabase()
        {
            _storedData = new List<T>();
            _inputDates = new List<DateTime>();
        }

        /// <summary>
        /// Menambahkan data baru dengan stempel waktu UTC saat ini.
        /// </summary>
        public void AddNewData(T data)
        {
            _storedData.Add(data);
            _inputDates.Add(DateTime.UtcNow);
        }

        /// <summary>
        /// Mencetak semua data yang tersimpan beserta stempel waktunya.
        /// </summary>
        public void PrintAllData()
        {
            for (int i = 0; i < _storedData.Count; i++)
            {
                Console.WriteLine($"Data {i + 1} berisi: {_storedData[i]}, yang disimpan
pada waktu UTC: {_inputDates[i]}");
            }
        }

        /// <summary>
        /// Class digunakan untuk melakukan generic mathematical operations.
        /// </summary>
        public class MathOperations
```

```
{
    /// <summary>
    /// Metode umum yang digunakan untuk menjumlahkan tiga nilai menggunakan
    /// </summary>
    public void AddThreeNumbers<T>(T value1, T value2, T value3)
    {
        dynamic a = value1;
        dynamic b = value2;
        dynamic c = value3;
        dynamic result = a + b + c;

        Console.WriteLine($"Hasil penjumlahan: {result}");
    }
}

/// <summary>
/// CClass utama yang digunakan untuk menjalankan program.
/// </summary>
public static class Program
{
    public static void Main()
    {
        var database = new SimpleDatabase<int>();

        Console.WriteLine("== Program SimpleDatabase ==\n");
        Console.WriteLine("Masukkan tiga angka dua digit dari NIM:");

        // Input 3 values from user
        for (int i = 0; i < 3; i++)
        {
            Console.Write($"Masukkan angka ke-{i + 1}: ");
            int inputNumber;
            while (!int.TryParse(Console.ReadLine(), out inputNumber) || inputNumber <
10 || inputNumber > 99)
            {
                Console.Write("Input tidak valid! Masukkan angka dua digit (10-99): ");
            }

            database.AddNewData(inputNumber);
        }

        Console.WriteLine("\nData yang telah dimasukkan:");
        database.PrintAllData();

        Console.WriteLine("\n== Program Penjumlahan ==\n");
        Console.Write("Masukkan NIM Anda: ");
        string nim = Console.ReadLine();

        if (string.IsNullOrEmpty(nim) || nim.Length < 8)
        {
```

```
Console.WriteLine("NIM harus memiliki minimal 8 digit.");
return;
}

// Extract 2-digit segments from NIM
string segment1 = nim.Substring(0, 2);
string segment2 = nim.Substring(2, 2);
string segment3 = nim.Substring(4, 2);
char lastDigit = nim[nim.Length - 1];

var calculator = new MathOperations();

try
{
    switch (lastDigit)
    {
        case '1':
        case '2':
            float f1 = float.Parse(segment1);
            float f2 = float.Parse(segment2);
            float f3 = float.Parse(segment3);
            calculator.AddThreeNumbers(f1, f2, f3);
            break;

        case '3':
        case '4':
        case '5':
            double d1 = double.Parse(segment1);
            double d2 = double.Parse(segment2);
            double d3 = double.Parse(segment3);
            calculator.AddThreeNumbers(d1, d2, d3);
            break;

        case '6':
        case '7':
        case '8':
            int i1 = int.Parse(segment1);
            int i2 = int.Parse(segment2);
            int i3 = int.Parse(segment3);
            calculator.AddThreeNumbers(i1, i2, i3);
            break;

        case '9':
        case '0':
            long l1 = long.Parse(segment1);
            long l2 = long.Parse(segment2);
            long l3 = long.Parse(segment3);
            calculator.AddThreeNumbers(l1, l2, l3);
            break;

        default:
```

```

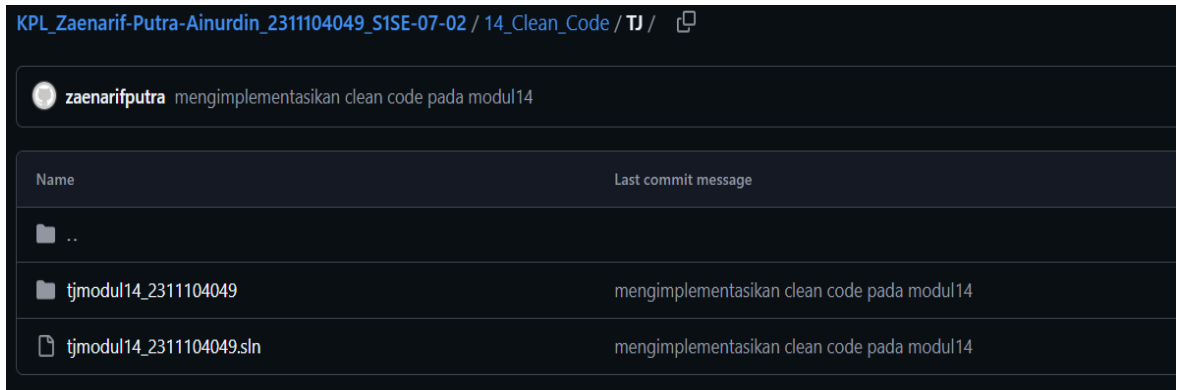
        Console.WriteLine("Digit terakhir NIM tidak valid.");
        break;
    }
}
catch (FormatException)
{
    Console.WriteLine("Format input tidak valid.");
}

Console.WriteLine("\nTekan Enter untuk keluar...");
Console.ReadLine();
}
}
}

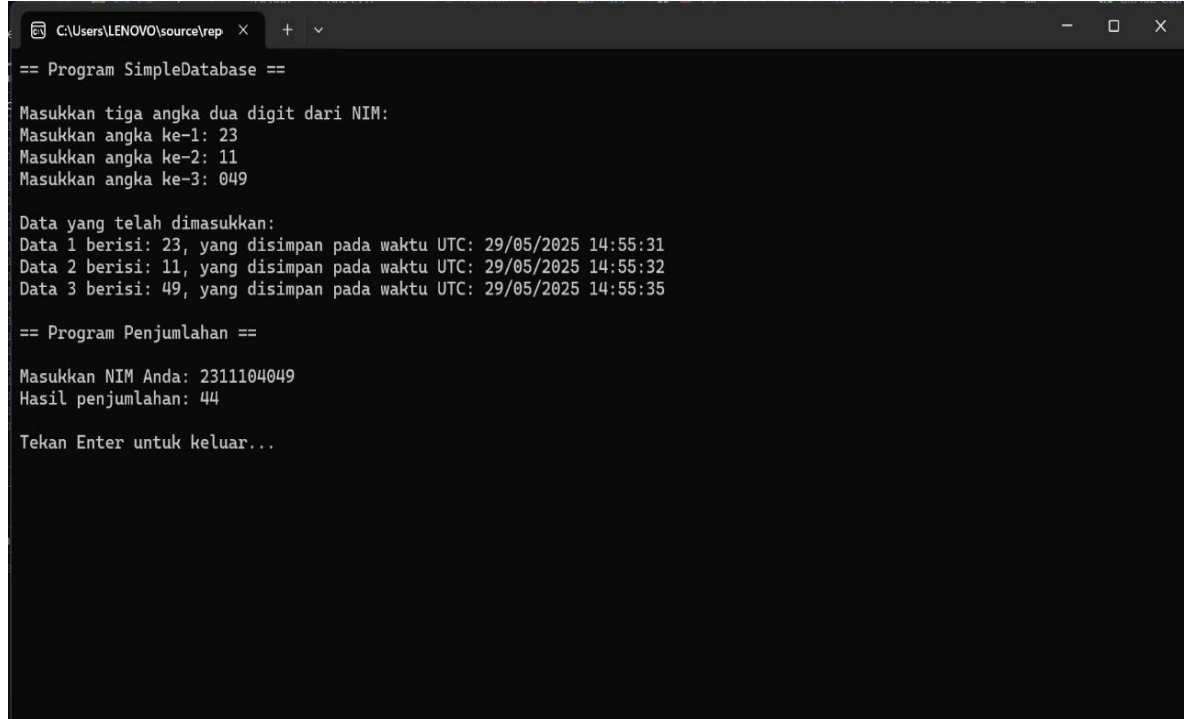
```

Pada versi yang telah diperbarui, dilakukan perbaikan naming convention dengan mengganti nama kelas dan method menjadi berbahasa Inggris dan mengikuti format PascalCase seperti SimpleDatabase dan AddThreeNumbers, agar sesuai dengan standar penulisan kode dalam bahasa C#. Selain itu, ditambahkan komentar XML pada setiap class dan method untuk menjelaskan fungsinya secara jelas dan informatif, sehingga memudahkan pemahaman bagi pembaca lain. Struktur program juga dirapikan dengan perbaikan indentasi, whitespace, dan pemisahan logika, menjadikan kode lebih bersih, konsisten, dan mudah dipelihara sesuai dengan prinsip clean code dan ketentuan refactoring Modul 14.

4. setelah semua implementasi yang dibutuhkan sudah berhasil di implementasikan semua selanjutnya melakukan push menuju cloud github yang dimana untuk melakukan push bisa melakukannya dengan perintah `git add .`, `git commit -m "mengimplementasikan Clean code pada modul 14"` dan kemudian terakhir melakukan `git push -u origin master`. jika berhasil maka akan di tampilkan seperti gambar berikut :



III. Hasil Running



```
C:\Users\LENOVO\source\rep >
== Program SimpleDatabase ==
Masukkan tiga angka dua digit dari NIM:
Masukkan angka ke-1: 23
Masukkan angka ke-2: 11
Masukkan angka ke-3: 049

Data yang telah dimasukkan:
Data 1 berisi: 23, yang disimpan pada waktu UTC: 29/05/2025 14:55:31
Data 2 berisi: 11, yang disimpan pada waktu UTC: 29/05/2025 14:55:32
Data 3 berisi: 49, yang disimpan pada waktu UTC: 29/05/2025 14:55:35

== Program Penjumlahan ==

Masukkan NIM Anda: 2311104049
Hasil penjumlahan: 44

Tekan Enter untuk keluar...
```

Kesimpulan yang bisa saya ambil adalah saya jadi lebih paham gimana pentingnya menerapkan konsep clean code dalam pemrograman, terutama soal penamaan yang konsisten, struktur kode yang rapi, dan penggunaan komentar supaya program lebih mudah dibaca dan dipahami. Dengan refactor dari versi sebelumnya, kode jadi kelihatan lebih profesional dan enak dilihat. Selain itu, saya juga belajar gimana caranya mengatur alur program dengan lebih jelas, serta gimana ngegabungin logika dengan pendekatan yang lebih bersih dan terstruktur. Intinya, setelah dibenerin pakai prinsip clean code, programnya jadi lebih keren, lebih gampang di-maintain, dan tentunya lebih sesuai standar yang diminta di modul praktikum.