

LAPORAN PRAKTIKUM
JURNAL MODUL 13



Nama :

Aulia Jasifa Br Ginting 2311104060

S1SE-07-02

Dosen :

Yudha Islami Sulistya, S.Kom., M.Kom

PROGRAM STUDI S1 REKAYASA PERANGKAT LUNAK

FAKULTAS INFORMATIKA

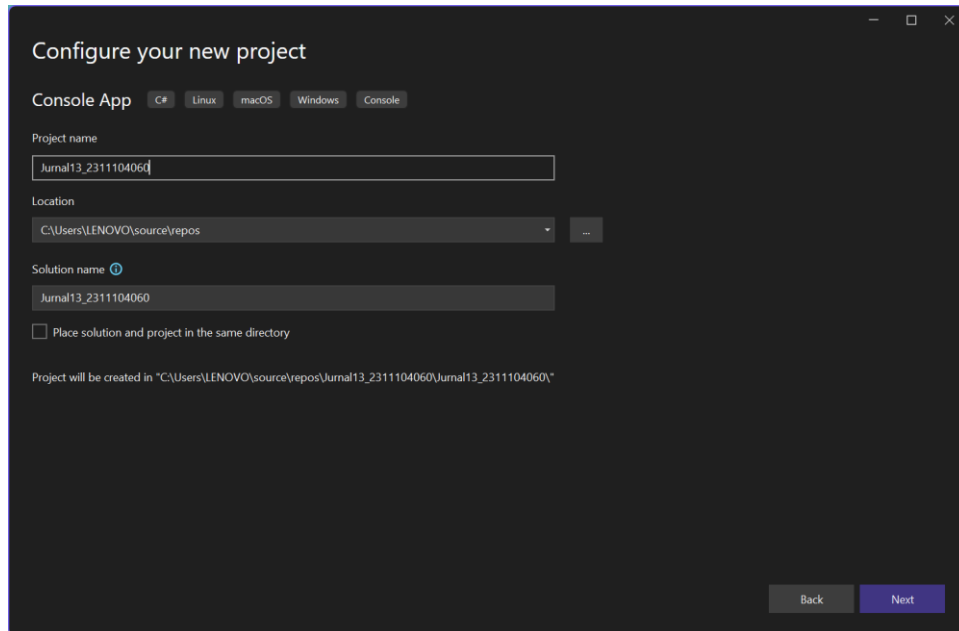
TELKOM UNIVERSITY PURWOKERTO

2025

I. LINK GITHUB

https://github.com/auliajsf06/KPL_Aulia-Jasifa-Br-Ginting_2311104060_SE-07-02

II. MEMBUAT PROJECT GUI BARU



III. MENJELASKAN DESIGN PATTERN SINGLETON

A. Berikan salah dua contoh kondisi dimana design pattern “Singleton” dapat digunakan.

1. Logging System (Sistem Pencatatan Log)
Dalam sebuah aplikasi, pencatatan log sebaiknya ditangani oleh satu objek tunggal agar hasil log konsisten dan tidak tumpang tindih. Dengan Singleton, semua komponen sistem dapat menggunakan instance logger yang sama.
2. Configuration Manager
Untuk membaca pengaturan dari file konfigurasi (seperti .config, .json, atau dari database), Singleton digunakan agar semua bagian sistem mengakses instance konfigurasi yang sama tanpa perlu memuat ulang data berkali-kali.

B. Berikan penjelasan singkat mengenai langkah-langkah dalam mengimplementasikan design pattern “Singleton”.

1. Buat Kelas Singleton: Definisikan kelas yang akan menjadi Singleton. Kelas ini harus memiliki konstruktor privat untuk mencegah instansiasi langsung dari luar kelas.
2. Buat Instance Statis: Di dalam kelas, buat field statis untuk menyimpan instance dari kelas Singleton. Field ini biasanya diinisialisasi dengan null.
3. Buat Metode Akses Statis: Implementasikan metode statis (misalnya, GetInstance) yang akan mengembalikan instance dari kelas Singleton. Di dalam metode ini, periksa apakah instance sudah ada; jika tidak, buat instance baru dan simpan di field statis.

4. Tandai Kelas sebagai Sealed: Untuk mencegah pewarisan dari kelas Singleton, tandai kelas tersebut sebagai sealed.
5. Implementasikan Logika Bisnis: Tambahkan metode dan logika bisnis yang diperlukan dalam kelas Singleton.

C. Berikan tiga kelebihan dan kekurangan dari design pattern “Singleton”.

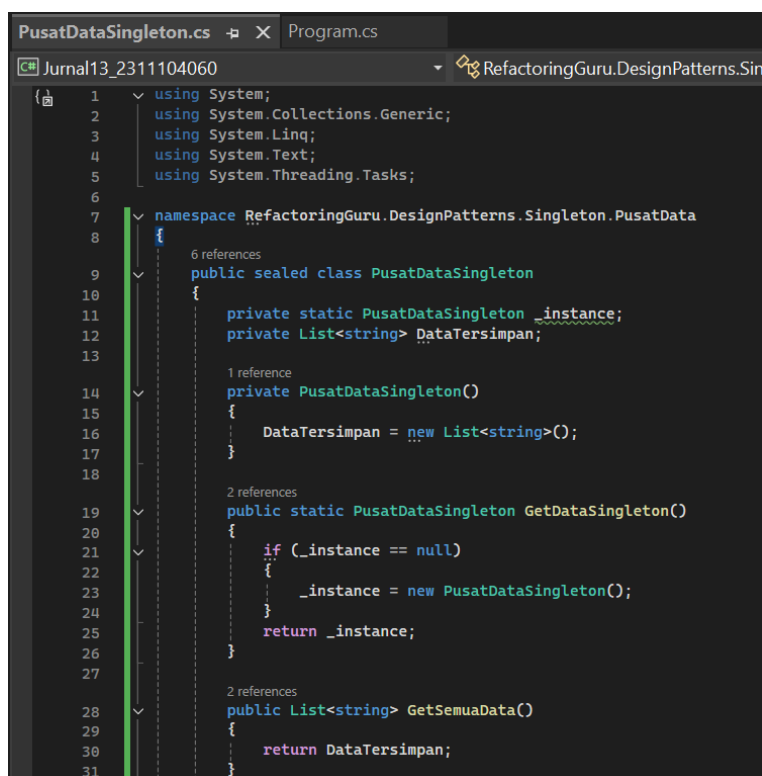
Kelebihan

1. Kontrol Akses Global: Singleton memberikan cara yang terpusat untuk mengakses instance, sehingga memudahkan pengelolaan dan penggunaan sumber daya yang terbatas.
2. Penghematan Memori: Dengan hanya memiliki satu instance, penggunaan memori dapat dioptimalkan, terutama untuk objek yang berat.
3. Konsistensi Data: Menggunakan Singleton memastikan bahwa semua bagian aplikasi menggunakan data yang sama, mengurangi risiko inkonsistensi.

Kekurangan

1. Kesulitan dalam Pengujian: Singleton dapat menyulitkan pengujian unit karena ketergantungan pada instance global, yang dapat membuat pengujian menjadi lebih kompleks.
2. Pelanggaran Prinsip Inversi Ketergantungan: Singleton dapat menyebabkan pelanggaran prinsip desain yang baik, di mana kelas menjadi terlalu terikat pada implementasi spesifik.
3. Masalah dalam Lingkungan Multithreaded: Implementasi Singleton yang tidak aman untuk multithreading dapat menyebabkan beberapa instance dibuat jika beberapa thread mencoba mengaksesnya secara bersamaan.

IV. IMPLEMENTASI DAN PEMAHAMAN DESIGN PATTERN SINGLETON



```

1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Threading.Tasks;
6
7  namespace RefactoringGuru.DesignPatterns.Singleton.PusatData
8  {
9      public sealed class PusatDataSingleton
10     {
11         private static PusatDataSingleton _instance;
12         private List<string> DataTersimpan;
13
14         private PusatDataSingleton()
15         {
16             DataTersimpan = new List<string>();
17         }
18
19         public static PusatDataSingleton GetDataSingleton()
20         {
21             if (_instance == null)
22             {
23                 _instance = new PusatDataSingleton();
24             }
25             return _instance;
26         }
27
28         public List<string> GetSemuaData()
29         {
30             return DataTersimpan;
31         }
32     }

```

```
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66

2 references
public void PrintSemuaData()
{
    if (DataTersimpan.Count == 0)
    {
        Console.WriteLine("Belum ada data yang tersimpan.");
        return;
    }

    Console.WriteLine("Data Tersimpan:");
    for (int i = 0; i < DataTersimpan.Count; i++)
    {
        Console.WriteLine($"[{i}] {DataTersimpan[i]}");
    }
}

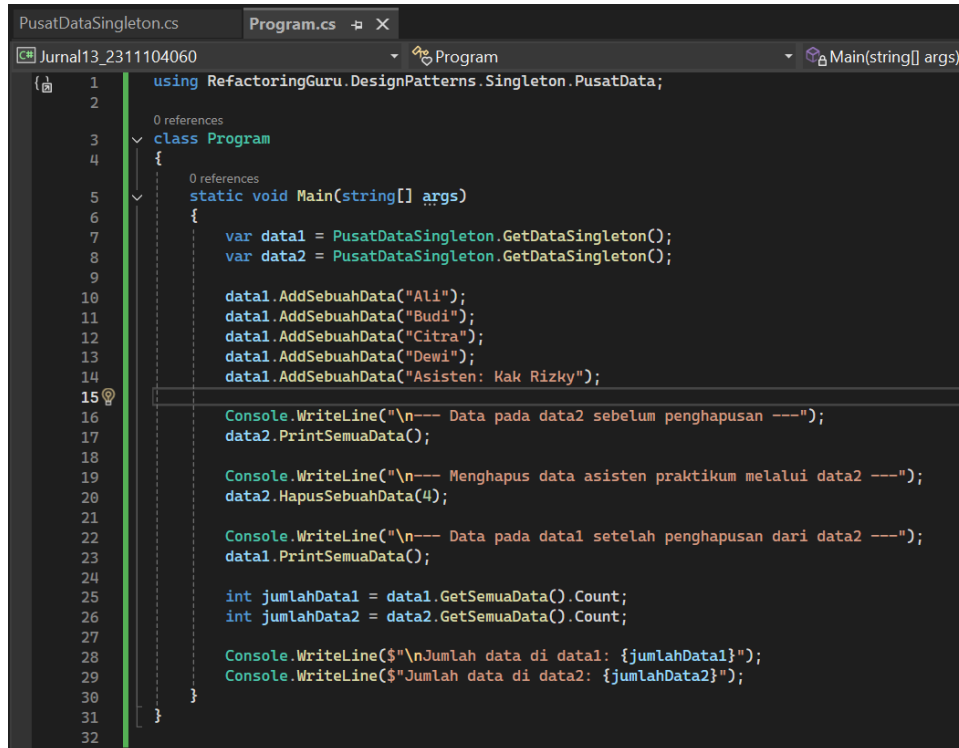
5 references
public void AddSebuahData(string input)
{
    DataTersimpan.Add(input);
    Console.WriteLine($"Data '{input}' berhasil ditambahkan.");
}

1 reference
public void HapusSebuahData(int index)
{
    if (index >= 0 && index < DataTersimpan.Count)
    {
        Console.WriteLine($"Data '{DataTersimpan[index]}' dihapus.");
        DataTersimpan.RemoveAt(index);
    }
    else
    {
        Console.WriteLine("Index tidak valid.");
    }
}
```

Penjelasan Syntax:

Class `PusatDataSingleton` merupakan implementasi dari design pattern Singleton, yaitu pola desain yang menjamin bahwa hanya ada satu instance dari class tersebut yang dibuat selama siklus hidup aplikasi. Hal ini dicapai dengan menjadikan konstruktor class sebagai `private`, sehingga tidak bisa diakses dari luar, dan menyediakan sebuah method statis `GetDataSingleton()` untuk mengontrol pembuatan instance. Jika instance belum ada (`null`), method ini akan membuat instance baru dan menyimpannya dalam variabel statis `_instance`; jika sudah ada, maka instance yang sama akan dikembalikan. Class ini memiliki atribut `DataTersimpan` yang bertipe `List<string>` untuk menyimpan data berupa string, seperti nama-nama anggota kelompok. Selain itu, class ini juga menyediakan method untuk menambahkan data (`AddSebuahData`), menampilkan semua data (`PrintSemuaData`), menghapus data berdasarkan indeks tertentu (`HapusSebuahData`), serta mengambil seluruh isi list (`GetSemuaData`). Dengan pendekatan ini, seluruh bagian aplikasi yang memanggil `PusatDataSingleton` akan mengakses dan memanipulasi data yang sama.

V. IMPLEMENTASI PROGRAM UTAMA



```
1 using RefactoringGuru.DesignPatterns.Singleton.PusatData;
2
3 class Program
4 {
5     static void Main(string[] args)
6     {
7         var data1 = PusatDataSingleton.GetDataSingleton();
8         var data2 = PusatDataSingleton.GetDataSingleton();
9
10        data1.AddSebuahData("Ali");
11        data1.AddSebuahData("Budi");
12        data1.AddSebuahData("Citra");
13        data1.AddSebuahData("Dewi");
14        data1.AddSebuahData("Asisten: Kak Rizky");
15
16        Console.WriteLine("\n--- Data pada data2 sebelum penghapusan ---");
17        data2.PrintSemuaData();
18
19        Console.WriteLine("\n--- Menghapus data asisten praktikum melalui data2 ---");
20        data2.HapusSebuahData(4);
21
22        Console.WriteLine("\n--- Data pada data1 setelah penghapusan dari data2 ---");
23        data1.PrintSemuaData();
24
25        int jumlahData1 = data1.GetSemuaData().Count;
26        int jumlahData2 = data2.GetSemuaData().Count;
27
28        Console.WriteLine($"Jumlah data di data1: {jumlahData1}");
29        Console.WriteLine($"Jumlah data di data2: {jumlahData2}");
30    }
31 }
32
```

Penjelasan Syntax:

Class Program berfungsi sebagai titik awal eksekusi program. Dalam method Main, dilakukan serangkaian langkah untuk mendemonstrasikan penggunaan class PusatDataSingleton. Pertama, dibuat dua variabel data1 dan data2 yang keduanya diisi menggunakan method GetDataSingleton(), yang mengembalikan instance yang sama dari singleton. Kemudian, melalui data1, ditambahkan beberapa nama anggota kelompok dan asisten praktikum ke dalam list data. Selanjutnya, data2 digunakan untuk mencetak seluruh data yang tersimpan, yang akan menampilkan data yang sebelumnya ditambahkan melalui data1. Setelah itu, dilakukan penghapusan terhadap nama asisten praktikum melalui data2. Karena data1 dan data2 merujuk pada instance yang sama, maka saat mencetak ulang data dari data1, nama asisten yang dihapus juga tidak akan muncul lagi. Di bagian akhir program, baik data1 maupun data2 memanggil GetSemuaData() untuk mencetak jumlah data yang tersisa, yang hasilnya akan sama. Hal ini membuktikan bahwa instance yang digunakan benar-benar satu dan konsisten, sebagaimana prinsip utama dari design pattern Singleton.

Output yang dihasilkan:

```
Microsoft Visual Studio Debu x + v
Data 'Citra' berhasil ditambahkan.
Data 'Dewi' berhasil ditambahkan.
Data 'Asisten: Kak Rizky' berhasil ditambahkan.

--- Data pada data2 sebelum penghapusan ---
Data Tersimpan:
[0] Ali
[1] Budi
[2] Citra
[3] Dewi
[4] Asisten: Kak Rizky

--- Menghapus data asisten praktikum melalui data2 ---
Data 'Asisten: Kak Rizky' dihapus.

--- Data pada data1 setelah penghapusan dari data2 ---
Data Tersimpan:
[0] Ali
[1] Budi
[2] Citra
[3] Dewi

Jumlah data di data1: 4
Jumlah data di data2: 3

C:\Users\LENOVO\source\repos\Jurnal13_2311104060\Jurnal13_2311104060\bin\Debug\net8.0\Jurnal13_2311104060.exe (process 18880) exited with code 0 (0x0).
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .|
```

VI. KESIMPULAN

Pada praktikum ini, mempelajari bagaimana diimplementasikan design pattern Singleton dalam bahasa C# dengan membuat kelas PusatDataSingleton yang memastikan hanya satu instance dari kelas tersebut yang dapat dibuat dan diakses secara global. Implementasi ini memungkinkan pengelolaan data yang konsisten melalui satu sumber data bersama, sebagaimana dibuktikan dengan manipulasi data dari dua variabel berbeda (data1 dan data2) yang tetap mengakses instance yang sama. Praktikum ini menunjukkan bahwa Singleton sangat berguna untuk menjaga integritas data dan kontrol akses terpusat, meskipun penggunaannya harus dilakukan secara bijak agar tidak mengganggu modularitas dan kemudahan pengujian dalam pengembangan perangkat lunak.