JURNAL MODUL 13

Nama : Ghaza Zidane Nurraihan
Nim : 2311104038
Kelas : S1-SE07-01

A. Berikan salah dua contoh kondisi dimana design pattern “Singleton” dapat digunakan.
Jawab :
1.Koneksi ke Database:

- Dalam sebuah aplikasi, umumnya hanya dibutuhkan satu koneksi utama ke database untuk menghemat sumber daya dan menghindari konflik. Dengan Singleton, kita bisa memastikan hanya satu instance koneksi database yang dibuat dan digunakan oleh seluruh bagian aplikasi.

2.Logger (Pencatat Log):

- Logger adalah komponen yang bertugas mencatat aktivitas program. Jika ada banyak instance logger, catatan log bisa tersebar atau tumpang tindih. Dengan Singleton, kita memastikan hanya satu objek logger yang dipakai secara global oleh semua modul.

B. Berikan penjelasan singkat mengenai langkah-langkah dalam mengimplementasikan design pattern “Singleton”.
Jawaban :
1.Tambahkan field statis (static) privat dalam kelas untuk menyimpan instance Singleton.
2.Buat konstruktor privat agar objek tidak bisa dibuat dari luar kelas menggunakan new.
3.Buat metode statis publik (misalnya getInstance) untuk:

- Mengecek apakah instance sudah dibuat.
- Jika belum, buat instance baru.
- Kembalikan instance yang sama setiap kali dipanggil.

4.Ubah kode klien agar menggunakan getInstance() sebagai satu-satunya cara untuk mengakses objek tersebut.

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

Kelebihan:
1.Menjamin hanya satu instance yang dibuat, cocok untuk sumber daya bersama seperti koneksi database.

2.Memberikan titik akses global yang mudah ke instance tersebut.
3.Inisialisasi tertunda (lazy initialization): objek baru dibuat hanya saat pertama kali dibutuhkan.

Kekurangan:
1.Melanggar prinsip Single Responsibility, karena menangani dua tanggung jawab sekaligus: kontrol instansiasi dan akses global.
2.Sulit di-test unit, karena konstruktor privat menyulitkan pembuatan mock object.
3.Perlu penanganan khusus di lingkungan multithread, untuk mencegah pembuatan banyak instance secara bersamaan.

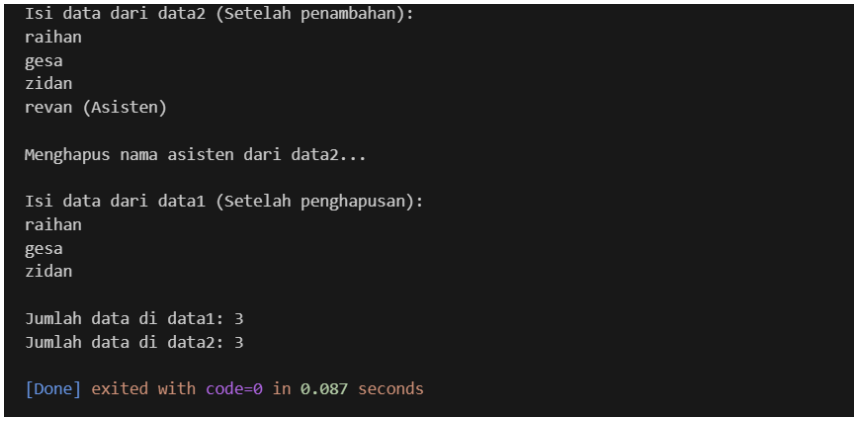
IMPLEMENTASI DAN PEMAHAMAN DESIGN PATTERN SINGLETON

```python
class PusatDataSingleton:
    # Properti statis untuk menyimpan instance tunggal
    _instance = None

    def __init__(self):
        # Konstruktor privat: hanya boleh dipanggil sekali
        if PusatDataSingleton._instance is not None:
            raise Exception("Gunakan metode GetDataSingleton() untuk mengakses instance.")
        self.DataTersimpan = []

    @classmethod
    def GetDataSingleton(cls):
        # Method untuk mengakses atau membuat instance tunggal
        if cls._instance is None:
            cls._instance = PusatDataSingleton()
        return cls._instance

    def GetSemuaData(self):
        # Mengembalikan seluruh data dalam list
        return self.DataTersimpan

    def PrintSemuaData(self):
        # Print satu per satu isi list
        if not self.DataTersimpan:
            print("Data kosong.")
        else:
            for data in self.DataTersimpan:
                print(data)

    def AddSebuahData(self, input):
        # Tambahkan data ke list
        self.DataTersimpan.append(input)

    def HapusSebuahData(self, index):
        # Hapus data berdasarkan index
        if 0 <= index < len(self.DataTersimpan):
            del self.DataTersimpan[index]
        else:
            print("Index tidak valid.")

```

Class PusatDataSingleton adalah implementasi pola desain Singleton menggunakan metaclass SingletonMeta yang memastikan hanya satu instance dari class ini dapat dibuat. Class ini memiliki atribut DataTersimpan, yaitu list string untuk menyimpan data, dan property _instance sebagai referensi ke instance tunggalnya. Melalui method GetDataSingleton(), pengguna bisa mengakses instance tersebut dengan aman. Class ini juga menyediakan method untuk menambah (AddSebuahData), menghapus (HapusSebuahData), melihat (GetSemuaData), dan mencetak semua data (PrintSemuaData). Pola ini cocok digunakan saat dibutuhkan pusat data bersama dalam program.

IMPLEMENTASI PROGRAM UTAMA

```python
class PusatDataSingleton:
    _instance = None

    def __init__(self):
        if PusatDataSingleton._instance is not None:
            raise Exception("Gunakan GetDataSingleton() untuk mendapatkan instance.")
        self.DataTersimpan = []

    @classmethod
    def GetDataSingleton(cls):
        if cls._instance is None:
            cls._instance = PusatDataSingleton()
        return cls._instance

    def GetSemuaData(self):
        return self.DataTersimpan

    def PrintSemuaData(self):
        if not self.DataTersimpan:
            print("Data kosong.")
        else:
            for data in self.DataTersimpan:
                print(data)

    def AddSebuahData(self, input):
        self.DataTersimpan.append(input)

    def HapusSebuahData(self, index):
        if 0 <= index < len(self.DataTersimpan):
            del self.DataTersimpan[index]
        else:
            print("Index tidak valid.")

if __name__ == "__main__":
    # A & B: Buat dua variabel dari hasil GetDataSingleton
    data1 = PusatDataSingleton.GetDataSingleton()
    data2 = PusatDataSingleton.GetDataSingleton()

    # C: Tambahkan data nama anggota kelompok dan asisten praktikum
    data1.AddSebuahData("raihan")              # Anggota kelompok
    data1.AddSebuahData("gesa")             # Anggota kelompok
    data1.AddSebuahData("zidan")            # Anggota kelompok
    data1.AddSebuahData("revan (Asisten)")   # Asisten praktikum

    # D: Print isi data2
    print("Isi data dari data2 (Setelah penambahan):")
    data2.PrintSemuaData()

    # E: Hapus nama asisten dari data2
    print("\nMenghapus nama asisten dari data2...")
    # Cari index asisten secara dinamis
    semua_data = data2.GetSemuaData()
    try:
        index_asisten = semua_data.index("revan (Asisten)")
        data2.HapusSebuahData(index_asisten)
    except ValueError:
        print("Nama asisten tidak ditemukan.")

    # F: Print isi data1 (asisten harus sudah hilang)
    print("\nIsi data dari data1 (Setelah penghapusan):")
    data1.PrintSemuaData()

    # G: Tampilkan jumlah data di data1 dan data2
    count1 = len(data1.GetSemuaData())
    count2 = len(data2.GetSemuaData())
    print(f"\nJumlah data di data1: {count1}")
    print(f"Jumlah data di data2: {count2}")

```



Program ini menggunakan design pattern Singleton untuk memastikan bahwa hanya ada satu instance dari class PusatDataSingleton yang menyimpan data secara global. Dengan atribut DataTersimpan berupa list string, class ini menyediakan method untuk menambahkan, menghapus, dan mencetak data. Dalam fungsi main, dua variabel (data1 dan data2) diisi dari instance yang sama melalui GetDataSingleton(). Penambahan data dilakukan melalui data1, dan saat data2 mencetak atau menghapus data, perubahan juga terlihat pada data1, membuktikan bahwa keduanya mengakses instance yang sama. Program ini menunjukkan bahwa data tersimpan secara terpusat dan konsisten antar semua variabel Singleton.