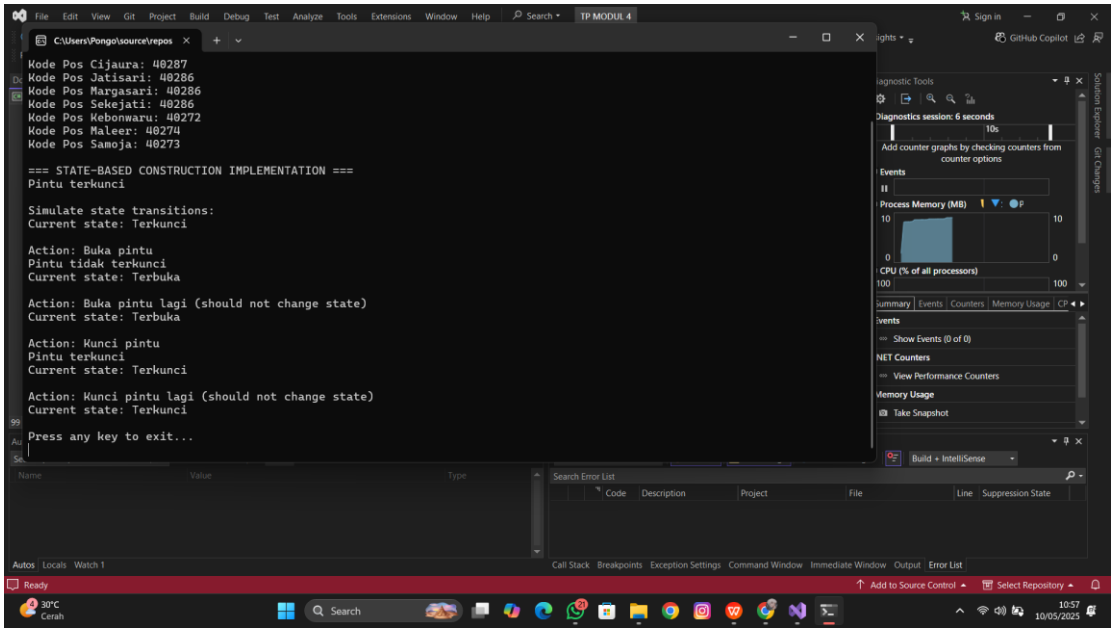
TP MODUL 4
Reza Afiansyah Wibowo
2311104062



1. Implementasi Table-Driven (KodePos.cs)
Table-driven adalah pendekatan pemrograman di mana logika program ditentukan oleh tabel atau struktur data (dalam kasus ini Dictionary) daripada menggunakan serangkaian kondisional seperti if-else atau switch-case. Pendekatan ini membuat kode lebih mudah dikelola, dibaca, dan dimodifikasi.

```
public class KodePos
{
    // Tabel berupa Dictionary yang memetakan kelurahan dengan kode pos
    private readonly Dictionary<string, string> kodePosTable = new Dictionary<string, string>
    {
        { "Batununggal", "40266" },
        { "Kujangsari", "40287" },
        { "Mengger", "40267" },
        // dan seterusnya...
    };

    // Method untuk mendapatkan kode pos berdasarkan nama kelurahan
    public string getKodePos(string kelurahan)
    {
        if (kodePosTable.ContainsKey(kelurahan))
        {
            return kodePosTable[kelurahan];
        }
        return "Kelurahan not found";
    }
}
```

Keuntungan implementasi table-driven:

Efisiensi: Pencarian dalam Dictionary memiliki kompleksitas waktu O(1) sehingga lebih efisien dibandingkan kondisional bersarang.
Maintainability: Menambah, mengubah, atau menghapus data kode pos sangat mudah dan tidak memerlukan perubahan logika program.
Readability: Kode lebih bersih dan mudah dibaca karena tidak ada kondisional yang kompleks.
Separation of concerns: Data (kode pos) dipisahkan dari logika (pencarian kode pos).

Dengan implementasi ini, kita cukup memanggil getKodePos("Nama Kelurahan") untuk mendapatkan kode pos yang sesuai.

2. Implementasi State-Based Construction (DoorMachine.cs)
State-based construction adalah teknik pemrograman di mana objek mengubah perilakunya berdasarkan state internalnya. Ini adalah implementasi dari pola desain State Pattern, yang memungkinkan objek mengubah perilakunya ketika state-nya berubah.

```
public class DoorMachine
{
    // Definisi state menggunakan enum
    public enum State
    {
        Terkunci,
        Terbuka
    }

    // State saat ini, diinisialisasi ke Terkunci
    private State currentState = State.Terkunci;

    // Constructor untuk menampilkan pesan awal
    public DoorMachine()
    {
        Console.WriteLine("Pintu terkunci");
    }

    // Method untuk aksi "Buka"
    public void Buka()
    {
        // Hanya mengubah state jika saat ini terkunci
        if (currentState == State.Terkunci)
        {
            currentState = State.Terbuka;
            Console.WriteLine("Pintu tidak terkunci");
        }
    }

    // Method untuk aksi "Kunci"
    public void Kunci()
    {
        // Hanya mengubah state jika saat ini terbuka
        if (currentState == State.Terbuka)
        {
            currentState = State.Terkunci;
            Console.WriteLine("Pintu terkunci");
        }
    }

    // Method untuk mendapatkan state saat ini
    public State GetCurrentState()
    {
        return currentState;
    }
}
```

Keuntungan implementasi state-based construction:

Pemisahan logika: Setiap state memiliki logika dan perilaku yang terpisah.
Enkapsulasi perubahan state: Transisi antar state dienkapsulasi dalam metode-metode yang jelas.

Kode yang mudah diperluas: Jika nanti diperlukan state tambahan, kita hanya perlu menambahkan nilai dalam enum dan menangani logika transisinya.
Reduksi kompleksitas: Tidak ada kondisional bersarang yang kompleks.

State machine dalam kode ini memiliki dua state (Terkunci dan Terbuka) dengan dua transisi possible:

Dari Terkunci ke Terbuka melalui aksi Buka()
Dari Terbuka ke Terkunci melalui aksi Kunci()

3. Program Utama (Program.cs)
Program utama mendemonstrasikan penggunaan kedua implementasi di atas:

```
class Program
{
    static void Main(string[] args)
    {
        // Implementasi Table-Driven
        Console.WriteLine("=== TABLE-DRIVEN IMPLEMENTATION ===");
        KodePos kodePos = new KodePos();
        Console.WriteLine("Kode Pos Batununggal: " + kodePos.getKodePos("Batununggal"));
        // dan seterusnya...

        // Implementasi State-Based Construction
        Console.WriteLine("\n=== STATE-BASED CONSTRUCTION IMPLEMENTATION ===");
        DoorMachine door = new DoorMachine();

        // Simulasi perubahan state
        Console.WriteLine("\nSimulate state transitions:");
        Console.WriteLine("Current state: " + door.GetCurrentState());

        Console.WriteLine("\nAction: Buka pintu");
        door.Buka();
        Console.WriteLine("Current state: " + door.GetCurrentState());

        // dan seterusnya...
    }
}
```

Keuntungan struktur program utama:

Demo yang jelas: Program mendemonstrasikan dengan jelas penggunaan kedua implementasi.
Output yang informatif: Program menampilkan pesan-pesan yang informatif tentang state dan aksi yang dilakukan.
Pemisahan implementasi: Kedua implementasi ditampilkan secara terpisah sehingga mudah dipahami.

Dengan implementasi ini, program akan menampilkan output berupa daftar kode pos berdasarkan kelurahan dan simulasi perubahan state pada pintu, menunjukkan bagaimana kedua pendekatan pemrograman (table-driven dan state-based) dapat diterapkan dalam konteks yang berbeda.