

Universitatea Tehnică a Moldovei
Facultatea Calculatoare Informatică și
Microelectronică
Departamentul Ingineria Software și Automatică

Proiect de an

Disciplina : Tehnici și mecanisme de proiectare a
produselor program
Tema: “HomePilot” – aplicație consola pentru controlul și monitorizarea
dispozitivelor dintr-o casă inteligentă

A efectuat:

Temciuc Vitalie, TI-224

A verificat:

Bîtca Ernest

Chișinău, 2025

Cuprins

| | |
|---|----|
| Introducere..... | 3 |
| Scopurile / obiectivele proiectului..... | 4 |
| Proiectarea sistemului..... | 5 |
| Implementarea..... | 9 |
| Utilizarea șabloanelor de proiectare..... | 9 |
| Factory Pattern..... | 9 |
| Singleton Pattern..... | 10 |
| Decorator Pattern..... | 10 |
| Facade Pattern..... | 11 |
| Observer Pattern..... | 11 |
| Importanța utilizării șabloanelor de proiectare în cod..... | 12 |
| Interacțiunea cu utilizatorul..... | 14 |
| Secvențe de cod..... | 14 |
| Documentarea Produsului..... | 17 |
| Concluzie..... | 19 |
| Anexa 1..... | 20 |
| Anexa 2..... | 23 |

Introducere

În ultimele decenii, evoluția tehnologiei a condus la dezvoltarea rapidă a sistemelor inteligente integrate în mediul casnic, cunoscute sub denumirea de case inteligente (Smart Homes). Acestea oferă utilizatorilor posibilitatea de a controla și automatiza dispozitivele electronice din locuințe, pentru a crește confortul, eficiența energetică și securitatea. O componentă esențială în dezvoltarea acestor sisteme o reprezintă software-ul capabil să gestioneze și să sincronizeze diverse tipuri de echipamente, de la iluminat și termostate, la camere de supraveghere și încuietori inteligente.

Proiectarea și implementarea unui astfel de sistem software ridică numeroase provocări, mai ales datorită complexității și diversității dispozitivelor și scenariilor de utilizare. În acest context, este necesară o arhitectură robustă, flexibilă și scalabilă, care să permită extinderea facilă a funcționalităților și întreținerea ușoară a codului. Pentru atingerea acestor obiective, folosirea șabloanelor de proiectare (design patterns) devine un instrument esențial pentru dezvoltatori.

Șabloanele de proiectare sunt soluții standardizate și recunoscute pentru probleme frecvente în dezvoltarea software. Ele oferă un limbaj comun și un set clar de reguli pentru organizarea codului, facilitând astfel dezvoltarea de aplicații modulare și ușor de întreținut. Prin aplicarea acestor șabloane, se poate asigura o separare clară a responsabilităților, reducerea cuplării și creșterea coeziunii componentelor, ceea ce conduce la un cod mai curat și mai eficient.

Acest proiect, intitulat HomePilot, exemplifică implementarea practică a unor șase șabloane de design: Factory, Singleton, Decorator, Observer, Command și Facade. Alegerea acestor pattern-uri urmărește acoperirea aspectelor fundamentale ale unei aplicații smart home, de la crearea flexibilă a obiectelor și gestionarea centralizată a dispozitivelor, până la extensibilitatea funcționalităților și facilitarea utilizării prin interfețe simple și scenarii complexe predefinite.

În concluzie, utilizarea șabloanelor de proiectare în acest proiect evidențiază modul în care bunele practici din domeniul arhitecturii software pot transforma un sistem complex într-o aplicație elegantă și funcțională, capabilă să răspundă nevoilor reale ale utilizatorilor casnici, oferind în același timp o bază solidă pentru dezvoltări viitoare.

Scopurile / obiectivele proiectului

Scopul principal al proiectului HomePilot este dezvoltarea unei aplicații software care să permită controlul centralizat și eficient al dispozitivelor inteligente din cadrul unei locuințe, oferind o interfață simplă, intuitivă și flexibilă pentru utilizator. Aplicația urmărește să demonstreze în mod practic și funcțional aplicarea șabloanelor de proiectare (design patterns) fundamentale, ca mijloc de organizare a codului, creștere a scalabilității și mentenanței sistemului.

Obiectivele proiectului:

- Implementarea șablonului Factory pentru crearea dispozitivelor smart într-un mod flexibil și decuplat. Prin această abordare, logica de creare a dispozitivelor este izolată de restul aplicației, permițând adăugarea facilă a unor noi tipuri de dispozitive fără a afecta codul existent.
- Utilizarea șablonului Singleton pentru asigurarea existenței unei singure instanțe globale a controller-ului care gestionează lista și starea dispozitivelor smart. Aceasta oferă un punct centralizat de acces și control asupra întregului sistem, garantând consistența datelor și sincronizarea operațiunilor.
- Aplicarea șablonului Decorator pentru extinderea dinamică a funcționalităților dispozitivelor, fără a modifica clasele de bază. Astfel, pot fi adăugate caracteristici suplimentare, precum funcția de vedere nocturnă la camere, oferind o arhitectură flexibilă și extensibilă.
- Implementarea șablonului Observer pentru notificarea automată a modificărilor stării dispozitivelor către componente interesate (ex. interfața utilizator, loguri). Acest model ajută la decuplarea componentelor și facilitează adăugarea de noi funcționalități de monitorizare.
- Folosirea șablonului Command pentru encapsularea comenzilor executate asupra dispozitivelor. Aceasta permite o gestionare unitară a acțiunilor, oferind o structură clară pentru execuția, anularea și posibilă înregistrare a comenzilor.

- Implementarea șablonului Facade pentru oferirea unei interfețe simplificate care grupează mai multe operații complexe într-un singur apel, exemplificat prin scenariul „mod vacanță”, ce combină stingerea luminilor, reglarea termostatlui și blocarea ușilor.
- Crearea unei interfețe CLI interactive care să permită utilizatorului final să controleze casa inteligentă într-un mod intuitiv și accesibil, prin meniuri clare și feedback imediat sub formă de notificări.
- Asigurarea modularității și scalabilității codului, astfel încât proiectul să poată fi extins în viitor cu noi tipuri de dispozitive, comenzi și funcționalități, fără a afecta arhitectura existentă.

Proiectarea sistemului

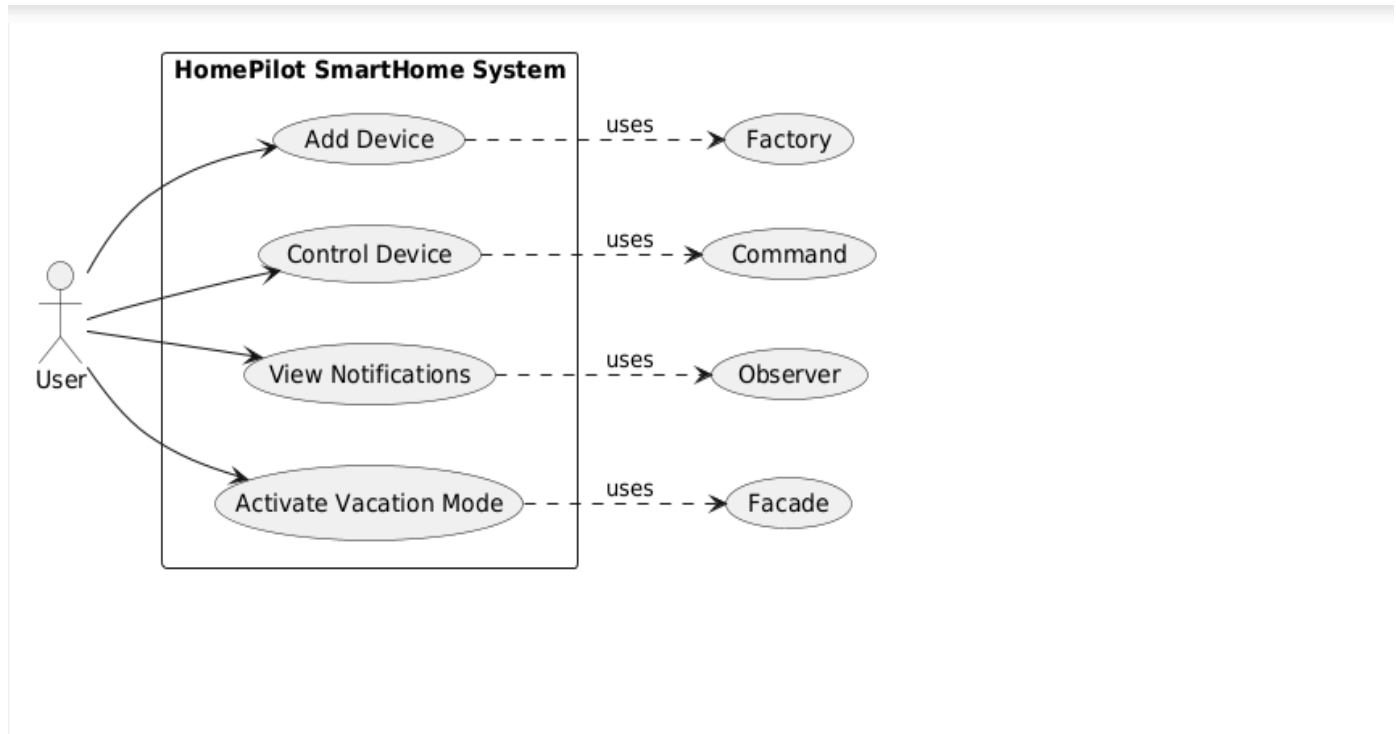


Figura 1 – Diagrama Use Case

Actor: Utilizator – interacționează cu aplicația prin linia de comandă (CLI).

Use case-uri:

- Aduă dispozitive – Crearea dispozitivelor smart folosind Factory Pattern.
- Controlează dispozitive – Executarea comenzilor asupra dispozitivelor prin Command Pattern.
- Vizualizează notificări – Primirea notificărilor de stare prin Observer Pattern.
- Activează modul Vacanță – Execută un scenariu complex printr-o interfață simplificată oferită de Facade Pattern.

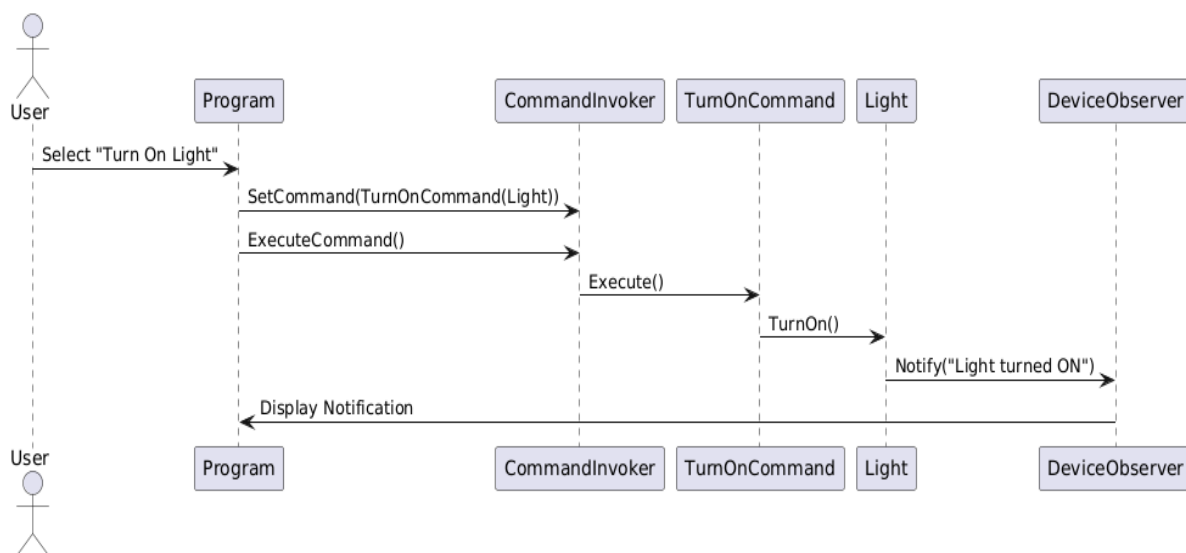


Figura 2 – Diagrama secvențial

Utilizatorul interacționează cu aplicația prin CLI (clasa Program).

Programul setează o comandă de tip TurnOnCommand prin intermediul CommandInvoker.

CommandInvoker execută comanda, care modifică starea dispozitivului Light.

Dispozitivul notifică observatorii despre schimbarea stării sale (aprins).

Observatorul afișează notificarea utilizatorului.

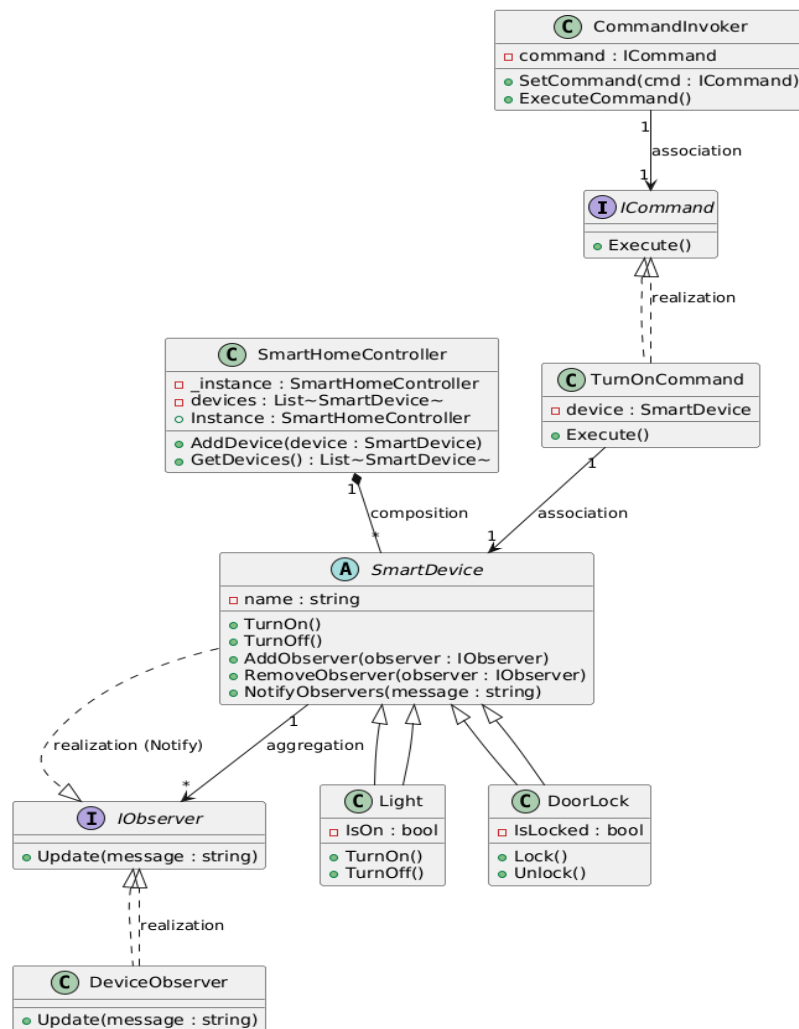


Figura 3 – Diagrama de clase

SmartDevice este clasa abstractă de bază pentru toate dispozitivele inteligente.

Light și DoorLock sunt implementări concrete ale dispozitivelor, fiecare cu propriile stări (aprins/stins, încuiat/descuiat).

DeviceFactory este responsabilă pentru crearea obiectelor SmartDevice fără ca restul aplicației să cunoască detalii concrete.

ICommand este interfața pentru comenzile executabile; TurnOnCommand este o implementare concretă care operează asupra unui SmartDevice.

CommandInvoker gestionează execuția comenzilor setate.

DeviceObserver implementează interfața IObserver și primește notificări de la dispozitivele smart.

SmartHomeController este Singleton și gestionează colecția globală de dispozitive.

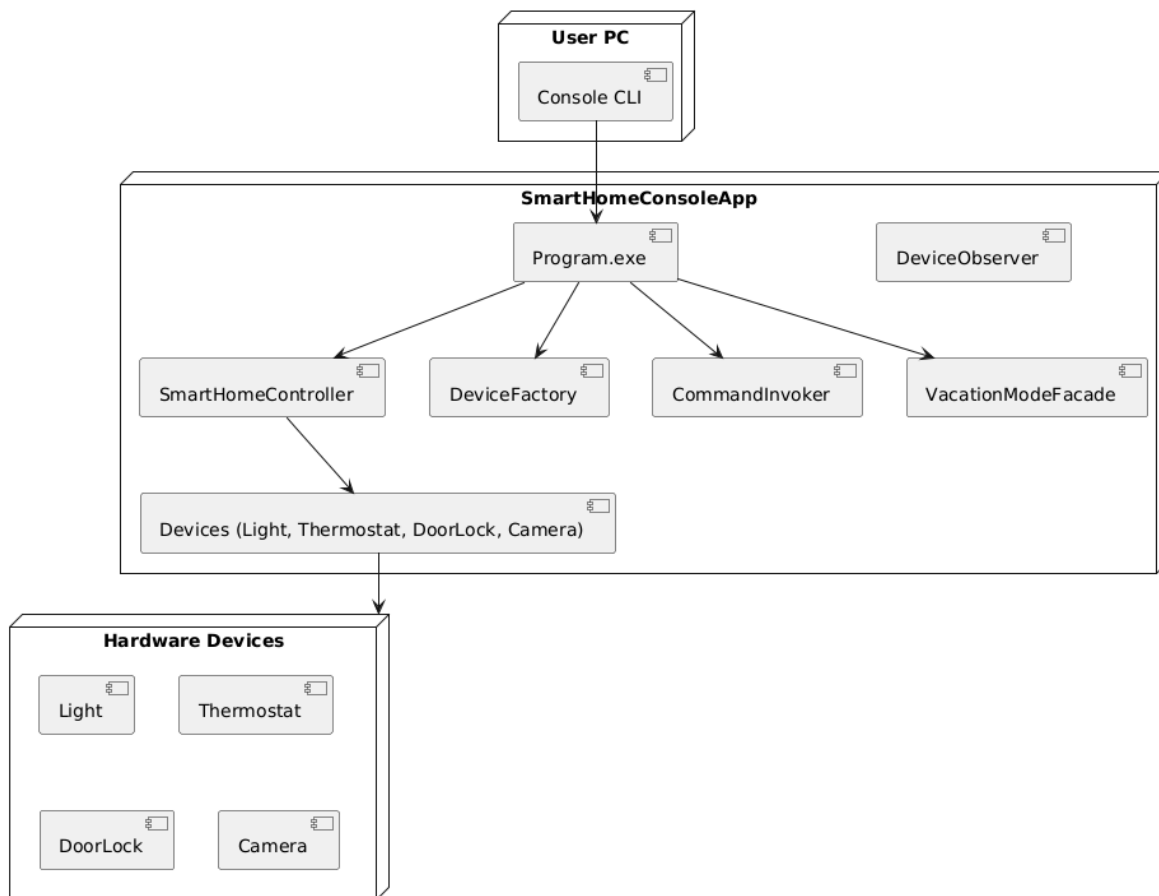


Figura 4 – Diagrama de deployment

User PC: Terminalul CLI de unde utilizatorul controlează aplicația.

SmartHomeConsoleApp: Codul executabil care include logica aplicației, Singleton-ul pentru controlul dispozitivelor, comenzi, observatori și factory.

Hardware Devices: Dispozitive fizice sau simulate (Light, DoorLock, Thermostat, Camera) ce sunt controlate prin aplicație.

Fluxul de comunicare începe de la consola utilizatorului, trece prin logica aplicației și ajunge la dispozitivele inteligente gestionate de controller.

Implementarea

Aplicația HomePilot a fost implementată folosind limbajul C#, în cadrul platformei .NET, fiind structurată modular și orientată pe obiecte. Scopul principal a fost integrarea a șase șabloane de proiectare fundamentale: Factory, Singleton, Decorator, Observer, Command și Facade, pentru a evidenția rolul fiecăruia în arhitectura generală a aplicației.

Aplicația rulează ca o aplicație console CLI, permițând utilizatorului să controleze dispozitive inteligente (lumini, termostate, încuietori) prin intermediul unui meniu interactiv. După executarea comenzilor, utilizatorul primește notificări în timp real despre modificările stărilor dispozitivelor.

Structura generală

Proiectul este organizat pe module clare, fiecare responsabil pentru o anumită funcționalitate a aplicației:

Devices/ – conține clasele pentru dispozitive inteligente: Light, Thermostat, DoorLock, Camera și clase abstracte/interfaces de bază.

Command/ – conține implementările comenzilor (ex: TurnOnCommand, LockDoorCommand) și invocatorul de comenzi.

Observer/ – conține interfețele pentru observatori și implementarea observatorului care afișează notificările.

Singleton/ – implementează clasa SmartHomeController, un Singleton care gestionează toate dispozitivele.

Factory/ – clasa DeviceFactory care creează dispozitivele în funcție de tip.

Facade/ – clasa VacationModeFacade care grupează operații complexe într-o comandă simplificată.

Program.cs – interfața principală CLI și fluxul principal de execuție..

Utilizarea șabloanelor de proiectare

Factory Pattern

Pentru a crea instanțe de dispozitive smart fără a expune clasele concrete, a fost implementată clasa DeviceFactory. Aceasta primește un tip de dispozitiv ca string și returnează o instanță corespunzătoare (ex: Light, Termostat). Astfel, logica de creare este centralizată și extensibilă.

Avantaje:

- Separarea responsabilităților între client și creare.
- Permite extinderea cu noi dispozitive fără modificarea codului client.
- Centralizarea logicii de creare.

Dezavantaje:

- Necesită actualizarea fabricii la adăugarea unui dispozitiv nou.
- Folosirea stringurilor poate duce la erori la runtime.

Singleton Pattern

Clasa SmartHomeController este implementată ca Singleton, asigurând o singură instanță care controlează toate dispozitivele smart. Aceasta oferă un punct global de acces și gestionează centralizat lista și starea dispozitivelor.

Avantaje:

- Acces global controlat și consistent.
- Evitarea duplicării instanțelor.
- Simplifică gestionarea stărilor dispozitivelor.

Dezavantaje:

- Testare dificilă în unit testing (dificultăți la mock).
- Poate ascunde dependențe.
- Risc de creștere excesivă a responsabilităților.

Decorator Pattern

Pentru a adăuga funcționalități suplimentare la dispozitive fără a le modifica (ex: vedere nocturnă la camere), s-a aplicat șablonul Decorator. Clasa CameraWithNightVision extinde funcționalitatea clasei Camera.

Avantaje:

- Flexibilitate mare în combinarea funcționalităților.
- Respectă principiul OCP (open/closed principle).
- Nu modifică clasele de bază.

Dezavantaje:

- Complexitate în urmărirea straturilor de decoratori.
- Debugging mai dificil.

Observer Pattern

Dispozitivele inteligente implementează interfața IObservable pentru a notifica automat observatorii (DeviceObserver) când starea lor se modifică (ex: aprindere lumină, blocare ușă).

Avantaje:

- Decuplare între emitent și observator.
- Scalabilitate pentru adăugarea de noi observatori.
- Extensibilitate ușoară.

Dezavantaje:

- Ordinea notificărilor nu este garantată.
- Debugging poate fi dificil.
- Pot apărea scurgeri de memorie dacă observatorii nu sunt dezabonați.

Command Pattern

Comenzile asupra dispozitivelor sunt încapsulate în obiecte de tip comandă (ex: TurnOnCommand, LockDoorCommand), iar executarea lor este gestionată de invocatorul de comenzi (CommandInvoker). Această structură asigură un mod flexibil de a administra acțiunile asupra dispozitivelor, facilitând extinderea sistemului cu comenzi noi sau funcții adiționale precum undo/redo.

Avantaje:

- Încapsularea acțiunilor ca obiecte independente.
- Facilitează adăugarea de noi comenzi fără modificări majore.
- Poate permite implementarea unor funcționalități complexe legate de execuția comenzilor.

Facade Pattern

Pentru a simplifica interacțiunea utilizatorului cu sistemul, clasa VacationModeFacade oferă o interfață unificată ce grupează mai multe operații complexe (ex: stingerea tuturor luminilor, reglarea termostatlui, blocarea ușilor) într-un singur apel, facilitând astfel utilizarea scenariilor avansate fără a expune complexitatea internă.

Avantaje:

- Oferă o interfață simplificată pentru funcționalități complexe.
- Reduce cuplarea dintre subsisteme și utilizatori.
- Crește lizibilitatea și simplitatea codului client.

Importanța utilizării șabloanelor de proiectare în cod

În dezvoltarea software-ului modern, complexitatea sistemelor este în continuă creștere, iar menținerea clarității, reutilizabilității și scalabilității codului devine o prioritate esențială. În acest context, șabloanele de proiectare (design patterns) oferă soluții testate, standardizate și general acceptate pentru problemele recurente întâlnite în procesul de dezvoltare software. Ele reprezintă un set de bune practici ce facilitează nu doar procesul de scriere a codului, ci și îmbunătățirea calității acestuia și colaborarea eficientă în echipe de dezvoltare.

Un prim beneficiu esențial al utilizării șabloanelor de proiectare este reutilizarea logicii consacrate. În loc să se reinventeze soluții pentru probleme comune, cum ar fi crearea obiectelor, gestionarea stărilor sau comunicarea între componente, șabloanele oferă structuri clare și eficiente care pot fi aplicate imediat. De exemplu, implementarea șablonului Factory pentru crearea dispozitivelor smart permite dezvoltatorilor să decupleze codul client de clasele concrete, ceea ce facilitează extinderea și întreținerea sistemului fără a afecta componentele deja funcționale.

În al doilea rând, șabloanele contribuie semnificativ la îmbunătățirea arhitecturii aplicației. Prin aplicarea unor pattern-uri precum Strategy, Observer sau Decorator, logica aplicației devine modulară, flexibilă și ușor de extins. Spre exemplu, logica de calcul sau comportamentul din cadrul unei aplicații poate fi modificat sau înlocuit dinamic, fără a interveni în structura de bază a codului. Acest aspect este critic pentru adaptabilitatea software-ului la cerințe noi sau în evoluție, făcând sistemul mai robust și

pregătit pentru schimbări pe termen lung.

Un alt aspect important este facilitarea colaborării și comunicării în cadrul echipei de dezvoltare. Atunci când o echipă utilizează șabloane standardizate, membrii pot înțelege rapid structura și comportamentul sistemului, chiar dacă nu au fost implicați în dezvoltarea inițială. Termeni precum Singleton, Factory, Decorator sau Observer devin un limbaj comun în echipă, reducând timpul necesar pentru onboarding și crescând eficiența schimbului de informații tehnice. Mai mult, aceste șabloane contribuie la o documentație implicită a arhitecturii, facilitând mentenanța și extinderea codului.

Utilizarea șabloanelor are, de asemenea, un impact major asupra testabilității și mentenanței codului. Separarea clară a responsabilităților și definirea unor interfețe bine structurate fac posibilă testarea unităților individuale izolat. Astfel, componentele precum un Observer sau o strategie de calcul pot fi validate independent, fără a necesita rularea întregii aplicații, ceea ce reduce timpul și costurile asociate testării și scade riscul apariției defectelor în producție.

Totuși, aplicarea unui șablon de design trebuie să fie făcută cu atenție și în contextul potrivit. Utilizarea nejustificată sau excesivă a șabloanelor poate complica inutil arhitectura și îngreuna înțelegerea codului. De aceea, un dezvoltator experimentat trebuie să cunoască nu doar funcționalitatea fiecărui șablon, ci și să înțeleagă când și de ce să-l aplice pentru a maximiza beneficiile și a evita problemele.

În concluzie, șabloanele de proiectare reprezintă un instrument esențial în arsenalul dezvoltatorului modern, iar aplicarea lor corectă contribuie la crearea unor aplicații robuste, scalabile și ușor de întreținut, cum este și cazul proiectului HomePilot.

Interacțiunea cu utilizatorul

Aplicația folosește biblioteca *readline-sync* pentru a primi input de la utilizator în terminal. Acesta poate selecta băutura dorită, toppingurile, tipul de preț și poate vizualiza rezultatul comenzii.

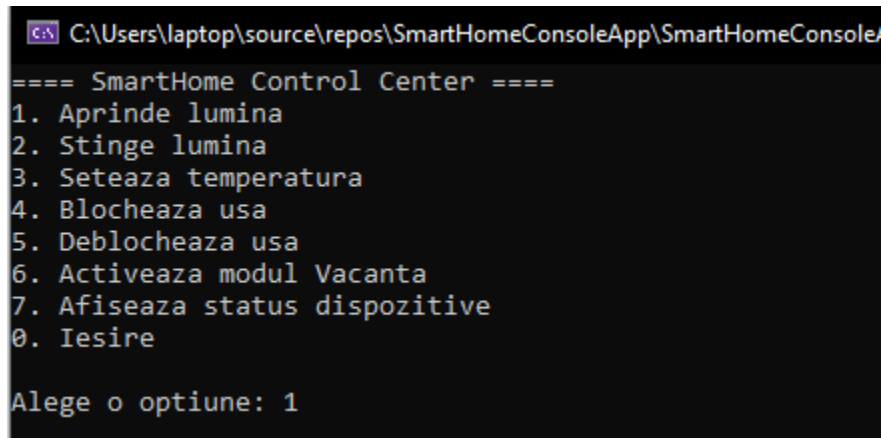
A screenshot of a terminal window with a black background and green text. The window title is 'C:\Users\laptop\source\repos\SmartHomeConsoleApp\SmartHomeConsoleApp'. The content shows a menu titled '==== SmartHome Control Center ===='. The menu items are: '1. Aprinde lumina', '2. Stingea lumina', '3. Seteaza temperatura', '4. Blocheaza usa', '5. Deblocheaza usa', '6. Activeaza modul Vacanta', '7. Afiseaza status dispozitive', and '0. Iesire'. At the bottom, it says 'Alege o optiune: 1'.

Figura 5 – Input de la utilizator

Secvențe de cod

```
public SmartDevice CreateDevice(string type, string name)
{
    switch (type.ToLower())
    {
        case "light": return new Light(name);
        case "thermostat": return new Thermostat(name);
        case "camera": return new Camera(name);
        case "doorlock": return new DoorLock(name);
        default: throw new ArgumentException("Invalid device type");
    }
}
```

Secvența 1 – Factory pattern

Acest șablon abstractizează procesul de creare a dispozitivelor smart, separând codul client de clasele concrete.


```

public sealed class SmartHomeController
{
    private static readonly SmartHomeController instance = new SmartHomeController();
    private List<SmartDevice> devices = new List<SmartDevice>();

    private SmartHomeController() { }

    public static SmartHomeController Instance => instance;

    public void AddDevice(SmartDevice device) => devices.Add(device);
}

```

Secvența 2 – Singleton pattern

Asigură o instanță unică responsabilă de gestionarea tuturor dispozitivelor, oferind un punct global de acces..

```

public class CameraWithNightVision : DeviceDecorator
{
    public CameraWithNightVision(Camera camera) : base(camera) { }

    public void EnableNightVision()
    {
        NotifyObservers($"{Name}: Night vision enabled.");
    }
}

```

Secvența 3 – Decorator pattern

Permite extinderea dinamică a funcționalităților unui dispozitiv fără a modifica clasa de bază.

```

public class TurnOnCommand : ICommand
{
    private SmartDevice device;

    public TurnOnCommand(SmartDevice device) => this.device = device;

    public void Execute() => device.TurnOn();
}

```

Secvența 4 – Command pattern

Încapsulează acțiunile asupra dispozitivelor pentru o execuție flexibilă și organizată.

```

public class DeviceObserver : IObserver
{
    public void Update(string message)
    {
        Console.WriteLine($"[Notification] {message}");
    }
}

```

Secvența 5 – Observer pattern

Notifică utilizatorul sau alte componente când starea dispozitivelor se schimbă.

```

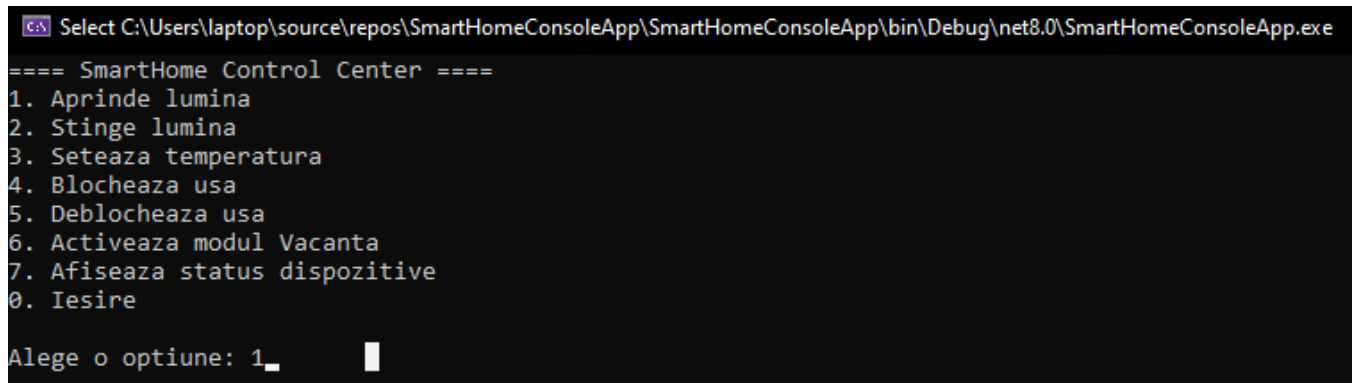
public void ActivateVacationMode()
{
    foreach (var device in
_controller.GetDevices())
    {
        if (device is Light)
            device.TurnOff();
        else if (device is Thermostat)
            ((Thermostat)device).SetTemperature(16);
        else if (device is DoorLock)
            ((DoorLock)device).Lock();
    }
}

```

Secvența 6 – Facade Pattern

Simplifică gestionarea unui scenariu complex într-o singură comandă.

Documentarea Produsului



```
C:\> Select C:\Users\laptop\source\repos\SmartHomeConsoleApp\SmartHomeConsoleApp\bin\Debug\net8.0\SmartHomeConsoleApp.exe

==== SmartHome Control Center ====
1. Aprinde lumina
2. Stinge lumina
3. Seteaza temperatura
4. Blocheaza usa
5. Deblocheaza usa
6. Activeaza modul Vacanta
7. Afiseaza status dispozitive
0. Iesire

Alege o optiune: 1_
```

Figura 6 – Meniul Inițial

Meniul principal afișat în consola aplicației, unde utilizatorul poate selecta opțiunile disponibile pentru controlul dispozitivelor inteligente.

```
C:\Users\laptop\source\repos\SmartHomeConsoleApp\SmartHomeConsoleApp\bin\Debug
3. Seteaza temperatura
4. Blocheaza usa
5. Deblocheaza usa
6. Activeaza modul Vacanta
7. Afiseaza status dispozitive
0. Iesire

Alege o optiune: 1
[NOTIFICARE]: Living Room Light was turned ON.

Alege o optiune: 1
[NOTIFICARE]: Living Room Light: The light is already ON!

Alege o optiune: 2
[NOTIFICARE]: Living Room Light was turned OFF.

Alege o optiune: 2
[NOTIFICARE]: Living Room Light: The light is already OFF!

Alege o optiune: 3
Seteaza temperatura (°C): 24
[NOTIFICARE]: Hall Thermostat: Temperature set to 24°C.

Alege o optiune: 5
[NOTIFICARE]: Front Door Lock: The door is already unlocked!

Alege o optiune: 4
[NOTIFICARE]: Front Door Lock is now LOCKED.

Alege o optiune: _
```

Figura 7 – Controlul dispozitivelor

Exemplu de interacțiune cu dispozitivele: aprinderea luminii, setarea temperaturii și blocarea ușii, cu feedback vizual imediat.

```
Alege o optiune: 6
[NOTIFICARE]: Living Room Light: The light is already OFF!
[NOTIFICARE]: Hall Thermostat: Temperature set to 16°C.
[NOTIFICARE]: Front Door Lock: The door is already locked!
SmartHome: Vacation Mode ACTIVATED.

Alege o optiune: _
```

Figura 8 – Activarea modului Vacanță

Notificările afișate în consolă în timpul activării scenariului complex „Mod Vacanță”, care gestionează simultan mai multe dispozitive.

```
Alege o optiune: 1
[NOTIFICARE]: Living Room Light was turned ON.

Alege o optiune: 3
Seteaza temperatura (°C): 24
[NOTIFICARE]: Hall Thermostat: Temperature set to 24°C.

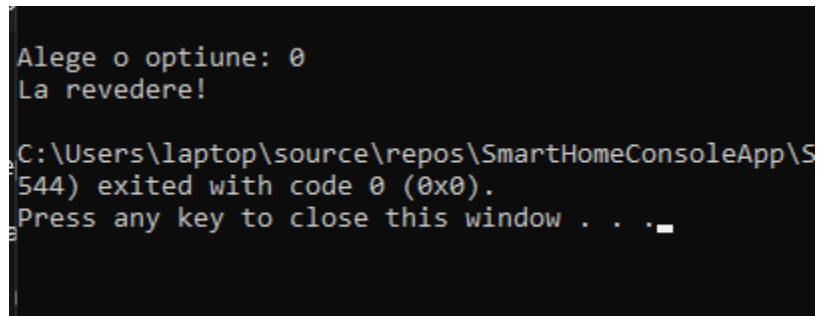
Alege o optiune: 4
[NOTIFICARE]: Front Door Lock is now LOCKED.

Alege o optiune: 7

--- Status dispozitive ---
Light: Living Room Light - ON
Termostat: Hall Thermostat - 24°C
Door: Front Door Lock - Locked
```

Figura 9 – Statusul dispozitivelor

După o comandă, putem imediat să inițializăm comanda următoare, observăm în figura 9 că prețul produsului depinde de adaosuri și de reducerea aplicată.

A screenshot of a Windows command prompt window with a black background and white text. The text displayed is: 'Alege o optiune: 0', 'La revedere!', 'C:\Users\laptop\source\repos\SmartHomeConsoleApp\SmartHomeConsoleApp.exe (544) exited with code 0 (0x0).', and 'Press any key to close this window . . .'. A cursor is visible at the end of the last line.

```
Alege o optiune: 0
La revedere!

C:\Users\laptop\source\repos\SmartHomeConsoleApp\SmartHomeConsoleApp.exe (544) exited with code 0 (0x0).
Press any key to close this window . . .
```

Figura 10 – Ieșirea din aplicație

Mesajul de încheiere a aplicației, afișat după ce utilizatorul alege să părăsească programul.

Concluzie

Proiectul realizat demonstrează importanța și utilitatea șabloanelor de proiectare în dezvoltarea aplicațiilor software moderne. Prin aplicarea practică a șase șabloane – Factory, Singleton, Decorator, Observer, Command și Facade – aplicația pentru gestionarea dispozitivelor inteligente într-o locuință a fost construită într-un mod modular, extensibil și ușor de întreținut.

Folosirea acestor șabloane a permis separarea clară a responsabilităților și evitarea duplicării codului, facilitând astfel adăugarea de noi funcționalități fără a afecta componentele existente. De exemplu, noi tipuri de dispozitive sau funcționalități pot fi integrate fără a modifica logica de bază, iar scenariile complexe pot fi gestionate simplificat printr-o interfață unificată.

În plus, șablonul Observer a oferit o soluție elegantă pentru notificarea automată a componentelor interesate de schimbările stării dispozitivelor, demonstrând beneficiile unei arhitecturi reactive și decuplate.

Aplicația poate fi extinsă cu ușurință pentru a include funcționalități suplimentare, precum integrarea cu o bază de date reală, dezvoltarea unei interfețe grafice sau adăugarea unor funcții avansate de raportare, păstrând în același timp claritatea și flexibilitatea arhitecturii existente.

În concluzie, proiectul HomePilot nu doar că îndeplinește cerințele tehnice ale temei propuse, dar oferă și un exemplu concret al modului în care șabloanele de proiectare pot îmbunătăți semnificativ calitatea codului și eficiența procesului de dezvoltare software. Aplicația poate fi utilizată pentru a ușura gestionarea locuințelor inteligente, facilitând o administrare simplă și eficientă a dispozitivelor casnice.

Anexa 1

<https://github.com/444real/HomePilot---Proiect-de-an-Tmpp.git> - repozitoriul cu aplicația creată însoțită de un ReadME file.