Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Кафедра інформаційних систем та технологій

Лабораторна робота №6

з дисципліни «Технології розроблення програмного забезпечення»

Тема: «Патерни проектування»

Виконав:
студент групи ІА-34
Тунік О.

Перевірив:
асистент кафедри ІСТ
Мягкий М.Ю.

Київ 2025

Зміст

Мета: Вивчити структуру шаблонів «Abstract Factory», «Factory Method», «Memento», «Observer», «Decorator» та навчитися застосовувати їх в реалізації програмної системи.

Теоретичні відомості

Шаблон «Abstract Factory»

Патерн «Абстрактна фабрика» дозволяє створювати сімейства взаємопов'язаних об'єктів без вказівки їх конкретних класів. Він виділяє загальний інтерфейс фабрики та конкретні реалізації для різних сімейств продуктів, що забезпечує узгодженість і взаємозамінність об'єктів. Прикладом є ADO.NET, де DbProviderFactory створює об'єкти під різні бази даних. Патерн спрощує розширення системи новими сімействами продуктів, хоча додавання нового типу продукту вимагає змін у всіх фабриках.

Шаблон «Factory Method»

Фабричний метод визначає інтерфейс для створення об'єктів певного базового типу і дозволяє підставляти підкласи замість базових класів. Це дозволяє розширювати поведінку системи без зміни існуючого коду. Патерн корисний, коли потрібно делегувати створення об'єктів підкласам, наприклад для різних мережевих пакетів (TcpPacket, UdpPacket). Мінусом є можливе ускладнення ієрархії класів при великій кількості продуктів.

Шаблон «Memento»

Патерн «Мементо» зберігає і відновлює стан об'єкта без порушення інкапсуляції. Originator створює об'єкт Memento для збереження свого стану, а Caretaker зберігає його та передає за потреби. Це дозволяє реалізувати скасовні дії, спрощує структуру об'єктів і не порушує інкапсуляцію. Недоліком є підвищене споживання пам'яті при великій кількості знімків стану.

Шаблон «Observer»

Патерн «Спостерігач» визначає залежність «один-до-багатьох», коли зміни стану одного об'єкта автоматично повідомляються всім зацікавленим. Він використовується, наприклад, у підписках на оновлення даних, MVVM або системах сповіщень. Серед переваг слабке зв'язування, можливість динамічного додавання/видалення спостерігачів і підтримка паралельної обробки. Серед недоліків – відсутність контролю над порядком повідомлень.

Шаблон «Decorator»

Патерн «Декоратор» дозволяє динамічно додавати об'єктам нову поведінку під час виконання програми. Він обгортає початковий об'єкт через агрегацію, зберігаючи його базові функції, і додає нові обов'язки. Приклад: додавання смуги прокрутки до візуальних елементів. Головні переваги це гнучкість і можливість

комбінувати дрібні об’єкти. Недоліки – складність конфігурації при багатошарових обгортках і велика кількість класів.

Хід роботи

Поставлене завдання

Flexible automation tool (strategy, command, abstract factory, facade, interpreter, SOA)

Інструмент автоматизації повинен забезпечувати найпростіші автоматичні дії для зручності користувача: завантаження нових фільмів / книг / файлів при випуску (наприклад, щоп'ятниці з'являються нові серії улюблених серіалів); встановити статуси в комунікаторах (skype – away при нульовій активності на тривалий час) і т.д. Автоматизація забезпечується шляхом введення правил (на зразок IFTTT.com сервісу), запису макросів (натискання клавіш, дії миші), планувальника завдань (о 5 ранку – початок роздачі торрент-файлів).

Реалізація паттерну

У системі автоматизації FlexibleAutomationTool реалізовали породжувальний патерн проектування «Абстрактна фабрика» (Abstract Factory) з метою відокремлення логіки створення платформно-залежних сервісів від логіки їх використання та забезпечення можливості заміни цілого сімейства таких сервісів без змін у ядрі застосунку. Використання цього патерну дозволяє працювати з платформними можливостями (діалоги, буфер обміну, вибір файлів, системний трей) виключно через абстракції, не прив’язуючись до конкретної UI-технології.

Патерн «Абстрактна фабрика» у даній архітектурі складається з ключових елементів: абстрактної фабрики (IPlatformFactory), абстрактних продуктів (інтерфейси платформних сервісів), конкретної фабрики (WinFormsPlatformFactory), конкретних продуктів (WinForms-реалізації сервісів) та клієнтів, які використовують створені об’єкти. Спроєктовану діаграму класів, що ілюструє використання патерну «Абстрактна фабрика», наведено на рис. 1.

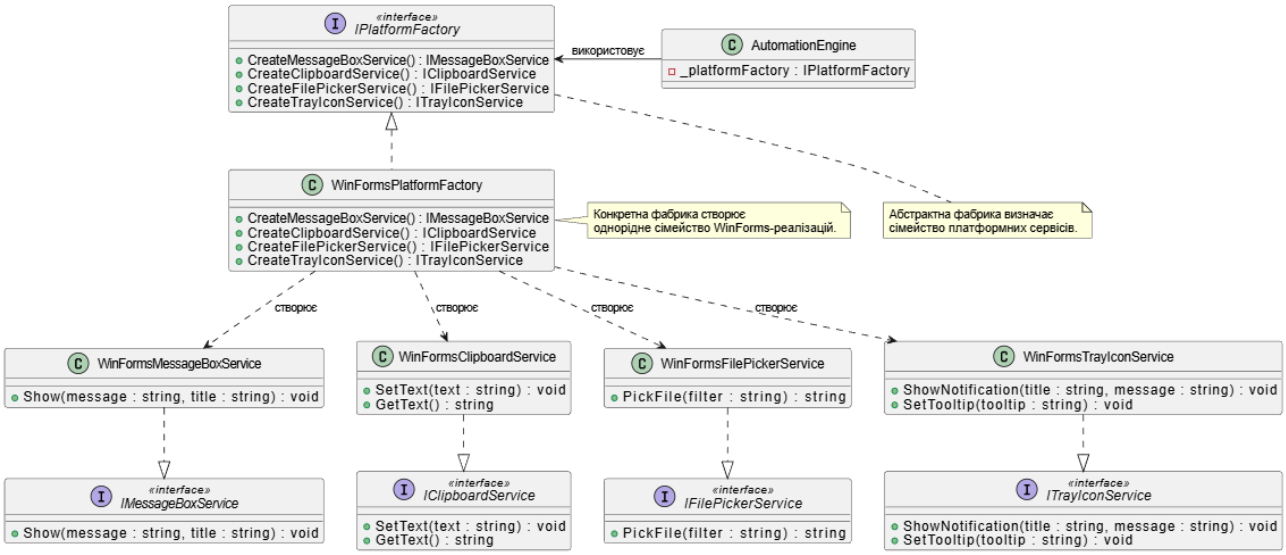


Рисунок 1. Діаграма класів паттерну Abstract Factory

Центральним елементом патерну є інтерфейс IPlatformFactory, розташований у ядрі системи. Він визначає єдиний контракт для створення всього сімейства платформних сервісів і містить методи CreateMessageBoxService(), CreateClipboardService(), CreateFilePickerService() та CreateTrayIconService(). Такий підхід гарантує, що всі сервіси, створені однією фабрикою, є узгодженими між собою та належать до однієї платформи, що повністю відповідає ідеї Abstract Factory.

Абстрактні продукти представлені інтерфейсами IMessageBoxService, IClipboardService, IFilePickerService та ITrayIconService, які описують доступні операції для роботи з відповідними платформними можливостями. Клієнтський код взаємодіє виключно з цими інтерфейсами, що дозволяє приховати деталі конкретної реалізації та забезпечити слабке зв’язування між компонентами системи.

Конкретною реалізацією абстрактної фабрики є клас WinFormsPlatformFactory, який створює WinForms-орієнтовані реалізації сервісів: WinFormsMessageBoxService, WinFormsClipboardService, WinFormsFilePickerService та WinFormsTrayIconService. Усі ці класи інкапсулюють WinForms-специфічну логіку та утворюють єдине сімейство продуктів. Додатково фабрика реєструє створені сервіси у ServiceManager, що дозволяє використовувати їх у різних частинах UI через механізм ін’єкції залежностей, не порушуючи при цьому концепцію патерну.

Клієнтом абстрактної фабрики в ядрі системи виступає AutomationEngine, який має залежність від IPlatformFactory, що передається йому під час ініціалізації. Під час виконання правил автоматизації Engine використовує фабрику для отримання необхідних платформних сервісів і передає їх у відповідні дії, наприклад

у MessageBoxAction. При цьому Engine не має жодної інформації про конкретну платформу, що відповідає принципу інверсії залежностей.

UI-компоненти системи, такі як MainForm, CreateRuleForm та EditRuleForm, зазвичай отримують IMessageBoxService напряму через DI, але за потреби можуть використовувати й інші сервіси, створені тією ж фабрикою. Завдяки цьому як ядро системи, так і користувацький інтерфейс залишаються незалежними від конкретної реалізації платформи.

У результаті використання патерну «Абстрактна фабрика» в FlexibleAutomationTool забезпечує узгодженість платформних сервісів, спрощує розширення системи новими UI-платформами (WPF, Console, тестові реалізації) та робить архітектуру більш гнучкою й структурованою. Водночас реалізація зберігає класичний недолік патерну: додавання нового типу продукту потребує розширення інтерфейсу фабрики та всіх її реалізацій, однак цей компроміс є виправданим з огляду на отримані переваги.

Фрагменти коду реалізації

Інтерфейс абстрактної фабрики IPlatformFactory визначає контракт для створення всього сімейства платформних сервісів. Клієнт працює лише через цей інтерфейс і не знає про конкретну реалізацію сервісів:

```
namespace FlexibleAutomationTool.Core.Factories
{
    public interface IPlatformFactory
    {
        IMessageBoxService CreateMessageBoxService();
        IClipboardService CreateClipboardService();
        IFilePickerService CreateFilePickerService();
        ITrayIconService CreateTrayIconService();
    }
}
```

Інтерфейс одного з продуктів IMessageBoxService. Він описує операції, які будь-який сервіс діалогового вікна/повідомлення повинен реалізувати. Клієнт може використовувати цей сервіс через інтерфейс, не знаючи, що конкретно робить WinForms чи інша реалізація:

```
namespace FlexibleAutomationTool.Core.Interfaces
{
    public interface IMessageBoxService
    {
        void Show(string message, string? title = null);
    }
}
```

Конкретна фабрика для WinForms WinFormsPlatformFactory. Реалізує IPlatformFactory та повертає конкретні продукти, усі вони належать до одного сімейства – WinForms. Реєстрація сервісів у ServiceManager забезпечує доступ до них у будь-якій частині UI:

```
namespace FlexibleAutomationTool.UI.Services
{
    public class WinFormsPlatformFactory : IPlatformFactory
    {
        private readonly IMessageBoxService _msgService;
        private readonly IClipboardService _clipboard;
        private readonly IFilePickerService _filePicker;
        private readonly ITrayIconService _trayIcon;
        private readonly ServiceManager _svcManager;

        public WinFormsPlatformFactory(IMessageBoxService msgService, ServiceManager
svcManager)
        {
            _msgService = msgService;
            _svcManager = svcManager;

            _clipboard = new WinFormsClipboardService();
            _filePicker = new WinFormsFilePickerService();
            _trayIcon = new WinFormsTrayIconService();

            _svcManager.Register(nameof(IMessageBoxService), _msgService);
            _svcManager.Register(nameof(IClipboardService), _clipboard);
            _svcManager.Register(nameof(IFilePickerService), _filePicker);
            _svcManager.Register(nameof(ITrayIconService), _trayIcon);
        }

        public IMessageBoxService CreateMessageBoxService() => _msgService;
        public IClipboardService CreateClipboardService() => _clipboard;
        public IFilePickerService CreateFilePickerService() => _filePicker;
        public ITrayIconService CreateTrayIconService() => _trayIcon;
    }
}
```

Конкретна реалізація сервісу WinFormsMessageBoxService. Реалізація сервісу показання вікна повідомлення для WinForms. Клас інкапсулює платформно-залежну логіку, клієнт працює лише через інтерфейс IMessageBoxService:

```
namespace FlexibleAutomationTool.UI.Services
{
    public class WinFormsMessageBoxService : IMessageBoxService
    {
        public void Show(string message, string? title = null)
        {
            MessageBox.Show(message, title ?? "Info");
        }
    }
}
```

Приклад використання фабрики у клієнті. AutomationEngine, або UI-компонент, можуть отримати сервіси через фабрику. Клієнт не знає про WinForms – він працює лише з інтерфейсами:

```
private readonly Factories.IPlatformFactory _factory;
var msgService = _factory.CreateMessageBoxService();
foreach (var a in actionList)
{
    if (a is MessageBoxAction mba)
    {
        mba.MessageBoxService = msgService;
    }
}
```

Обґрунтування використання паттерну

Використання патерну «Абстрактна фабрика» (Abstract Factory) у системі FlexibleAutomationTool є обґрунтованим з огляду на необхідність гнучкого та узгодженого створення платформно-залежних сервісів. У реальних сценаріях використання системи потрібна робота з різними елементами платформи – від відображення повідомлень та роботи з буфером обміну до вибору файлів і взаємодії з системним треєм. Без застосування патерну Abstract Factory логіка створення цих сервісів була б розкидана по різних частинах системи, що призвело б до дублювання коду, жорсткої прив’язки UI-компонентів до конкретної платформи та ускладнення підтримки системи.

Патерн Abstract Factory дозволяє інкапсулювати створення всього сімейства платформних сервісів у єдиній абстрактній фабриці (IPlatformFactory), а конкретні фабрики (наприклад, WinFormsPlatformFactory) реалізують ці сервіси для певної платформи. Клієнти, такі як AutomationEngine або UI-компоненти, працюють виключно через інтерфейси сервісів і не залежать від конкретної реалізації. Такий підхід відповідає принципу інверсії залежностей та зменшує зв’язність між компонентами системи.

Застосування патерну також дозволяє чітко розділити відповідальності між елементами архітектури. Фабрика відповідає лише за створення об’єктів, сервіси інкапсулюють платформно-залежну логіку, а клієнти використовують сервіси через абстракції. Наприклад, WinFormsClipboardService реалізує роботу з буфером обміну лише для WinForms, а клієнт працює через інтерфейс IClipboardService, що дозволяє легко замінити реалізацію на WPF або тестовий stub без зміни коду клієнтів.

Важливою перевагою використання Abstract Factory є можливість забезпечити узгодженість усіх сервісів у межах одного сімейства: всі об’єкти, створені конкретною фабрикою, належать до однієї платформи, що зменшує ризик помилок взаємодії між сервісами. Крім того, патерн спрощує тестування системи, оскільки фабрика дозволяє підставляти мок-реалізації сервісів, не змінюючи бізнес-логіку Engine або UI-компонентів.

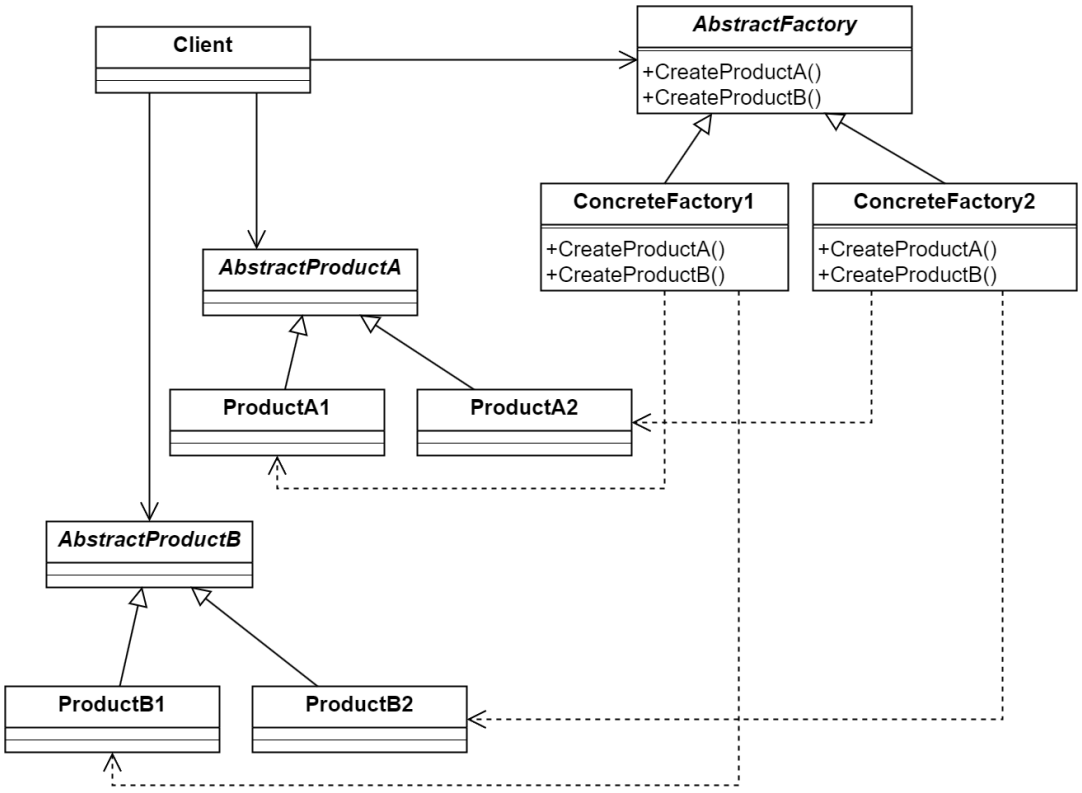
Порівняно з альтернативними підходами, такими як створення сервісів безпосередньо у клієнтах або жорстка прив’язка до конкретних класів, використання Abstract Factory забезпечує більш чисту та масштабовану архітектуру. Додавання нового сімейства сервісів або нової платформи відбувається через реалізацію нової конкретної фабрики та сервісів, без зміни існуючого коду клієнтів, що зберігає принцип відкритості-закритості і підтримує модульність системи.

Відповіді на контрольні запитання

1. Яке призначення шаблону «Абстрактна фабрика»?

Створювати сімейства взаємопов'язаних об'єктів без вказівки їх конкретних класів.

2. Нарисуйте структуру шаблону «Абстрактна фабрика».



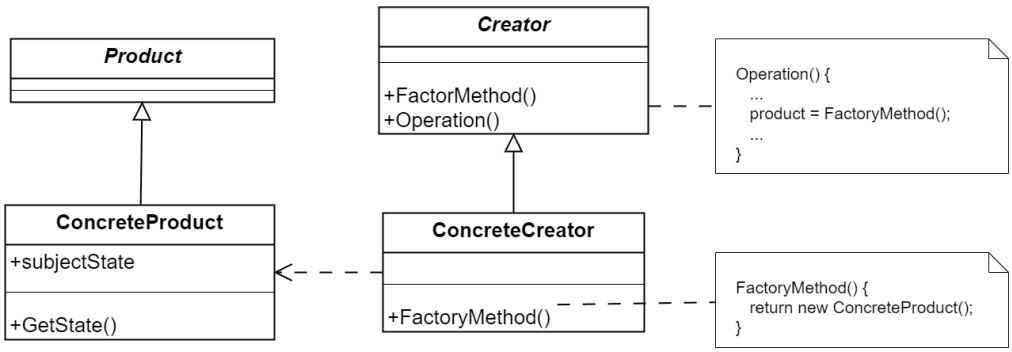
3. Які класи входять в шаблон «Абстрактна фабрика», та яка між ними взаємодія?

IPlatformFactory (абстрактна фабрика) створює інтерфейси продуктів (IMessageBoxService, IClipboardService тощо); конкретна фабрика реалізує їх і повертає клієнту через інтерфейси.

4. Яке призначення шаблону «Фабричний метод»?

Визначає інтерфейс для створення об'єктів певного типу, дозволяючи підставляти підкласи замість базового класу.

5. Нарисуйте структуру шаблону «Фабричний метод».



6. Які класи входять в шаблон «Фабричний метод», та яка між ними взаємодія?

Creator містить фабричний метод для створення продуктів, ConcreteCreator реалізує метод для конкретного продукту, клієнт використовує продукт через абстракцію Product.

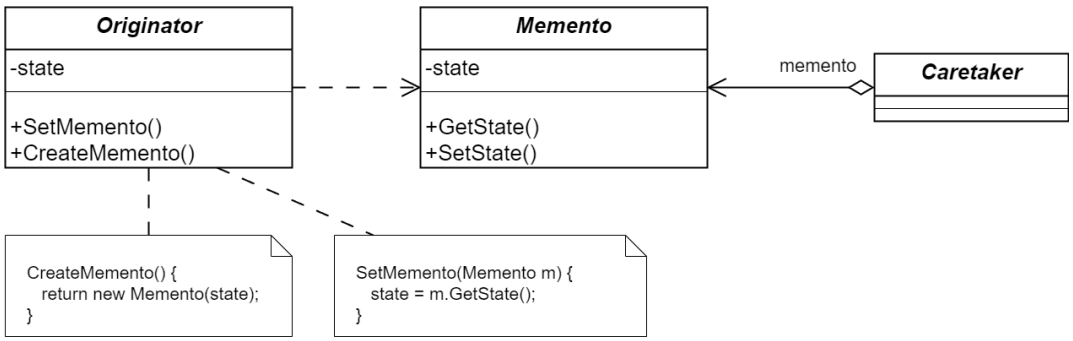
7. Чим відрізняється шаблон «Абстрактна фабрика» від «Фабричний метод»?

Abstract Factory створює ціле сімейство пов’язаних об’єктів, а Factory Method створює один об’єкт певного типу.

8. Яке призначення шаблону «Знімок»?

Зберігати і відновлювати стан об’єкта без порушення інкапсуляції.

9. Нарисуйте структуру шаблону «Знімок».



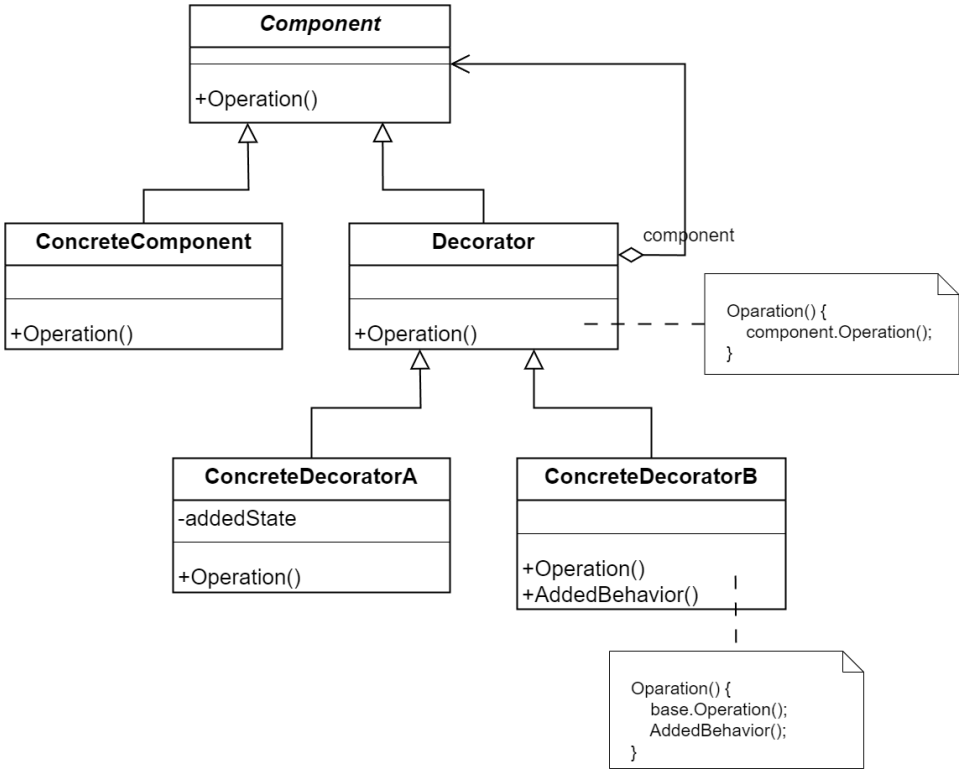
10. Які класи входять в шаблон «Знімок», та яка між ними взаємодія?

Originator містить стан, Memento зберігає стан, Caretaker управляє збереженням; клієнт через Caretaker може відновити стан Originator.

11. Яке призначення шаблону «Декоратор»?

Динамічно додавати об’єктам нову поведінку, обгортаючи їх без зміни базового класу.

12. Нарисуйте структуру шаблону «Декоратор».



13. Які класи входять в шаблон «Декоратор», та яка між ними взаємодія?

Component визначає інтерфейс, ConcreteComponent реалізує його, Decorator містить посилання на Component і розширює функціонал, ConcreteDecorator додає конкретну поведінку.

14. Які є обмеження використання шаблону «декоратор»?

Ускладнюється структура через велику кількість дрібних класів та важко конфігурувати багатошарові обгортки одночасно.

Висновки

У ході виконання лабораторної роботи розглянули та реалізовали породжувальний патерн проєктування «Абстрактна фабрика» (Abstract Factory) у системі автоматизації FlexibleAutomationTool. Використання цього патерну дозволило централізовано інкапсулювати логіку створення платформно-залежних сервісів і забезпечити можливість заміни цілого сімейства сервісів без змін у клієнтському коді. Реалізація з абстрактною фабрикою IPlatformFactory, конкретними фабриками (WinFormsPlatformFactory), абстрактними продуктами та їх конкретними реалізаціями демонструє чітке розділення відповідальностей між компонентами та слабке зв’язування клієнтів від конкретної платформи.

Аналіз результатів підтверджує ефективність застосування патерну Abstract Factory у випадках, коли необхідно працювати з різними платформними можливостями (повідомлення, буфер обміну, вибір файлів, системний трей) через абстракції. Патерн забезпечує узгодженість всіх сервісів одного сімейства, спрощує тестування системи за допомогою мок-реалізацій, а також дозволяє легко додавати нові платформи або розширювати існуючі без зміни бізнес-логіки. Загалом реалізація підтверджує доцільність використання Abstract Factory для створення модульних, масштабованих і зручних у супроводі програмних систем.