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

Лабораторна робота №5

Технології розроблення програмного забезпечення

«Шаблони «Adapter», «Builder», «Command», «Chain of Responsibility», «Prototype»»

Варіант 24

Виконав
студент групи ІА–33:
Вовчок М.М.

Перевірив
Мягкий М. Ю.

Київ — 2025

Тема

Flexible Automation Tool. Реалізація шаблону проектування Builder для побудови правил автоматизації.

Короткі теоретичні відомості

Шаблони проектування — це стандартизовані рішення типових задач, які часто виникають під час розробки програмного забезпечення. Вони дозволяють зробити код гнучкішим, розширюваним і зрозумілим, зменшити зв'язність між компонентами системи.

У даній лабораторній роботі розглядаються такі шаблони:

• Adapter — використовується для узгодження інтерфейсів. Дозволяє змусити працювати разом класи з несумісними інтерфейсами за рахунок створення проміжного адаптера.

• Builder — забезпечує поетапне створення складних об'єктів, відокремлюючи процес побудови від кінцевого представлення. Особливо корисний, коли об'єкт має багато параметрів або може існувати в різних конфігураціях.

• Command — інкапсулює запит у вигляді окремого об'єкта, що дозволяє ставити команди в чергу, логувати, скасовувати або повторно виконувати їх.

• Chain of Responsibility — організовує послідовну обробку запиту ланцюгом обробників, де кожен елемент сам вирішує, обробляти запит чи передавати далі.

• Prototype — забезпечує створення нових об'єктів шляхом копіювання існуючих (клонування), що корисно при дорогій ініціалізації об'єктів.

У реалізації варіанта 24 (Flexible Automation Tool) було обрано шаблон Builder, оскільки система працює з правилами автоматизації, які містять багато полів та можуть задаватися в різних конфігураціях (різні типи

тригерів, умов, дій, розкладів). Використання Builder дозволяє поетапно конструювати правило на основі вхідних параметрів користувача або конфігураційного файлу.

Хід роботи

Завдання

1. Ознайомитися з короткими теоретичними відомостями.
2. Реалізувати частину функціоналу системи Flexible Automation Tool у вигляді класів та їх взаємодії.
3. Застосувати один з розглянутих шаблонів (Builder) при реалізації програми.
4. Скласти звіт про виконану роботу.

Реалізація шаблону Builder

У системі Flexible Automation Tool користувач створює правила автоматизації, що складаються з назви, типу тригера, умов виконання, набору дій та розкладу. Такі правила можуть мати різну конфігурацію, тому доцільно застосувати шаблон Builder для поетапного конструювання об'єкта AutomationRule.

Було створено інтерфейс IAutomationRuleBuilder, який визначає кроки побудови правила: задання назви, тригера, умови, дій, розкладу та фінальне створення екземпляра AutomationRule. Конкретна реалізація SimpleAutomationRuleBuilder будує стандартне правило для запуску дій за розкладом або певною подією. Клас RuleDirector виступає в ролі "розпорядника" (director), який керує послідовністю виклику методів будівельника на основі вхідної DTO-моделі від користувача.

Інтерфейс IAutomationRuleBuilder

```
public interface IAutomationRuleBuilder
{
   IAutomationRuleBuilder SetName(string name);
   IAutomationRuleBuilder SetSchedule(string cronExpression);
   IAutomationRuleBuilder SetTrigger(TriggerType triggerType, string triggerData);
   IAutomationRuleBuilder SetCondition(string? expression);
   IAutomationRuleBuilder AddAction(ActionType actionType, string parameters);
```

```
    AutomationRule Build();
}
```

Допоміжні перерахування та клас AutomationRule

```
public enum TriggerType
{
    Schedule,
    Event
}

public enum ActionType
{
    DownloadFile,
    SetStatus,
    SendNotification
}

public class AutomationRule
{
    public int Id { get; set; }
    public string Name { get; set; } = string.Empty;
    public TriggerType TriggerType { get; set; }
    public string TriggerData { get; set; } = string.Empty;
    public string? ConditionExpression { get; set; }
    public List<(ActionType Type, string Parameters)> Actions { get; } = new();
    public string? ScheduleExpression { get; set; }
}
```

Конкретний будівельник SimpleAutomationRuleBuilder

```
public class SimpleAutomationRuleBuilder : IAutomationRuleBuilder
{
    private readonly AutomationRule _rule;

    public SimpleAutomationRuleBuilder()
    {
        _rule = new AutomationRule();
    }

    public IAutomationRuleBuilder SetName(string name)
    {
        _rule.Name = name;
        return this;
    }

    public IAutomationRuleBuilder SetSchedule(string cronExpression)
    {
        _rule.ScheduleExpression = cronExpression;
```

```
        return this;
    }

    public IAutomationRuleBuilder SetTrigger(TriggerType triggerType, string triggerData)
    {
        _rule.TriggerType = triggerType;
        _rule.TriggerData = triggerData;
        return this;
    }

    public IAutomationRuleBuilder SetCondition(string? expression)
    {
        _rule.ConditionExpression = expression;
        return this;
    }

    public IAutomationRuleBuilder AddAction(ActionType actionType, string parameters)
    {
        _rule.Actions.Add((actionType, parameters));
        return this;
    }

    public AutomationRule Build()
    {
            if (string.IsNullOrWhiteSpace(_rule.Name))
            throw new InvalidOperationException("Rule name must be specified");
        if (_rule.Actions.Count == 0)
            throw new InvalidOperationException("At least one action must be added");
        return _rule;
    }
}
```

Director (RuleDirector) та DTO-модель

```
public class AutomationRuleDto
{
    public string Name { get; set; } = string.Empty;
    public TriggerType TriggerType { get; set; }
    public string TriggerData { get; set; } = string.Empty;
    public string? ConditionExpression { get; set; }
    public string? ScheduleExpression { get; set; }
    public List<(ActionType Type, string Parameters)> Actions { get; set; } = new();
}

public class RuleDirector
{
    private readonly IAutomationRuleBuilder _builder;
```

```
    public RuleDirector(IAutomationRuleBuilder builder)
    {
        _builder = builder;
    }

    public AutomationRule Construct(AutomationRuleDto dto)
    {
        _builder
            .SetName(dto.Name)
            .SetTrigger(dto.TriggerType, dto.TriggerData)
            .SetCondition(dto.ConditionExpression ?? string.Empty);

        if (!string.IsNullOrWhiteSpace(dto.ScheduleExpression))
        {
            _builder.SetSchedule(dto.ScheduleExpression);
        }

        foreach (var action in dto.Actions)
        {
            _builder.AddAction(action.Type, action.Parameters);
        }

        return _builder.Build();
    }
}
```

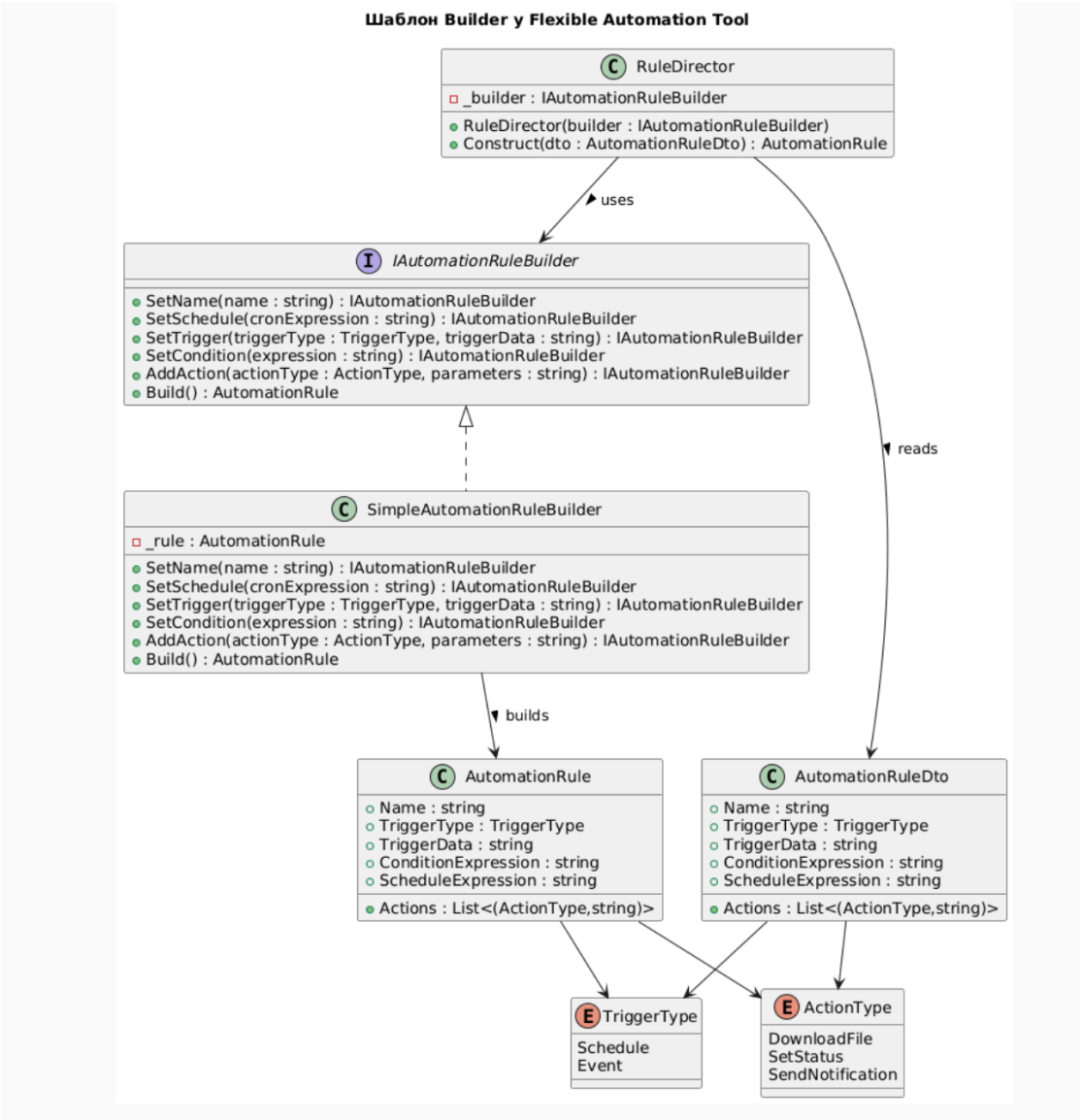
Приклад використання Builder у Program.cs

```
var dto = new AutomationRuleDto
{
    Name = "Download new episodes and notify",
    TriggerType = TriggerType.Schedule,
    TriggerData = "0 18 * * 5", // щоп'ятниці о 18:00
    ConditionExpression = "user.isPremium == true",
    ScheduleExpression = "0 18 * * 5",
    Actions =
    {
        (ActionType.DownloadFile, "url=https://example.com/episodes"),
        (ActionType.SendNotification, "channel=email;template=new-episode")
    }
};

var builder = new SimpleAutomationRuleBuilder();
var director = new RuleDirector(builder);
AutomationRule rule = director.Construct(dto);

Console.WriteLine($"Rule '{rule.Name}' with {rule.Actions.Count} actions constructed
successfully.");
```

Діаграма класів



Діаграма класів (рис. 1)

Зображено взаємодію основних елементів шаблону Builder у системі Flexible Automation Tool: інтерфейс IAutomationRuleBuilder, конкретний клас SimpleAutomationRuleBuilder, клас AutomationRule (складний об'єкт, що будується), DTO-клас AutomationRuleDto та клас-розпорядник RuleDirector. RuleDirector не знає деталей створення правила і використовує лише інтерфейс будівельника, що забезпечує гнучкість та можливість заміни реалізації будівельника.

Висновок

У ході виконання лабораторної роботи було розглянуто групу шаблонів проектування Adapter, Builder, Command, Chain of Responsibility та Prototype. Для системи Flexible Automation Tool було обрано і реалізовано шаблон Builder, який використано для поетапного створення складного об'єкта AutomationRule. Застосування Builder дозволило відокремити процес побудови правила від його внутрішньої структури, спростити додавання нових конфігурацій правил та покращити читабельність коду.