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

Лабораторна робота №5

з дисципліни «Технології розроблення програмного забезпечення»

Тема: «Шаблони «Adapter», «Builder», «Command», «Chain of responsibility», «Prototype»»

Варіант №15

Виконав:
студент групи ІА-23
Лядський Д.С.

Перевірив:
Мягкий М. Ю.

Київ 2024

Зміст

Тема

Шаблони «Adapter», «Builder», «Command», «chain of responsibility», «Prototype»

Мета

Ознайомитися з основними концепціями, принципами та застосуванням шаблонів проектування «Adapter», «Builder», «Command», «Chain of Responsibility» та «Prototype». Навчитися реалізовувати ці шаблони в програмному коді для вирішення практичних завдань, підвищення гнучкості, масштабованості та повторного використання коду в програмних системах.

Короткі теоретичні відомості

Анти-патерни (anti-patterns)

Анти-патерни — це типові помилки в проєктуванні чи програмуванні, яких варто уникати. Вони ілюструють неправильні рішення, що ведуть до проблем у системах.

Анти-патерни управління розробкою ПЗ

- Дим і дзеркала — демонстрація вигляду ще не створеного функціоналу.
- Роздування ПЗ — вимоги нових версій до ресурсів надміру зростають.
- Функції для галочки — набір функцій, що заявлені, але погано реалізовані.

Анти-патерни в розробці ПЗ

- Інверсія абстракції — приховування функціоналу, який може бути корисним.
- Великий клубок бруду — система без чіткої структури.
- Роздування інтерфейсу — надмірно складний і громіздкий інтерфейс.

Анти-патерни об'єктно-орієнтованого програмування

- Божественний об'єкт — занадто багато функцій в одному класі.
- Полтергейст — об'єкти, які тільки передають дані іншим.
- Самотність — невиправдане використання патерну «одинак».

Анти-патерни програмування

- Спагеті-код — заплутана структура коду.
- Жорстке кодування — залежність від жорстко заданих значень.
- Передчасна оптимізація — спроби оптимізації без реальної потреби.

Організаційні анти-патерни

- Аналітичний параліч — надмірний акцент на аналізі.
- Повзуче покращення — додавання функцій на шкоду якості.
- Єдина обізнана людина — критична залежність від однієї особи.

Шаблон «Adapter»

Призначення: дозволяє адаптувати інтерфейс одного об'єкта до іншого без змін у вихідному коді.

Проблема: несумісність форматів чи інтерфейсів.

Рішення: створення об'єкта-адаптера, що перетворює дані або методи для сумісності.

Приклад: адаптер для європейської зарядки в США.

Переваги: приховує складність трансформацій.

Недоліки: ускладнює структуру коду.

Шаблон «Builder»

Призначення: відокремлює процес створення об'єкта від його представлення, що корисно при створенні складних об'єктів.

Проблема: складний процес побудови, що потребує модульності.

Рішення: використання методів будівельника для кожного етапу створення.

Приклад: створення відповіді веб-сервера.

Переваги: гнучкість і незалежність від змін.

Недоліки: залежність клієнта від конкретного будівельника.

Шаблон «Command»

Призначення: перетворює виклики методів у класи, що дозволяє працювати з діями як з об'єктами.

Сценарії використання:

Потрібна система з можливістю додавання нових команд.

Потрібно додати командам можливість скасування або логування.

Необхідність створення ланцюжків команд.

Завдання

1. Ознайомитися з короткими теоретичними відомостями.
2. Реалізувати частину функціоналу робочої програми у вигляді класів та їх взаємодій для

досягнення конкретних функціональних можливостей.

3. Застосування одного з розглянутих шаблонів при реалізації програми.

Обрана тема

15 E-mail клієнт (singleton, builder, decorator, template method, interpreter, SOA)

Поштовий клієнт повинен нагадувати функціонал поштових програм Mozilla Thunderbird, The Bat і т.д. Він повинен сприймати і коректно обробляти pop3/smtp/imap протоколи, мати функції автонастройки основних поштових провайдерів для України (gmail, ukr.net, i.ua), розділяти повідомлення на

папки/категорії/важливість, зберігати чернетки незавершених повідомлень, прикріплювати і обробляти прикріплені файли.

Хід роботи

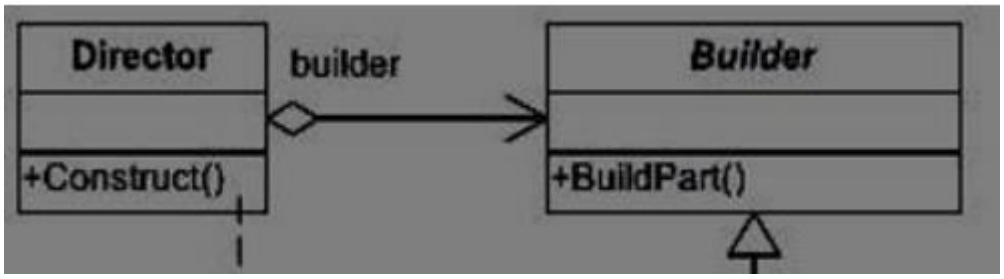

Рисунок 1. Структура шаблона Builder

Реалізація патерну Builder:

```
@Getter
public class EmailAccount {
    1 usage
    private Long id;
    1 usage
    private String emailAddress;
    1 usage
    private String password;
    1 usage
    private ServerConnection incomingServer;
    1 usage
    private ServerConnection outgoingServer;

    1 usage
    private Session incomingServerSession;

    1 usage
    private Session outgoingServerSession;
    1 usage
    private Boolean autoconfig;

    1 usage  dLiadsk
    private EmailAccount(EmailAccountBuilder emailAccountBuilder) {
        this.id = emailAccountBuilder.id;
        this.emailAddress = emailAccountBuilder.emailAddress;
        this.password = emailAccountBuilder.password;
        this.incomingServer = emailAccountBuilder.incomingServer;
        this.outgoingServer = emailAccountBuilder.outgoingServer;
        this.incomingServerSession = emailAccountBuilder.incomingServerSession;
```

Рисунок 2. Клас EmailAccount

```
        this.incomingServerSession = emailAccountBuilder.incomingServerSession;
        this.outgoingServerSession = emailAccountBuilder.outgoingServerSession;
        this.autoconfig = emailAccountBuilder.autoconfig;
    }


    10 usages  dLiadsk *
    public static class EmailAccountBuilder {
        2 usages
        private Long id;
        3 usages
        private String emailAddress;
        2 usages
        private String password;
        5 usages
        private ServerConnection incomingServer;
        5 usages
        private ServerConnection outgoingServer;

        2 usages
        private Session incomingServerSession;

        2 usages
        private Session outgoingServerSession;
        2 usages
        private Boolean autoconfig;
```

Рисунок 3. Клас EmailAccount

```
    public EmailAccountBuilder(String emailAddress, String password) {
        this.emailAddress = emailAddress;
        this.password = password;
    }
    3 usages  dLiadsk
    public EmailAccountBuilder setAutoconfig(Boolean autoconfig){
        if (autoconfig){
            autoConfigure();
        }
        this.autoconfig = autoconfig;
        return this;
    }
    dLiadsk
    public EmailAccountBuilder setId(Long id){
        this.id = id;
        return this;
    }

    dLiadsk
    public EmailAccountBuilder setIncomingServer(ServerConnection serverConnection){
        this.incomingServer = serverConnection;
        return this;
    }
    dLiadsk
    public EmailAccountBuilder setOutgoingServer(ServerConnection serverConnection){
        this.outgoingServer = serverConnection;
        return this;
    }
```

Рисунок 4. Клас EmailAccount

```
    public EmailAccountBuilder setIncomingServerSession(Session session){
        this.incomingServerSession = session;
        return this;
    }
    1 usage  dLiadsk
    public EmailAccountBuilder setOutgoingServerSession(Session session){
        this.outgoingServerSession = session;
        return this;
    }
    1 usage  dLiadsk
    private void autoConfigure() {
        String domain = this.emailAddress.split( regex: "@")[1];
        switch (domain) {
            case "gmail.com" -> {
                this.incomingServer = new ServerConnection( host: "imap.gmail.com",  port: 993, ProtocolType.IMAP);
                this.outgoingServer = new ServerConnection( host: "smtp.gmail.com",  port: 465, ProtocolType.SMTP);
            }
            case "ukr.net" -> {
                this.incomingServer = new ServerConnection( host: "imap.ukr.net",  port: 993, ProtocolType.IMAP);
                this.outgoingServer = new ServerConnection( host: "smtp.ukr.net",  port: 465, ProtocolType.SMTP);
            }
            case "i.ua" -> {
                this.incomingServer = new ServerConnection( host: "imap.i.ua",  port: 993, ProtocolType.IMAP);
                this.outgoingServer = new ServerConnection( host: "smtp.i.ua",  port: 465, ProtocolType.SMTP);
            }
            default -> System.out.println("Автонастройка недоступна для домену: " + domain);
        }
    }
    3 usages  dLiadsk
    public EmailAccount build() { return new EmailAccount( emailAccountBuilder: this); }
}
```

Рисунок 5. Клас EmailAccount

Клас EmailAccount призначений для опису облікового запису електронної пошти з підтримкою налаштувань серверів вхідної та вихідної пошти. Він використовує підхід Builder для спрощення створення об'єктів із різними параметрами.

Основні поля:

- id: Унікальний ідентифікатор облікового запису.
- emailAddress: Адреса електронної пошти.
- password: Пароль для доступу.
- incomingServer: Параметри сервера вхідної пошти (типу IMAP/POP3).
- outgoingServer: Параметри сервера вихідної пошти (SMTP).
- incomingServerSession: Сесія для сервера вхідної пошти.
- outgoingServerSession: Сесія для сервера вихідної пошти.
- autoconfig: Вказує, чи використовується автоматична конфігурація серверів.

Особливості:

Автонастройка: Метод autoConfigure налаштовує сервери залежно від домену електронної пошти (підтримуються gmail.com, ukr.net, i.ua).

Builder-паттерн: Клас EmailAccountBuilder дозволяє поступово налаштовувати параметри облікового запису перед створенням об'єкта EmailAccount.

```
EmailAccount emailAccount = new EmailAccount.EmailAccountBuilder(email, password)
        .setAutoconfig(true)
        .setId(generateEmailAccountId())
        .build();
```

Рисунок 6. Приклад використання

Висновок

У ході виконання лабораторної роботи було розглянуто шаблони проектування «Adapter», «Builder», «Command», «Chain of Responsibility» та

«Prototype». Реалізація цих шаблонів дозволяє спрощувати архітектуру програмних систем, забезпечувати гнучкість та зручність модифікації коду, а також сприяти повторному використанню рішень. Отримані знання та практичні навички є важливими для розробки програмного забезпечення з якісною архітектурою.