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

Лабораторна робота №7

з дисципліни «Технології розроблення програмного забезпечення»

Тема: «Шаблон «Mediator», «Facade», «Bridge», «Template method»»

Варіант №15

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

Перевірив:
Мягкий М. Ю.

Київ 2024

Зміст

Тема

Шаблон «Mediator», «Facade», «Bridge», «Template method»

Мета

Метою цієї лабораторної роботи є дослідження та застосування шаблонів проектування «Mediator», «Facade», «Bridge» і «Template Method». В рамках роботи студенти повинні розібратися в концепціях кожного шаблону, їхньому застосуванню для вирішення реальних задач, а також реалізувати приклади їх використання в коді.

Короткі теоретичні відомості

Принципи проектування:

Don't Repeat Yourself (DRY):

Повторення коду слід уникати, щоб зробити його читабельним, легким у підтримці й уникнути помилок. Використовуються техніки рефакторингу, наприклад, винесення загального функціоналу в методи, інтерфейси або класи.

Keep It Simple, Stupid (KISS):

Компоненти системи мають бути простими. Краще створювати багато простих компонентів, ніж одну складну систему. Це підвищує надійність і спрощує підтримку.

You Only Load It Once (YOLO):

Ініціалізаційні змінні слід завантажувати один раз при запуску програми, щоб уникнути затримок і проблем з продуктивністю.

Принцип Парето (80/20):

80% результату досягається за 20% зусиль.

Наприклад, 80% помилок можна виправити, закривши 20% багів.

You Ain’t Gonna Need It (YAGNI):

Не варто реалізовувати функціонал, який, ймовірно, не знадобиться. Просте рішення краще, ніж складне і універсальне.

Шаблон «Mediator» (Посередник):

Призначення: Організовує взаємодію між об'єктами через окремий об'єкт-посередник, що зменшує кількість залежностей між компонентами.

Застосування: Використовується для керування складними взаємодіями, наприклад, в діалогах, де взаємодіють численні елементи.

Переваги:

- Усуває прямі залежності між компонентами.
- Централізує управління.

Недоліки:

- Посередник може стати занадто складним.

Шаблон «Facade» (Фасад):

Призначення: Надає спрощений інтерфейс доступу до складної системи.

Застосування: Використовується для ізоляції клієнта від деталей реалізації складної підсистеми, наприклад, бібліотек або фреймворків.

Переваги:

- Спрощує роботу з підсистемою.
- Зменшує кількість залежностей.

Недоліки:

- Фасад може стати божественним об'єктом.

Шаблон «Bridge» (Міст):

Призначення: Відокремлює абстракцію від її реалізації, дозволяючи розвивати їх незалежно.

Застосування:Використовується, коли потрібно уникнути множинного спадкування через перехресні комбінації властивостей, наприклад, кольору й форми.

Переваги:

- Забезпечує платформонезалежність.
- Реалізує принцип відкритості/закритості.

Недоліки:

- Ускладнює код через введення додаткових класів.

Шаблон «Template Method» (Шаблонний метод):

Призначення: Дозволяє визначити основну структуру алгоритму в базовому класі, залишивши підкласи для уточнення окремих його кроків.

Застосування: Використовується для створення повторюваних алгоритмів з варіативними кроками, наприклад, обробка файлів різних форматів.

Переваги:

- Усуває дублювання коду.
- Полегшує підтримку й розширення.

Недоліки:

- Може бути важко налаштувати для складних алгоритмів.

Завдання

1. Ознайомитися з короткими теоретичними відомостями

2. Реалізувати частину функціонала робочої програми у вигляді класів і їх взаємодій для досягнення конкретних функціональних можливостей.

3. Застосування одного з даних 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

Реалізація патерну template method:

```
@Slf4j
public abstract class EmailProtocolHandler {
    2 usages
    public final Session authorize(EmailAccount emailAccount, ServerConnection serverConnection) {
        Properties properties = setupProperties(serverConnection, emailAccount);
        Session session = createSession(properties, emailAccount);

        if (testConnection(session, serverConnection, emailAccount)) {
            log.info("Authorization successful for server: " + serverConnection.getHost());
            return session;
        } else {
            log.warn("Authorization failed for server: " + serverConnection.getHost());
            return null;
        }
    }

    // Метод для налаштування властивостей (шаблонний крок)
    1 usage  2 implementations
    protected abstract Properties setupProperties(ServerConnection serverConnection, EmailAccount emailAccount);

    // Тестування з'єднання (шаблонний крок)
    1 usage  2 implementations
    protected abstract boolean testConnection(Session session, ServerConnection serverConnection, EmailAccount emailAccount);

    1 usage
    private Session createSession(Properties properties, EmailAccount emailAccount) {
        return Session.getInstance(properties, getPasswordAuthentication() -> {
                return new PasswordAuthentication(emailAccount.getEmailAddress(), emailAccount.getPassword());
        });
    }
}
```

Рисунок 1. Абстрактний клас EmailProtocolHandler

```java
@Slf4j
public class IncomingServerHandler extends EmailProtocolHandler {
    1 usage
    @Override
    protected Properties setupProperties(ServerConnection serverConnection, EmailAccount emailAccount) {
        Properties properties = new Properties();
        String protocol = serverConnection.getProtocol().name().toLowerCase();
        properties.put("mail." + protocol + ".host", serverConnection.getHost());
        properties.put("mail." + protocol + ".port", serverConnection.getPort());
        properties.put("mail." + protocol + ".ssl.enable", "true");
        return properties;
    }

    1 usage
    @Override
    protected boolean testConnection(Session session, ServerConnection serverConnection, EmailAccount emailAccount) {
        try (Store store = session.getStore(serverConnection.getProtocol().name().toLowerCase())) {
            store.connect(serverConnection.getHost(), emailAccount.getEmailAddress(), emailAccount.getPassword());
            return true;
        } catch (MessagingException e) {
            log.warn("Incoming server connection failed: " + e.getMessage());
            return false;
        }
    }
}
```

Рисунок 2. Клас IncomingServerHandler

```java
@Slf4j
public class OutgoingServerHandler extends EmailProtocolHandler {
    1 usage
    @Override
    protected Properties setupProperties(ServerConnection serverConnection, EmailAccount emailAccount) {
        Properties properties = new Properties();
        properties.put("mail.smtp.host", serverConnection.getHost());
        properties.put("mail.smtp.port", serverConnection.getPort());
        properties.put("mail.smtp.ssl.enable", "true");
        properties.put("mail.smtp.auth", "true");
        properties.put("mail.smtp.starttls.enable", "true");

        return properties;
    }

    1 usage
    @Override
    protected boolean testConnection(Session session, ServerConnection serverConnection, EmailAccount emailAccount) {
        try {
            Transport transport = session.getTransport( protocol: "smtp");
            transport.connect();
            transport.close();
            return true;
        } catch (MessagingException e) {
            log.warn("Outgoing server connection failed: " + e.getMessage());
            return false;
        }
    }
}
```

Рисунок 3. Клас EmailAccount

Даний код реалізує систему обробки протоколів електронної пошти для Java-додатка. Він розроблений на основі шаблонного методу, що дозволяє

забезпечити загальну логіку авторизації для різних типів серверів (вхідних та вихідних), а специфічні аспекти реалізуються в дочірніх класах.

Опис класів:

1. EmailProtocolHandler

Цей абстрактний клас є базовим для роботи з протоколами електронної пошти. Він реалізує загальну логіку авторизації та передбачає методи для налаштування специфічних властивостей і тестування з'єднання, які повинні бути реалізовані в дочірніх класах.

Методи:

authorize: головний метод для створення сесії, що виконує:

- Налаштування властивостей через setupProperties.
- Створення Session.
- Перевірку з'єднання через testConnection.

setupProperties: абстрактний метод для налаштування властивостей залежно від протоколу.

testConnection: абстрактний метод для перевірки з'єднання.

createSession: приватний метод для створення сесії з використанням облікових даних.

2. IncomingServerHandler

Реалізує специфічну логіку для роботи з вхідними серверами (IMAP/POP3).

Особливості:

- Використовує протокол (IMAP або POP3), визначений у ServerConnection.
- Налаштовує SSL для безпечного з'єднання.
- Використовує Store для тестування з'єднання.

Методи:

setupProperties: налаштовує властивості для вхідного сервера (хост, порт, SSL).

testConnection: перевіряє з'єднання через Store.

3. OutgoingServerHandler

Реалізує специфічну логіку для роботи з вихідними серверами (SMTP).

Особливості:

- Використовує SMTP для відправлення повідомлень.
- Додає властивості для автентифікації та підтримки TLS.
- Використовує Transport для тестування з'єднання.

Методи:

setupProperties: налаштовує властивості для SMTP (хост, порт, автентифікація, SSL, TLS).

testConnection: перевіряє з'єднання через Transport.

Основні принципи проєктування:

Шаблонний метод (Template Method): Загальний алгоритм авторизації реалізований в базовому класі EmailProtocolHandler, тоді як специфічні кроки налаштування властивостей (setupProperties) і перевірки з'єднання (testConnection) делегуються дочірнім класам.

Полегшення розширення: Код легко розширити для інших протоколів (наприклад, EWS або протоколів майбутнього).

Логгування: Використовується @Slf4j для журналювання, що дозволяє відслідковувати успішні і невдалі спроби з'єднання.

```java
public EmailAccount authorizeEmail(EmailAccount emailAccount) {
    EmailProtocolHandler incomingHandler = new IncomingServerHandler();
    Session incomingSession = incomingHandler.authorize(emailAccount, emailAccount.getIncomingServer());

    if (incomingSession == null) {
        log.warn("Authorization failed on incoming server for: " + emailAccount.getEmailAddress());
        return null;
    }

    EmailProtocolHandler outgoingHandler = new OutgoingServerHandler();
    Session outgoingSession = outgoingHandler.authorize(emailAccount, emailAccount.getOutgoingServer());

    if (outgoingSession == null) {
        log.warn("Authorization failed on outgoing server for: " + emailAccount.getEmailAddress());
        return null;
    }

    EmailAccount authorizedEmailAccount = new EmailAccount.EmailAccountBuilder(emailAccount.getEmailAddress(), emailAccount.g
            .setId(emailAccount.getId())
            .setAutoconfig(true)
            .setIncomingServerSession(incomingSession)
            .setOutgoingServerSession(outgoingSession)
            .build();

    log.info("Authorization successful for: " + emailAccount.getEmailAddress());
    return authorizedEmailAccount;
}
```

Рисунок 4. Застосування в EmailAccountService

```java
public static void main(String[] args) throws SQLException {
//      DbConnection dbConnection = DbConnection.getInstance();
//      launch();
    testEmailAccount();
//      List<EmailMessage> emailMessages = new EmailFilterService().emailUnreadMessageDateFilter(emailMessages(), LocalDate.of(2023, 11, 1), LocalDate.of(2023,
//      emailMessages.forEach(System.out::println);
}

1 usage
public static void testEmailAccount(){
    EmailAccountService emailAccountService = new EmailAccountService(new EmailAccountRepository(), new UserRepository());
//      try {
//          emailAccountService.addEmailAccount(new User(), new EmailAccount("qqwe35949@gmail.com", "Asdfgh321"));
//      } catch (SQLException e) {
//          throw new RuntimeException(e);
//      }
    EmailAccount emailAccount = new EmailAccount.EmailAccountBuilder( emailAddress: "qwertyeee@ukr.net",  password: "04oiRBZT1bj5V8my").setAutoconfig(true).build();
    emailAccountService.authorizeEmail(emailAccount);

}
```

```
Main
"C:\Program Files\Java\jdk-17\bin\java.exe" ...
дек. 09, 2024 11:24:35 PM jakarta.mail.Session loadResource
WARNING: expected resource not found: /META-INF/javamail.default.address.map
23:24:37.042 [main] INFO org.example.model.handler.EmailProtocolHandler -- Authorization successful for server: imap.ukr.net
дек. 09, 2024 11:24:37 PM jakarta.mail.Session loadResource
WARNING: expected resource not found: /META-INF/javamail.default.address.map
23:24:42.604 [main] INFO org.example.model.handler.EmailProtocolHandler -- Authorization successful for server: smtp.ukr.net
23:24:42.604 [main] INFO org.example.service.EmailAccountService -- Authorization successful for: qwertyeee@ukr.net
```

Рисунок 6. Демонстрація роботи

Висновок

Під час виконання лабораторної роботи було успішно вивчено чотири популярних шаблони проектування: «Mediator», «Facade», «Bridge» і «Template Method». Застосування кожного з цих шаблонів дозволяє знизити складність

програмних систем, зробити їх більш гнучкими, зрозумілими та масштабованими.