Міністерство освіти і науки України Національний технічний університет України “Київський політехнічний інститут імені Ігоря Сікорського” Факультет інформатики та обчислювальної техніки Кафедра інформаційних систем та технологій

Лабораторна робота №5

із дисципліни «**Технології розроблення програмного забезпечення**»

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

Виконав:

студент групи ІА–24

Яблонський Д.Б.

Перевірив:

Мягкий М.Ю.

Київ 2024

Зміст

Мета: метою виконання лабораторної роботи є вивчення та практичне застосування шаблонів проєктування, таких як «Adapter», «Builder», «Command», «Chain of Responsibility», і «Prototype», для створення ефективних і гнучких програмних рішень. Ця лабораторна робота спрямована на розвиток навичок у використанні різних патернів для вирішення задач у розробці програмного забезпечення.

Теоретичні відомості

Шаблони проєктування є одним із фундаментальних елементів сучасної розробки програмного забезпечення. Вони представляють собою готові рішення для типових задач у програмуванні, знижують складність системи та забезпечують її масштабованість. Шаблони допомагають уникнути дублювання коду, підвищують його зрозумілість і зручність у підтримці.

Антипатерни, навпаки, демонструють погані рішення та практики, які, хоча й можуть працювати в певних умовах, зазвичай ведуть до зниження продуктивності й якості програмного забезпечення. Розуміння антипатернів є важливим аспектом професійного зростання, адже дозволяє розробникам уникати типових помилок і виявляти слабкі місця в системах.

У цій лабораторній роботі розглядаються наступні шаблони:

Adapter

Шаблон Adapter використовується для забезпечення сумісності між несумісними класами. Його головна ідея полягає в створенні проміжного класу, який перетворює інтерфейс одного класу на інтерфейс, зрозумілий іншому. Adapter дозволяє взаємодіяти об'єктам, які зазвичай не можуть працювати разом, без змін у їх коді.

Builder

Шаблон Builder забезпечує створення складних об'єктів покроково. Він дозволяє ізолювати процес створення об'єкта від його представлення, що робить код більш гнучким і полегшує підтримку. Використання Builder актуальне, коли об'єкт має багато параметрів або може існувати в різних конфігураціях.

Command

Command — це шаблон, який інкапсулює дію або запит у вигляді об'єкта. Це дозволяє передавати дії як параметри, створювати черги команд, підтримувати скасування та повторне виконання операцій. Основна перевага Command у тому, що він сприяє зниженню зв'язності між відправником і виконавцем дії.

Chain of Responsibility

Цей шаблон організовує обробку запитів через ланцюг обробників. Кожен обробник приймає рішення, чи варто обробити запит, або ж передати його наступному обробнику в ланцюзі. Chain of Responsibility дозволяє динамічно змінювати послідовність обробки запитів і додає гнучкості в реалізацію логіки.

Prototype

Шаблон Prototype використовується для створення нових об'єктів шляхом копіювання вже існуючих. Він корисний, коли створення об'єкта є дорогим процесом, а копіювання забезпечує значне зниження витрат. Prototype дозволяє зберігати стан об'єкта, що особливо важливо у випадках складних конфігурацій.

Хід роботи

1. Ознайомитися з короткими теоретичними відомостями.
2. Реалізувати частину функціоналу робочої програми у вигляді класів та їхньої взаємодії для досягнення конкретних функціональних можливостей.
3. Застосування одного з розглянутих шаблонів при реалізації програми.

Тема: Mind-mapping software

Реалізація шаблону Prototype

В цій лабораторній роботі я використовуватиму шаблон Prototype. Використання цього шаблону полягатиме в створенні практично аналогічних об’єктів класу Cookie, шляхом клонування.

При реєстрації або логіні користувача, якщо він правильно ввів всі дані і реєстрація або авторизація пройшла успішно, застосунок повинен створити контекст цього користувача для того, шоб цей користувач міг далі зручно

користуватись сервісом. Для цього, буде створюватись кукі, значенням якого є ідентифікатор користувача, що зараз працює в застосунку. І коли Користувач викликатиме дії, що потребують ідентифікувати користувача, який їх викликає, буде братись кукі, значенням якого і є цей ідентифікатор.

```
@PostMapping("/register")  yablonya
public ResponseEntity<?> registerUser(
        @RequestBody UserRegistrationRequest request,
        HttpServletResponse response
) {
    try {
        User registeredUser = userService.registerUser(request.getName(), request.getEmail(), request.getPassword());
        userService.addUserIdToCookie(response, registeredUser);

        logger.info("User registered with email: {}", registeredUser.getEmail());
        return ResponseEntity.ok(registeredUser);
    } catch (IllegalArgumentException e) {
        logger.error("Registration error: {}", e.getMessage());
        return ResponseEntity.badRequest().body(e.getMessage());
    }
}
```

Рис. 1 - Метод registerUser класу HomeController

```
@PostMapping("/login")  yablonya
public ResponseEntity<?> loginUser(
        @RequestBody UserLoginRequest request,
        HttpServletResponse response
) {
    try {
        User user = userService.loginUser(request.getEmail(), request.getPassword());
        userService.addUserIdToCookie(response, user);

        return ResponseEntity.ok(user);
    } catch (IllegalArgumentException e) {
        return ResponseEntity.status(401).body(e.getMessage());
    }
}
```

Рис. 2 - Метод loginUser класу HomeController

```java
@PostMapping("/logout")  yablonya
public ResponseEntity<?> logoutUser(HttpServletResponse response) {
    try {
        userService.clearUserCookie(response);
        logger.info("User logged out successfully.");
        return ResponseEntity.ok( body: "Logout successful.");
    } catch (Exception e) {
        logger.error("Logout error: {}", e.getMessage());
        return ResponseEntity.status(500).body("An error occurred during logout.");
    }
}
```

Рис. 3 - Метод logoutUser класу HomeController

На рис. 1-3 ми бачимо методи, які відповідатимуть на запити користувача, а саме запит на реєстрацію, авторизацію та вихід з акаунту. Всі ці методи потребують взаємодії з кукі. Ця взаємодія реалізована в класі UserService.

UserService.java

```java
@Service  7 usages  yablonya
public class UserService {
    private final UserRepository userRepository;  4 usages
    private final CookiePrototype userCookiePrototype = new CookiePrototype( name: "userId",  maxAge: 24 * 60 * 60);
    private static final Logger logger = LoggerFactory.getLogger(UserService.class);  6 usages

    @Autowired  yablonya
    public UserService(UserRepository userRepository) {
        this.userRepository = userRepository;
    }

    public User registerUser(String name, String email, String password) {  1 usage  yablonya
        if (userRepository.findByEmail(email) != null) {
            throw new IllegalArgumentException("User with this email already exists.");
        }

        User user = createUser(name, email, password);
        User savedProfile = userRepository.save(user);
        logger.info("User profile created: {}", savedProfile.getEmail());

        return savedProfile;
    }

    public User loginUser(String email, String password) {  1 usage  yablonya
        logger.info("Attempting to login user with email: {}", email);

        User user = userRepository.findByEmail(email);

        if (user == null || !user.getPassword().equals(password)) {
            logger.warn("Invalid login attempt for email: {}", email);
            throw new IllegalArgumentException("Invalid email or password.");
        }

        logger.info("User logged in successfully: {}", email);
        return user;
    }
```

Рис. 4 - Клас UserService та його методи що використовуються в класі HomeController

```
private User createUser(String name, String email, String password) {  1 usage  yablonya
    User user = new User();
    user.setName(name);
    user.setEmail(email);
    user.setPassword(password);
    return user;
}

public void addUserIdToCookie(HttpServletResponse response, User user) {  2 usages  yab
    Cookie cookie = userCookiePrototype.cloneWithValue(String.valueOf(user.getId()));
    response.addCookie(cookie);
    logger.info("User cookie added with ID: {}", user.getId());
}

public void clearUserCookie(HttpServletResponse response) {  1 usage  yablonya
    Cookie cookie = userCookiePrototype.cloneAsCleared();
    response.addCookie(cookie);
    logger.info("User cookie cleared.");
}
```

Рис. 5 - Допоміжні методи класу UserService

На рис. 4-5 ми бачимо як відбувається робота з кукі при створенні користувача та його авторизації. Можна помітити, що єдиний параметр кукі, який змінюється, це значення ідентифікатора користувача, а всі інші параметри можна копіювати. В ситуації коли користувач виходить з акаунту, застосунок повинен очистити кукі, і для цього щоразу використовується об’єкт з одними і тими ж параметрами, що дозволяє повністю його копіювати.

CookiePrototype.java

```java
package org.example.mindmappingsoftware.prototypes;

import jakarta.servlet.http.Cookie;

public class CookiePrototype {  3 usages  yablonya
    private final Cookie prototype;  6 usages

    public CookiePrototype(String name, int maxAge) {  1 usag
        this.prototype = new Cookie(name,  value: null);
        this.prototype.setPath("/");
        this.prototype.setHttpOnly(true);
        this.prototype.setMaxAge(maxAge);
    }

    public Cookie cloneWithValue(String value) {  1 usage  y
        Cookie clonedCookie = (Cookie) prototype.clone();
        clonedCookie.setValue(value);
        return clonedCookie;
    }

    public Cookie cloneAsCleared() {  1 usage  yablonya
        Cookie clearedCookie = (Cookie) prototype.clone();
        clearedCookie.setMaxAge(0);
        return clearedCookie;
    }
}
```

Рис. 6 - Клас CookiePrototype, що реалізує механізм клонування об’єкта

На рис. 6 показано клас CookiePrototype, що відповідає за клонування об’єкту. Маємо метод cloneWithValue для клонування кукі, що використовується при реєстрації та логіні, а також cloneAsCleared, що потрібен для очищення кукі.

Весь оновлений код можна переглянути в даній директорії репозиторію проєкту: https://github.com/yablonya/TRPZ_labs_yablonskyi_ia-24/tree/lab-5/src/main/java/org/example/mindmappingsoftware

Висновок: У ході виконання цієї лабораторної роботи я реалізував шаблон проєктування Prototype, який дозволяє створювати об'єкти шляхом їх клонування. Робота з цим патерном дала мені змогу краще зрозуміти, як можна ефективно використовувати копіювання об'єктів для зниження витрат на їх ініціалізацію, особливо у випадках, коли створення нового екземпляра є ресурсомістким.

Я переконався, що Prototype є надзвичайно корисним для задач, де необхідно створювати багато схожих об'єктів із мінімальними змінами. Наприклад, це може бути актуально у випадках складних конфігурацій або об'єктів зі значною кількістю параметрів. Крім того, я отримав практичний досвід роботи з методами клонування об'єктів у Java, що дозволило закріпити мої знання з об’єктно-орієнтованого програмування.

Ця лабораторна робота також допомогла мені краще зрозуміти важливість шаблонів проєктування загалом, адже вони сприяють створенню якісного коду, який легко масштабувати й підтримувати. Опанування шаблону Prototype зокрема стане гарним доповненням до знань про оптимізацію використання ресурсів, якими оперує застосунок.