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 №7

Технології розробки програмного забезпечення

ШАБЛОН «MEDIATOR», «FACADE», «BRIDGE», «TEMPLATE METHOD»

Виконала:
студентка групи ІА-24
Орловська А. В.
Перевірив:
Мягкий М. Ю.

Київ 2024

Зміст

Тема: ШАБЛОН «MEDIATOR», «FACADE», «BRIDGE», «TEMPLATE METHOD»

Короткі теоретичні відомості

Шаблон Mediator (Посередник) є поведінковим шаблоном, який зменшує взаємодії між об'єктами, спрямовуючи всі комунікації через єдиний центральний об'єкт-посередник. Це дозволяє знизити зв'язність між компонентами та спростити їх взаємодію, замінюючи прямі зв'язки між ними на посередницькі. Такий підхід часто використовується в ситуаціях, коли необхідно знизити складність системи з багатьма взаємодіючими елементами.

Facade (Фасад) — це структурний шаблон, який забезпечує спрощений інтерфейс для роботи зі складною підсистемою. Завдяки фасаду клієнт може взаємодіяти з підсистемою через один простий інтерфейс, при цьому внутрішня складність системи приховується. Цей шаблон дозволяє знизити складність використання системи, коли клієнт не потребує доступу до всіх її компонентів.

Шаблон Bridge (Міст) дозволяє розділити абстракцію від її реалізації, що дає можливість змінювати ці дві частини незалежно одна від одної. Це дозволяє створювати гнучкі та масштабовані системи, де нові абстракції та їх реалізації можна додавати без порушення існуючого коду. Це особливо корисно, коли необхідно працювати з різними варіантами реалізації абстракцій, наприклад, для різних платформ чи пристроїв.

Template Method (Шаблонний метод) — це поведінковий шаблон, який визначає загальну структуру алгоритму, дозволяючи підкласам змінювати деякі його частини. В основному класі задається загальна послідовність кроків алгоритму, а підкласи можуть реалізовувати конкретні етапи. Цей шаблон допомагає знизити дублювання коду та забезпечує однакову структуру виконання алгоритмів, що є корисним у фреймворках і бібліотеках.

Хід роботи

***Система для колективних покупок*(proxy, builder, decorator, facade, composite).** Система дозволяє створити список групи для колективної покупки, список що потрібно купити з орієнтовною вартістю кожної позиції та орієнтовною загальною вартістю, запланувати хто що буде купляти. Щоб користувач міг відмітити що він купив, за яку суму, з можливістю прикріпити чек. Система дозволяє користувачу вести списки бажаних для нього покупок, з можливістю позначати списки, які будуть доступні для друзів (як списки, що можна подарувати користувачеві). Система дозволяє добавляти інших користувачів в друзі.

Основні принципи та застосування Facade:

Фасад забезпечує спрощений доступ до складних підсистем, приховуючи їхні деталі реалізації та надаючи клієнту зручний інтерфейс для взаємодії. Він інкапсулює взаємодію клієнта з підсистемою, захищаючи клієнтський код від змін у внутрішній структурі системи. Замість того, щоб працювати з великою кількістю взаємозалежних компонентів, клієнт звертається лише до одного фасаду, що суттєво зменшує залежності між ними.

Коли застосовувати Facade:

Шаблон Facade доцільно використовувати, коли потрібно приховати складність підсистеми, надавши клієнту простий і зрозумілий інтерфейс. Він також корисний у випадках, коли система складається з багатьох взаємозалежних класів, і є необхідність зменшити рівень зв'язності між ними. Крім того, Facade допомагає полегшити тестування та впровадження змін у великих системах, забезпечуючи єдиний контрольований доступ до функціональності підсистеми.

Реалізація шаблону проєктування для майбутньої системи

Фасад CollectivePurchasesFacade служить для спрощення взаємодії між контролерами та сервісами. Він об'єднує виклики до різних сервісів в одному місці, забезпечуючи єдиний інтерфейс для виконання різних операцій. Основні дії фасаду включають:

Аутентифікація користувачів: Викликає методи сервісу аутентифікації для перевірки облікових даних користувачів.

Управління дружбою: Викликає методи сервісу дружби для додавання, видалення та отримання дружніх зв'язків користувачів.

Управління списками бажань: Викликає методи проксі-сервісу списків бажань для створення, оновлення, видалення та отримання списків бажань користувачів.

Реєстрація користувачів: Викликає методи сервісу реєстрації для створення нових користувачів та перевірки наявності електронної пошти.

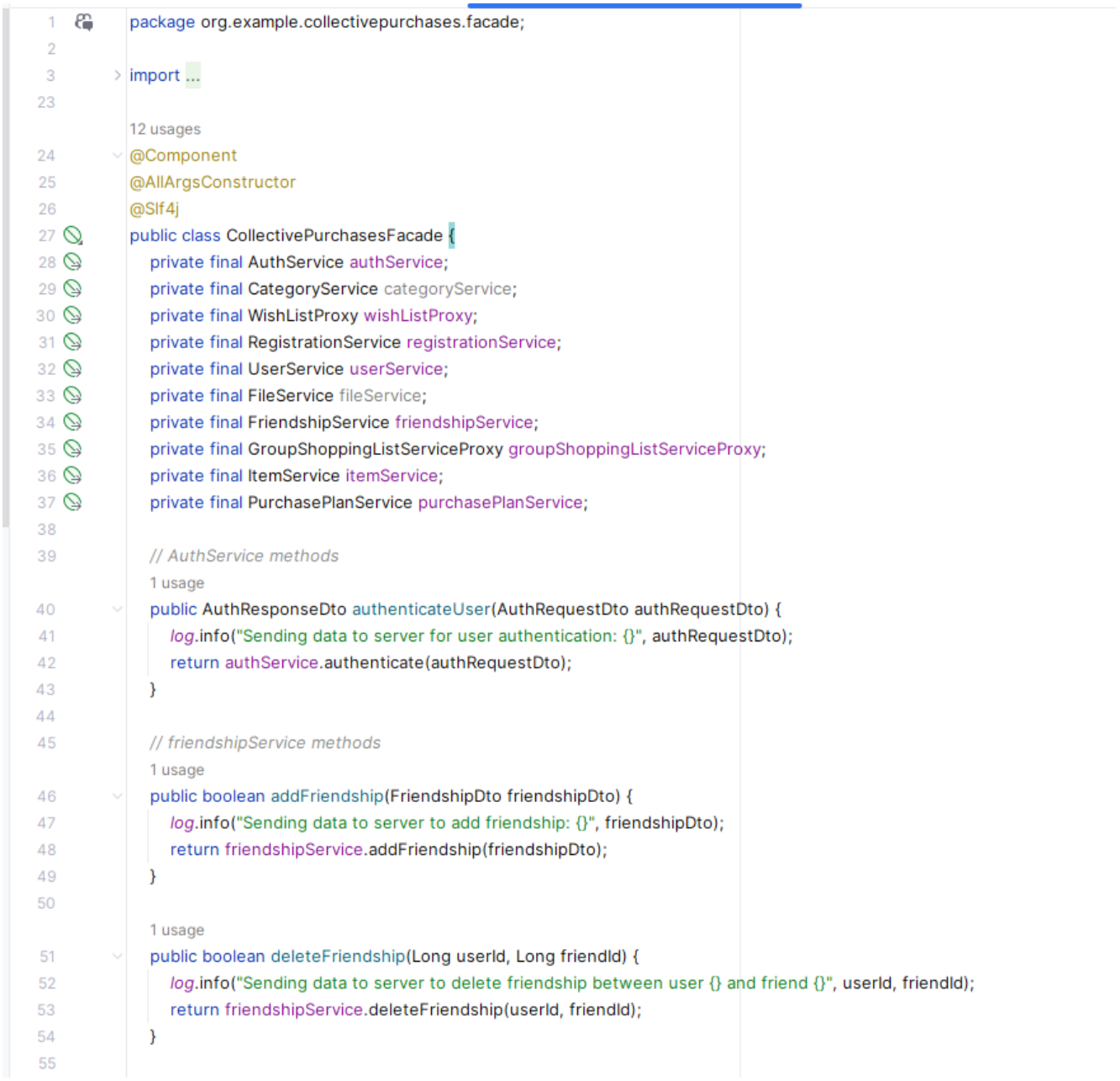
Управління користувачами: Викликає методи сервісу користувачів для отримання інформації про користувачів, перевірки їх існування та отримання користувачів за токеном.

Управління предметами: Викликає методи сервісу предметів для створення, оновлення, видалення та отримання предметів, а також для отримання предметів, згрупованих за категоріями.

Управління планами покупок: Викликає методи сервісу планів покупок для створення, оновлення, видалення та отримання планів покупок, а також для роботи з важливими планами покупок.

Управління груповими списками покупок: Викликає методи проксі-сервісу групових списків покупок для створення, оновлення, видалення та отримання групових списків покупок, а також для додавання користувачів до цих списків.

Фасад забезпечує централізоване місце для викликів сервісів, що спрощує код контролерів і робить його більш читабельним та підтримуваним.



```java
package org.example.collectivepurchases.facade;

import ...

12 usages
@Component
@AllArgsConstructor
@Slf4j
public class CollectivePurchasesFacade {
    private final AuthService authService;
    private final CategoryService categoryService;
    private final WishListProxy wishListProxy;
    private final RegistrationService registrationService;
    private final UserService userService;
    private final FileService fileService;
    private final FriendshipService friendshipService;
    private final GroupShoppingListServiceProxy groupShoppingListServiceProxy;
    private final ItemService itemService;
    private final PurchasePlanService purchasePlanService;

    // AuthService methods
    1 usage
    public AuthResponseDto authenticateUser(AuthRequestDto authRequestDto) {
        log.info("Sending data to server for user authentication: {}", authRequestDto);
        return authService.authenticate(authRequestDto);
    }

    // friendshipService methods
    1 usage
    public boolean addFriendship(FriendshipDto friendshipDto) {
        log.info("Sending data to server to add friendship: {}", friendshipDto);
        return friendshipService.addFriendship(friendshipDto);
    }

    1 usage
    public boolean deleteFriendship(Long userId, Long friendId) {
        log.info("Sending data to server to delete friendship between user {} and friend {}", userId, friendId);
        return friendshipService.deleteFriendship(userId, friendId);
    }

```

Рис. 1.1 – Код класу CollectivePurchasesFacade

```
1 usage
56  public List<FriendshipDto> getFriendshipsByUserId(Long userId) {
57      log.info("Sending data to server to get friendships for user {}", userId);
58      return friendshipService.getFriendshipsByUserId(userId);
59  }
60
61  // WishListProxy methods
    1 usage
62  public WishListDto createWishList(WishListDto wishListDto) {
63      log.info("Sending data to server to create wishlist: {}", wishListDto);
64      return wishListProxy.createWishList(wishListDto);
65  }
66
    1 usage
67  public List<WishListDto> getWishListsByUserId(Long userId) {
68      log.info("Sending data to server to get wishlists for user {}", userId);
69      return wishListProxy.getWishListsByUserId(userId);
70  }
71
    1 usage
72  public WishListDto getWishListById(Long id, String token) {
73      log.info("Sending data to server to get wishlist by id: {}", id);
74      return wishListProxy.getWishListById(id, token);
75  }
76
    1 usage
77  public void deleteWishListById(Long id) {
78      log.info("Sending data to server to delete wishlist by id: {}", id);
79      wishListProxy.deleteWishListById(id);
80  }
81
    1 usage
82  public WishListDto updateWishList(WishListDto wishListDto) {
83      log.info("Sending data to server to update wishlist: {}", wishListDto);
84      return wishListProxy.updateWishList(wishListDto);
85  }
```

Рис. 1.2 – Код класу CollectivePurchasesFacade

```
1 usage
88  public boolean createUser(UserDto userDTO) {
89      log.info("Sending data to server to create user: {}", userDTO);
90      return registrationService.createUser(userDTO);
91  }
92
    no usages
93  public boolean isEmailAlreadyInUse(String email) {
94      log.info("Sending data to server to check if email is already in use: {}", email
95      return registrationService.isEmailAlreadyInUse(email);
96  }
97
98  // UserService methods
    no usages
99  public User getUserById(Long id) {
100     log.info("Sending data to server to get user by id: {}", id);
101     return userService.getUserById(id);
102 }
103
    no usages
104 public List<User> getAllUsers() {
105     log.info("Sending data to server to get all users");
106     return userService.getAllUsers();
107 }
108
    no usages
109 public boolean userExists(Long userId) {
110     log.info("Sending data to server to check if user exists with id: {}", userId);
111     return userService.userExists(userId);
112 }
113
    no usages
114 public User getUserByToken(String token) {
115     log.info("Sending data to server to get user by token: {}", token);
116     return userService.getUserByToken(token);
117 }
118
```

Рис. 1.3 – Код класу CollectivePurchasesFacade

```java
public class ItemController {
    private final CollectivePurchasesFacade collectivePurchasesFacade;

    @PostMapping("/create")
    public ResponseEntity<Object> createItem(@RequestBody ItemDto itemDto) {
        ItemDto createdItem = collectivePurchasesFacade.createItem(itemDto);
        return ResponseEntity.ok().body(Map.of( k1: "message",  v1: "Item created successfully",  k2: "item", createdItem));
    }

    @GetMapping("/all")
    public ResponseEntity<Object> getAllItems() {
        List<ItemDto> items = collectivePurchasesFacade.getAllItems();
        return ResponseEntity.ok(Map.of( k1: "items", items));
    }

    @GetMapping("/{itemId}")
    public ResponseEntity<Object> getItemById(@PathVariable Long itemId) {
        ItemDto item = collectivePurchasesFacade.getItemById(itemId);
        return ResponseEntity.ok(Map.of( k1: "item", item));
    }

    @PutMapping("/update/{itemId}")
    public ResponseEntity<Object> updateItem(@PathVariable Long itemId, @RequestBody Item item) {
        ItemDto updatedItem = collectivePurchasesFacade.updateItem(itemId, item);
        return ResponseEntity.ok().body(Map.of( k1: "message",  v1: "Item updated successfully",  k2: "item", updatedItem));
    }

    @DeleteMapping("/delete/{itemId}")
    public ResponseEntity<Object> deleteItem(@PathVariable Long itemId) {
        collectivePurchasesFacade.deleteItem(itemId);
        return ResponseEntity.ok().body(Map.of( k1: "message",  v1: "Item deleted successfully"));
    }

    @GetMapping("/grouped-by-category")
    public ResponseEntity<Object> getItemsGroupedByCategory() {
        Map<String, List<ItemDto>> itemsGroupedByCategory = collectivePurchasesFacade.getItemsGroupedByCategory();
        return ResponseEntity.ok(Map.of( k1: "itemsGroupedByCategory", itemsGroupedByCategory));
    }
}
```

Рис. 2 – Приклад використання CollectivePurchasesFacade

```
o.s.web.servlet.DispatcherServlet       : Initializing Servlet 'dispatcherServlet'
o.s.web.servlet.DispatcherServlet       : Completed initialization in 1 ms
o.e.c.facade.CollectivePurchasesFacade  : Sending data to server to create item: ItemDto(id=null, name=Example Item, category=Example Category, price=10.0, quantity=5.0)

o.e.c.models.composite.Item             : Setting name: Example Item
o.e.c.models.composite.Item             : Setting price: 10.0
c1_0 where c1_0.category_id=?
o.e.c.models.composite.Item             : Setting category: Category(id=1, name=Example Category, components=[])
o.e.c.models.composite.Item             : Building Item with id: null, name: Example Item, category: Category(id=1, name=Example Category, components=[]), price: 10.0, quantity: 0.0
```

Рис. 3 – Приклад роботи CollectivePurchasesFacade

Зображення структури шаблону

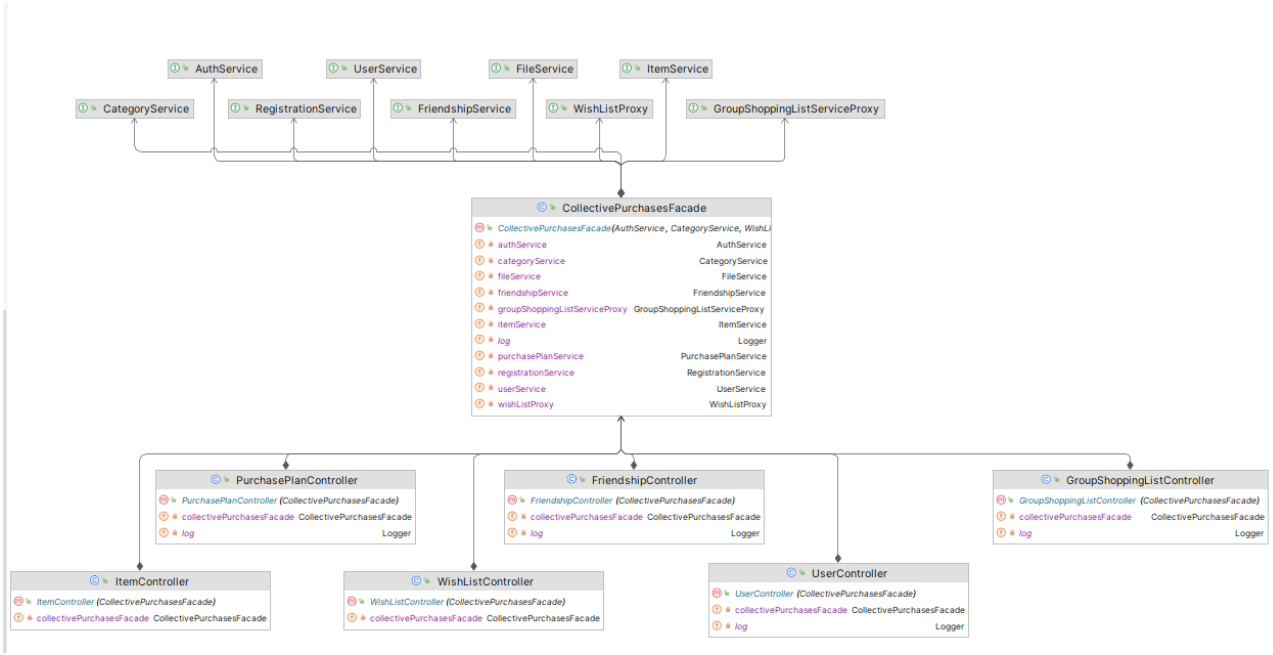


Рис. 4 – Структура шаблону

Посилання на репозиторій: https://github.com/annaorlovskaaa/TRPZ_labs.git

Висновок:

У даній лабораторній роботі ми реалізували і дослідили кілька структурних і поведінкових шаблонів проєктування, включаючи **Mediator**, **Facade**, **Bridge**, та **Template Method**. Впровадження цих шаблонів дозволило оптимізувати архітектуру програмного забезпечення, покращити його підтримуваність та розширюваність, а також забезпечити чітке розділення відповідальностей між компонентами.

Facade (Фасад)

Шаблон *Facade* було використано для спрощення взаємодії між підсистемами та надання клієнтам єдиного інтерфейсу доступу. Він забезпечує зменшення складності архітектури, приховуючи деталі реалізації підсистем і зменшуючи кількість залежностей. Завдяки цьому шаблону:

- Ми змогли інкапсулювати складну логіку взаємодії між підсистемами.
- Полегшили використання системи для зовнішніх клієнтів шляхом надання єдиної точки доступу.
- Підвищили гнучкість у випадках необхідності модифікації окремих підсистем, оскільки зовнішні клієнти залишаються незалежними від деталей реалізації.

  Використання *Facade* є важливим кроком у зменшенні технічного боргу та покращенні загальної якості коду.