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 №4

Технології розробки програмного забезпечення

«ШАБЛОНИ «SINGLETON», «ITERATOR», «PROXY», «STATE», «STRATEGY»»

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

Київ 2024

Зміст

Тема: ШАБЛОНИ «SINGLETON», «ITERATOR», «PROXY», «STATE», «STRATEGY»

Короткі теоретичні відомості

Singleton (Одинак) – це шаблон проєктування, який гарантує, що клас матиме лише один екземпляр, і забезпечує глобальну точку доступу до цього екземпляра. Він використовується в ситуаціях, коли необхідно централізовано керувати доступом до певного ресурсу, наприклад, конфігурації або логування. Важливим аспектом є забезпечення потокобезпеки у багатопоточному середовищі.

Iterator (Ітератор) – це поведінковий шаблон, який надає спосіб послідовного доступу до елементів колекції без розкриття її внутрішньої структури. Він дозволяє обходити колекцію в різних напрямках (прямо, назад) і приховує складність реалізації самої структури даних.

Proxy (Замісник) – це структурний шаблон, який представляє об'єкт-замісник, що контролює доступ до реального об'єкта. Використовується для оптимізації роботи, наприклад, у разі відкладеної ініціалізації або контролю доступу. Замісник може додавати функціонал, наприклад, кешування чи авторизацію, перед викликом реального об'єкта.

State (Стан) – це поведінковий шаблон, який дозволяє об'єкту змінювати свою поведінку залежно від свого стану. Вся логіка поведінки розділена між окремими класами станів, що полегшує масштабування та підтримку. Ключовою особливістю є те, що об'єкт змінює свій клас під час виконання, дозволяючи керувати переходами між станами.

Strategy (Стратегія) – це поведінковий шаблон, який визначає сімейство алгоритмів, інкапсулює кожен з них і робить їх взаємозамінними. Це дозволяє змінювати спосіб виконання операцій без зміни коду клієнта. Шаблон використовується для реалізації гнучкої архітектури, де можна вибирати різні стратегії динамічно під час виконання.

Хід роботи

***Система для колективних покупок*(proxy, builder, decorator, façade, 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.

Реалізація шаблону проєктування для майбутньої системи

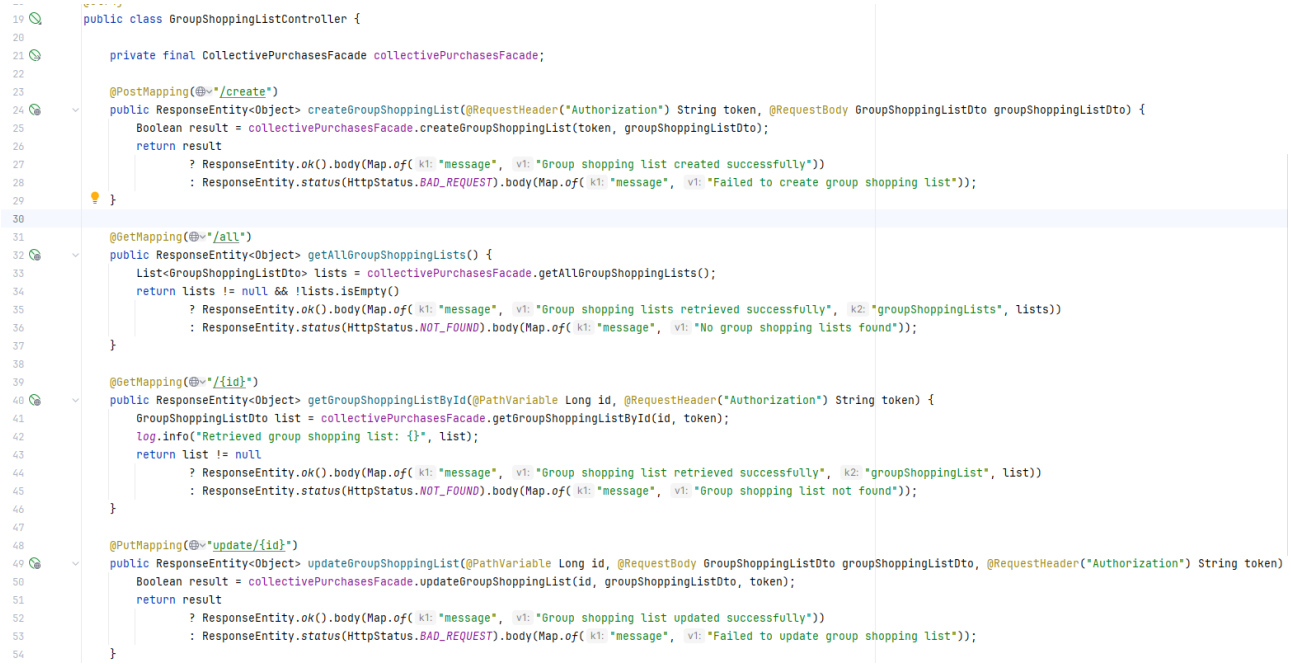
Шаблон **Proxy (Замісник)** є одним із структурних шаблонів проєктування. Його основна мета – створити об'єкт-замісник, який контролює доступ до реального об'єкта, додаючи додаткову функціональність або обмеження. Замісник виступає посередником між клієнтом і реальним об'єктом, зберігаючи ту саму інтерфейсну структуру, що й реальний об'єкт.

Основні принципи та застосування Proxy:

1. **Контроль доступу:** Proxy дозволяє додати механізми аутентифікації чи авторизації перед тим, як клієнт отримає доступ до основного об'єкта.
2. **Оптимізація ресурсів:** Реальний об'єкт може бути створений лише тоді, коли це дійсно необхідно (наприклад, у випадку відкладеної ініціалізації).
3. **Логування та моніторинг:** Proxy може вести журнал викликів методів або здійснювати моніторинг запитів.
4. **Кешування:** Замісник може зберігати проміжні дані, щоб уникнути надмірних викликів до реального об'єкта.

У класі GroupShoppingListServiceProxyImpl Proxy використовується для перевірки доступу користувача до групового списку покупок. Перед викликом

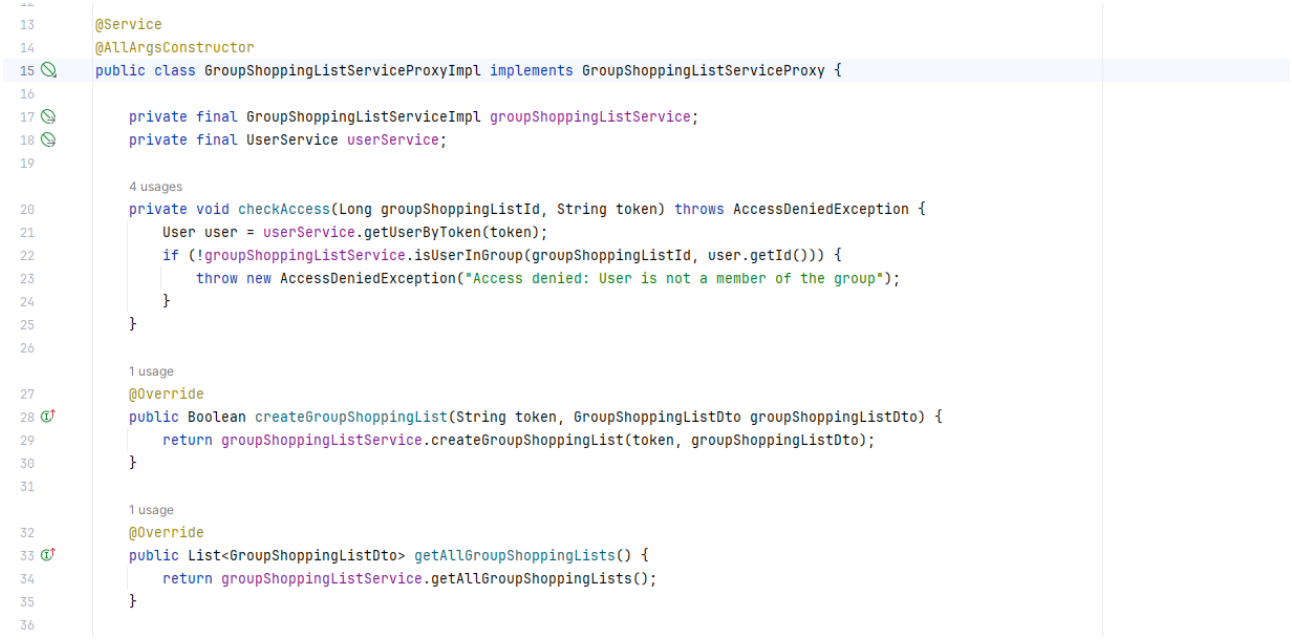
основних методів сервісу, проксі-клас перевіряє, чи користувач є членом групи. Якщо доступ дозволено, виклик перенаправляється до реального сервісу (GroupShoppingListServiceImpl), інакше – викидається виняток. Таким чином, Proxy додає рівень безпеки та спрощує основну логіку сервісу, залишаючи перевірку доступу за замісником.



```
public class GroupShoppingListController {

    private final CollectivePurchasesFacade collectivePurchasesFacade;

    @PostMapping("/create")
    public ResponseEntity<Object> createGroupShoppingList(@RequestHeader("Authorization") String token, @RequestBody GroupShoppingListDto groupShoppingListDto) {
        Boolean result = collectivePurchasesFacade.createGroupShoppingList(token, groupShoppingListDto);
        return result
                ? ResponseEntity.ok().body(Map.of( k1: "message",  v1: "Group shopping list created successfully"))
                : ResponseEntity.status(HttpStatus.BAD_REQUEST).body(Map.of( k1: "message",  v1: "Failed to create group shopping list"));
    }

    @GetMapping("/all")
    public ResponseEntity<Object> getAllGroupShoppingLists() {
        List<GroupShoppingListDto> lists = collectivePurchasesFacade.getAllGroupShoppingLists();
        return lists != null && !lists.isEmpty()
                ? ResponseEntity.ok().body(Map.of( k1: "message",  v1: "Group shopping lists retrieved successfully",  k2: "groupShoppingLists", lists))
                : ResponseEntity.status(HttpStatus.NOT_FOUND).body(Map.of( k1: "message",  v1: "No group shopping lists found"));
    }

    @GetMapping("/{id}")
    public ResponseEntity<Object> getGroupShoppingListById(@PathVariable Long id, @RequestHeader("Authorization") String token) {
        GroupShoppingListDto list = collectivePurchasesFacade.getGroupShoppingListById(id, token);
        log.info("Retrieved group shopping list: {}", list);
        return list != null
                ? ResponseEntity.ok().body(Map.of( k1: "message",  v1: "Group shopping list retrieved successfully",  k2: "groupShoppingList", list))
                : ResponseEntity.status(HttpStatus.NOT_FOUND).body(Map.of( k1: "message",  v1: "Group shopping list not found"));
    }

    @PutMapping("update/{id}")
    public ResponseEntity<Object> updateGroupShoppingList(@PathVariable Long id, @RequestBody GroupShoppingListDto groupShoppingListDto, @RequestHeader("Authorization") String token)
        Boolean result = collectivePurchasesFacade.updateGroupShoppingList(id, groupShoppingListDto, token);
        return result
                ? ResponseEntity.ok().body(Map.of( k1: "message",  v1: "Group shopping list updated successfully"))
                : ResponseEntity.status(HttpStatus.BAD_REQUEST).body(Map.of( k1: "message",  v1: "Failed to update group shopping list"));
    }
```

Рис. 1 – Код класу GroupShoppingConntroller

Контролер використовує проксі-сервіс для виконання операцій з груповими списками покупок.



```java
@Service
@AllArgsConstructor
public class GroupShoppingListServiceProxyImpl implements GroupShoppingListServiceProxy {

    private final GroupShoppingListServiceImpl groupShoppingListService;
    private final UserService userService;

    4 usages
    private void checkAccess(Long groupShoppingListId, String token) throws AccessDeniedException {
        User user = userService.getUserByToken(token);
        if (!groupShoppingListService.isUserInGroup(groupShoppingListId, user.getId())) {
            throw new AccessDeniedException("Access denied: User is not a member of the group");
        }
    }

    1 usage
    @Override
    public Boolean createGroupShoppingList(String token, GroupShoppingListDto groupShoppingListDto) {
        return groupShoppingListService.createGroupShoppingList(token, groupShoppingListDto);
    }

    1 usage
    @Override
    public List<GroupShoppingListDto> getAllGroupShoppingLists() {
        return groupShoppingListService.getAllGroupShoppingLists();
    }
```

Рис. 2 – Код класу GroupShoppingListServiceProxyImpl

Перш ніж передати виклик до реального сервісу (GroupShoppingListServiceImpl), проксі-клас перевіряє, чи є користувач членом групи, до якої належить список покупок. Якщо перевірка успішна, запит передається до реального сервісу. У протилежному випадку викликається виняток, і метод не виконується Після успішної передачі запиту реальний сервіс обробляє запит і повертає результат через проксі.

```java
@Service
@AllArgsConstructor
@Slf4j
public class GroupShoppingListServiceImpl implements GroupShoppingListService {

    private final GroupShoppingListRepository groupShoppingListRepository;
    private final UserService userService;

    1 usage
    @Override
    public Boolean createGroupShoppingList(String token, GroupShoppingListDto groupShoppingListDto) {
        log.info("Creating group shopping list with name: {}", groupShoppingListDto.getName());

        User user = userService.getUserByToken(token);
        if (user == null) {
            log.warn("User not found for token: {}", token);
            return false;
        }

        GroupShoppingList groupShoppingList = buildGroupShoppingList(groupShoppingListDto);
        groupShoppingList.setUser(user);
        groupShoppingListRepository.save(groupShoppingList);
        return true;
    }


    1 usage
    @Override
    public List<GroupShoppingListDto> getAllGroupShoppingLists() {
        log.info("Retrieving all group shopping lists");
        List<GroupShoppingList> lists = groupShoppingListRepository.findAll();
        return lists.stream() Stream<GroupShoppingList>
                .map(this::buildGroupShoppingListDto) Stream<GroupShoppingListDto>
                .collect(Collectors.toList());
    }
```

Рис. 3 – Код класу GroupShoppingConntroller

Зображення структури шаблону

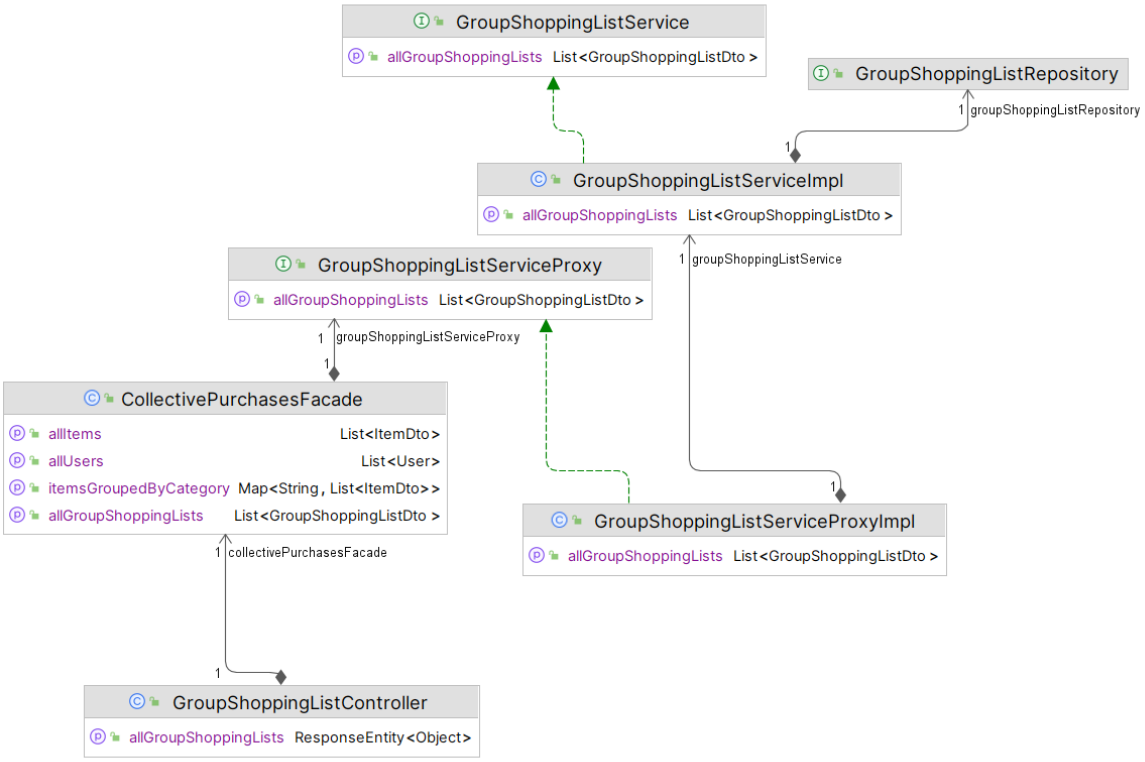


Рис. 4 – Структура шаблону

Посилання на репозиторій:

https://github.com/annaorlovskaaa/collectivePurchases.git

Висновок: У процесі виконання цієї лабораторної роботи було проаналізовано структуру та призначення п'яти основних шаблонів проєктування: Singleton, Iterator, Proxy, State та Strategy. Кожен із цих шаблонів має свої сильні та слабкі сторони, а їх правильне застосування дозволяє значно підвищити ефективність розробки програмного забезпечення.

Також було детально розглянуто саме призначення кожного шаблону, їх переваги та недоліки. На основі отриманих знань було реалізовано проксі-клас GroupShoppingListServiceProxyImpl, який застосовує шаблон Proxy. Завдяки цьому вдалося реалізувати перевірку прав доступу (чи є користувач членом групи) перед викликом методів реального сервісу. Використання шаблону Proxy дозволило досягти чіткого розподілу відповідальності між перевіркою прав доступу та основною бізнес-логікою, що сприяє підтримуваності та масштабованості коду.