Лекция 15

Практика ООП: разработка мини-проекта

1. Введение в практическое применение ООП

Объектно-ориентированное программирование (ООП) — это мощная парадигма, которая помогает разработчикам создавать гибкие, поддерживаемые и легко расширяемые системы. В рамках этой лекции мы разработаем мини-проект, используя принципы ООП, такие как инкапсуляция, наследование и полиморфизм. Мы также применим паттерны проектирования и принципы SOLID, что позволит создать структурированный и устойчивый к изменениям код.

2. Описание мини-проекта: Система управления библиотекой

Мы создадим приложение для управления библиотекой, которое будет выполнять следующие функции:

- Управлять списком книг и пользователей.
- Предоставлять возможность пользователям брать книги и возвращать их.
- Поддерживать запись об истории выданных книг.
- Реализовать простой интерфейс для добавления книг и пользователей.

3. Структура проекта и анализ требований

Для организации проекта определим несколько ключевых классов:

1. **Book** — класс, представляющий книгу, которая будет содержать информацию о названии, авторе и статусе (выдана/доступна).
2. **User** — класс, представляющий пользователя библиотеки.
3. **Library** — основной класс, который будет управлять книгами и пользователями.
4. **TransactionHistory** — класс для ведения истории операций.

Проект будет организован с учетом следующих требований:

- Пользователь может взять книгу, если она доступна.
- Пользователь может вернуть книгу, если она была выдана ему.
- Библиотека должна вести историю всех операций выдачи и возврата книг.

4. Создание классов проекта

4.1 Класс Book

Класс Book будет содержать атрибуты для хранения информации о книге и метод для изменения её статуса.

python
Копировать код

```
class Book:
    def __init__(self, title, author):
        self.title = title
        self.author = author
        self.is_available = True

    def borrow(self):
        if self.is_available:
            self.is_available = False
            return True
        return False

    def return_book(self):
        self.is_available = True

    def __str__(self):
        status = "доступна" if self.is_available else "выдана"
        return f"{self.title} by {self.author} ({status})"
```

Методы класса Book:

- borrow() — изменяет статус книги на «выдана».
- return_book() — изменяет статус книги на «доступна».
- __str__() — возвращает строковое представление книги.

4.2 Класс User

Класс User будет представлять пользователя библиотеки, содержащего ID, имя и список книг, которые пользователь взял.

python
Копировать код

```
class User:
    def __init__(self, user_id, name):
        self.user_id = user_id
        self.name = name
        self.borrowed_books = []
```

```
    def borrow_book(self, book):
        if book.borrow():
            self.borrowed_books.append(book)
            return True
        return False

    def return_book(self, book):
        if book in self.borrowed_books:
            book.return_book()
            self.borrowed_books.remove(book)
            return True
        return False

    def __str__(self):
        return f"Пользователь: {self.name} (ID: {self.user_id})"
```

Методы класса User:

- borrow_book() — добавляет книгу в список взятых книг.
- return_book() — удаляет книгу из списка взятых книг и возвращает её в библиотеку.
- __str__() — возвращает строковое представление пользователя.

4.3 Класс TransactionHistory

Этот класс будет вести запись операций (взятие и возврат книг), чтобы хранить информацию о том, кто и когда брал книги.

python
Копировать код

```
from datetime import datetime

class TransactionHistory:
    def __init__(self):
        self.history = []

    def record_borrow(self, user, book):
        entry = {
            "action": "взял",
            "user": user.name,
            "book": book.title,
            "date": datetime.now()
        }
        self.history.append(entry)
```

```
    def record_return(self, user, book):
        entry = {
            "action": "вернул",
            "user": user.name,
            "book": book.title,
            "date": datetime.now()
        }
        self.history.append(entry)

    def show_history(self):
        for entry in self.history:
            date = entry["date"].strftime("%Y-%m-%d %H:%M:%S")
            print(f"{entry['user']} {entry['action']} книгу '{entry['book']}' ({date})")
```

Методы класса TransactionHistory:

- record_borrow() — добавляет запись о взятии книги.
- record_return() — добавляет запись о возврате книги.
- show_history() — отображает историю операций.

4.4 Класс Library

Класс Library будет управлять книгами, пользователями и журналом операций.

python
Копировать код

```
class Library:
    def __init__(self):
        self.books = []
        self.users = []
        self.history = TransactionHistory()

    def add_book(self, title, author):
        book = Book(title, author)
        self.books.append(book)

    def add_user(self, user_id, name):
        user = User(user_id, name)
        self.users.append(user)

    def find_user(self, user_id):
        for user in self.users:
            if user.user_id == user_id:
```

```
                return user
        return None

    def find_book(self, title):
        for book in self.books:
            if book.title == title:
                return book
        return None

    def borrow_book(self, user_id, book_title):
        user = self.find_user(user_id)
        book = self.find_book(book_title)
        if user and book:
            if user.borrow_book(book):
                self.history.record_borrow(user, book)
                print(f"Книга '{book.title}' выдана пользователю {user.name}.")
            else:
                print(f"Книга '{book.title}' уже выдана.")
        else:
            print("Пользователь или книга не найдены.")

    def return_book(self, user_id, book_title):
        user = self.find_user(user_id)
        book = self.find_book(book_title)
        if user and book:
            if user.return_book(book):
                self.history.record_return(user, book)
                print(f"Книга '{book.title}' возвращена пользователем {user.name}.")
            else:
                print("Пользователь не брал эту книгу.")
        else:
            print("Пользователь или книга не найдены.")
```

Методы класса Library:

- add_book() — добавляет новую книгу в библиотеку.
- add_user() — регистрирует нового пользователя.
- find_user() и find_book() — помогают находить пользователей и книги по ID или названию.
- borrow_book() и return_book() — позволяют пользователям брать и возвращать книги.

5. Применение принципов SOLID в проекте

Принципы SOLID помогают улучшить качество кода и поддерживаемость проекта.

- **SRP (Принцип единственной ответственности)**: Каждый класс выполняет строго определенную задачу — Book отвечает за информацию о книге, User управляет пользователями, а TransactionHistory ведет журнал операций.
- **OCP (Принцип открытости/закрытости)**: Если потребуется расширить функциональность, можно добавить новые классы (например, для новых типов пользователей или книг), не изменяя существующие.
- **LSP (Принцип подстановки Лисков)**: Если будут созданы подклассы Book (например, для электронных или печатных книг), они должны сохранять поведение Book.
- **ISP (Принцип разделения интерфейса)**: Методы класса Library могут быть организованы в интерфейсы, если требуется разделить операции управления книгами и пользователями.
- **DIP (Принцип инверсии зависимостей)**: Класс Library зависит от абстракции TransactionHistory, что облегчает замену истории транзакций на другой класс без изменения основного кода.

6. Пример использования системы

Теперь мы можем протестировать разработанную систему, добавив пользователей и книги, а также выполнив операции.

python
Копировать код
```
# Создание библиотеки
library = Library()

# Добавление книг
library.add_book("1984", "George Orwell")
library.add_book("To Kill a Mockingbird", "Harper Lee")

# Добавление пользователей
library.add_user(1, "Alice")
library.add_user(2, "Bob")

# Взятие книги пользователем
library.borrow_book(1, "1984")

# Попытка взять уже выданную книгу
library.borrow_book(2, "1984")
```

```
# Возврат книги
library.return_book(1, "1984")

# Просмотр истории операций
library.history.show_history()
```

7. Заключение

На примере создания системы управления библиотекой мы рассмотрели ключевые аспекты применения ООП в проектировании и разработке программного обеспечения. Использование классов для представления различных сущностей (книги, пользователи, библиотека) позволяет легко управлять данными и логикой приложения. Применение принципов SOLID делает код более гибким и поддерживаемым, позволяя легко адаптировать его под изменяющиеся требования.