Паттерн MVC

Паттерн MVC (Model-View-Controller) — это архитектурный шаблон, который используется для разделения приложения на три взаимосвязанных компонента: модель (Model), представление (View) и контроллер (Controller). Этот подход помогает организовать код, сделать его более понятным и упростить поддержку и масштабирование.

1. Модель (Model)

Модель отвечает за управление данными и бизнес-логикой. Она получает данные, обрабатывает их и предоставляет их в нужном формате для представления.

Основные задачи модели:

- Получение данных из базы данных или других источников.
- Обработка данных и выполнение бизнес-логики.
- Управление состоянием данных.

```python
class Product:
    def __init__(self, name, price):
        self.name = name
        self.price = price

    def apply_discount(self, percentage):
        self.price -= self.price * (percentage / 100)
```

2. Представление (View)

Представление отвечает за отображение данных пользователю. Оно получает данные из модели и форматирует их для показа.

Основные задачи представления:

- Отображение данных в удобном для пользователя виде.
- Обновление интерфейса при изменении данных.

```python
class ProductView:
    def show_product(self, product):
        print(f"Product: {product.name}")
        print(f"Price: ${product.price:.2f}")

    def show_discount(self, discount):
        print(f"Discount applied: {discount}%")
```

3. Контроллер (Controller)

Контроллер отвечает за взаимодействие между моделью и представлением. Он принимает пользовательские вводы, обрабатывает их, изменяет данные в модели и обновляет представление.

Основные задачи контроллера:

- Обработка пользовательских запросов.
- Взаимодействие с моделью для изменения данных.
- Обновление представления после изменения данных.

```python
class ProductController:
    def __init__(self, model, view):
        self.model = model
        self.view = view

    def set_discount(self, percentage):
        self.model.apply_discount(percentage)
        self.view.show_discount(percentage)
        self.view.show_product(self.model)
```

Пример использования MVC. Объединение всех компонентов:

```
# Создание модели
product = Product("Laptop", 1500.00)

# Создание представления
view = ProductView()

# Создание контроллера
controller = ProductController(product, view)

# Отображение продукта до применения скидки
view.show_product(product)

# Применение скидки через контроллер
controller.set_discount(10)
```

Дополнительные аспекты паттерна MVC

1. Обратная связь (Observer Pattern):
   - Модель может использовать паттерн наблюдатель для уведомления представления об изменениях данных.
2. Разделение логики представления:
   - В реальных приложениях логика представления может быть разделена на несколько частей, таких как шаблоны и компоненты.
3. Применение в web-разработке:

В web-разработке MVC часто используется в фреймворках, таких как Django и Flask, где:

- Модель представляет собой ORM модели.
- Представление может быть HTML-шаблоном.
- Контроллер представлен в виде view-функций или методов классов.

Заключение

Паттерн MVC помогает организовать код, сделать его модульным и поддерживаемым. Он четко разделяет обязанности между различными компонентами, что упрощает разработку и тестирование приложения.

Задание: Управление библиотекой. Приложение будет включать добавление книг, просмотр всех книг и поиск книг по названию.

- model.py: Содержит классы Book и Library. Book управляет данными о книге, а Library управляет коллекцией книг.
- view.py: Содержит класс LibraryView, который отвечает за взаимодействие с пользователем: отображение книг, вывод сообщений и получение данных от пользователя.
- controller.py: Содержит класс LibraryController, который управляет взаимодействием между моделью и представлением. Метод main() создает экземпляры модели, представления и контроллера, а затем запускает меню для взаимодействия с пользователем.

Задание: Управление студентами и курсами

Создайте приложение для управления студентами и курсами. Приложение должно позволять добавлять студентов, добавлять курсы, записывать студентов на курсы, а также просматривать список студентов и курсов.

- model.py: Содержит классы Student, Course и School. Student и Course управляют данными о студентах и курсах, а School управляет коллекцией студентов и курсов.
- view.py: Содержит класс SchoolView, который отвечает за взаимодействие с пользователем: отображение студентов и курсов, вывод сообщений и получение данных от пользователя.
- controller.py: Содержит класс SchoolController, который управляет взаимодействием между моделью и представлением. Метод main() создает экземпляры модели, представления и контроллера, а затем запускает меню для взаимодействия с пользователем.