Министерство цифрового развития, связи и массовых коммуникаций Российской Федерации
Ордена Трудового Красного Знамени федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский технический университет связи и информатики»
(МТУСИ)

Кафедра «Программная инженерия»
дисциплина «Рефакторинг баз данных и приложений»

Лабораторная работа № 4

«Рефакторинг приложений с изучением перекрестного код-ревью»

Выполнил: студент группы БПИ2303
Березовец Илья Сергеевич

Москва 2025

Цель работы:
1. Ознакомиться с основными принципами и задачами код-ревью.
2. Научиться проводить перекрестное код-ревью, выявлять проблемы в коде и предлагать улучшения.
3. Применить полученные рекомендации для рефакторинга кода.
4. Развить навыки командной работы и взаимодействия в процессе разработки.

Ход работы:

Код до рефакторинга:

```python
class Shop:
    def __init__(self):
        self.products = []
        self.order_manager = OrderManager()

    def add_product(self, name, price, quantity):
        new_product = Product(name, price, quantity)
        self.products.append(new_product)


    def get_product(self, product_name):
        for product in self.products:
            if product.name == product_name:
                return product
        return None

    def place_order(self, product_name, quantity):
        product = self.get_product(product_name)
        if product is None:
            return "Товар не найден"
        if product.quantity < quantity:
            return "Недостаточно товара на складе"

        product.quantity -= quantity
        order = self.order_manager.create_order(product_name, quantity, product.price)
        return f"Заказ оформлен: {order.quantity} x {order.product_name} на сумму {order.total_price}"

    def show_orders(self):
        self.order_manager.list_orders()
```

Замечания одногруппника:
1. Избежание дублирования кода
2. Обработка ошибок с помощью исключений
3. Разделение методов на более мелкие функции

Код после рефакторинга:

```python
class Shop:
    def __init__(self):
        self.products = []
        self.order_manager = OrderManager()

    def add_product(self, name, price, quantity):
        new_product = Product(name, price, quantity)
        self.products.append(new_product)

    def get_product(self, product_name):
        """Получить товар по имени."""
        for product in self.products:
            if product.name == product_name:
                return product
        raise ValueError(f"Товар с именем {product_name} не найден")  # Исключение для ошибки

    def check_product_availability(self, product, quantity):
        """Проверка, есть ли достаточное количество товара."""
        if product.quantity < quantity:
            raise ValueError(f"Недостаточно товара на складе. Доступно {product.quantity} единиц.")

    def place_order(self, product_name, quantity):
        try:
            product = self.get_product(product_name)
            self.check_product_availability(product, quantity)

            # Создание заказа
            product.quantity -= quantity
            order = self.order_manager.create_order(product_name, quantity, product.price)
            return f"Заказ оформлен: {order.quantity} x {order.product_name} на сумму {order.total_price}"

        except ValueError as e:
            return str(e)  # Возвращаем ошибку как строку, если возникло исключение

    def show_orders(self):
        """Отобразить все заказы."""
        self.order_manager.list_orders()
```

Описание изменений:

1. Метод check_product_availability:
Вынес проверку наличия товара в отдельный метод. Это делает код более читаемым и облегчает тестирование логики проверки.

2. Исключения вместо строк с ошибками:
В методах get_product и place_order теперь используются исключения для обработки ошибок (например, если товар не найден или его недостаточно). Это улучшает стабильность программы и делает обработку ошибок более централизованной.

3. Чистая и более компактная логика в place_order:
Теперь метод place_order стал проще и понятнее, так как все проверки вынесены в отдельные методы. Это улучшает читаемость и уменьшает количество кода в одном методе.

Вывод:
Попрактиковал применение рефакторинга кода по рекомендациям одногруппника.

Ссылка на GitHub:
https://github.com/iberezaa/RBDiP