

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 № 3
«Изучение рефакторинга приложений»


Выполнил: студент группы БПИ2303
Берездовец Илья Сергеевич

Цель работы:

1. Ознакомиться с основными принципами и задачами рефакторинга.
2. Научиться выявлять проблемные участки кода (code smells) и устранять их.
3. Применить техники рефакторинга для улучшения читаемости, структуры и производительности кода.
4. Развить навыки анализа и улучшения существующего кода.

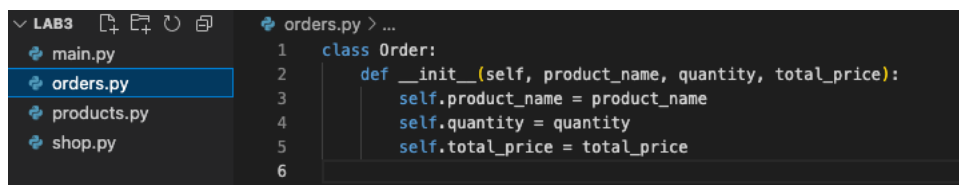
Ход работы:

1) Открыл проект до рефакторинга:



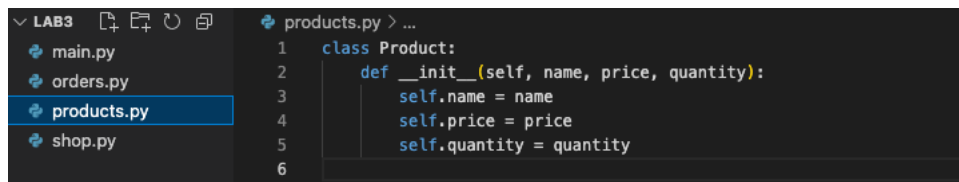
```
shop.py > ...
1 from orders import Order
2 from products import Product
3
4 class Shop:
5     def __init__(self):
6         self.products = []
7         self.orders = []
8
9     def add_product(self, name, price, quantity):
10        new_product = Product(name, price, quantity)
11        self.products.append(new_product)
12
13    def place_order(self, product_name, quantity):
14        for product in self.products:
15            if product.name == product_name:
16                if product.quantity >= quantity:
17                    product.quantity -= quantity
18                    new_order = Order(product_name, quantity, product.price * quantity)
19                    self.orders.append(new_order)
20                    return f"Заказ оформлен: {quantity} x {product_name}"
21                else:
22                    return "Недостаточно товара на складе"
23        return "Товар не найден"
24
25    def show_orders(self):
26        for order in self.orders:
27            print(f"Заказ: {order.product_name}, Количество: {order.quantity}, Сумма: {order.total_price}")
28
```

Рис. 1 – файл shop.py



```
orders.py > ...
1 class Order:
2     def __init__(self, product_name, quantity, total_price):
3         self.product_name = product_name
4         self.quantity = quantity
5         self.total_price = total_price
6
```

Рис. 2 – файл orders.py



```
products.py > ...
1 class Product:
2     def __init__(self, name, price, quantity):
3         self.name = name
4         self.price = price
5         self.quantity = quantity
6
```

Рис. 3 – файл products.py

```
LAB3
├── main.py
├── orders.py
├── products.py
└── shop.py
```

```
main.py > ...
1  from shop import Shop
2
3  shop = Shop()
4
5  shop.add_product("Телефон", 50000, 10)
6  shop.add_product("Ноутбук", 80000, 5)
7
8  print(shop.place_order("Телефон", 2))
9  print(shop.place_order("Ноутбук", 3))
10
11 shop.show_orders()
12 |
```

Рис. 4 – файл main.py

2) Выявил проблемы, которые подлежат рефакторингу.

1. Дублирование кода:

Проверка наличия товара повторяется при оформлении заказа.

2. Отсутствие обработки исключений:

Нет защиты от ввода некорректных данных (например, отрицательного количества товара).

3. Плохая читаемость и нарушение SRP (Single Responsibility Principle):

Shop выполняет сразу несколько задач: хранит товары, оформляет заказы и уменьшает их количество.

3) Провел рефакторинг кода.

```
LAB3
├── main.py
├── orders.py
├── products.py
└── shop.py

shop.py > ...
1 from products import Product
2 from orders import OrderManager
3
4 class Shop:
5     def __init__(self):
6         self.products = []
7         self.order_manager = OrderManager()
8
9     def add_product(self, name, price, quantity):
10        if price < 0 or quantity < 0:
11            raise ValueError("Цена и количество не могут быть отрицательными")
12        new_product = Product(name, price, quantity)
13        self.products.append(new_product)
14
15    def get_product(self, product_name):
16        """Возвращает продукт по названию, если он есть в магазине"""
17        for product in self.products:
18            if product.name == product_name:
19                return product
20        return None
21
22    def place_order(self, product_name, quantity):
23        """Передаёт заказ менеджеру заказов и обновляет количество товара"""
24        product = self.get_product(product_name)
25        if product is None:
26            return "Товар не найден"
27        if product.quantity < quantity:
28            return "Недостаточно товара на складе"
29
30        product.quantity -= quantity
31        order = self.order_manager.create_order(product_name, quantity, product.price)
32        return f"Заказ оформлен: {order.quantity} x {order.product_name} на сумму {order.total_price}"
33
34    def show_orders(self):
35        """Выводит список оформленных заказов"""
36        self.order_manager.list_orders()
```

Рис. 5 – shop.py после рефакторинга

```
LAB3
├── main.py
├── orders.py
├── products.py
└── shop.py

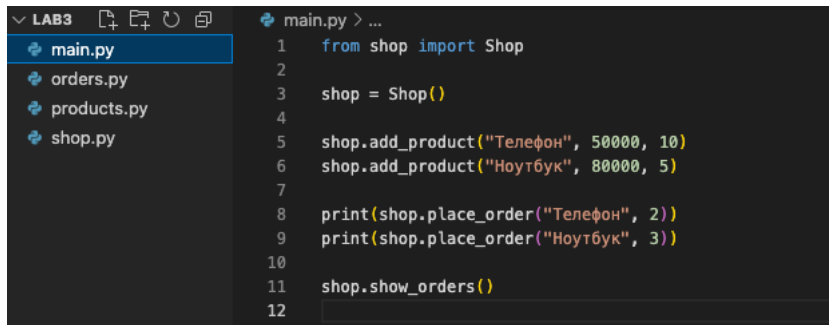
orders.py > ...
1 class Order:
2     def __init__(self, product_name, quantity, total_price):
3         self.product_name = product_name
4         self.quantity = quantity
5         self.total_price = total_price
6
7 class OrderManager:
8     def __init__(self):
9         self.orders = []
10
11    def create_order(self, product_name, quantity, price):
12        """Создает новый заказ и добавляет его в список"""
13        order = Order(product_name, quantity, price * quantity)
14        self.orders.append(order)
15        return order
16
17    def list_orders(self):
18        """Выводит список всех заказов"""
19        if not self.orders:
20            print("Заказов пока нет")
21        for order in self.orders:
22            print(f"Заказ: {order.product_name}, Количество: {order.quantity}, Сумма: {order.total_price}")
23
```

Рис. 6 – orders.py после рефакторинга

```
LAB3
├── main.py
├── orders.py
├── products.py
└── shop.py

products.py > ...
1 class Product:
2     def __init__(self, name, price, quantity):
3         if price < 0 or quantity < 0:
4             raise ValueError("Цена и количество должны быть неотрицательными")
5         self.name = name
6         self.price = price
7         self.quantity = quantity
8
```

Рис. 7 – products.py после рефакторинга



```
LAB3 main.py > ...
main.py
orders.py
products.py
shop.py

1 from shop import Shop
2
3 shop = Shop()
4
5 shop.add_product("Телефон", 50000, 10)
6 shop.add_product("Ноутбук", 80000, 5)
7
8 print(shop.place_order("Телефон", 2))
9 print(shop.place_order("Ноутбук", 3))
10
11 shop.show_orders()
12
```

Рис. 8 – main.py

4) Улучшения после рефакторинга:

1. Код стал более читаемым и разделенным по ответственности:
Shop управляет товарами.
OrderManager управляет заказами.
Product хранит данные о продукте с валидацией.
2. Устранено дублирование кода:
Логика заказов теперь находится в OrderManager, а не в Shop.
Функция `get_product()` теперь возвращает продукт, вместо поиска вручную.
3. Добавлены проверки:
Product теперь не может иметь отрицательную цену или количество.
Shop.add_product() тоже проверяет корректность данных.

5) Добавил тесты:

```
LABS
> venv
main.py
orders.py
products.py
shop.py
test_shop.py

test_shop.py > TestShop > test_place_order_product_not_found
1 import unittest
2 from shop import Shop
3 from orders import OrderManager
4 from products import Product
5
6 class TestShop(unittest.TestCase):
7
8     def setUp(self):
9         """Создаёт тестовый магазин перед каждым тестом"""
10        self.shop = Shop()
11        self.shop.add_product("Телефон", 50000, 10)
12        self.shop.add_product("Ноутбук", 80000, 5)
13
14    def test_add_product(self):
15        """Тест добавления товаров"""
16        self.shop.add_product("Планшет", 30000, 7)
17        product = self.shop.get_product("Планшет")
18        self.assertIsNotNone(product)
19        self.assertEqual(product.price, 30000)
20        self.assertEqual(product.quantity, 7)
21
22    def test_place_order_success(self):
23        """Тест успешного оформления заказа"""
24        result = self.shop.place_order("Телефон", 2)
25        self.assertEqual(result, "Заказ оформлен: 2 x Телефон на сумму 100000")
26
27        product = self.shop.get_product("Телефон")
28        self.assertEqual(product.quantity, 8)
29
30    def test_place_order_insufficient_stock(self):
31        """Тест оформления заказа при недостаточном количестве товара"""
32        result = self.shop.place_order("Ноутбук", 10)
33        self.assertEqual(result, "Недостаточно товара на складе")
34
35    def test_place_order_product_not_found(self):
36        """Тест оформления заказа на несуществующий товар"""
37        result = self.shop.place_order("Камера", 1)
38        self.assertEqual(result, "Товар не найден")
39
40    def test_product_negative_values(self):
41        """Тест попытки создания товара с отрицательной ценой или количеством"""
42        with self.assertRaises(ValueError):
43            Product("Часы", -1000, 5)
44        with self.assertRaises(ValueError):
45            Product("Часы", 1000, -5)
46
47    def test_order_manager_create_order(self):
48        """Тест создания заказа через OrderManager"""
49        order_manager = OrderManager()
50        order = order_manager.create_order("Мышка", 2, 1500)
51        self.assertEqual(order.product_name, "Мышка")
52        self.assertEqual(order.quantity, 2)
53        self.assertEqual(order.total_price, 3000)
54
55    if __name__ == '__main__':
56        unittest.main()
57
```

Рис. 9 – файл test_shop.py (тесты)

Тесты прошли успешно:

```
(venv) berezoff@Mac lab3 % python -m unittest test_shop.py
.....
Ran 6 tests in 0.000s
OK
```

Вывод:

В ходе выполнения лабораторной работы был проведен рефакторинг исходного кода с целью улучшения его читаемости и структуры. Были устранены проблемы дублирования кода, большие функции были разделены на более мелкие, а имена переменных и методов приведены к более понятному виду. Добавлена обработка ошибок для повышения надежности. В завершение были написаны и успешно выполнены юнит-тесты для проверки функциональности основных методов классов. Работа позволила развить навыки рефакторинга и улучшить качество кода.

Ссылка на GitHub: <https://github.com/iberezaa>