Московский государственный технический университет им. Н.Э. Баумана

Факультет “Информатика и системы управления”
Кафедра “Системы обработки информации и управления”

Курс «Парадигмы и конструкции языков программирования»

Отчет по лабораторной работе №4

«Шаблоны проектирования и модульное тестирование в Python»

Выполнил:
студент группы ИУ5 - 32Б:
Васильев Н. Д.
Подпись и дата:

Проверил:
преподаватель каф. ИУ5
Гапанюк Ю. Е.
Подпись и дата:

Москва, 2025 г.

Задание:

1. Необходимо для произвольной предметной области реализовать от одного до трех шаблонов проектирования: один порождающий, один структурный и один поведенческий. В качестве справочника шаблонов можно использовать следующий каталог. Для сдачи лабораторной работы в минимальном варианте достаточно реализовать один паттерн.
2. В модульных тестах необходимо применить следующие технологии:
    - TDD - фреймворк.
    - BDD - фреймворк.
    - Создание Mock-объектов.

Код программы:

Файл main.py

```
from abc import ABC, abstractmethod
from enum import Enum
from typing import List


class ProductCategory(Enum):
    ELECTRONICS = "электроника"
    CLOTHING = "одежда"
    BOOKS = "книги"


class Product(ABC):
    def __init__(self, name: str, price: float):
        self.name = name
        self.price = price

    @abstractmethod
    def get_description(self) -> str:
        pass

    @abstractmethod
    def calculate_discount(self) -> float:
        pass


class ElectronicsProduct(Product):
    def get_description(self) -> str:
        return f"Электроника: {self.name} - {self.price} руб."
```

```
    def calculate_discount(self) -> float:
        return self.price * 0.9


class ClothingProduct(Product):
    def get_description(self) -> str:
        return f"Одежда: {self.name} - {self.price} руб."

    def calculate_discount(self) -> float:
        return self.price * 0.8


class BooksProduct(Product):
    def get_description(self) -> str:
        return f"Книги: {self.name} - {self.price} руб."

    def calculate_discount(self) -> float:
        return self.price * 0.85

#Порождающий патерн(Фабричный метод)
class ProductFactory:
    @staticmethod
    def create_product(category: ProductCategory, name: str, price: float) ->
Product:
        if category == ProductCategory.ELECTRONICS:
            return ElectronicsProduct(name, price)
        elif category == ProductCategory.CLOTHING:
            return ClothingProduct(name, price)
        elif category == ProductCategory.BOOKS:
            return BooksProduct(name, price)
        else:
            raise ValueError(f"Неизвестная категория: {category}")

#Структурный патерн(Адаптер)
class LegacyPaymentSystem:
    def process_payment_legacy(self, amount: float, customer_id: str) -> dict:
        return {
            "transaction_id": f"TXN_{customer_id}",
            "status": "completed",
            "amount_processed": amount,
            "customer": customer_id
        }


class IPaymentProcessor(ABC):
    @abstractmethod
    def process_payment(self, amount: float, customer_email: str) -> str:
        pass
```

```
class PaymentAdapter(IPaymentProcessor):
    def __init__(self, legacy_system: LegacyPaymentSystem):
        self.legacy_system = legacy_system

    def process_payment(self, amount: float, customer_email: str) -> str:
        legacy_result = self.legacy_system.process_payment_legacy(amount,
customer_email)
        return f"Платеж {legacy_result['transaction_id']}:
{legacy_result['status']}"

#Поведенческий патерн(Наблюдатель)
class OrderObserver(ABC):
    @abstractmethod
    def update(self, order_id: int, status: str):
        pass


class Customer(OrderObserver):
    def __init__(self, name: str, email: str):
        self.name = name
        self.email = email

    def update(self, order_id: int, status: str):
        print(f"Клиент {self.name} получил уведомление: Заказ #{order_id} -
{status}")


class Warehouse(OrderObserver):
    def update(self, order_id: int, status: str):
        if status == "обработан":
            print(f"Склад: начать сборку заказа #{order_id}")


class OrderSubject:
    def __init__(self):
        self._observers = []

    def attach(self, observer: OrderObserver):
        self._observers.append(observer)

    def detach(self, observer: OrderObserver):
        self._observers.remove(observer)

    def notify(self, order_id: int, status: str):
        for observer in self._observers:
            observer.update(order_id, status)
```

```
class Order(OrderSubject):
    _order_counter = 0

    def __init__(self, customer: Customer):
        super().__init__()
        Order._order_counter += 1
        self.order_id = Order._order_counter
        self.customer = customer
        self.products = []
        self.status = "создан"
        self.total_amount = 0.0

        self.attach(customer)
        self.attach(Warehouse())

    def add_product(self, product: Product):
        self.products.append(product)
        self.total_amount += product.price
        self.status = "обновлен"
        self.notify(self.order_id, self.status)

    def process_order(self, payment_processor: IPaymentProcessor):
        try:
            payment_result = payment_processor.process_payment(self.total_amount,
self.customer.email)
            print(f"Платеж обработан: {payment_result}")

            self.status = "обработан"
            self.notify(self.order_id, self.status)
            return True
        except Exception as e:
            self.status = "ошибка оплаты"
            self.notify(self.order_id, self.status)
            return False

    def get_order_info(self) -> str:
        products_info = "\n".join([f"  - {product.get_description()}" for product
in self.products])
        return f"""Заказ #{self.order_id}
Клиент: {self.customer.name}
Статус: {self.status}
Общая сумма: {self.total_amount} руб.
Товары:
{products_info}"""
```

```
def main():
    print("=== СИСТЕМА УПРАВЛЕНИЯ ЗАКАЗАМИ ===\n")

    customer = Customer("Иван Петров", "ivan@mail.com")
    order = Order(customer)

    laptop = ProductFactory.create_product(ProductCategory.ELECTRONICS, "Ноутбук",
50000)
    tshirt = ProductFactory.create_product(ProductCategory.CLOTHING, "Футболка",
1500)
    book = ProductFactory.create_product(ProductCategory.BOOKS, "Python
Programming", 1200)

    order.add_product(laptop)
    order.add_product(tshirt)
    order.add_product(book)

    legacy_system = LegacyPaymentSystem()
    payment_adapter = PaymentAdapter(legacy_system)

    print("Обработка заказа...")
    order.process_order(payment_adapter)

    print("\n" + "=" * 50)
    print(order.get_order_info())


if __name__ == "__main__":
    main()
```

Файл test_mocks.py

```
import unittest
from unittest.mock import Mock
from main import PaymentAdapter, LegacyPaymentSystem, Order, Customer,
ProductFactory, ProductCategory


class TestOrderSystemMocks(unittest.TestCase):

    def test_payment_adapter_with_mock(self):
        mock_legacy_system = Mock(spec=LegacyPaymentSystem)
        mock_legacy_system.process_payment_legacy.return_value = {
            "transaction_id": "TXN_12345",
            "status": "completed",
            "amount_processed": 1000.0,
            "customer": "test@mail.com"
        }

        adapter = PaymentAdapter(mock_legacy_system)
        result = adapter.process_payment(1000.0, "test@mail.com")

        self.assertIn("TXN_12345", result)
        mock_legacy_system.process_payment_legacy.assert_called_once_with(1000.0,
"test@mail.com")

    def test_order_processing_with_mock_payment(self):
        customer = Customer("Тест Клиент", "test@mail.com")
        order = Order(customer)

        product = ProductFactory.create_product(ProductCategory.ELECTRONICS,
"Телефон", 10000)
        order.add_product(product)

        mock_payment_processor = Mock()
        mock_payment_processor.process_payment.return_value = "Платеж успешно
обработан"

        result = order.process_order(mock_payment_processor)

        self.assertTrue(result)
        mock_payment_processor.process_payment.assert_called_once_with(10000,
"test@mail.com")


if __name__ == '__main__':
    unittest.main()
```

Файл test_tdd.py

```
import unittest
from main import ProductFactory, ProductCategory, ElectronicsProduct, Order,
Customer


class TestOrderSystemTDD(unittest.TestCase):

    def test_product_factory_creates_correct_type(self):
        electronics = ProductFactory.create_product(
            ProductCategory.ELECTRONICS, "Смартфон", 30000
        )
        self.assertIsInstance(electronics, ElectronicsProduct)

    def test_product_discount_calculation(self):
        clothing = ProductFactory.create_product(
            ProductCategory.CLOTHING, "Джинсы", 2000
        )
        discounted_price = clothing.calculate_discount()
        self.assertEqual(discounted_price, 1600)

    def test_order_total_amount(self):
        customer = Customer("Тест Клиент", "test@mail.com")
        order = Order(customer)

        product1 = ProductFactory.create_product(ProductCategory.ELECTRONICS,
"Планшет", 25000)
        product2 = ProductFactory.create_product(ProductCategory.BOOKS, "Книга",
500)

        order.add_product(product1)
        order.add_product(product2)

        self.assertEqual(order.total_amount, 25500)

    def test_order_auto_increment_id(self):
        customer = Customer("Тест Клиент", "test@mail.com")
        order1 = Order(customer)
        order2 = Order(customer)

        self.assertEqual(order1.order_id + 1, order2.order_id)


if __name__ == '__main__':
    unittest.main()
```

Скриншоты работы:

```
(.venv) PS C:\Users\Николай\Desktop\Python_Course_3Sem\Lab4-3Sem> python main.py
=== СИСТЕМА УПРАВЛЕНИЯ ЗАКАЗАМИ ===

Клиент Иван Петров получил уведомление: Заказ #1 - обновлен
Клиент Иван Петров получил уведомление: Заказ #1 - обновлен
Клиент Иван Петров получил уведомление: Заказ #1 - обновлен
Обработка заказа...
Платеж обработан: Платеж TXN_ivan@mail.com: completed
Клиент Иван Петров получил уведомление: Заказ #1 - обработан
Склад: начать сборку заказа #1

==================================================
Заказ #1
Клиент: Иван Петров
Статус: обработан
Общая сумма: 52700.0 руб.
Товары:
  - Электроника: Ноутбук - 50000 руб.
  - Одежда: Футболка - 1500 руб.
  - Книги: Python Programming - 1200 руб.
(.venv) PS C:\Users\Николай\Desktop\Python_Course_3Sem\Lab4-3Sem>
```

```
(.venv) PS C:\Users\Николай\Desktop\Python_Course_3Sem\Lab4-3Sem> python test_tdd.py
.Клиент Тест Клиент получил уведомление: Заказ #3 - обновлен
Клиент Тест Клиент получил уведомление: Заказ #3 - обновлен
...
----------------------------------------------------------------------
Ran 4 tests in 0.001s

OK
(.venv) PS C:\Users\Николай\Desktop\Python_Course_3Sem\Lab4-3Sem> python test_mocks.py
Клиент Тест Клиент получил уведомление: Заказ #1 - обновлен
Платеж обработан: Платеж успешно обработан
Клиент Тест Клиент получил уведомление: Заказ #1 - обработан
Склад: начать сборку заказа #1
..
----------------------------------------------------------------------
Ran 2 tests in 0.002s

OK
```