Московский государственный технический университет им. Н.Э. Баумана

Факультет «Информатика и системы управления» Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Парадигмы и конструкции языков программирования» Отчёт по лабораторной работе №4

Шаблоны проектирования и модульное тестирование в Python

Выполнил:
студент группы ИУ5-35Б
Александров Георгий

Проверил: преподаватель каф. ИУ5
Нардид Анатолий Николаевич

Москва, 2025 г.

Описание задания

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.**

Текст программы

main.py

```
from abc import ABC, abstractmethod
from typing import List
from datetime import datetime
import unittest
from unittest.mock import Mock, patch

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

    @abstractmethod
    def prepare(self):
        pass

    def __str__(self):
        return f"{self.name} - {self.price} руб."

class Pizza(MenuItem):
    def __init__(self, name: str, price: float, size: str):
        super().__init__(name, price)
        self.size = size

    def prepare(self):
        print(f"Готовим пиццу '{self.name}' размера {self.size}")
        return f"Пицца '{self.name}' готова!"

class Drink(MenuItem):
    def __init__(self, name: str, price: float, volume: float):
        super().__init__(name, price)
        self.volume = volume

    def prepare(self):
        print(f"Наливаем напиток '{self.name}' {self.volume}л")
        return f"Напиток '{self.name}' готов!"

class Dessert(MenuItem):
    def __init__(self, name: str, price: float, is_cold: bool = True):
        super().__init__(name, price)
        self.is_cold = is_cold

    def prepare(self):
        temperature = "охлажденный" if self.is_cold else "теплый"
        print(f"Готовим {temperature} десерт '{self.name}'")
        return f"Десерт '{self.name}' готов!"
```

```
class MenuItemFactory(ABC):
    @abstractmethod
    def create_item(self, *args, **kwargs) -> MenuItem:
        pass

class PizzaFactory(MenuItemFactory):
    def create_item(self, name: str, price: float, size: str = "medium") -> Pizza:
        return Pizza(name, price, size)

class DrinkFactory(MenuItemFactory):
    def create_item(self, name: str, price: float, volume: float = 0.5) -> Drink:
        return Drink(name, price, volume)

class DessertFactory(MenuItemFactory):
    def create_item(self, name: str, price: float, is_cold: bool = True) -> Dessert:
        return Dessert(name, price, is_cold)

class OrderObserver(ABC):
    @abstractmethod
    def update(self, order: 'Order'):
        pass

class KitchenObserver(OrderObserver):
    def update(self, order: 'Order'):
        print(f"[КУХНЯ] Получен новый заказ #{order.order_id}")
        for item in order.items:
            print(f"[КУХНЯ] Готовим: {item.name}")

class CustomerObserver(OrderObserver):
    def __init__(self, customer_name: str):
        self.customer_name = customer_name

    def update(self, order: 'Order'):
        print(f"[КЛИЕНТ {self.customer_name}] Статус вашего заказа #{order.order_id}: {order.status}")

class PaymentObserver(OrderObserver):
    def update(self, order: 'Order'):
        if order.status == "оплачен":
            print(f"[ПЛАТЕЖНАЯ СИСТЕМА] Заказ #{order.order_id} оплачен. Сумма:
{order.total_price} руб.")

class Order:
    _order_counter = 0

    def __init__(self):
        Order._order_counter += 1
        self.order_id = Order._order_counter
```

```
        self.items: List[MenuItem] = []
        self.status = "создан"
        self.order_time = datetime.now()
        self._observers: List[OrderObserver] = []

    def add_item(self, item: MenuItem):
        self.items.append(item)

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

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

    def notify_observers(self):
        for observer in self._observers:
            observer.update(self)

    def change_status(self, new_status: str):
        self.status = new_status
        print(f"\nЗаказ #{self.order_id} изменил статус на: {new_status}")
        self.notify_observers()

    @property
    def total_price(self):
        return sum(item.price for item in self.items)

    def __str__(self):
        items_str = "\n".join(f"  - {item}" for item in self.items)
        return f"Заказ #{self.order_id} ({self.status}):\n{items_str}\nИтого: {self.total_price} руб."

class CafeOrderSystem:
    def __init__(self):
        self.pizza_factory = PizzaFactory()
        self.drink_factory = DrinkFactory()
        self.dessert_factory = DessertFactory()
        self.kitchen = KitchenObserver()
        self.payment_system = PaymentObserver()

    def create_combo_order(self, customer_name: str, pizza_type: str = "Маргарита"):
        print(f"\nСоздание комбо-заказа для {customer_name}...")

        order = Order()
        customer = CustomerObserver(customer_name)

        order.add_observer(self.kitchen)
        order.add_observer(customer)
```

```
        order.add_observer(self.payment_system)

        pizza = self.pizza_factory.create_item(pizza_type, 450, "large")
        drink = self.drink_factory.create_item("Кола", 120, 0.5)
        dessert = self.dessert_factory.create_item("Тирамису", 200, True)

        order.add_item(pizza)
        order.add_item(drink)
        order.add_item(dessert)

        order.change_status("принят")
        order.change_status("готовится")

        print("\nНачинаем приготовление:")
        for item in order.items:
            print(f"   {item.prepare()}")

        order.change_status("готов")
        order.change_status("оплачен")
        order.change_status("выдан")

        return order

    def create_custom_order(self, customer_name: str, items_data: List[dict]):
        print(f"\nСоздание кастомного заказа для {customer_name}...")

        order = Order()
        customer = CustomerObserver(customer_name)

        order.add_observer(self.kitchen)
        order.add_observer(customer)
        order.add_observer(self.payment_system)

        factories = {
            "pizza": self.pizza_factory,
            "drink": self.drink_factory,
            "dessert": self.dessert_factory
        }

        for item_data in items_data:
            item_type = item_data.get("type")
            factory = factories.get(item_type)

            if factory:
                item_data_copy = item_data.copy()
                item_data_copy.pop("type", None)
                item = factory.create_item(**item_data_copy)
```

```
                order.add_item(item)

        order.change_status("принят")
        return order

class TestCafeSystemTDD(unittest.TestCase):
    def test_menu_item_creation(self):
        pizza = Pizza("Пепперони", 500, "large")
        self.assertEqual(pizza.name, "Пепперони")
        self.assertEqual(pizza.price, 500)
        self.assertEqual(pizza.size, "large")

    def test_factory_method(self):
        pizza_factory = PizzaFactory()
        pizza = pizza_factory.create_item("4 сыра", 550, "medium")
        self.assertIsInstance(pizza, Pizza)
        self.assertEqual(pizza.name, "4 сыра")

    def test_order_total_price(self):
        order = Order()
        pizza = Pizza("Маргарита", 450, "medium")
        drink = Drink("Сок", 100, 0.3)
        order.add_item(pizza)
        order.add_item(drink)
        self.assertEqual(order.total_price, 550)

    def test_observer_pattern(self):
        order = Order()
        mock_observer = Mock(spec=OrderObserver)
        order.add_observer(mock_observer)
        order.change_status("готовится")
        mock_observer.update.assert_called_once_with(order)

    def test_facade_pattern(self):
        cafe_system = CafeOrderSystem()
        order = cafe_system.create_combo_order("Иван Иванов")
        self.assertEqual(len(order.items), 3)
        self.assertEqual(order.status, "выдан")

class TestCafeSystemBDD:
    def test_order_flow(self):
        cafe_system = CafeOrderSystem()
        order = cafe_system.create_combo_order("Петр Петров")
        assert order.total_price > 0
        assert len(order.items) == 3
        assert order.status == "выдан"
```

```
    def test_custom_order_creation(self):
        cafe_system = CafeOrderSystem()
        items_data = [
            {"type": "pizza", "name": "Гавайская", "price": 480, "size": "medium"},
            {"type": "drink", "name": "Чай", "price": 80, "volume": 0.3}
        ]
        order = cafe_system.create_custom_order("Мария", items_data)
        assert len(order.items) == 2
        assert order.total_price == 560

class TestCafeSystemWithMocks(unittest.TestCase):
    def test_mock_factory(self):
        mock_factory = Mock(spec=MenuItemFactory)
        mock_item = Mock(spec=MenuItem)
        mock_item.name = "Тестовый продукт"
        mock_item.price = 100
        mock_factory.create_item.return_value = mock_item
        item = mock_factory.create_item(name="Тест", price=100)
        self.assertEqual(item.name, "Тестовый продукт")
        mock_factory.create_item.assert_called_once_with(name="Тест", price=100)

    @patch('__main__.KitchenObserver')
    def test_mock_kitchen_observer(self, MockKitchen):
        mock_kitchen_instance = MockKitchen.return_value
        order = Order()
        order.add_observer(mock_kitchen_instance)
        order.change_status("готовится")
        mock_kitchen_instance.update.assert_called_with(order)

    def test_mock_menu_item_preparation(self):
        mock_item = Mock(spec=MenuItem)
        mock_item.name = "Мок-пицца"
        mock_item.price = 999
        mock_item.prepare.return_value = "Мок-приготовление завершено"
        order = Order()
        order.add_item(mock_item)
        result = mock_item.prepare()
        self.assertEqual(result, "Мок-приготовление завершено")
        mock_item.prepare.assert_called_once()

def main():
    print("=" * 60)
    print("ДЕМОНСТРАЦИЯ ПАТТЕРНОВ ПРОЕКТИРОВАНИЯ")
    print("=" * 60)

    cafe_system = CafeOrderSystem()
```

```
print("\n1. Создание комбо-заказа через фасад:")
print("-" * 40)
order1 = cafe_system.create_combo_order("Анна Сидорова", "Пепперони")
print(f"\n{order1}")

print("\n2. Создание кастомного заказа:")
print("-" * 40)
custom_items = [
    {"type": "pizza", "name": "Диабло", "price": 520, "size": "large"},
    {"type": "dessert", "name": "Чизкейк", "price": 250, "is_cold": False}
]
order2 = cafe_system.create_custom_order("Сергей", custom_items)
print(f"\n{order2}")

print("\n3. Демонстрация фабричного метода:")
print("-" * 40)
pizza_factory = PizzaFactory()
drink_factory = DrinkFactory()
pizza = pizza_factory.create_item("Карбонара", 490, "medium")
drink = drink_factory.create_item("Кофе", 150, 0.2)
print(f"Создана через фабрику: {pizza}")
print(f"Создан через фабрику: {drink}")

print("\n4. Запуск модульных тестов:")
print("-" * 40)

print("Запуск TDD тестов...")
suite = unittest.TestLoader().loadTestsFromTestCase(TestCafeSystemTDD)
runner = unittest.TextTestRunner(verbosity=2)
runner.run(suite)

print("\nЗапуск BDD тестов...")
bdd_tester = TestCafeSystemBDD()
try:
    bdd_tester.test_order_flow()
    print("✓ BDD тест order_flow пройден")
except AssertionError as e:
    print(f"✗ BDD тест order_flow не пройден: {e}")

try:
    bdd_tester.test_custom_order_creation()
    print("✓ BDD тест custom_order_creation пройден")
except AssertionError as e:
    print(f"✗ BDD тест custom_order_creation не пройден: {e}")

print("\nЗапуск тестов с Mock-объектами...")
mock_suite = unittest.TestLoader().loadTestsFromTestCase(TestCafeSystemWithMocks)
```

```
    runner.run(mock_suite)

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

test_cafe.py

```
import unittest
from unittest.mock import Mock, patch
from main import *

class TestCafeSystem(unittest.TestCase):
    def setUp(self):
        self.cafe = CafeOrderSystem()
        self.test_order = Order()
        self.pizza = Pizza("Тестовая", 100, "small")
        self.drink = Drink("Тестовый", 50, 0.5)

    def test_pizza_creation(self):
        self.assertEqual(self.pizza.name, "Тестовая")
        self.assertEqual(self.pizza.price, 100)
        self.assertEqual(self.pizza.size, "small")

    def test_order_add_item(self):
        self.test_order.add_item(self.pizza)
        self.assertEqual(len(self.test_order.items), 1)
        self.assertEqual(self.test_order.items[0].name, "Тестовая")

    def test_order_total_price(self):
        self.test_order.add_item(self.pizza)
        self.test_order.add_item(self.drink)
        self.assertEqual(self.test_order.total_price, 150)

    def test_observer_addition(self):
        observer = Mock()
        self.test_order.add_observer(observer)
        self.assertEqual(len(self.test_order._observers), 1)

    def test_observer_notification(self):
        observer = Mock()
        self.test_order.add_observer(observer)
        self.test_order.change_status("готовится")
        observer.update.assert_called_once_with(self.test_order)

    def test_factory_method_pizza(self):
        factory = PizzaFactory()
        pizza = factory.create_item("Тест", 200, "large")
```

```
        self.assertIsInstance(pizza, Pizza)
        self.assertEqual(pizza.size, "large")

    def test_facade_combo_order(self):
        order = self.cafe.create_combo_order("Тестовый Клиент")
        self.assertEqual(order.status, "выдан")
        self.assertEqual(len(order.items), 3)

    @patch('main.KitchenObserver')
    def test_mock_kitchen_in_facade(self, MockKitchen):
        mock_kitchen = MockKitchen.return_value
        self.cafe.kitchen = mock_kitchen
        order = self.cafe.create_combo_order("Тест")
        self.assertTrue(mock_kitchen.update.called)

    def test_menu_item_preparation(self):
        result = self.pizza.prepare()
        self.assertIn("готов", result.lower())

def run_tests():
    suite = unittest.TestLoader().loadTestsFromTestCase(TestCafeSystem)
    runner = unittest.TextTestRunner(verbosity=2)

    print("=" * 60)
    print("ЗАПУСК МОДУЛЬНЫХ ТЕСТОВ")
    print("=" * 60)

    result = runner.run(suite)

    print(f"\nРезультаты тестов:")
    print(f"Всего тестов: {result.testsRun}")
    print(f"Пройдено: {result.testsRun - len(result.failures) - len(result.errors)}")
    print(f"Провалено: {len(result.failures)}")
    print(f"Ошибок: {len(result.errors)}")

    return result

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

Листинг кода

main.py

```
(venv) PS C:\Users\MSI-Crosshair16\Documents\лабы\lab_python_fp\lab_chabl> python main.py
============================================================
ДЕМОНСТРАЦИЯ ПАТТЕРНОВ ПРОЕКТИРОВАНИЯ
============================================================

1. Создание комбо-заказа через фасад:
--------------------------------------

Создание комбо-заказа для Анна Сидорова...

Заказ #1 изменил статус на: принят
[КУХНЯ] Получен новый заказ #1
[КУХНЯ] Готовим: Пепперони
[КУХНЯ] Готовим: Кола
[КУХНЯ] Готовим: Тирамису
[КЛИЕНТ Анна Сидорова] Статус вашего заказа #1: принят

Заказ #1 изменил статус на: готовится
[КУХНЯ] Получен новый заказ #1
[КУХНЯ] Готовим: Пепперони
[КУХНЯ] Готовим: Кола
[КУХНЯ] Готовим: Тирамису
[КЛИЕНТ Анна Сидорова] Статус вашего заказа #1: готовится

Начинаем приготовление:
Готовим пиццу 'Пепперони' размера large
  Пицца 'Пепперони' готова!
Наливаем напиток 'Кола' 0.5л
  Напиток 'Кола' готов!
Готовим охлажденный десерт 'Тирамису'
  Десерт 'Тирамису' готов!

Заказ #1 изменил статус на: готов
[КУХНЯ] Получен новый заказ #1
[КУХНЯ] Готовим: Пепперони
[КУХНЯ] Готовим: Кола
[КУХНЯ] Готовим: Тирамису
[КЛИЕНТ Анна Сидорова] Статус вашего заказа #1: готов
```

```
Заказ #1 изменил статус на: оплачен
[КУХНЯ] Получен новый заказ #1
[КУХНЯ] Готовим: Пепперони
[КУХНЯ] Готовим: Кола
[КУХНЯ] Готовим: Тирамису
[КЛИЕНТ Анна Сидорова] Статус вашего заказа #1: оплачен
[ПЛАТЕЖНАЯ СИСТЕМА] Заказ #1 оплачен. Сумма: 770 руб.

Заказ #1 изменил статус на: выдан
[КУХНЯ] Получен новый заказ #1
[КУХНЯ] Готовим: Пепперони
[КУХНЯ] Готовим: Кола
[КУХНЯ] Готовим: Тирамису
[КЛИЕНТ Анна Сидорова] Статус вашего заказа #1: выдан

Заказ #1 (выдан):
  - Пепперони - 450 руб.
  - Кола - 120 руб.
  - Тирамису - 200 руб.
Итого: 770 руб.

2. Создание кастомного заказа:
----------------------------------------

Создание кастомного заказа для Сергей...

Заказ #2 изменил статус на: принят
[КУХНЯ] Получен новый заказ #2
[КУХНЯ] Готовим: Диабло
[КУХНЯ] Готовим: Чизкейк
[КЛИЕНТ Сергей] Статус вашего заказа #2: принят

Заказ #2 (принят):
  - Диабло - 520 руб.
  - Чизкейк - 250 руб.
Итого: 770 руб.

3. Демонстрация фабричного метода:
----------------------------------------
Создана через фабрику: Карбонара - 490 руб.
Создан через фабрику: Кофе - 150 руб.

4. Запуск модульных тестов:
----------------------------------------
Запуск TDD тестов...
test_facade_pattern (__main__.TestCafeSystemTDD.test_facade_pattern) ...
Создание комбо-заказа для Иван Иванов...

Заказ #3 изменил статус на: принят
[КУХНЯ] Получен новый заказ #3
[КУХНЯ] Готовим: Маргарита
[КУХНЯ] Готовим: Кола
[КУХНЯ] Готовим: Тирамису
[КЛИЕНТ Иван Иванов] Статус вашего заказа #3: принят

Заказ #3 изменил статус на: готовится
[КУХНЯ] Получен новый заказ #3
[КУХНЯ] Готовим: Маргарита
[КУХНЯ] Готовим: Кола
[КУХНЯ] Готовим: Тирамису
[КЛИЕНТ Иван Иванов] Статус вашего заказа #3: готовится

Начинаем приготовление:
Готовим пиццу 'Маргарита' размера large
  Пицца 'Маргарита' готова!
Наливаем напиток 'Кола' 0.5л
  Напиток 'Кола' готов!
```

```
Готовим охлажденный десерт 'Тирамису'
  Десерт 'Тирамису' готов!

Заказ #3 изменил статус на: готов
[КУХНЯ] Получен новый заказ #3
[КУХНЯ] Готовим: Маргарита
[КУХНЯ] Готовим: Кола
[КУХНЯ] Готовим: Тирамису
[КЛИЕНТ Иван Иванов] Статус вашего заказа #3: готов

Заказ #3 изменил статус на: оплачен
[КУХНЯ] Получен новый заказ #3
[КУХНЯ] Готовим: Маргарита
[КУХНЯ] Готовим: Кола
[КУХНЯ] Готовим: Тирамису
[КЛИЕНТ Иван Иванов] Статус вашего заказа #3: оплачен
[ПЛАТЕЖНАЯ СИСТЕМА] Заказ #3 оплачен. Сумма: 770 руб.

Заказ #3 изменил статус на: выдан
[КУХНЯ] Получен новый заказ #3
[КУХНЯ] Готовим: Маргарита
[КУХНЯ] Готовим: Кола
[КУХНЯ] Готовим: Тирамису
[КЛИЕНТ Иван Иванов] Статус вашего заказа #3: выдан
ok
test_factory_method (__main__.TestCafeSystemTDD.test_factory_method) ... ok
test_menu_item_creation (__main__.TestCafeSystemTDD.test_menu_item_creation) ... ok
test_observer_pattern (__main__.TestCafeSystemTDD.test_observer_pattern) ...
Заказ #4 изменил статус на: готовится
ok
test_order_total_price (__main__.TestCafeSystemTDD.test_order_total_price) ... ok

----------------------------------------------------------------------
Ran 5 tests in 0.007s

OK
```

```
Запуск BDD тестов...

Создание комбо-заказа для Петр Петров...

Заказ #6 изменил статус на: принят
[КУХНЯ] Получен новый заказ #6
[КУХНЯ] Готовим: Маргарита
[КУХНЯ] Готовим: Кола
[КУХНЯ] Готовим: Тирамису
[КЛИЕНТ Петр Петров] Статус вашего заказа #6: принят

Заказ #6 изменил статус на: готовится
[КУХНЯ] Получен новый заказ #6
[КУХНЯ] Готовим: Маргарита
[КУХНЯ] Готовим: Кола
[КУХНЯ] Готовим: Тирамису
[КЛИЕНТ Петр Петров] Статус вашего заказа #6: готовится

Начинаем приготовление:
Готовим пиццу 'Маргарита' размера large
  Пицца 'Маргарита' готова!
Наливаем напиток 'Кола' 0.5л
  Напиток 'Кола' готов!
Готовим охлажденный десерт 'Тирамису'
  Десерт 'Тирамису' готов!

Заказ #6 изменил статус на: готов
[КУХНЯ] Получен новый заказ #6
[КУХНЯ] Готовим: Маргарита
[КУХНЯ] Готовим: Кола
[КУХНЯ] Готовим: Тирамису
[КЛИЕНТ Петр Петров] Статус вашего заказа #6: готов
```

```
Заказ #6 изменил статус на: оплачен
[КУХНЯ] Получен новый заказ #6
[КУХНЯ] Готовим: Маргарита
[КУХНЯ] Готовим: Кола
[КУХНЯ] Готовим: Тирамису
[КЛИЕНТ Петр Петров] Статус вашего заказа #6: оплачен
[ПЛАТЕЖНАЯ СИСТЕМА] Заказ #6 оплачен. Сумма: 770 руб.

Заказ #6 изменил статус на: выдан
[КУХНЯ] Получен новый заказ #6
[КУХНЯ] Готовим: Маргарита
[КУХНЯ] Готовим: Кола
[КУХНЯ] Готовим: Тирамису
[КЛИЕНТ Петр Петров] Статус вашего заказа #6: выдан
✓ BDD тест order_flow пройден

Создание кастомного заказа для Мария...

Заказ #7 изменил статус на: принят
[КУХНЯ] Получен новый заказ #7
[КУХНЯ] Готовим: Гавайская
[КУХНЯ] Готовим: Чай
[КЛИЕНТ Мария] Статус вашего заказа #7: принят
✓ BDD тест custom_order_creation пройден

Запуск тестов с Mock-объектами...
test_mock_factory (__main__.TestCafeSystemWithMocks.test_mock_factory) ... ok
test_mock_kitchen_observer (__main__.TestCafeSystemWithMocks.test_mock_kitchen_observer) ...
Заказ #8 изменил статус на: готовится
ok
test_mock_menu_item_preparation (__main__.TestCafeSystemWithMocks.test_mock_menu_item_preparation) ... ok

----------------------------------------------------------------------
Ran 3 tests in 0.006s

OK
```

test_cafe.py

Запуск конкретного теста test_order_total_price из класса TestCafeSystem в модуле test_cafe.py

```
(venv) PS C:\Users\MSI-Crosshair16\Documents\лабы\lab_python_fp\lab_chabl> python -m unittest test_cafe.TestCafeSystem.
test_order_total_price
.
----------------------------------------------------------------------
Ran 1 test in 0.000s

OK
```