Факультет «Информатика и системы управления»

Кафедра ИУ5 «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 ИУ5-31Б
Ван
Чжувэнь

Проверил:
Гапанюк Ю.Е.

2025 г.

Листинг кода

main.py

```
# === 1. Фабричный метод (порождающий паттерн) ===
class Tour:
    def __init__(self, name, price):
        self.name = name
        self.price = price

    def calculate_cost(self, days):
        return self.price * days

class AdventureTour(Tour):
    def get_type(self):
        return "Приключенческий тур"

class CulturalTour(Tour):
    def get_type(self):
        return "Культурный тур"

class TourFactory:
    @staticmethod
    def create_tour(tour_type, name, price):
        if tour_type == "adventure":
            return AdventureTour(name, price)
        elif tour_type == "cultural":
            return CulturalTour(name, price)
        else:
            raise ValueError(f"Неизвестный тип тура: {tour_type}")

# === 2. Адаптер (структурный паттерн) ===
class PayPalSystem:
    def pay_in_usd(self, amount_usd):
        return f"PayPal: Оплачено ${amount_usd}"

class StripeSystem:
    def charge(self, amount_cents):
        return f"Stripe: Списано {amount_cents} центов"

class PaymentAdapter:
    def __init__(self, system_type):
        self.system_type = system_type
        if system_type == "paypal":
            self.system = PayPalSystem()
        elif system_type == "stripe":
            self.system = StripeSystem()
        else:
            raise ValueError(f"Неизвестная система: {system_type}")

    def pay(self, amount_rub):
        if self.system_type == "paypal":
            amount_usd = amount_rub / 75
            return self.system.pay_in_usd(amount_usd)
```

```
        else:  # stripe
            amount_cents = int((amount_rub / 75) * 100)
            return self.system.charge(amount_cents)

# === 3. Стратегия (поведенческий паттерн) ===
class DiscountStrategy:
    def calculate_discount(self, price):
        return 0

class StudentDiscount(DiscountStrategy):
    def calculate_discount(self, price):
        return price * 0.15  # 15% скидка

class GroupDiscount(DiscountStrategy):
    def calculate_discount(self, price):
        return price * 0.10  # 10% скидка

class PriceCalculator:
    def __init__(self, discount_strategy=None):
        self.discount_strategy = discount_strategy or DiscountStrategy()

    def calculate_final_price(self, price):
        discount = self.discount_strategy.calculate_discount(price)
        return price - discount

# === Основная программа ===
def main():
    print("=== Демонстрация паттернов ===")

    # 1. Фабричный метод
    print("\n1. Фабричный метод:")
    tour1 = TourFactory.create_tour("adventure", "Поход в горы", 5000)
    tour2 = TourFactory.create_tour("cultural", "Музейный тур", 3000)
    print(f"{tour1.name}: {tour1.get_type()}")
    print(f"{tour2.name}: {tour2.get_type()}")

    # 2. Стратегия
    print("\n2. Паттерн Стратегия:")
    calculator = PriceCalculator()
    price = 10000

    calculator.discount_strategy = StudentDiscount()
    print(f"Студенческая скидка: {calculator.calculate_final_price(price)} руб.")

    calculator.discount_strategy = GroupDiscount()
    print(f"Групповая скидка: {calculator.calculate_final_price(price)} руб.")

    # 3. Адаптер
    print("\n3. Паттерн Адаптер:")
    adapter1 = PaymentAdapter("paypal")
    adapter2 = PaymentAdapter("stripe")

    print(adapter1.pay(7500))
```

```
    print(adapter2.pay(7500))

    print("\n=== Завершение ===")

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

test_simple.py

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

# Импортируем классы из основного файла
import main

class TestTourFactory(unittest.TestCase):
    """Тесты фабричного метода"""

    def test_create_adventure_tour(self):
        """Создание приключенческого тура"""
        tour = main.TourFactory.create_tour("adventure", "Рафтинг", 7000)
        self.assertIsInstance(tour, main.AdventureTour)
        self.assertEqual(tour.name, "Рафтинг")
        self.assertEqual(tour.get_type(), "Приключенческий тур")

    def test_create_cultural_tour(self):
        """Создание культурного тура"""
        tour = main.TourFactory.create_tour("cultural", "Экскурсия", 4000)
        self.assertIsInstance(tour, main.CulturalTour)
        self.assertEqual(tour.get_type(), "Культурный тур")

    def test_invalid_tour_type(self):
        """Тест с неверным типом тура"""
        with self.assertRaises(ValueError):
            main.TourFactory.create_tour("spa", "SPA-тур", 10000)

class TestPaymentAdapter(unittest.TestCase):
    """Тесты адаптера"""

    def test_paypal_adapter(self):
        """Тест PayPal адаптера"""
        adapter = main.PaymentAdapter("paypal")
        result = adapter.pay(7500)
        self.assertIn("PayPal", result)
        self.assertIn("Оплачено", result)

    def test_stripe_adapter(self):
        """Тест Stripe адаптера"""
        adapter = main.PaymentAdapter("stripe")
        result = adapter.pay(7500)
        self.assertIn("Stripe", result)
        self.assertIn("Списано", result)

class TestDiscountStrategy(unittest.TestCase):
```

```
    """Тесты стратегии скидок"""

    def test_student_discount(self):
        """Тест студенческой скидки"""
        discount = main.StudentDiscount()
        self.assertEqual(discount.calculate_discount(10000), 1500)

    def test_group_discount(self):
        """Тест групповой скидки"""
        discount = main.GroupDiscount()
        self.assertEqual(discount.calculate_discount(10000), 1000)

    def test_no_discount(self):
        """Тест отсутствия скидки"""
        discount = main.DiscountStrategy()
        self.assertEqual(discount.calculate_discount(10000), 0)

class TestWithMocks(unittest.TestCase):
    """Тесты с использованием Mock объектов"""

    @patch('main.PayPalSystem')
    def test_paypal_mock(self, mock_paypal):
        """Тест с моком PayPal"""
        mock_instance = Mock()
        mock_instance.pay_in_usd.return_value = "Mock PayPal Payment"
        mock_paypal.return_value = mock_instance

        adapter = main.PaymentAdapter("paypal")
        result = adapter.pay(10000)

        self.assertEqual(result, "Mock PayPal Payment")
        mock_instance.pay_in_usd.assert_called_once()

    def test_tour_mock(self):
        """Тест с моком тура"""
        mock_tour = Mock()
        mock_tour.name = "Mock Tour"
        mock_tour.price = 5000
        mock_tour.calculate_cost.return_value = 15000

        result = mock_tour.calculate_cost(3)
        self.assertEqual(result, 15000)
        mock_tour.calculate_cost.assert_called_once_with(3)

if __name__ == '__main__':
    unittest.main(verbosity=2)
```

test_bdd.py

```
import main

def test_booking_scenario():
```

```
    """Простой тест сценария бронирования"""
    print("=== Тестирование сценария бронирования ===")

    # Given (Дано) - предварительные условия
    tour = main.TourFactory.create_tour("adventure", "Скалолазание", 6000)
    calculator = main.PriceCalculator(main.StudentDiscount())

    # When (Когда) - действие
    final_price = calculator.calculate_final_price(tour.price * 3)

    # Then (Тогда) - проверка результата
    assert tour.get_type() == "Приключенческий тур"
    assert final_price == (6000 * 3) * 0.85  # 15% скидка
    print("✓ Сценарий успешно пройден")

    return True

def test_payment_scenario():
    """Простой тест сценария оплаты"""
    print("\n=== Тестирование сценария оплаты ===")

    # Given
    adapter = main.PaymentAdapter("stripe")

    # When
    result = adapter.pay(5000)

    # Then
    assert "Stripe" in result
    print("✓ Оплата через Stripe успешно обработана")

    return True

if __name__ == "__main__":
    # Запуск BDD тестов
    test_booking_scenario()
    test_payment_scenario()
```

Результат выполнения

```
zhuwen@boxj:~/Zhuwen-CS-Labs-2025/lab4$ python3 main.py
=== Демонстрация паттернов ===

1. Фабричный метод:
Поход в горы: Приключенческий тур
Музейный тур: Культурный тур

2. Паттерн Стратегия:
Студенческая скидка: 8500.0 руб.
Групповая скидка: 9000.0 руб.

3. Паттерн Адаптер:
PayPal: Оплачено $100.0
Stripe: Списано 10000 центов
```

```
zhuwen@boxj:~/Zhuwen-CS-Labs-2025/lab4$ python3 test_bdd.py
=== Тестирование сценария бронирования ===
✓ Сценарий успешно пройден

=== Тестирование сценария оплаты ===
✓ Оплата через Stripe успешно обработана
```