**Московский государственный технический**

**университет им. Н.Э. Баумана**

**Факультет ИУ**

**Кафедра ИУ5**

**Курс «Парадигмы и конструкции языков программирования»**

**Отчет лабораторной работе №5-6**

Выполнил студент группы ИУ5-33Б:

Костяев В.Д.

Подпись и дата:

Проверил преподаватель каф.:

Гапанюк Ю. Е.

Подпись и дата:

Москва, 2024 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.**](https://ru.wikipedia.org/wiki/%D0%A8%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)**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:**

**order\_factory.py (Фабричный метод)**

**from abc import ABC, abstractmethod**

**# Абстрактный класс заказа**

**class Order(ABC):**

**@abstractmethod**

**def process(self):**

**pass**

**# Конкретные типы заказов**

**class PhysicalOrder(Order):**

**def process(self):**

**return "Processing physical order. Packaging and shipping."**

**class DigitalOrder(Order):**

**def process(self):**

**return "Processing digital order. Sending download link."**

**# Фабрика для создания заказов**

**class OrderFactory(ABC):**

**@abstractmethod**

**def create\_order(self):**

**pass**

**class PhysicalOrderFactory(OrderFactory):**

**def create\_order(self):**

**return PhysicalOrder()**

**class DigitalOrderFactory(OrderFactory):**

**def create\_order(self):**

**return DigitalOrder()**

**2. payment\_adapter.py (Адаптер)**

**# Сторонний платежный интерфейс**

**class ThirdPartyPaymentGateway:**

**def make\_payment(self, amount):**

**return f"Paid {amount} using Third Party Payment Gateway."**

**# Адаптер для приведения интерфейса в удобный вид**

**class PaymentAdapter:**

**def \_\_init\_\_(self, gateway):**

**self.gateway = gateway**

**def pay(self, amount):**

**return self.gateway.make\_payment(amount)**

**3. delivery\_strategy.py (Стратегия)**

**from abc import ABC, abstractmethod**

**# Стратегия доставки**

**class DeliveryStrategy(ABC):**

**@abstractmethod**

**def deliver(self):**

**pass**

**class CourierDelivery(DeliveryStrategy):**

**def deliver(self):**

**return "Delivered via Courier."**

**class PostDelivery(DeliveryStrategy):**

**def deliver(self):**

**return "Delivered via Post Office."**

**class DeliveryContext:**

**def \_\_init\_\_(self, strategy: DeliveryStrategy):**

**self.strategy = strategy**

**def execute\_delivery(self):**

**return self.strategy.deliver()**

**4. test\_order\_factory.py (TDD тесты)**

**import unittest**

**from order\_factory import PhysicalOrderFactory, DigitalOrderFactory**

**class TestOrderFactory(unittest.TestCase):**

**def test\_physical\_order(self):**

**factory = PhysicalOrderFactory()**

**order = factory.create\_order()**

**self.assertEqual(order.process(), "Processing physical order. Packaging and shipping.")**

**def test\_digital\_order(self):**

**factory = DigitalOrderFactory()**

**order = factory.create\_order()**

**self.assertEqual(order.process(), "Processing digital order. Sending download link.")**

**if \_\_name\_\_ == "\_\_main\_\_":**

**unittest.main()**

**5. test\_payment\_adapter.py (Mock-тест)**

**from unittest.mock import Mock**

**from payment\_adapter import PaymentAdapter**

**def test\_payment\_adapter():**

**# Создаем mock для стороннего платежного интерфейса**

**mock\_gateway = Mock()**

**mock\_gateway.make\_payment.return\_value = "Paid 100 using Mock Gateway."**

**# Используем адаптер с mock-объектом**

**adapter = PaymentAdapter(mock\_gateway)**

**result = adapter.pay(100)**

**# Проверяем результат**

**assert result == "Paid 100 using Mock Gateway."**

**mock\_gateway.make\_payment.assert\_called\_with(100)**

**if \_\_name\_\_ == "\_\_main\_\_":**

**test\_payment\_adapter()**

**print("Payment Adapter Mock Test Passed!")**

**6. features/order.feature (BDD сценарии)**

**Feature: Order Processing**

**Scenario: Process Physical Order**

**Given a physical order factory**

**When I create a physical order**

**Then the result should be "Processing physical order. Packaging and shipping."**

**Scenario: Process Digital Order**

**Given a digital order factory**

**When I create a digital order**

**Then the result should be "Processing digital order. Sending download link."**

**7. features/steps/order\_steps.py (Шаги BDD)**

**from order\_factory import PhysicalOrderFactory, DigitalOrderFactory**

**from behave import given, when, then**

**@given('a physical order factory')**

**def step\_given\_physical\_factory(context):**

**context.factory = PhysicalOrderFactory()**

**@given('a digital order factory')**

**def step\_given\_digital\_factory(context):**

**context.factory = DigitalOrderFactory()**

**@when('I create a physical order')**

**def step\_when\_create\_physical(context):**

**context.result = context.factory.create\_order().process()**

**@when('I create a digital order')**

**def step\_when\_create\_digital(context):**

**context.result = context.factory.create\_order().process()**

**@then('the result should be "{expected}"')**

**def step\_then\_result\_should\_be(context, expected):**

**assert context.result == expected**

**Экранные формы с примерами выполнения программы:**

