Московский государственный технический университет им. Н.Э. Баумана

Факультет Радиотехнический
Кафедра РТ5

Курс «Программирование на основе классов и шаблонов»

Отчет по лабораторной работе №5-6
«Шаблоны проектирования и модульное тестирование в Python»

Выполнил:

студент группы РТ5-31Б:
Салищев И.Д.

Подпись и дата:

Проверил:

преподаватель каф. ИУ5
Гапанюк Ю.Е.

Подпись и дата:

Москва, 2023

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

1. Адаптер метод с тестом BDD

```
from abc import ABC, abstractmethod
import unittest
from unittest.mock import MagicMock

# Тест для Адаптера
class TestAdapter(unittest.TestCase):
    def test_payment_adapter(self):
        # Создаем Mock-объект для первого стороннего сервиса
        service_mock_1 = MagicMock()
        service_mock_1.process_payment.return_value = "Payment processed by Service 1"

        # Создаем Mock-объект для второго стороннего сервиса
        service_mock_2 = MagicMock()
        service_mock_2.make_payment.return_value = "Payment processed by Service 2"

        # Используем Mock-объекты в адаптерах
        adapter_1 = PaymentAdapter(service_mock_1, "process_payment")
        adapter_2 = PaymentAdapter(service_mock_2, "make_payment")

        # Проверяем вызовы методов через адаптеры
        self.assertEqual(adapter_1.process_payment(), "Processed via Adapter: Payment processed by Service 1")
        service_mock_1.process_payment.assert_called_once()

        self.assertEqual(adapter_2.process_payment(), "Processed via Adapter: Payment processed by Service 2")
        service_mock_2.make_payment.assert_called_once()

# Целевой интерфейс нашей системы платежей
class PaymentSystem(ABC):
    @abstractmethod
    def process_payment(self):
```

```
        pass

# Адаптер для интеграции сторонних сервисов с нашей системой
class PaymentAdapter(PaymentSystem):
    def __init__(self, third_party_service, method_name):
        self.third_party_service = third_party_service
        self.method_name = method_name

    def process_payment(self):
        method = getattr(self.third_party_service, self.method_name)
        return f"Processed via Adapter: {method()}"

# Использование
# Представление двух различных сторонних сервисов
class ThirdPartyService1:
    def process_payment(self):
        return "Payment processed successfully by Service 1"

class ThirdPartyService2:
    def make_payment(self):
        return "Payment processed successfully by Service 2"

service1 = ThirdPartyService1()
service2 = ThirdPartyService2()

adapter1 = PaymentAdapter(service1, "process_payment")
adapter2 = PaymentAdapter(service2, "make_payment")

print(adapter1.process_payment())
print(adapter2.process_payment())

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

2. Фабричный метод с тестом TTD

```
from abc import ABC, abstractmethod
import unittest

# Тест для Фабричного метода
class TestFactoryMethod(unittest.TestCase):
    def test_product_creation(self):
        # Создаем магазин книг и проверяем создание книги
        book_store = BookStore()
        book_product = book_store.create_product()
        self.assertIsInstance(book_product, Book)
        self.assertEqual(book_product.display_info(), "Book: 1984 by
George Orwell")

```

```
        # Создаем магазин электроники и проверяем создание электронного устройства
        electronics_store = ElectronicsStore()
        electronics_product = electronics_store.create_product()
        self.assertIsInstance(electronics_product, ElectronicDevice)
        self.assertEqual(electronics_product.display_info(), "Electronic Device: Apple Smartphone")

# Продукт - Товар в интернет-магазине
class Product(ABC):
    @abstractmethod
    def display_info(self):
        pass

# Конкретный продукт - Книга
class Book(Product):
    def __init__(self, title, author):
        self.title = title
        self.author = author

    def display_info(self):
        return f"Book: {self.title} by {self.author}"

# Конкретный продукт - Электронное устройство
class ElectronicDevice(Product):
    def __init__(self, name, brand):
        self.name = name
        self.brand = brand

    def display_info(self):
        return f"Electronic Device: {self.brand} {self.name}"

# Создатель - Интернет-магазин
class OnlineStore(ABC):
    @abstractmethod
    def create_product(self):
        pass

    def sell_product(self):
        product = self.create_product()
        result = f"{self.store_type} sold: {product.display_info()}"
        return result

# Конкретный создатель - Интернет-магазин книг
class BookStore(OnlineStore):
    store_type = "BookStore"

    def create_product(self):
        return Book("1984", "George Orwell")

# Конкретный создатель - Интернет-магазин электронных устройств
```

```
class ElectronicsStore(OnlineStore):
    store_type = "ElectronicsStore"

    def create_product(self):
        return ElectronicDevice("Smartphone", "Apple")

# Использование
book_store = BookStore()
book_result = book_store.sell_product()
print(book_result)

electronics_store = ElectronicsStore()
electronics_result = electronics_store.sell_product()
print(electronics_result)

# Запуск тестов
if __name__ == "__main__":
    unittest.main()
```

3. Наблюдатель с тестом создание Mock-объектов

```
from abc import ABC, abstractmethod
import unittest
from unittest.mock import Mock

# Наблюдатель
class Observer(ABC):
    @abstractmethod
    def update(self, message):
        pass

# Субъект - Корзина покупок
class ShoppingCart:
    def __init__(self):
        self._observers = []
        self._items = []

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

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

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

    def add_item(self, item):
        self._items.append(item)
        self.notify_observers(f"Товар добавлен: {item}")

```

```
    def remove_item(self, item):
        if item in self._items:
            self._items.remove(item)
            self.notify_observers(f"Товар удален: {item}")

    def display_items(self):
        return self._items

# Конкретный наблюдатель - Клиент магазина
class Customer(Observer):
    def __init__(self, name):
        self.name = name

    def update(self, message):
        print(f"{self.name} получил/а уведомление! {message}")

# Тест для Наблюдателя
cart = ShoppingCart()

customer1 = Customer("Никита")
customer2 = Customer("Екатерина")

# Добавляем клиентов в качестве наблюдателей
cart.add_observer(customer1)
cart.add_observer(customer2)

# Добавляем товары в корзину
cart.add_item("Баскетбольный мяч")
cart.add_item("Ноутбук")

# Удаляем товар из корзины
cart.remove_item("Ноутбук")

# Отображаем текущие товары в корзине
print("Товар в корзине:", cart.display_items())
```

Экранные формы с примерами выполнения программы

1. Adapter.py

```
Processed via Adapter: Payment processed successfully by Service 1
Processed via Adapter: Payment processed successfully by Service 2
.
----------------------------------------------------------------------
Ran 1 test in 0.007s

OK
```

2. Factory Method.py

```
BookStore sold: Book: 1984 by George Orwell
ElectronicsStore sold: Electronic Device: Apple Smartphone
.
----------------------------------------------------------------------
Ran 1 test in 0.002s

OK
```

2. Observer.py

```
Никита получил/а уведомление! Товар добавлен: Баскетбольный мяч
Екатерина получил/а уведомление! Товар добавлен: Баскетбольный мяч
Никита получил/а уведомление! Товар добавлен: Ноутбук
Екатерина получил/а уведомление! Товар добавлен: Ноутбук
Никита получил/а уведомление! Товар удален: Ноутбук
Екатерина получил/а уведомление! Товар удален: Ноутбук
Товар в корзине: ['Баскетбольный мяч']
```