**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: |  | Проверил: |
| студент группы РТ5-31Б:  Салищев И.Д. |  | преподаватель каф. ИУ5  Гапанюк Ю.Е. |
|  |  |  |
| Подпись и дата: |  | Подпись и дата: |

Москва, 2023

**Описание задания**

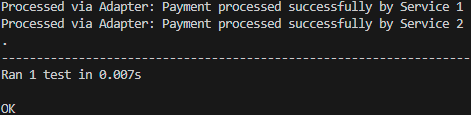
1. Необходимо для произвольной предметной области реализовать от одного до трех шаблонов проектирования: один порождающий, один структурный и один поведенческий. В качестве справочника шаблонов можно использовать [следующий каталог.](https://ru.wikipedia.org/wiki/%D0%A8%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%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**

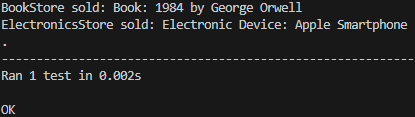
1. Адаптер метод с тестом BDD
2. from abc import ABC, abstractmethod
3. import unittest
4. from unittest.mock import MagicMock
5. # Тест для Адаптера
6. class TestAdapter(unittest.TestCase):
7. def test\_payment\_adapter(self):
8. # Создаем Mock-объект для первого стороннего сервиса
9. service\_mock\_1 = MagicMock()
10. service\_mock\_1.process\_payment.return\_value = "Payment processed by Service 1"
11. # Создаем Mock-объект для второго стороннего сервиса
12. service\_mock\_2 = MagicMock()
13. service\_mock\_2.make\_payment.return\_value = "Payment processed by Service 2"
14. # Используем Mock-объекты в адаптерах
15. adapter\_1 = PaymentAdapter(service\_mock\_1, "process\_payment")
16. adapter\_2 = PaymentAdapter(service\_mock\_2, "make\_payment")
17. # Проверяем вызовы методов через адаптеры
18. self.assertEqual(adapter\_1.process\_payment(), "Processed via Adapter: Payment processed by Service 1")
19. service\_mock\_1.process\_payment.assert\_called\_once()
20. self.assertEqual(adapter\_2.process\_payment(), "Processed via Adapter: Payment processed by Service 2")
21. service\_mock\_2.make\_payment.assert\_called\_once()
22. # Целевой интерфейс нашей системы платежей
23. class PaymentSystem(ABC):
24. @abstractmethod
25. def process\_payment(self):
26. pass
27. # Адаптер для интеграции сторонних сервисов с нашей системой
28. class PaymentAdapter(PaymentSystem):
29. def \_\_init\_\_(self, third\_party\_service, method\_name):
30. self.third\_party\_service = third\_party\_service
31. self.method\_name = method\_name
32. def process\_payment(self):
33. method = getattr(self.third\_party\_service, self.method\_name)
34. return f"Processed via Adapter: {method()}"
35. # Использование
36. # Представление двух различных сторонних сервисов
37. class ThirdPartyService1:
38. def process\_payment(self):
39. return "Payment processed successfully by Service 1"
40. class ThirdPartyService2:
41. def make\_payment(self):
42. return "Payment processed successfully by Service 2"
43. service1 = ThirdPartyService1()
44. service2 = ThirdPartyService2()
45. adapter1 = PaymentAdapter(service1, "process\_payment")
46. adapter2 = PaymentAdapter(service2, "make\_payment")
47. print(adapter1.process\_payment())
48. print(adapter2.process\_payment())
49. # Запуск тестов
50. if \_\_name\_\_ == '\_\_main\_\_':
51. unittest.main()
52. Фабричный метод с тестом TTD
53. from abc import ABC, abstractmethod
54. import unittest
55. # Тест для Фабричного метода
56. class TestFactoryMethod(unittest.TestCase):
57. def test\_product\_creation(self):
58. # Создаем магазин книг и проверяем создание книги
59. book\_store = BookStore()
60. book\_product = book\_store.create\_product()
61. self.assertIsInstance(book\_product, Book)
62. self.assertEqual(book\_product.display\_info(), "Book: 1984 by George Orwell")
63. # Создаем магазин электроники и проверяем создание электронного устройства
64. electronics\_store = ElectronicsStore()
65. electronics\_product = electronics\_store.create\_product()
66. self.assertIsInstance(electronics\_product, ElectronicDevice)
67. self.assertEqual(electronics\_product.display\_info(), "Electronic Device: Apple Smartphone")
68. # Продукт - Товар в интернет-магазине
69. class Product(ABC):
70. @abstractmethod
71. def display\_info(self):
72. pass
73. # Конкретный продукт - Книга
74. class Book(Product):
75. def \_\_init\_\_(self, title, author):
76. self.title = title
77. self.author = author
78. def display\_info(self):
79. return f"Book: {self.title} by {self.author}"
80. # Конкретный продукт - Электронное устройство
81. class ElectronicDevice(Product):
82. def \_\_init\_\_(self, name, brand):
83. self.name = name
84. self.brand = brand
85. def display\_info(self):
86. return f"Electronic Device: {self.brand} {self.name}"
87. # Создатель - Интернет-магазин
88. class OnlineStore(ABC):
89. @abstractmethod
90. def create\_product(self):
91. pass
92. def sell\_product(self):
93. product = self.create\_product()
94. result = f"{self.store\_type} sold: {product.display\_info()}"
95. return result
96. # Конкретный создатель - Интернет-магазин книг
97. class BookStore(OnlineStore):
98. store\_type = "BookStore"
99. def create\_product(self):
100. return Book("1984", "George Orwell")
101. # Конкретный создатель - Интернет-магазин электронных устройств
102. class ElectronicsStore(OnlineStore):
103. store\_type = "ElectronicsStore"
104. def create\_product(self):
105. return ElectronicDevice("Smartphone", "Apple")
106. # Использование
107. book\_store = BookStore()
108. book\_result = book\_store.sell\_product()
109. print(book\_result)
110. electronics\_store = ElectronicsStore()
111. electronics\_result = electronics\_store.sell\_product()
112. print(electronics\_result)
113. # Запуск тестов
114. if \_\_name\_\_ == "\_\_main\_\_":
115. unittest.main()
116. Наблюдатель с тестом создание Mock-объектов
117. from abc import ABC, abstractmethod
118. import unittest
119. from unittest.mock import Mock
120. # Наблюдатель
121. class Observer(ABC):
122. @abstractmethod
123. def update(self, message):
124. pass
125. # Субъект - Корзина покупок
126. class ShoppingCart:
127. def \_\_init\_\_(self):
128. self.\_observers = []
129. self.\_items = []
130. def add\_observer(self, observer):
131. self.\_observers.append(observer)
132. def remove\_observer(self, observer):
133. self.\_observers.remove(observer)
134. def notify\_observers(self, message):
135. for observer in self.\_observers:
136. observer.update(message)
137. def add\_item(self, item):
138. self.\_items.append(item)
139. self.notify\_observers(f"Товар добавлен: {item}")
140. def remove\_item(self, item):
141. if item in self.\_items:
142. self.\_items.remove(item)
143. self.notify\_observers(f"Товар удален: {item}")
144. def display\_items(self):
145. return self.\_items
146. # Конкретный наблюдатель - Клиент магазина
147. class Customer(Observer):
148. def \_\_init\_\_(self, name):
149. self.name = name
150. def update(self, message):
151. print(f"{self.name} получил/а уведомление! {message}")
152. # Тест для Наблюдателя
153. cart = ShoppingCart()
154. customer1 = Customer("Никита")
155. customer2 = Customer("Екатерина")
156. # Добавляем клиентов в качестве наблюдателей
157. cart.add\_observer(customer1)
158. cart.add\_observer(customer2)
159. # Добавляем товары в корзину
160. cart.add\_item("Баскетбольный мяч")
161. cart.add\_item("Ноутбук")
162. # Удаляем товар из корзины
163. cart.remove\_item("Ноутбук")
164. # Отображаем текущие товары в корзине
165. print("Товар в корзине:", cart.display\_items())

**Экранные формы с примерами выполнения программы**

1. Adapter.py



1. Factory Method.py



1. Observer.py

