|  |  |
| --- | --- |
|  | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

|  |  |
| --- | --- |
| ФАКУЛЬТЕТ | Инженерный бизнес и менеджмент |
| КАФЕДРА | Промышленная логистика |

**Лабораторная работа №5-6  
«Парадигмы и конструкции языков программирования»**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Студент |  |  |  |  |  |
|  | (группа) |  | (подпись, дата) |  | (И.О. Фамилия) |
| Преподаватель |  |  |  |  |  |
|  |  |  | (подпись, дата) |  | (И.О. Фамилия) |
|  |  |  |  |  |  |

Москва, 2024 г

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.

Первый шаблон

class NotificationSettings:

"""

Шаблон Singleton для хранения настроек уведомлений.

"""

\_instance = None

def \_\_new\_\_(cls, \*args, \*\*kwargs):

if not cls.\_instance:

cls.\_instance = super().\_\_new\_\_(cls)

cls.\_instance.settings = {}

return cls.\_instance

def set\_setting(self, key, value):

self.settings[key] = value

def get\_setting(self, key):

return self.settings.get(key, None)

Второй шаблон

class EmailSender:

"""

Реализация отправки email.

"""

def send\_email(self, recipient, subject, message):

print(f"Отправка email на {recipient}: {subject} - {message}")

class SMSSender:

"""

Реализация отправки SMS.

"""

def send\_sms(self, phone\_number, message):

print(f"Отправка SMS на {phone\_number}: {message}")

class NotificationAdapter:

"""

Адаптер для приведения разных способов уведомлений к единому интерфейсу.

"""

def \_\_init\_\_(self, sender):

self.sender = sender

def send(self, recipient, message):

if isinstance(self.sender, EmailSender):

self.sender.send\_email(recipient, "Уведомление", message)

elif isinstance(self.sender, SMSSender):

self.sender.send\_sms(recipient, message)

else:

raise ValueError("Неподдерживаемый тип отправителя")

Третий шаблон

class Observable:

"""

Класс, который уведомляет подписчиков об изменениях.

"""

def \_\_init\_\_(self):

self.subscribers = []

def subscribe(self, observer):

self.subscribers.append(observer)

def unsubscribe(self, observer):

self.subscribers.remove(observer)

def notify\_subscribers(self, message):

for subscriber in self.subscribers:

subscriber.update(message)

class Customer:

"""

Подписчик, который получает уведомления.

"""

def \_\_init\_\_(self, name):

self.name = name

def update(self, message):

print(f"{self.name} получил уведомление: {message}")

Тесты

TDD

import unittest

from unittest.mock import patch

from main import NotificationSettings, NotificationAdapter, EmailSender, SMSSender, Observable, Customer

class TestSingleton(unittest.TestCase):

def test\_singleton\_instance(self):

settings1 = NotificationSettings()

settings2 = NotificationSettings()

self.assertIs(settings1, settings2) # Ожидается, что это один и тот же объект

def test\_singleton\_settings(self):

settings = NotificationSettings()

settings.set\_setting("test\_key", "test\_value")

self.assertEqual(settings.get\_setting("test\_key"), "test\_value")

class TestAdapter(unittest.TestCase):

@patch("main.EmailSender.send\_email")

def test\_email\_adapter(self, mock\_send\_email):

email\_sender = EmailSender()

adapter = NotificationAdapter(email\_sender)

adapter.send("test@example.com", "Test message")

mock\_send\_email.assert\_called\_once\_with("test@example.com", "Уведомление", "Test message")

@patch("main.SMSSender.send\_sms")

def test\_sms\_adapter(self, mock\_send\_sms):

sms\_sender = SMSSender()

adapter = NotificationAdapter(sms\_sender)

adapter.send("+1234567890", "Test SMS message")

mock\_send\_sms.assert\_called\_once\_with("+1234567890", "Test SMS message")

class TestObserver(unittest.TestCase):

def test\_observer\_notification(self):

shop = Observable()

customer = Customer("Тестовый клиент")

with patch.object(customer, "update") as mock\_update:

shop.subscribe(customer)

shop.notify\_subscribers("Тестовое сообщение")

mock\_update.assert\_called\_once\_with("Тестовое сообщение")

BDD

**from pytest\_bdd import scenario, given, when, then**

**from main import NotificationSettings, Observable, Customer**

**@scenario("features/notification\_settings.feature", "Test singleton behavior")**

**def test\_singleton\_behavior():**

**pass**

**@given("a NotificationSettings instance")**

**def notification\_settings():**

**return NotificationSettings()**

**@when("I set a setting with a key and value")**

**def set\_setting(notification\_settings):**

**notification\_settings.set\_setting("test\_key", "test\_value")**

**@then("I should be able to retrieve the same value using the same key")**

**def check\_setting(notification\_settings):**

**assert notification\_settings.get\_setting("test\_key") == "test\_value"**