Московский государственный технический университет им. Н.Э. Баумана.

Факультет «Информатика и управление»
Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Базовые компоненты интернет-технологий»

Отчет по лабораторной работе № 4
« Шаблоны проектирования и модульное тестирование в Python»

Выполнил:
студент группы ИУ5-31Б
Зелинский Даниил Михайлович

Подпись и дата:

Проверил:
преподаватель каф. ИУ5
Гапанюк Юрий Евгеньевич

Подпись и дата:

Москва, 2021 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. 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.

Library.py

```python
from __future__ import annotations
from abc import ABC, abstractmethod

EnglishBookStandartName='Pride & Prejudice'
RussianBookStandartName='Война и мир'

#абстрактная фабрика
class AbstractFactory(ABC):
    @abstractmethod
    def create_book(self, name) -> AbstractBook:
        pass
    @abstractmethod
    def create_reader(self) -> AbstractReader:
        pass

# фабрика английских книг и читателей
class English(AbstractFactory):
    def create_book(self, name=EnglishBookStandartName) -> AbstractBook:
        return EnglishBook(name)

    def create_reader(self) -> AbstractReader:
        return EnglishReader()

# фабрика русских книг и читателей
class Russian(AbstractFactory):
    def create_book(self, name=RussianBookStandartName) -> AbstractBook:
        return RussianBook(name)

    def create_reader(self) -> AbstractReader:
        return RussianReader()

# абстрактная книга
class AbstractBook(ABC):
    @abstractmethod
    def read(self) -> str:
        pass


class EnglishBook(AbstractBook):
    def __init__(self, name=EnglishBookStandartName):
        self.name=name
    def read(self) -> str:
        return f"The book named <<{self.name}>>"

class RussianBook(AbstractBook):
    def __init__(self, name=RussianBookStandartName):
        self.name=name
    def read(self) -> str:
        return f"Книга под названием <<{self.name}>>"

# абстрактный читатель
class AbstractReader(ABC):
    @abstractmethod
    def tell_something(self) -> None:
        pass
    @abstractmethod
    def read_a_book(self, collaborator: AbstractBook) -> None:
        pass

# английский читатель может читать только книги на английском языке
class EnglishReader(AbstractReader):
    def tell_something(self) -> str:
        return "I wanna read something."
    def read_a_book(self, collaborator: AbstractBook) -> str:
        result = collaborator.read()
        if (self.check_language(collaborator)):
            return f"I have read {result} and enjoyed it."
        else:
            return f"I don't understand <<{result}>> and I hate it."

    def check_language(self, book: AbstractBook)->bool:
        return isinstance(book, EnglishBook)

# русский читатель может читать только книги на русском языке
class RussianReader(AbstractReader):
    def tell_something(self) -> str:
        return "Почитать хочется."

    def read_a_book(self, collaborator: AbstractBook) -> str:
        result = collaborator.read()
        if (self.check_language(collaborator)):
            return f"{result} мне очень понравилась!"
        else:
            return f"Я не понимаю <<{result}>>, что за ерунда?"

    def check_language(self, book: AbstractBook)->bool:
        return isinstance(book, RussianBook)

def client_code(factory: AbstractFactory, bookName) -> str:
    book=factory.create_book(bookName)
    reader=factory.create_reader()
    return f"{reader.read_a_book(book)}"
#адаптер
class TranslatorToRussian(RussianBook, EnglishBook):
    def __init__(self, enBook:EnglishBook):
        self.enBook=enBook

    def translate(self)->RussianBook:
        return RussianBook(f"Перевод: {self.enBook.read()} - теперь доступно
на русском языке!")


if __name__ == "__main__":

    '''    enbook=EnglishBook('The Old Man and the Sea')
        translator=TranslatorToRussian(enbook)
        rubook=translator.translate()
        print(rubook.name)
        print(client_code(Russian(), rubook.name))'''
```

test.py

```python
import unittest
from Library import *

class MyTestCase(unittest.TestCase):
    def setUp(self):
        self.Olga=RussianReader()
        self.John=EnglishReader()
        self.ruBook=RussianBook('Руслан и Людмила')
        self.enBook=EnglishBook('Romeo and Juliet')
    def test_client_code(self):
        result=client_code(Russian(), 'Анна Каренина')
        self.assertEqual(result, "Книга под названием <<Анна Каренина>> мне очень понравилась!")
        result=client_code(English(), 'Tom Sawyer')
        self.assertEqual(result, "I have read The book named <<Tom Sawyer>> and enjoyed it.")

    def test_client_code_without_a_book_name(self):
        with self.assertRaises(TypeError):
            client_code(Russian())

    def test_acceptable_books(self):
        self.assertEqual(self.Olga.read_a_book(self.ruBook),
                         "Книга под названием <<Руслан и Людмила>> мне очень понравилась!")
        self.assertEqual(self.John.read_a_book(self.enBook),
                         "I have read The book named <<Romeo and Juliet>> and enjoyed it.")

    def test_unacceptable_books(self):
        self.assertEqual(self.Olga.read_a_book(self.enBook),
                         "Я не понимаю <<The book named <<Romeo and Juliet>>>>, что за ерунда?")
        self.assertEqual(self.John.read_a_book(self.ruBook),
                         "I don't understand <<Книга под названием <<Руслан и Людмила>>>> and I hate it.")

    def test_standart_names(self):
        self.assertEqual(RussianBook().name, RussianBookStandartName)
        self.assertEqual(EnglishBook().name, EnglishBookStandartName)

    def test_translator_to_Russian(self):
        self.assertEqual(TranslatorToRussian(self.enBook).translate().name,
                         "Перевод: The book named <<Romeo and Juliet>> - теперь доступно на русском языке!")
    def test_client_code_with_translator(self):
        rubook=TranslatorToRussian(self.enBook).translate()
        self.assertEqual(client_code(Russian(), rubook.name),
                         "Книга под названием <<Перевод: The book named <<Romeo and Juliet>> - теперь доступно на русском языке!>> мне очень понравилась!")

    def test_English_reader_with_wrong_translation(self):
        enbook=TranslatorToRussian(self.enBook).translate()
        self.assertEqual(self.John.read_a_book(enbook),
                         "I don't understand <<Книга под названием <<Перевод: The book named <<Romeo and Juliet>> - теперь доступно на русском языке!>>>> and I hate it.")

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

steps.py

```python
# -*- coding: utf-8 -*-
from Library import*
from behave import given, when, then

@given(u'There is an English book named "{bookName}"')
def step_impl(context, bookName:str):
    context.book=EnglishBook(bookName)

@given(u'There is an English reader')
def step_impl(context):
    context.reader=EnglishReader()


@given(u'There is a Russian book named "{bookName}"')
def step_impl(context, bookName: str):
    context.book = RussianBook(bookName)

@given(u'There is a Russian reader')
def step_impl(context):
    context.reader = RussianReader()


@when(u'The reader tries to read the book')
def step_impl(context):
    context.result=context.reader.read_a_book(context.book)


@then(u'The next result is expected: "{result}"')
def step_impl(context, result):
    assert context.result==result

@when(u'The book is translated')
def step_impl(context):
    context.book=TranslatorToRussian(context.book).translate()
    context.result=context.book.name
```

Library.feature

```gherkin
Feature: just simple reading
  Scenario: English reader tries to read English book
    Given There is an English book named "Harry Potter and the Philosopher's
Stone"
    Given There is an English reader
    When The reader tries to read the book
    Then The next result is expected: "I have read The book named <<Harry
Potter and the Philosopher's Stone>> and enjoyed it."
  Scenario: Russian reader tries to read Russian book
    Given There is a Russian book named "Таня Гроттер и волшебный контрабас"
    Given There is a Russian reader
    When The reader tries to read the book
    Then The next result is expected: "Книга под названием <<Таня Гроттер и
волшебный контрабас>> мне очень понравилась!"
  Scenario: English reader tries to read Russian book
    Given There is a Russian book named "Таня Гроттер и волшебный контрабас"
    Given There is an English reader
    When The reader tries to read the book
    Then The next result is expected: "I don't understand <<Книга под
названием <<Таня Гроттер и волшебный контрабас>>>> and I hate it."
  Scenario: Russian reader tries to read English book
    Given There is an English book named "Harry Potter and the Philosopher's
Stone"
    Given There is a Russian reader
    When The reader tries to read the book
    Then The next result is expected: "Я не понимаю <<The book named <<Harry
Potter and the Philosopher's Stone>>>>, что за ерунда?"
```

Translate.feature

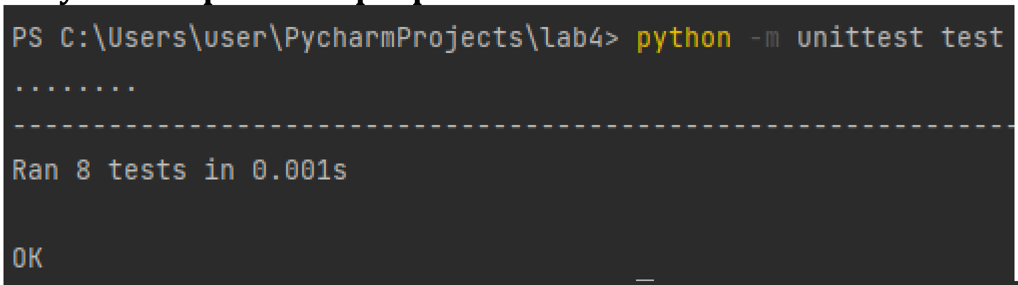
```
Feature: reading the translated books
  Scenario: Translate the book to Russian
    Given There is an English book named "Harry Potter and the Philosopher's
Stone"
    When The book is translated
    Then The next result is expected: "Перевод: The book named <<Harry Potter
and the Philosopher's Stone>> - теперь доступно на русском языке!"
  Scenario: Russian reader tries to read the book translated
    Given There is an English book named "Harry Potter and the Philosopher's
Stone"
    Given There is a Russian reader
    When The book is translated
    When The reader tries to read the book
    Then The next result is expected: "Книга под названием <<Перевод: The
book named <<Harry Potter and the Philosopher's Stone>> - теперь доступно на
русском языке!>> мне очень понравилась!"
  Scenario: English reader tries to read the book translated to Russian
    Given There is an English book named "Harry Potter and the Philosopher's
Stone"
    Given There is an English reader
    When The book is translated
    When The reader tries to read the book
    Then The next result is expected: "I don't understand <<Книга под
названием <<Перевод: The book named <<Harry Potter and the Philosopher's
Stone>> - теперь доступно на русском языке!>>>> and I hate it."
```

mockTests.py

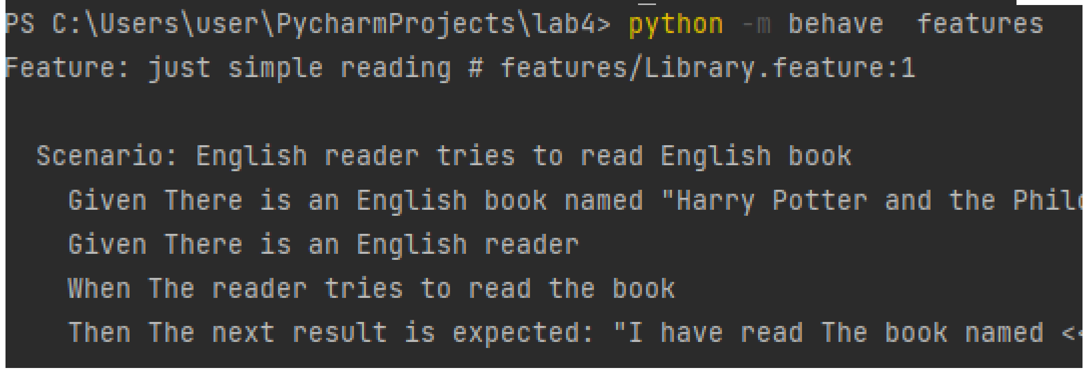
```
from Library import *
from unittest import TestCase
from unittest.mock import patch

class TestLibrary(TestCase):

    @patch('Library.client_code', side_effect=[
        "Книга под названием <<Анна Каренина>> мне очень понравилась!",
        "I have read The book named <<Tom Sawyer>> and enjoyed it.",

    ])
    def test_client_code(self, ccode):
        self.assertEqual(ccode(Russian(), 'Анна Каренина'),
                         "Книга под названием <<Анна Каренина>> мне очень
понравилась!")
        self.assertEqual(ccode(English(), 'Tom Sawyer'),
                         "I have read The book named <<Tom Sawyer>> and enjoyed
it.")

    @patch('Library.client_code', side_effect=TypeError)
    def test_error_cc(self, ccode):
        self.assertRaises(TypeError,(Russian()))

    @patch.object(RussianReader(),'read_a_book', side_effect=[
    "I have read The book named <<Romeo and Juliet>> and enjoyed it.",
     "Книга под названием <<Руслан и Людмила>> мне очень понравилась!",
     "I don't understand <<Книга под названием <<Руслан и Людмила>>>> and I
hate it.",
     "Я не понимаю <<The book named <<Romeo and Juliet>>>>, что за ерунда?"

])
```

```
    def test_readers_with_books(self, rbook, side_effect):
        self.assertEqual(rbook(RussianBook("Romeo and Juliet")),
side_effect[0])
```

Результаты работы программы.

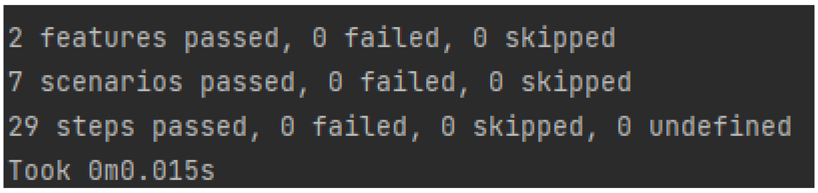


```
PS C:\Users\user\PycharmProjects\lab4> python -m unittest test
........
----------------------------------------------------------------
Ran 8 tests in 0.001s

OK
```



```
PS C:\Users\user\PycharmProjects\lab4> python -m behave  features
Feature: just simple reading # features/Library.feature:1

  Scenario: English reader tries to read English book
    Given There is an English book named "Harry Potter and the Phil
    Given There is an English reader
    When The reader tries to read the book
    Then The next result is expected: "I have read The book named <
```

…



```
2 features passed, 0 failed, 0 skipped
7 scenarios passed, 0 failed, 0 skipped
29 steps passed, 0 failed, 0 skipped, 0 undefined
Took 0m0.015s
```