Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования

«Московский государственный технический университет имени Н.Э. Баумана»

(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ Информатика и системы управления

КАФЕДРА Системы обработки информации и управления (ИУ5)

О т ч е т

по лабораторной работе №4

Шаблоны проектирования и модульное тестирование в Python

Дисциплина: Разработка Интернет-Приложений

Студент гр. ИУ5-53Б _____ Терентьев В.О.

(Подпись, дата) (Фамилия И.О.)

Преподаватель _____ Гапанюк Ю.Е.

(Подпись, дата) (Фамилия И.О.)

Москва, 2020

1. Описание задания

1. Необходимо для произвольной предметной области реализовать три шаблона проектирования: один порождающий, один структурный и один поведенческий. В качестве справочника шаблонов можно использовать следующий каталог.
2. Для каждой реализации шаблона необходимо написать модульный тест. В модульных тестах необходимо применить следующие технологии:
   - TDD - фреймворк.
   - BDD - фреймворк.
   - Создание Mock-объектов.

2. Текст программы

2.1. Порождающий шаблон проектирования (Строитель) builder.py

```
from __future__ import annotations
from abc import ABC, abstractmethod, abstractproperty
from typing import Any


class Builder(ABC):

    @abstractproperty
    def car(self) -> None:
        pass

    @abstractmethod
    def buildCarBody(self) -> None:
        pass

    @abstractmethod
    def setEngine(self) -> None:
        pass

    @abstractmethod
    def setTransmission(self) -> None:
        pass

    @abstractmethod
    def setCruiseControl(self) -> None:
        pass

    @abstractmethod
    def setAirConditioning(self) -> None:
        pass


class SedanCarBuilder(Builder):

    def __init__(self) -> None:
```

```
        """
        Новый экземпляр строителя должен содержать пустой объект продукта,
        который используется в дальнейшей сборке.
        """
        self.reset()

    def reset(self) -> None:
        self._product = Product1()

    @property
    def car(self) -> Product1:
        car = self._product
        self.reset()
        return car

    def buildCarBody(self) -> None:
        self._product.add("Sedan")

    def setEngine(self) -> None:
        self._product.add("VR38DETT")

    def setTransmission(self) -> None:
        self._product.add("6-speed dual-clutch")

    def setCruiseControl(self) -> None:
        self._product.add("Cruise Control")

    def setAirConditioning(self) -> None:
        self._product.add("Air Conditioning")


class HatchbackCarBuilder(Builder):

    def __init__(self) -> None:
        self.reset()

    def reset(self) -> None:
        self._product = Product1()

    @property
    def car(self) -> Product1:
        car = self._product
        self.reset()
        return car

    def buildCarBody(self) -> None:
        self._product.add("Hatchback")

    def setEngine(self) -> None:
        self._product.add("VR38DETT")
```

```
    def setTransmission(self) -> None:
        self._product.add("6-speed dual-clutch")

    def setCruiseControl(self) -> None:
        self._product.add("Cruise Control")

    def setAirConditioning(self) -> None:
        self._product.add("Air Conditioning")


class Product1():

    def __init__(self) -> None:
        self.parts = []

    def add(self, part: Any) -> None:
        self.parts.append(part)

    def list_parts(self):
        print(f"Product parts: {', '.join(self.parts)}", end="")
        return self.parts


class Director:

    def __init__(self) -> None:
        self._builder = None

    @property
    def builder(self) -> Builder:
        return self._builder

    @builder.setter
    def builder(self, builder: Builder) -> None:
        self._builder = builder

    def build_minimal_version_car(self) -> None:
        self.builder.buildCarBody()
        self.builder.setEngine()
        self.builder.setTransmission()

    def build_full_version_car(self) -> None:
        self.builder.buildCarBody()
        self.builder.setEngine()
        self.builder.setTransmission()
        self.builder.setCruiseControl()
        self.builder.setAirConditioning()


if __name__ == "__main__":
```

```
    director = Director()

    print("Sedans: ")
    builder = SedanCarBuilder()
    director.builder = builder

    print("Standard basic car: ")
    director.build_minimal_version_car()
    builder.car.list_parts()

    print("\n")

    print("Standard full version car: ")
    director.build_full_version_car()
    builder.car.list_parts()

    print("\n")

    print("Hatchbacks: ")
    builder = HatchbackCarBuilder()
    director.builder = builder

    print("Standard basic car: ")
    director.build_minimal_version_car()
    builder.car.list_parts()

    print("\n")

    print("Standard full version car: ")
    director.build_full_version_car()
    builder.car.list_parts()

    print("\n")

    print("Custom hatchback car: ")
    builder.buildCarBody()
    builder.setEngine()
    builder.setTransmission()
    builder.setAirConditioning()
    builder.car.list_parts()
```

2.2. Структурный шаблон проектирования (Заместитель) structural.py

```
from abc import ABC, abstractmethod
import time
import random

cache = {}
```

```python
class Video(ABC):

    @abstractmethod
    def downloading(self) -> None:
        pass

    def real_downloading(self):
        pass


class RealVideo(Video):
    def __init__(self, name):
        self.name = name

    def downloading(self) -> None:
        print("Начало скачивания...")
        cache.update({self.name: self.real_downloading()})
        print("Скачивание завершено.")

    def real_downloading(self):
        time.sleep(3)
        return random.randrange(10000, 99999)


class Proxy(Video):
    def __init__(self, real_subject: RealVideo) -> None:
        self._real_subject = real_subject

    def downloading(self) -> None:
        if self.check_cache():
            self.cache_access()
        else:
            self._real_subject.downloading()

    def check_cache(self) -> bool:
        print("Поиск видео в кэше...")
        return self._real_subject.name in cache

    def cache_access(self) -> None:
        print("Видео найдено в кэше.")


def client_code(subject: Video) -> None:

    # ...

    subject.downloading()

    # ...
```

```
def main():
    print("Без кэша...")
    print("Видео1:")
    real_subject1 = RealVideo("video1")
    client_code(real_subject1)
    print("Видео2:")
    real_subject2 = RealVideo("video2")
    client_code(real_subject2)
    print("Видео1:")
    client_code(real_subject1)

    print("")

    print("С кэшем...")
    print("Видео3:")
    real_subject3 = RealVideo("video3")
    proxy = Proxy(real_subject3)
    client_code(proxy)
    print("Видео4:")
    real_subject4 = RealVideo("video4")
    proxy = Proxy(real_subject4)
    client_code(proxy)
    print("Видео3:")
    proxy = Proxy(real_subject3)
    client_code(proxy)


if __name__ == "__main__":
    main()
```

2.3. Поведенческий шаблон проектирования (Наблюдатель) behavioral.py

```
from __future__ import annotations
from abc import ABC, abstractmethod
from typing import List


class Subject(ABC):

    @abstractmethod
    def follow(self, observer: Observer) -> None:
        pass

    @abstractmethod
    def unfollow(self, observer: Observer) -> None:
        pass

    @abstractmethod
    def notify(self) -> None:
```

```
        pass


class SportNews(Subject):

    _state: str = None

    _observers: List[Observer] = []

    def follow(self, observer: Observer) -> None:
        print("Новый подписчик.")
        self._observers.append(observer)

    def unfollow(self, observer: Observer) -> None:
        print("Пользователь отписался.")
        self._observers.remove(observer)

    def notify(self) -> None:

        print("Отправка уведомлений...")
        for observer in self._observers:
            observer.update(self)

    def some_logic(self, new) -> None:

        print("\nСобытие в мире спорта.")
        self._state = new

        print(f"{self._state}: выпуск новой новости.")
        self.notify()


class Observer(ABC):

    @abstractmethod
    def update(self, subject: Subject) -> None:
        pass


class FootballObserver(Observer):
    def update(self, subject: Subject) -> None:
        if subject._state == 'Футбол':
            print("Футбольный болельщик получил уведомление.")


class HockeyObserver(Observer):
    def update(self, subject: Subject) -> None:
        if subject._state == 'Хоккей':
            print("Хоккейный болельщик получил уведомление.")
```

```
if __name__ == "__main__":

    subject = SportNews()

    observer_a = FootballObserver()
    subject.follow(observer_a)

    observer_b = HockeyObserver()
    subject.follow(observer_b)

    subject.some_logic('Футбол')
    subject.some_logic('Хоккей')

    subject.unfollow(observer_a)

    subject.some_logic('Футбол')
```

2.4. Модульный тест с применением TDD-фреймворка. unitest.py

```
from builder import SedanCarBuilder, HatchbackCarBuilder
import unittest


class SetTest(unittest.TestCase):
    def setUp(self):
        self.sedan_car = SedanCarBuilder()
        self.hatchback_car = HatchbackCarBuilder()

    def test_setEngine_Sedan(self):
        self.sedan_car.setEngine()
        self.assertEqual(self.sedan_car.car.list_parts(), [
                         'VR38DETT'], "Should be VR38DETT")

    def test_setAirConditioning_Hatchback(self):
        self.hatchback_car.setCruiseControl()
        self.hatchback_car.setAirConditioning()
        self.assertEqual(self.hatchback_car.car.list_parts(), [
                         'Cruise Control', 'Air Conditioning'], "Should be Air Co
nditioning")


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

2.5. Модульный тест с применением BDD-фреймворка. bdd.py

```
from pytest_bdd import scenario, given, when, then
from behavioral import SportNews, FootballObserver
```

```
@given("I am a sports news user.")
def user(SportNews):
    subject = SportNews()


@when("I followed to football news")
def go_to_article(subject, FootballObserver):
    observer_a = FootballObserver()
    subject.follow(observer_a)


@then("I should be followed to football news")
def article_is_published(subject):
    assert subject.followed
```

2.6. Модульный тест с созданием Mock-объектов. mock.py

```
import structural
import random
import unittest
from unittest.mock import patch


class Test(unittest.TestCase):
    @patch('structural.RealVideo.real_downloading', return_value=random.randrange
(10000, 99999))
    def test_real_downloading(self, real_downloading):
        structural.main()


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

3. Экранные формы с примерами выполнения программы

3.1. builder.py

```
PS C:\Users\webma\iCloudDrive\MGTU\5\1DIA\github\DIA\lab4> & C:/Users/webma/AppData/Local/Programs/P
ython/Python39/python.exe c:/Users/webma/iCloudDrive/MGTU/5/1DIA/github/DIA/lab4/builder.py
Sedans:
Standard basic car:
Product parts: Sedan, VR38DETT, 6-speed dual-clutch

Standard full version car:
Product parts: Sedan, VR38DETT, 6-speed dual-clutch, Cruise Control, Air Conditioning

Hatchbacks:
Standard basic car:
Product parts: Hatchback, VR38DETT, 6-speed dual-clutch

Standard full version car:
Product parts: Hatchback, VR38DETT, 6-speed dual-clutch, Cruise Control, Air Conditioning

Custom hatchback car:
Product parts: Hatchback, VR38DETT, 6-speed dual-clutch, Air Conditioning
PS C:\Users\webma\iCloudDrive\MGTU\5\1DIA\github\DIA\lab4>
```

3.2. structural.py

```
PS C:\Users\webma\iCloudDrive\MGTU\5\1DIA\github\DIA\lab4> & C:/Users/webma/AppData/Local/Programs/P
ython/Python39/python.exe c:/Users/webma/iCloudDrive/MGTU/5/1DIA/github/DIA/lab4/structural.py
Без кэша...
Видео1:
Начало скачивания...
Скачивание завершено.
Видео2:
Начало скачивания...
Скачивание завершено.
Видео1:
Начало скачивания...
Скачивание завершено.

С кэшем...
Видео3:
Поиск видео в кэше...
Начало скачивания...
Скачивание завершено.
Видео4:
Поиск видео в кэше...
Начало скачивания...
Скачивание завершено.
Видео3:
Поиск видео в кэше...
Видео найдено в кэше.
PS C:\Users\webma\iCloudDrive\MGTU\5\1DIA\github\DIA\lab4>
```

3.3. behavioral.py

```
PS C:\Users\webma\iCloudDrive\MGTU\5\1DIA\github\DIA\lab4> & C:/Users/webma/AppData/Local/Programs/P
ython/Python39/python.exe c:/Users/webma/iCloudDrive/MGTU/5/1DIA/github/DIA/lab4/behavioral.py
Новый подписчик.
Новый подписчик.

Событие в мире спорта.
Футбол: выпуск новой новости.
Отправка уведомлений...
Футбольный болельщик получил уведомление.

Событие в мире спорта.
Хоккей: выпуск новой новости.
Отправка уведомлений...
Хоккейный болельщик получил уведомление.
Пользователь отписался.

Событие в мире спорта.
Футбол: выпуск новой новости.
Отправка уведомлений...
PS C:\Users\webma\iCloudDrive\MGTU\5\1DIA\github\DIA\lab4>
```

3.4. unitest.py

```
PS C:\Users\webma\iCloudDrive\MGTU\5\1DIA\github\DIA\lab4> & C:/Users/webma/AppData/Local/Programs/P
ython/Python39/python.exe c:/Users/webma/iCloudDrive/MGTU/5/1DIA/github/DIA/lab4/unitest.py
..
----------------------------------------------------------------------
Ran 2 tests in 0.000s

OK
Product parts: Cruise Control, Air ConditioningProduct parts: VR38DETT
PS C:\Users\webma\iCloudDrive\MGTU\5\1DIA\github\DIA\lab4>
```

3.5. mock.py

Mock-объект заменяет реализацию метода *real_downloading* класса *RealVideo*. Этот метод имитирует загрузку видеофайла за 3 секунды, mock-объект пропускает загрузку и сразу выдает видеофайл.

```
PS C:\Users\webma\iCloudDrive\MGTU\5\1DIA\github\DIA\lab4> & C:/Users/webma/AppData/Local/Programs/P
ython/Python39/python.exe c:/Users/webma/iCloudDrive/MGTU/5/1DIA/github/DIA/lab4/mock.py
Без кэша...
Видео1:
Начало скачивания...
Скачивание завершено.
Видео2:
Начало скачивания...
Скачивание завершено.
Видео1:
Начало скачивания...
Скачивание завершено.

С кэшем...
Видео3:
Поиск видео в кэше...
Начало скачивания...
Скачивание завершено.
Видео4:
Поиск видео в кэше...
Начало скачивания...
Скачивание завершено.
Видео3:
Поиск видео в кэше...
Видео найдено в кэше.
.
----------------------------------------------------------------------
Ran 1 test in 0.002s

OK
PS C:\Users\webma\iCloudDrive\MGTU\5\1DIA\github\DIA\lab4>
```

4. Ссылка на репозиторий

https://github.com/iYroglif/DIA