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)

Лабораторная работа №4
по курсу «Разработка интернет-приложений»

Выполнил

студент группы ИУ5-54Б

Сысойкин Е.М.

Москва, 2020

1 Общее 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

patterns/adapter_pattern.py

```python
#!/usr/bin/env python
import math


class Hole(object):
    def __init__(self, r):
        self.r = r

    def put(self, obj):
        try:
            if self.r >= obj.r:
                return True
            else:
                return False
        except AttributeError:
            print("Can not use this object!")


class Square(object):
    def __init__(self, x, h):
        self.x = x
        self.h = h


class SquareHoleAdapter(object):
    def __init__(self, sq_obj):
        self.sq_obj = sq_obj

    @property
    def r(self):
        return math.sqrt(2*(self.sq_obj.x**2))/2


if __name__ == "__main__":
    h1 = Hole(5)
    h2 = Hole(2)
    s1 = Square(5, 7)
    s2 = Square(3, 3)
    sa = SquareHoleAdapter(s2)

    print("Square(5,7) into Hole(5): ")
    print(h1.put(s1))
    print("\nSquare(5,7) into Hole(5) via adapter: ")
    print(h1.put(sa))
    print("\nSquare(3,3) into Hole(2): ")
    print(h2.put(sa))
```

patterns/builder_pattern.py

```python
#!/usr/bin/env python
from __future__ import annotations
from abc import ABC, abstractmethod, abstractproperty
from typing import Any


class ImageBuilderInterface(ABC):

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

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

    @abstractmethod
    def placeholder(self, uri) -> None:
        pass

    @abstractmethod
    def size(self, width, height) -> None:
        pass

```

```python

class ImageBuilder(ImageBuilderInterface):

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

    def reset(self) -> None:
        self._image = Image()

    @property
    def image(self) -> Image:
        image = self._image
        self.reset()
        return image

    def round(self) -> None:
        self._image._is_round = True

    def placeholder(self, uri) -> None:
        self._image.placeholder = uri

    def size(self, width, height) -> None:
        self._image.set_size(width, height)


class Image():

    def __init__(self) -> None:
        self._width = 0
        self._height = 0
        self.placeholder = "empty"
        self._is_round = False

    def is_round(self):
        return self._is_round

    def set_size(self, width, height):
        self._width = width
        self._height = height

    def show_image(self):
        print("Image with placeholder {}, size {}x{}, isRound {}\n".format(self.placeholder, self._wi


class ImageDirector:

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

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

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

    def produce_standart_image(self) -> None:
        self.builder.size(220, 220)
        self.builder.placeholder("uri://placeholders/default_placeholder")

    def produce_rounded_image(self) -> None:
        self.builder.round()

    def produce_full_image(self, width, height, placeholder) -> None:
        self.builder.placeholder(placeholder)
        self.builder.size(width, height)


if __name__ == "__main__":

    director = ImageDirector()
    builder = ImageBuilder()
    director.builder = builder

    print("Standard default image: ")
    director.produce_standart_image()
    builder.image.show_image()

    print("Standard full featured product: ")
    director.produce_full_image(128, 128, "uri://placeholder/my_placeholder")
    builder.image.show_image()

    # can be used without director
    print("Custom image: ")
    builder.placeholder("uri://placeholder/custom_placeholder")
    builder.round()
    builder.size(300, 300)
    builder.image.show_image()
```

patterns/observer_pattern.py

```python
#!/usr/bin/env python
from __future__ import annotations
from abc import ABC, abstractmethod
from random import randrange
from typing import List


class Subject(ABC):

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

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

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


class SomeContentManager(Subject):

    _state: int = None
    _observers: List[Observer] = []

    def attach(self, observer: Observer) -> None:
        print("SomeContentManager: Attached an observer.")
        self._observers.append(observer)

    def detach(self, observer: Observer) -> None:
        self._observers.remove(observer)

    def notify(self) -> None:
        print("SomeContentManager: Notifying observers...")
        for observer in self._observers:
            observer.update(self)

    def request_content(self) -> None:

        print("\nGoing to the internet(not) and getting the data.")
        self._state = randrange(0, 10)

        self.notify()
```

```python


class Observer(ABC):

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



class MyObserverA(Observer):
    def update(self, subject: Subject) -> None:
        print("MyObserverA: update {}".format(subject._state))


class MyObserverB(Observer):
    def update(self, subject: Subject) -> None:
        print("MyObserverA: update {}".format(subject._state))


if __name__ == "__main__":
    content_manager = SomeContentManager()

    observer_a = MyObserverA()
    content_manager.attach(observer_a)

    observer_b = MyObserverB()
    content_manager.attach(observer_b)

    content_manager.request_content()
    content_manager.request_content()

    content_manager.detach(observer_a)
    content_manager.request_content()
    content_manager.detach(observer_b)
    content_manager.request_content()
```

tests.py

```python
#!/usr/bin/env python
from patterns.builder_pattern import ImageDirector, ImageBuilder
from patterns.adapter_pattern import Hole, Square, SquareHoleAdapter
from mock import patch

if __name__ == "__main__":
    director = ImageDirector()
    builder = ImageBuilder()
    director.builder = builder

    h1 = Hole(5)
    h2 = Hole(2)
    s1 = Square(5, 7)
    s2 = Square(3, 3)
    sa = SquareHoleAdapter(s2)

    def tdd():
        director.produce_standart_image()
        assert builder.image.placeholder == "uri://placeholders/default_placeholder"

        builder.round()
        assert builder.image.is_round() == True

        assert type(h1.put(sa)) == type(True)


    def mock():
        with patch("patterns.builder_pattern.Image.is_round") as m:
            builder.round()
            m.return_value = builder.image.is_round()

        with patch("patterns.adapter_pattern.Hole.put") as m:
            m.return_value = h1.put(sa)

    tdd()
    mock()
```

features/test.feature

```
Feature: build standart image
Scenario: build standart image
Given build standart image
Then image is standart
```

features/steps/test.py

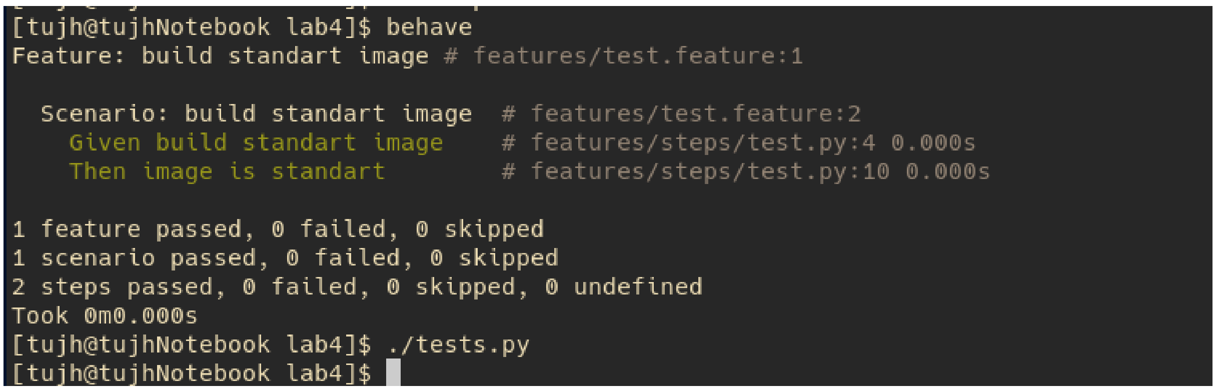
```
from behave import *
from patterns.builder_pattern import ImageDirector, ImageBuilder

@given("build standart image")
def step_impl(context):
    context.director = ImageDirector()
    context.builder = ImageBuilder()
    context.director.builder = context.builder
```

```python

@then("image is standart")
def step_impl(context):
    context.director.produce_standart_image()
    image = context.builder.image
    assert image.placeholder == "uri://placeholders/default_placeholder"
    assert image.is_round() == False
```

3 Экранные формы



```
[tujh@tujhNotebook lab4]$ behave
Feature: build standart image # features/test.feature:1

  Scenario: build standart image  # features/test.feature:2
    Given build standart image    # features/steps/test.py:4 0.000s
    Then image is standart        # features/steps/test.py:10 0.000s

1 feature passed, 0 failed, 0 skipped
1 scenario passed, 0 failed, 0 skipped
2 steps passed, 0 failed, 0 skipped, 0 undefined
Took 0m0.000s
[tujh@tujhNotebook lab4]$ ./tests.py
[tujh@tujhNotebook lab4]$
```