МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ИМЕНИ Н.Э. БАУМАНА

Факультет Информатика и системы управления

Кафедра Системы обработки информации и управления (ИУ5)

Разработка интернет-приложений

Отчет по лабораторной работе №4

Выполнил: Торжков Максим Сергеевич

Группа: ИУ5-51Б

Преподаватель: Гапанюк Юрий Евгеньевич

Дата: 30.11.19

Подпись:

Москва, 2020 г.

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.

Текст программы и результаты:

Файл pizzabuilder.py:

```
from __future__ import annotations
from abc import ABC, abstractmethod, abstractproperty
from typing import Any
from watcher import NewPizzaObserver, FinishPizzaObserver
from facade import Facade

class Pizza():

    def __init__(self) -> None:
        self.dough = []
        self.sauce = []
        self.topping = []

    def set_dough(self, part: Any) -> None:
        self.dough.append(part)
    def set_sauce(self, part: Any) -> None:
        self.sauce.append(part)
    def set_topping(self, part: Any) -> None:
        self.topping.append(part)

    def list_parts(self) -> None:
        print(f"Dough: {', '.join(self.dough)}", end="\n")
        print(f"Sauce: {', '.join(self.sauce)}", end="\n")
        print(f"Topping: {', '.join(self.topping)}", end="\n")

class PizzaBuilder(ABC):
    pizza_type = None

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

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

    @abstractmethod
    def produce_sauce(self) -> None:
        pass
```

```
    @abstractmethod
    def produce_topping(self) -> None:
        pass


class SpicyPizzaBuilder(PizzaBuilder):

    def __init__(self) -> None:
        self.pizza_type = "Spicy Pizza"
        self.reset()

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


    @property
    def product(self) -> Pizza:
        product = self._product
        self.reset()
        return product

    def produce_dough(self) -> None:
        self._product.set_dough("Standart")

    def produce_sauce(self) -> None:
        self._product.set_sauce("Spicy")

    def produce_topping(self) -> None:
        self._product.set_topping("Tomatoes")

class HawaiianPizzaBuilder(PizzaBuilder):

    def __init__(self) -> None:
        self.pizza_type = "Hawaiian Pizza"
        self.reset()

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


    @property
    def product(self) -> Pizza:
        product = self._product
        self.reset()
        return product

    def produce_dough(self) -> None:
        self._product.set_dough("Fat")

    def produce_sauce(self) -> None:
        self._product.set_sauce("Katchup")

    def produce_topping(self) -> None:
        self._product.set_topping("Pineapple")


class Cook:

    def __init__(self, name, new_observer, finish_observer) -> None:
        self._builder = None
        self.name = name
        self._new_pizza_observer = new_observer or NewPizzaObserver()
        self._finish_pizza_observer = finish_observer or
```

```
FinishPizzaObserver()

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

    @builder.setter
    def builder(self, builder: PizzaBuilder) -> None:
        self._builder = builder
        self.notify(self._new_pizza_observer)

    def build_only_sauce_pizza(self) -> None:
        self.builder.produce_dough()
        self.builder.produce_sauce()
        self.notify(self._finish_pizza_observer)

    def build_full_pizza(self) -> None:
        self.builder.produce_dough()
        self.builder.produce_sauce()
        self.builder.produce_topping()
        self.notify(self._finish_pizza_observer)

    def notify(self, observer: PizzaObserver) -> None:
        print("Наблюдатель говорит:")
        observer.update(self)


if __name__ == "__main__":

    director1 = Cook("NewCook1", None, None)
    builder1 = SpicyPizzaBuilder()
    director1.builder = builder1

    director2 = Cook("NewCook2", None, None)
    builder2 = HawaiianPizzaBuilder()
    director2.builder = builder2

    cookers = Facade(director1, director2)
    cookers.cooke1_operation()
    cookers.cooker2_operation()
```

Файл watcher.py:

```
from __future__ import annotations
from abc import ABC, abstractmethod
#from pizzabuilder import Cook


class PizzaObserver(ABC):

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


class NewPizzaObserver(PizzaObserver):
    def update(self, subject: Cook) -> None:
        print(f"{subject.name} is cooking {subject.builder.pizza_type}\n")


class FinishPizzaObserver(PizzaObserver):
    def update(self, subject: Cook) -> None:
```

```
        print(f"{subject.name} закончил готовить пиццу
{subject.builder.pizza_type}")
        print(f"Ингридиенты:")
        subject.builder.product.list_parts()
```

Файл facade.py:

```
from __future__ import annotations


class Facade:

    def __init__(self, cooker1: Cook, cooker2: Cook) -> None:
        self.cooker1 = cooker1
        self.cooker2 = cooker2

    def cooke1_operation(self) -> None:

        print("Only sauce pizza: ")
        self.cooker1.build_only_sauce_pizza()

        print("\n")

        print("Full pizza: ")
        self.cooker1.build_full_pizza()

        print("\n")


    def cooker2_operation(self) -> None:
        print("Only sauce pizza: ")
        self.cooker2.build_only_sauce_pizza()

        print("\n")

        print("Full pizza: ")
        self.cooker2.build_full_pizza()

        print("\n")
```

Файл test_watcher.py:

```
from unittest import main
from unittest import TestCase
from unittest.mock import patch
from pizzabuilder import Cook, SpicyPizzaBuilder



class TestWatcher(TestCase):
    @patch('watcher.NewPizzaObserver')
    def test_new_pizza_watcher(self, MockObserver):
        observer = MockObserver
        cooker = Cook("Cooker", observer, None)
        builder1 = SpicyPizzaBuilder()
        cooker.builder = builder1
        observer.update.assert_called_once()

    @patch('watcher.FinishPizzaObserver')
    def test_finish_pizza_watcher(self, MockObserver):
```

```
        observer = MockObserver
        cooker = Cook("Cooker", None, observer)
        builder1 = SpicyPizzaBuilder()
        cooker.builder = builder1
        cooker.build_full_pizza()
        observer.update.assert_called_once()

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

Файл test_facade.py:

```
import unittest
from facade import Facade
from pizzabuilder import Cook

test_cooker1 = Cook("TestCook1", None, None)
test_cooker2 = Cook("TestCook2", None, None)


class TestFacade(unittest.TestCase):
    def test_facade_create_cooker(self):
        facade = Facade(test_cooker1, test_cooker2)
        unknown_cooker1 = facade.cooker1
        unknown_cooker2 = facade.cooker2

        self.assertEqual(unknown_cooker1.name, test_cooker1.name)

        self.assertEqual(unknown_cooker2.name, test_cooker2.name)

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

Файл test.py:

```
import unittest
from test_facade import TestFacade
from test_watcher import TestWatcher

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

Файл pizzabuilder.feature:

```
Feature: Cook
  Test Cook

  Scenario: Create new Cook
    Given I want to create Cook named Cooker
    When I create it
    Then I expect that his name will be Cooker
```

Файл test_pizzabuilder.py:

```
from radish import given, when, then
from pizzabuilder import Cook
```

```
@given("I want to create Cook named {name: w}")
def cook_name(step, name):
    step.context.name = name

@when("I create it")
def create_cook(step):
    step.context.cook = Cook(step.context.name, None, None)

@then("I expect that his name will be {new_name: w}")
def expect_name(step, new_name):
    assert step.context.cook.name == new_name
```

Результаты программы:

Наблюдатель говорит:

NewCook1 is cooking Spicy Pizza

Наблюдатель говорит:

NewCook2 is cooking Hawaiian Pizza

Only sauce pizza:

Наблюдатель говорит:

NewCook1 закончил готовить пиццу Spicy Pizza

Ингридиенты:

Dough: Standart

Sauce: Spicy

Topping:

Full pizza:

Наблюдатель говорит:

NewCook1 закончил готовить пиццу Spicy Pizza

Ингридиенты:

Dough: Standart

Sauce: Spicy

Topping: Tomatoes

Only sauce pizza:

Наблюдатель говорит:

NewCook2 закончил готовить пиццу Hawaiian Pizza

Ингридиенты:

Dough: Fat

Sauce: Katchup

Topping:

Full pizza:

Наблюдатель говорит:

NewCook2 закончил готовить пиццу Hawaiian Pizza

Ингридиенты:

Dough: Fat

Sauce: Katchup

Topping: Pineapple