Московский государственный технический университет

имени Н. Э. Баумана

Факультет «Информатика и системы управления»

Кафедра ИУ5 «Системы обработки информации и управления»

Отчёт по лабораторной работе №4

«Шаблоны проектирования и модульное тестирование в Python»

Выполнила:

Рыжкова Юлия Николаевна

Группа ИУ5-31Б

Проверил:

Канев Антон Игоревич

Кафедра ИУ5

Москва 2021г.

Задание

1. Необходимо для произвольной предметной области реализовать от одного до трех шаблонов проектирования: один порождающий, один структурный и один поведенческий.

2. В модульных тестах необходимо применить следующие технологии:
    - TDD - фреймворк.
    - BDD - фреймворк.
    - Создание Mock-объектов.

Текст программы и результаты её работы

Файл **builder.py**

```
'''Шаблон: Строитель'''
from __future__ import annotations
from abc import ABC, abstractmethod
from typing import Any

class Cook(ABC):
    @property
    @abstractmethod
    def salad(self) -> None:
        pass
    @abstractmethod
    def add_vegetables(self) -> None:
        pass
    @abstractmethod
    def add_meat(self) -> None:
        pass
    @abstractmethod
    def add_sauce(self) -> None:
        pass

class SaladCook(Cook):
    def __init__(self) -> None:
        self.reset()
    def reset(self) -> None:
        self._salad = CookedSalad()
    @property
    def salad(self) -> CookedSalad:
        salad = self._salad
        self.reset()
        return salad
    def add_vegetables(self) -> None:
        self._salad.add("vegetables")
    def add_meat(self) -> None:
        self._salad.add("meat")
    def add_sauce(self) -> None:
        self._salad.add("sauce")

class CookedSalad():
    def __init__(self) -> None:
        self.ingredients = []
    def add(self, ingredient: Any) -> None:
        self.ingredients.append(ingredient)
```

```
    def list_ingredients(self) -> None:
        print(f"Salad ingredients: {',
'.join(self.ingredients)}", end="")

class Chief:
    def __init__(self) -> None:
        self._cook = None
    @property
    def cook(self) -> Cook:
        return self._cook
    @cook.setter
    def cook(self, cook: Cook) -> None:
        self._cook = cook
    def cook_meat_salad(self) -> None:
        self.cook.add_vegetables()
        self.cook.add_meat()
        self.cook.add_sauce()
    def cook_vegetarian_salad(self) -> None:
        self.cook.add_vegetables()
        self.cook.add_sauce()

if __name__ == "__main__":
    chief = Chief()
    cook = SaladCook()
    chief.cook = cook
    print("Cooking meat salad: ")
    chief.cook_meat_salad()
    cook.salad.list_ingredients()
    print("Cooking vegetarian salad: ")
    chief.cook_vegetarian_salad()
    cook.salad.list_ingredients()
```

Результаты работы **builder.py**

```
Run:  builder
/Users/artisia/PycharmProjects/lab_python_test/venv/bin/python /Users/
Cooking meat salad:
Salad ingredients: vegetables, meat, sauceCooking vegetarian salad:
Salad ingredients: vegetables, sauce
Process finished with exit code 0
```

Файл **bridge.py**

```python
'''Шаблон: Мост + Строитель'''
from __future__ import annotations
from abc import ABC, abstractmethod
from typing import Any

# Абстрактный класс
class Chief:
    def __init__(self, cook: Cook) -> None:
        self.cook = cook
    def StartCooking(self) -> str:
        return ('Started cooking!')

class SaladChief(Chief):
    def cook_meat_salad(self) -> str:
        self.cook.add_vegetables()
        self.cook.add_meat()
        self.cook.add_sauce()
        return "Cooking meat salad..."
    def cook_vegetarian_salad(self) -> None:
        self.cook.add_vegetables()
        self.cook.add_sauce()
        return "Cooking vegetarian salad..."

#Реализация
class Cook(ABC):
    @property
    @abstractmethod
    def salad(self) -> None:
        pass
    @abstractmethod
    def add_vegetables(self) -> None:
        pass
    @abstractmethod
    def add_meat(self) -> None:
        pass
    @abstractmethod
    def add_sauce(self) -> None:
        pass

class SaladCook(Cook):
    def __init__(self) -> None:
        self.reset()
```

```
    def reset(self) -> None:
        self._salad = CookedSalad()
    @property
    def salad(self) -> CookedSalad:
        salad = self._salad
        self.reset()
        return salad
    def add_vegetables(self) -> None:
        self._salad.add("vegetables")
    def add_meat(self) -> None:
        self._salad.add("meat")
    def add_sauce(self) -> None:
        self._salad.add("sauce")

class CookedSalad():
    def __init__(self) -> None:
        self.ingredients = []
    def add(self, ingredient: Any) -> None:
        self.ingredients.append(ingredient)
    def list_ingredients(self) -> str:
        if not self.ingredients:
            return "There is no salad ready"
        else:
            return f"Salad ingredients: {',
'.join(self.ingredients)}"

if __name__ == "__main__":
    cook = SaladCook()
    chief = SaladChief(cook)
    print(cook.salad.list_ingredients())
    cooking = chief.cook_meat_salad()
    print(cooking)
    meal = cook.salad.list_ingredients()
    print(meal)
```

Результаты работы **bridge.py**

```
Run:  bridge
/Users/artisia/PycharmProjects/lab_python_test/venv/
There is no salad ready
Cooking meat salad...
Salad ingredients: vegetables, meat, sauce

Process finished with exit code 0
```

Файл **Comand.py**

```python
'''Шаблон: Команда+ Мост + Строитель'''
from __future__ import annotations
from abc import ABC, abstractmethod
from typing import Any

class Order(ABC):
    @abstractmethod
    def execute(self) -> None:
        pass

class SaladOrder(Order):
    def __init__(self, reciever: SaladChief, orderText:
str) -> None:
        self._reciever = reciever
        self._orderText = orderText
    def execute(self) -> None:
        print("*Order is passed to the Chief*")
        self._reciever.start_cooking(self._orderText)

# Абстрактный класс
class Chief:
    def __init__(self, cook: Cook) -> None:
        self.cook = cook

class SaladChief(Chief):
    def cook_meat_salad(self) -> None:
        self.cook.add_vegetables()
        self.cook.add_meat()
        self.cook.add_sauce()
```

```
        print("*The meat salad is ready*")
    def cook_vegetarian_salad(self) -> None:
        self.cook.add_vegetables()
        self.cook.add_sauce()
        print("*The vegetable salad is ready*")
    def start_cooking(self, orderText: str) -> None:
        if orderText == "meat salad":
            self.cook_meat_salad()
        elif orderText == "vegetarian salad":
            self.cook_vegetarian_salad()
        else:
            print("Chief: 'Sorry, but we can't cook that'")

#Реализация
class Cook(ABC):
    @property
    @abstractmethod
    def salad(self) -> None:
        pass
    @abstractmethod
    def add_vegetables(self) -> None:
        pass
    @abstractmethod
    def add_meat(self) -> None:
        pass
    @abstractmethod
    def add_sauce(self) -> None:
        pass

class SaladCook(Cook):
    def __init__(self) -> None:
        self.reset()
    def reset(self) -> None:
        self._salad = CookedSalad()
    @property
    def salad(self) -> CookedSalad:
        salad = self._salad
        self.reset()
        return salad
    def add_vegetables(self) -> None:
        self._salad.add("vegetables")
    def add_meat(self) -> None:
        self._salad.add("meat")
    def add_sauce(self) -> None:
        self._salad.add("sauce")

class CookedSalad():
```

```
    def __init__(self) -> None:
        self.ingredients = []
    def add(self, ingredient: Any) -> None:
        self.ingredients.append(ingredient)
    def list_ingredients(self) -> None:
        print(f"Salad ingredients: {',
'.join(self.ingredients)}", end="")
```

```
class Waiter:
    took_order = None
    def took_order(self, order: Order):
        self._took_order = order
    def pass_order(self) -> None:
        if isinstance(self._took_order, Order):
            print("\nWaiter: 'I'm going to tell the Chief
about your order'")
            self._took_order.execute()
        print("Waiter: 'Enjoy'")
```

```
if __name__ == "__main__":
    waiter = Waiter()
    cook = SaladCook()
    chief = SaladChief(cook)
    waiter.took_order(SaladOrder(chief, "meat salad"))
    waiter.pass_order()
```

Результаты работы **Comand.py**

```
Run:  Comand ×
/Users/artisia/PycharmProjects/lab_python_test/venv/bin/python

Waiter: 'I'm going to tell the Chief about your order'
*Order is passed to the Chief*
*The meat salad is ready*
Waiter: 'Enjoy'

Process finished with exit code 0
```

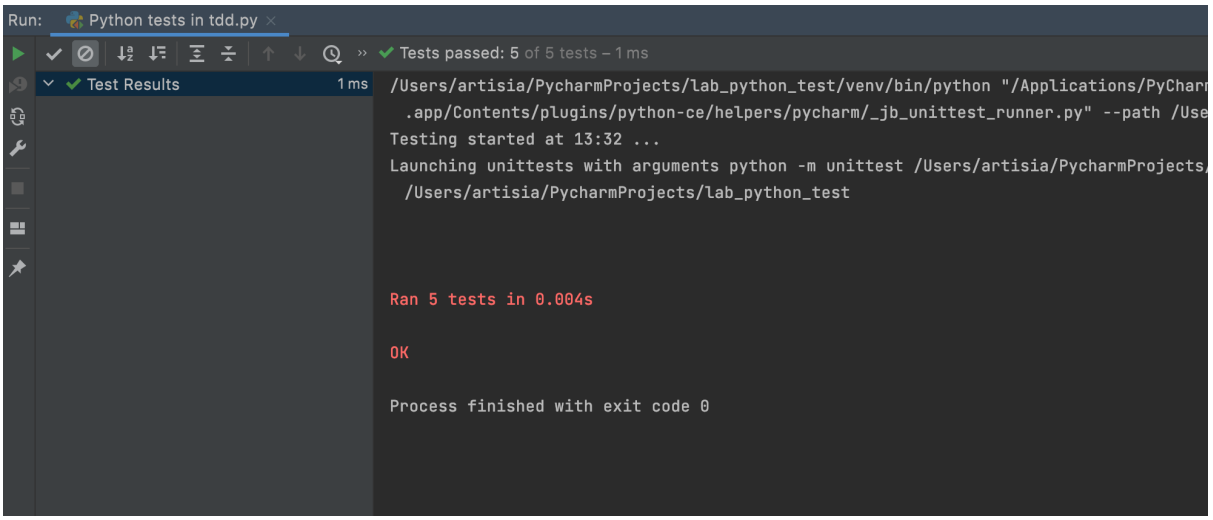
Файл **tdd.py**

```
import unittest
from bridge import *
```

```python
class Tests(unittest.TestCase):
    def setUp(self):
        self.cook = SaladCook()
        self.chief = SaladChief(self.cook)
    def test_start_cooking(self):
        self.assertEqual(self.chief.StartCooking(),
"Started cooking!")
    def test_cooking(self):
      self.assertEqual(self.chief.cook_meat_salad(),
"Cooking meat salad...")
      self.assertEqual(self.chief.cook_vegetarian_salad(),
"Cooking vegetarian salad...")
    def test_ingredients(self):
      self.chief.cook_meat_salad()
      self.assertEqual(self.cook.salad.list_ingredients(),
"Salad ingredients: vegetables, meat, sauce")
      self.chief.cook_vegetarian_salad()
      self.assertEqual(self.cook.salad.list_ingredients(),
"Salad ingredients: vegetables, sauce")
    def test_no_cooking(self):
      self.assertEqual(self.cook.salad.list_ingredients(),
"There is no salad ready")
    def test_no_recipe(self):
        with self.assertRaises(AttributeError):
            self.chief.cook_pizza()

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

Результаты **tdd - тестирования**



Файл **bdd.py**

```
from behave import given, when, then
from bridge import *

@given(u'Chief starts cooking')
def step_impl(context):
    context.cook = SaladCook()
    context.chief = SaladChief(context.cook)
@when(u'Someone ordered a {order}')
def step_impl(context, order: str):
    if order == "meat salad":
        context.result = "Cooking meat salad..."
    elif order == "vegetarian salad":
        context.result = "Cooking vegetarian salad..."

@given(u'A meat salad')
def step_impl(context):
    context.cook = SaladCook()
    context.chief = SaladChief(context.cook)
    context.chief.cook_meat_salad()
@given(u'A vegetarian salad')
def step_impl(context):
    context.cook = SaladCook()
    context.chief = SaladChief(context.cook)
    context.chief.cook_vegetarian_salad()
@given(u'No salad')
def step_impl(context):
    context.cook = SaladCook()
    context.chief = SaladChief(context.cook)
@when(u'Someone asked the ingredients')
def step_impl(context):
    context.result = context.cook.salad.list_ingredients()

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

Файл **bdd.feature**

```
Feature: cooking salads
 Scenario: Chief starts cooking meat salad
    Given Chief starts cooking
    When Someone ordered a meat salad
    Then The next result is expected: "Cooking meat
salad..."
```

```
 Scenario: Chief starts cooking vegetarian salad
   Given Chief starts cooking
   When Someone ordered a vegetarian salad
   Then The next result is expected: "Cooking
vegetarian salad..."
 Scenario: Someone asked the ingredients of a meat
salad
   Given A meat salad
   When Someone asked the ingredients
   Then The next result is expected: "Salad
ingredients: vegetables, meat, sauce"
 Scenario: Someone asked the ingredients of a
vegetarian salad
   Given A vegetarian salad
   When Someone asked the ingredients
   Then The next result is expected: "Salad
ingredients: vegetables, sauce"
 Scenario: Someone asked the ingredients of an
unready salad
   Given No salad
   When Someone asked the ingredients
   Then The next result is expected: "There is no
salad ready"
```

Результаты **bdd - тестирования**

```
Terminal:  Local  +
(venv) artisia@MacBook-Air-Ulia lab_python_test % behave features/bdd.feature
Feature: cooking salads # features/bdd.feature:1

  Scenario: Chief starts cooking meat salad                    # features/bdd.feature:2
    Given Chief starts cooking                                 # features/steps/bdd.py:4 0.002s
    When Someone ordered a meat salad                          # features/steps/bdd.py:8 0.000s
    Then The next result is expected: "Cooking meat salad..." # features/steps/bdd.py:33 0.000s

  Scenario: Chief starts cooking vegetarian salad                    # features/bdd.feature:6
    Given Chief starts cooking                                       # features/steps/bdd.py:4 0.000s
    When Someone ordered a vegetarian salad                          # features/steps/bdd.py:8 0.001s
    Then The next result is expected: "Cooking vegetarian salad..." # features/steps/bdd.py:33 0.000s

  Scenario: Someone asked the ingredients of a meat salad                        # features/bdd.feature:10
    Given A meat salad                                                           # features/steps/bdd.py:15 0.001s
    When Someone asked the ingredients                                           # features/steps/bdd.py:29 0.001s
    Then The next result is expected: "Salad ingredients: vegetables, meat, sauce" # features/steps/bdd.py:33 0.000s

  Scenario: Someone asked the ingredients of a vegetarian salad              # features/bdd.feature:14
    Given A vegetarian salad                                                 # features/steps/bdd.py:20 0.001s
    When Someone asked the ingredients                                       # features/steps/bdd.py:29 0.001s
    Then The next result is expected: "Salad ingredients: vegetables, sauce" # features/steps/bdd.py:33 0.000s

  Scenario: Someone asked the ingredients of an unready salad   # features/bdd.feature:18
    Given No salad                                              # features/steps/bdd.py:25 0.000s
    When Someone asked the ingredients                          # features/steps/bdd.py:29 0.000s
    Then The next result is expected: "There is no salad ready" # features/steps/bdd.py:33 0.000s

1 feature passed, 0 failed, 0 skipped
5 scenarios passed, 0 failed, 0 skipped
15 steps passed, 0 failed, 0 skipped, 0 undefined
Took 0m0.009s
(venv) artisia@MacBook-Air-Ulia lab_python_test %

Run  TODO  Problems  Terminal  Python Packages  Python Console
```

Файл **mock.py**

```python
from bridge import *
from unittest import TestCase
from unittest.mock import patch

class Tests(TestCase):
    @patch('bridge.CookedSalad.list_ingredients',
return_value="Salad ingredients: vegetables, meat, sauce")
    def test_meat_ingr(self, list_igredients):
        self.cook = SaladCook()

self.assertEqual(self.cook.salad.list_ingredients(),
"Salad ingredients: vegetables, meat, sauce")

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

Результаты **mock - тестирования**