|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **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)** |

**Факультет «Информатика и системы управления»**

**Кафедра ИУ5 «Системы обработки информации и управления»**

Курс «Разработка интернет-приложений»

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

Выполнила:

студент группы ИУ5-51Б

Павловская А.А.

26.12.2020

Проверил:

|  |
| --- |
| преподаватель каф. ИУ5 |
| Гапанюк Ю.Е. |

Москва, 2020 г.

**Цель лабораторной работы:** изучение реализации шаблонов проектирования и возможностей модульного тестирования в языке Python.

**Задание:**

Необходимо для произвольной предметной области реализовать три шаблона проектирования: один порождающий, один структурный и один поведенческий. В качестве справочника шаблонов можно использовать следующий каталог.

Для каждой реализации шаблона необходимо написать модульный тест. В модульных тестах необходимо применить следующие технологии:

TDD - фреймворк.

BDD - фреймворк.

Создание Mock-объектов.

**Текст программы:**

**IcecreamBuilder.py**

from \_\_future\_\_ import annotations

from abc import ABC, abstractmethod, abstractproperty

from typing import Any

from Observer import StartIcecreamObserver, FinishIcecreamObserver,IcecreamObserver

from Facade import Facade

#from icecream\_cafe.Observer import StartIcecreamObserver, FinishIcecreamObserver,IcecreamObserver

#from icecream\_cafe.Facade import Facade

class Icecream():

def \_\_init\_\_(self) -> None:

self.cream = []

self.syrop = []

self.addition = []

def set\_cream(self, part: Any) -> None:

self.cream.append(part)

def set\_syrop(self, part: Any) -> None:

self.syrop.append(part)

def set\_addition(self, part: Any) -> None:

self.addition.append(part)

def list\_parts(self) -> None:

print(f"Base - "," ".join(self.cream))

print(f"Syrop - "," ".join(self.syrop))

print(f"Addition - "," ".join(self.addition))

class IcecreamBuilder(ABC):

icecream\_type = None

@abstractproperty

def icecream(self) -> None:

pass

@abstractmethod

def produce\_cream(self) -> None:

pass

@abstractmethod

def produce\_syrop(self) -> None:

pass

@abstractmethod

def produce\_addition(self) -> None:

pass

class Type1IcecreamBuilder(IcecreamBuilder):

def \_\_init\_\_(self) -> None:

self.icecream\_type = "Type 1 icecream"

self.reset()

def reset(self) -> None:

self.\_icecream = Icecream()

@property

def icecream(self) -> Icecream:

icecream = self.\_icecream

self.reset()

return icecream

def produce\_cream(self) -> None:

self.\_icecream.set\_cream("Plombir")

def produce\_syrop(self) -> None:

self.\_icecream.set\_syrop("Caramel")

def produce\_addition(self) -> None:

self.\_icecream.set\_addition("Crushed nuts")

class Type2IcecreamBuilder(IcecreamBuilder):

def \_\_init\_\_(self) -> None:

self.icecream\_type = "Type 2 icecream"

self.reset()

def reset(self) -> None:

self.\_icecream = Icecream()

@property

def icecream(self) -> Icecream:

icecream = self.\_icecream

self.reset()

return icecream

def produce\_cream(self) -> None:

self.\_icecream.set\_cream("Creme brulee")

def produce\_syrop(self) -> None:

self.\_icecream.set\_syrop("Chocolate syrop")

def produce\_addition(self) -> None:

self.\_icecream.set\_addition("Cookies")

class Iceman:

def \_\_init\_\_(self, name, start\_observer, finish\_observer) -> None:

self.\_builder = None

self.name = name

self.\_start\_icecream\_observer = start\_observer or StartIcecreamObserver()

self.\_finish\_icecream\_observer = finish\_observer or FinishIcecreamObserver()

@property

def builder(self) -> IcecreamBuilder:

return self.\_builder

@builder.setter

def builder(self, builder: IcecreamBuilder) -> None:

self.\_builder = builder

self.notify(self.\_start\_icecream\_observer)

def build\_pure\_icecream(self) -> None:

self.builder.produce\_cream()

self.notify(self.\_finish\_icecream\_observer)

def build\_complex\_icecream(self) -> None:

self.builder.produce\_cream()

self.builder.produce\_syrop()

self.builder.produce\_addition()

self.notify(self.\_finish\_icecream\_observer)

def notify(self, observer: IcecreamObserver) -> None:

print("Waiter speaks:")

observer.update(self)

if \_\_name\_\_ == "\_\_main\_\_":

director1 = Iceman("Iceman1", None, None)

builder1 = Type1IcecreamBuilder()

director1.builder = builder1

director2 = Iceman("Iceman2", None, None)

builder2 = Type2IcecreamBuilder()

director2.builder = builder2

icemans = Facade(director1, director2)

icemans.iceman1\_operation()

icemans.iceman2\_operation()

print("-------------------------------------------------------------")

director = Iceman("Iceman1", None, None)

builder = Type1IcecreamBuilder()

director.builder = builder

print("Basic icecream: ")

director.build\_pure\_icecream()

print("\n")

print("Complex icecream: ")

director.build\_complex\_icecream()

print("\n")

print("Custom icecream: ")

builder.produce\_cream()

builder.produce\_syrop()

builder.icecream.list\_parts()

print("\n")

**Observer.py**

from \_\_future\_\_ import annotations

from abc import ABC, abstractmethod

#from icecream\_cafe.IcecreamBuilder import Iceman

class IcecreamObserver(ABC):

@abstractmethod

def update(self, subject: Iceman) -> None:

pass

class StartIcecreamObserver(IcecreamObserver):

def update(self, subject: Iceman) -> None:

print(f"{subject.name} makes {subject.builder.icecream\_type}\n")

class FinishIcecreamObserver(IcecreamObserver):

def update(self, subject: Iceman) -> None:

print(f"{subject.name} finished {subject.builder.icecream\_type}")

print(f"Parts:")

subject.builder.icecream.list\_parts()

**Facade.py**

from \_\_future\_\_ import annotations

#from icecream\_cafe.IcecreamBuilder import Iceman

class Facade:

def \_\_init\_\_(self, iceman1: Iceman, iceman2: Iceman) -> None:

self.iceman1 = iceman1

self.iceman2 = iceman2

def iceman1\_operation(self) -> None:

print("Basic icecream: ")

self.iceman1.build\_pure\_icecream()

print("\n")

print("Complex icecream: ")

self.iceman1.build\_complex\_icecream()

print("\n")

def iceman2\_operation(self) -> None:

print("Basic icecream: ")

self.iceman2.build\_pure\_icecream()

print("\n")

print("Complex icecream: ")

self.iceman2.build\_complex\_icecream()

print("\n")

**test\_facade.py**

import unittest

#from icecream\_cafe.IcecreamBuilder import Iceman

#from icecream\_cafe.Facade import Facade

from IcecreamBuilder import Iceman

from Facade import Facade

test\_iceman1 = Iceman("TestIceman1", None, None)

test\_iceman2 = Iceman("TestIceman2", None, None)

class TestFacade(unittest.TestCase):

def test\_facade\_create\_iceman(self):

facade = Facade(test\_iceman1, test\_iceman2)

unknown\_iceman1 = facade.iceman1

unknown\_iceman2 = facade.iceman2

self.assertEqual(unknown\_iceman1.name, test\_iceman1.name)

self.assertEqual(unknown\_iceman2.name, test\_iceman2.name)

if \_\_name\_\_ == '\_\_main\_\_':

unittest.main()

**test\_observer.py**

from unittest import main

from unittest import TestCase

from unittest.mock import patch

#from icecream\_cafe.IcecreamBuilder import Iceman, Type1IcecreamBuilder

#from icecream\_cafe.Observer import StartIcecreamObserver,FinishIcecreamObserver

from IcecreamBuilder import Iceman, Type1IcecreamBuilder

from Observer import StartIcecreamObserver,FinishIcecreamObserver

class TestObserver(TestCase):

@patch('Observer.StartIcecreamObserver')

def test\_start\_icecream\_observer(self, MockObserver):

observer = MockObserver

iceman = Iceman("Iceman", observer, None)

builder1 = Type1IcecreamBuilder()

iceman.builder = builder1

observer.update.assert\_called\_once()

@patch('Observer.FinishIcecreamObserver')

def test\_finish\_icecream\_observer(self, MockObserver):

observer = MockObserver

iceman = Iceman("Iceman", None, observer)

builder1 = Type1IcecreamBuilder()

iceman.builder = builder1

iceman.build\_complex\_icecream()

observer.update.assert\_called\_once()

if \_\_name\_\_ == '\_\_main\_\_':

main()

**test\_icecreambuilder.py**

from radish import given, when, then

from IcecreamBuilder import Type1IcecreamBuilder

@given("I am creating Type1 Builder")

def create\_type1builder(step):

step.context.builder = Type1IcecreamBuilder()

@when("I make him do cream only icecream")

def builder\_produce\_only\_cream(step):

step.context.builder.produce\_cream()

@then("Then I expect that cream will be {cream\_name: w}")

def expect\_cream\_name(step, cream\_name):

assert step.context.icecream.cream == [cream\_name]

**f\_icecreambuilder.feature**

Feature: Type1IcecreamBuilder

The goal is to test Type1IcecreamBuilder

Scenario: Check Type1IcecreamBuilder results

Given I am creating Type1 Builder

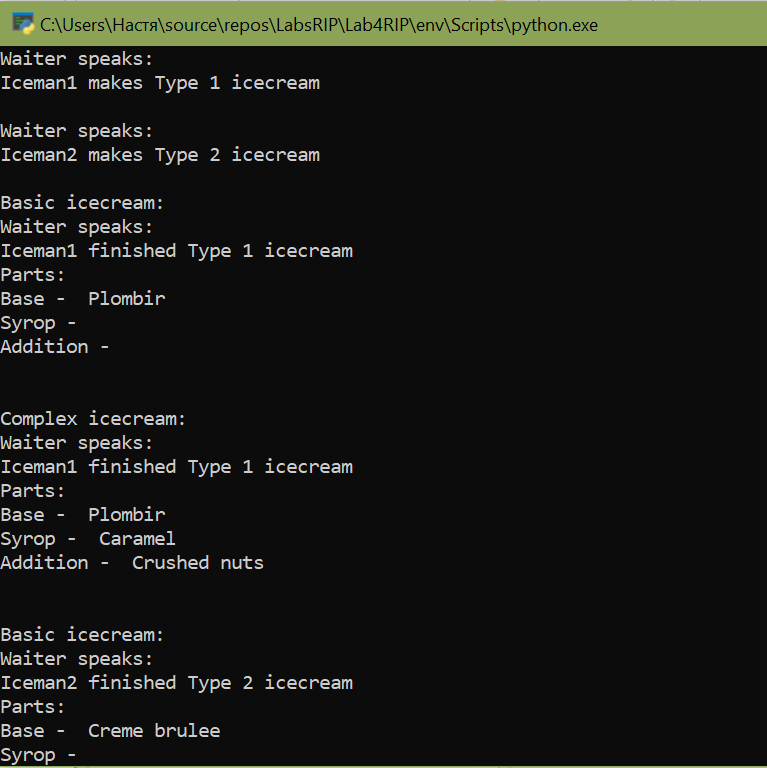
When I make him do cream only icecream

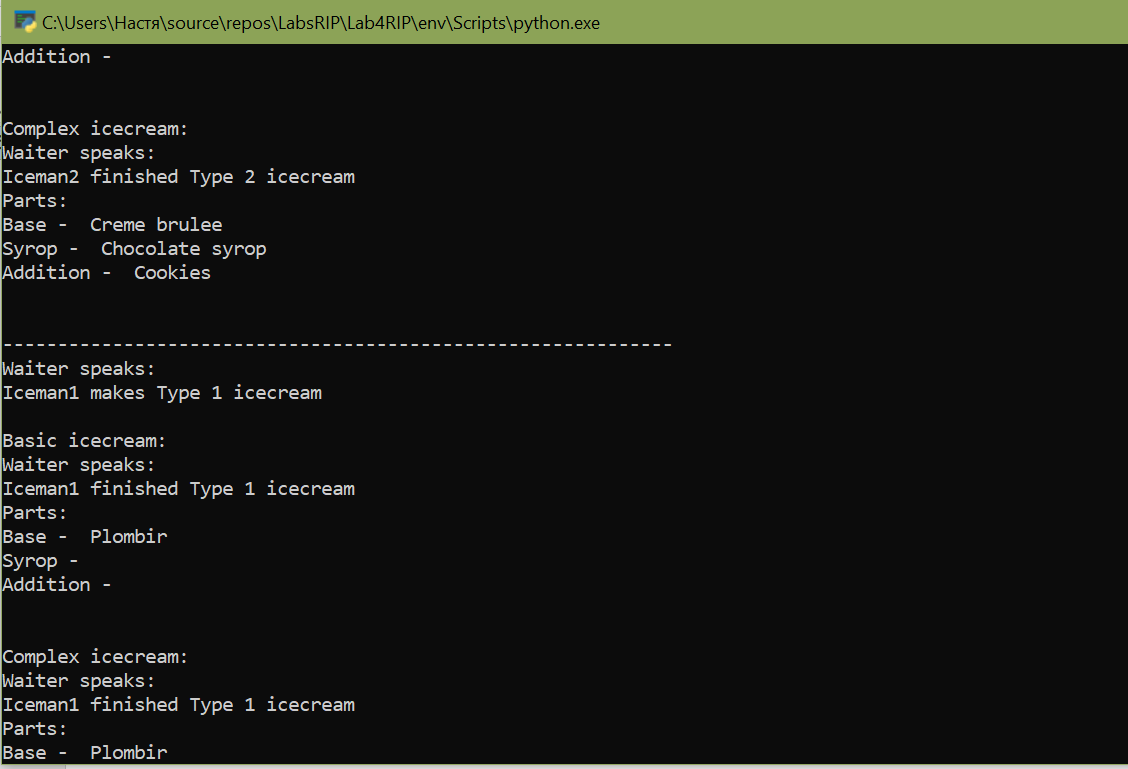
Then I expect that cream will be Plombir

**Ссылка на репозиторий Github:**

<https://github.com/PavlAA79/LabsRIP.git>

**Экранные формы с примерами выполнения программы:**

****

****

