Московский государственный технический

университет им. Н.Э. Баумана

Факультет «Радиотехника»
Кафедра «Информатика и вычислительная техника»

Курс «Парадигмы и конструкции языков программирования»

Отчет по лабораторной работе №4

« Шаблоны проектирования и модульное тестирование в Python»

Выполнил:	Проверил:
студент группы РТ5-31Б:	преподаватель каф. ИУ5
Михеева В.А.	Гапанюк Ю.Е.
Подпись и дата:	Подпись и дата:

Москва, 2023 г.

Постановка задачи:

1. Необходимо для произвольной предметной области реализовать от одного до трех шаблонов проектирования: один порождающий, один структурный и один поведенческий. В качестве справочника шаблонов можно использовать следующий каталог. Для сдачи лабораторной работы в минимальном варианте достаточно реализовать один паттерн.
2. В модульных тестах необходимо применить следующие технологии:
    a. TDD - фреймворк.
    b. BDD - фреймворк.
    c. Создание Mock-объектов.

шаблоны проектирования

Файл Decorator.py

```
from abc import ABC, abstractmethod
class Component(ABC):
  @abstractmethod
  def do_work(self):
      pass
class Window(Component):
  def do_work(self):
      return('Empty Window')
class Decorator(Component):
  def __init__(self, component=''):
      self.component = component
  @abstractmethod
  def do_work(self):
      pass
class Border(Decorator):
  def do_work(self):
      return(f"<Border>{self.component.do_work()}</Border>")
class Thick(Decorator):
  def do_work(self):
      return(f"<Thick>{self.component.do_work()}</Thick>")

if __name__ == "__main__":
    a=Window()
    print(Thick(Border(a)).do_work())
```

Файл Fabric_method.py

```
import math
class Point(object):
```

```
    def show(self, a, b):
       raise NotImplementedError
    def Print(self, a, b):
        self.show(a, b)
        return f'x: {self.x}, y: {self.y}'
class Polar(Point):
    def show(self, r, theta):
        self.x = r * math.cos(theta)
        self.y = r * math.sin(theta)

class Decart(Point):
    def show(self, x, y):
        self.x=x
        self.y=y

if __name__ == '__main__':
    print(Polar().Print(1, 2))
    print(Decart().Print(1, 2))
```

Файл formock.py

```
def formock():
    path = "res\primer_faila.txt"
    with open(path, 'r') as a:
        contents = a.read()
    print(contents)
if __name__ == "__main__":
    formock()
```

Файл Observer.py

```
class WidowSystemNotification:

    def __init__(self):
        self.observers = []

    def add(self, observer):
        self.observers.append(observer)
        print(f'add window {observer.number}')

    def delete(self, observer):
        self.observers.remove(observer)
        print(f'add window {observer.number}')

    def notification(self):
        for observer in self.observers:
            observer.get_notification()
class AbstractObserver(ABC):
    @abstractmethod
    def get_notification(self):
```

```
        pass

class Window_web(AbstractObserver):
    def __init__(self, number):
        self.number = number

    def get_notification(self):
        print(f'web-window {self.number} get notification')
class Window_mobile(AbstractObserver):
    def __init__(self, number):
        self.number = number

    def get_notification(self):
        print(f'mobile-window {self.number} get notification')


if __name__ == "__main__":
    window_web_1 = Window_web(1)
    window_web_2 = Window_web(2)
    window_web_3 = Window_web(3)

    window_mobile_1 = Window_mobile(1)
    window_mobile_2 = Window_mobile(2)

    system = WidowSystemNotification()
    system.add(window_web_1)
    system.add(window_web_2)
    system.add(window_web_3)

    system.add(window_mobile_1)
    system.add(window_mobile_2)
    system.notification()
```

2. Для модульных тестов используется фреймворк behave
Файл forDecorator.py

```
from behave import *
from Decorator import Window, Border, Thick

@given('an empty window')
def step_given_empty_window(context):
    context.window = Window()

@given('the border is given')
def step_given_border(context):
    context.border = Border(context.window)
```

```
@given('a thick is given')
def step_given_thick(context):
    context.thick = Thick(context.border)

@when('the window works flawlessly')
def step_when_window_works(context):
    context.result = context.window.do_work()

@when('performing work at the border')
def step_when_border_works(context):
    context.result = context.border.do_work()

@when('wonderful work in a thick border')
def step_when_thick_border_works(context):
    context.result = context.thick.do_work()

@then('get the result {expected}')
def step_then_expected_result(context, expected):
    assert context.result == expected, f"Error: expect
{expected}, get {context.result}"
```

Файл myfeatureSteps.py

```
from behave import *
import Fabric_method

@given('a polar point with radius {radius:d} and angle
{angle:d} degrees')
def step_impl(context, radius, angle):
    context.radius = radius
    context.angle = angle

@when('converting the polar point to Cartesian
coordinates')
def step_impl(context):
    context.result =
Fabric_method.Polar().Print(context.radius, context.angle)

@then('the resulting Cartesian coordinates should be x:
{result_x:f}, y: {result_y:f}')
```

```
def step_impl(context, result_x, result_y):
    assert str(context.result) == f'x: {result_x}, y:
{result_y}', f"Error: expected {result_x}, {result_y} but
got {context.result_x}, {context.result_y}."


@given('a Cartesian point with x-coordinate {x:d} and y-
coordinate {y:d}')
def step_impl(context, x, y):
    context.x = x
    context.y = y

@when('converting the Cartesian point to polar
coordinates')
def step_impl(context):
    context.result =
Fabric_method.Decart().Print(context.x, context.y)

@then('the resulting coordinates should be x:
{result_x:w}, y: {result_y:w}')
def step_impl(context, result_x, result_y):
    assert str(context.result) == f'x: {result_x}, y:
{result_y}', f"Error: expected {result_x}, {result_y} but
got {context.result_x}, {context.result_y}."
```

Файл Decoratorfeature.feature

```
Feature: Decorator
  Scenario: Creating an empty window
    Given an empty window
    When the window works flawlessly
    Then get the result Empty Window

  Scenario: Adding a border to a window
    Given an empty window
    And the border is given
    When performing work at the border
    Then get the result <Border>Empty Window</Border>

  Scenario: Adding a property-thick to border around the
```

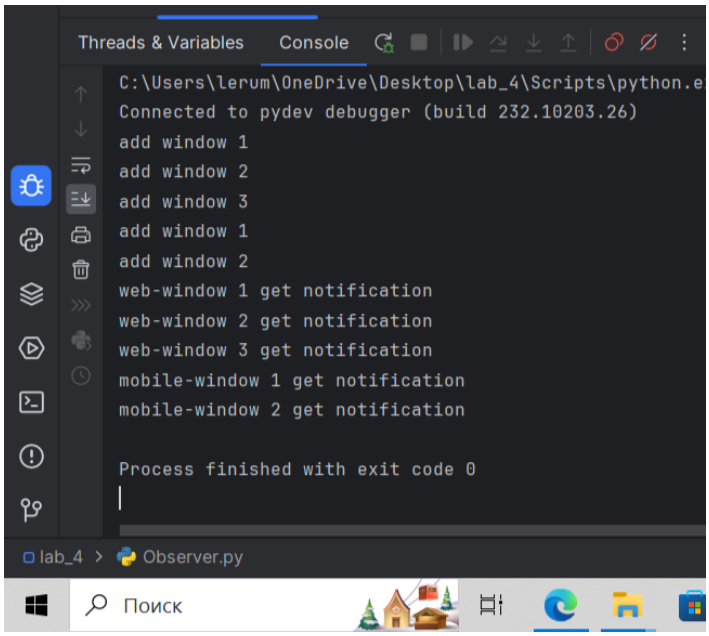
```
window.
    Given an empty window
    And the border is given
    And a thick is given
    When wonderful work in a thick border
    Then get the result <Thick><Border>Empty
Window</Border></Thick>
```

Файл myfeature.feature

```
Feature: Point Conversion

    Scenario: Convert Cartesian coordinates to polar
coordinates
        Given a Cartesian point with x-coordinate 3 and y-
coordinate 4
        When converting the Cartesian point to polar
coordinates
        Then the resulting coordinates should be x: 3, y:
4

    Scenario: Convert polar coordinates to Cartesian
coordinates
        Given a polar point with radius 1 and angle 2
degrees
        When converting the polar point to Cartesian
coordinates
        Then the resulting Cartesian coordinates should be
x: -0.4161468365471424, y: 0.9092974268256817
```

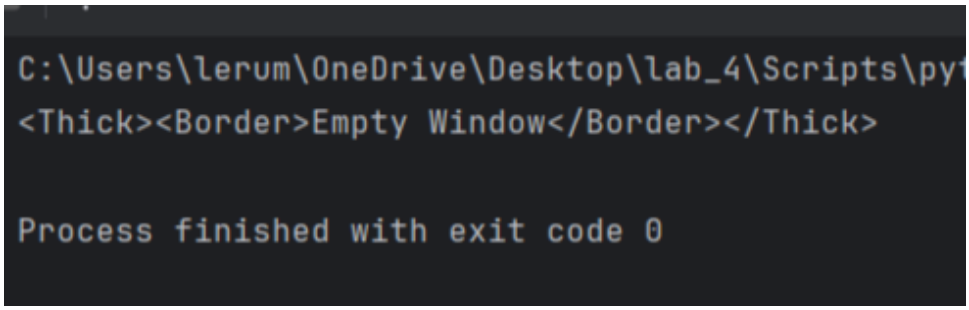
Выполнение программы

Файл Observer.py



Файл formock.py

Файл Decorator.py

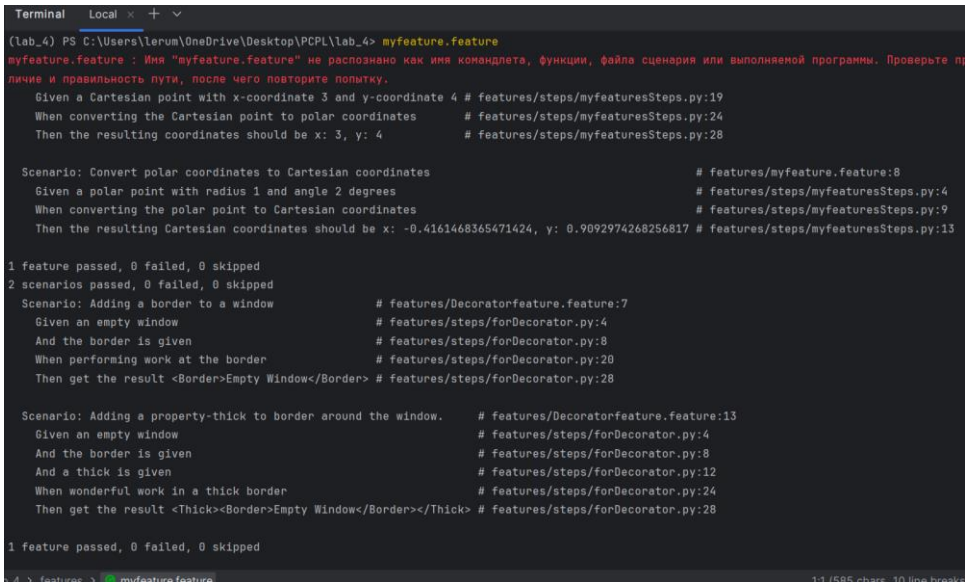


Файл Fabric_method.py

```
C:\Users\lerum\OneDrive\Desktop\lab_4\Scripts\python.exe C:\Users\lerum\OneDrive\Desktop\PCPL\
x: -0.4161468365471424, y: 0.9092974268256817
x: 1, y: 2
```

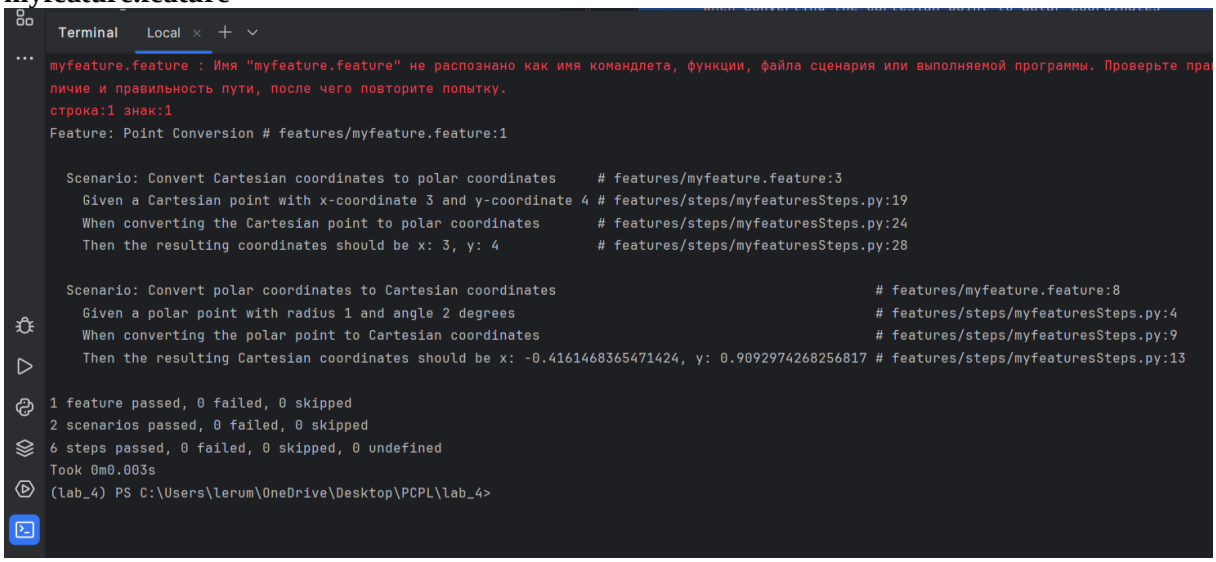
Тесты:

Decoratorfeature.feature



```
Terminal  Local
(lab_4) PS C:\Users\lerum\OneDrive\Desktop\PCPL\lab_4> myfeature.feature
myfeature.feature : Имя "myfeature.feature" не распознано как имя командлета, функции, файла сценария или выполняемой программы. Проверьте пр
личие и правильность пути, после чего повторите попытку.
    Given a Cartesian point with x-coordinate 3 and y-coordinate 4 # features/steps/myfeaturesSteps.py:19
    When converting the Cartesian point to polar coordinates       # features/steps/myfeaturesSteps.py:24
    Then the resulting coordinates should be x: 3, y: 4            # features/steps/myfeaturesSteps.py:28

  Scenario: Convert polar coordinates to Cartesian coordinates                                      # features/myfeature.feature:8
    Given a polar point with radius 1 and angle 2 degrees                                           # features/steps/myfeaturesSteps.py:4
    When converting the polar point to Cartesian coordinates                                        # features/steps/myfeaturesSteps.py:9
    Then the resulting Cartesian coordinates should be x: -0.4161468365471424, y: 0.9092974268256817 # features/steps/myfeaturesSteps.py:13

1 feature passed, 0 failed, 0 skipped
2 scenarios passed, 0 failed, 0 skipped
  Scenario: Adding a border to a window              # features/Decoratorfeature.feature:7
    Given an empty window                            # features/steps/forDecorator.py:4
    And the border is given                          # features/steps/forDecorator.py:8
    When performing work at the border               # features/steps/forDecorator.py:20
    Then get the result <Border>Empty Window</Border> # features/steps/forDecorator.py:28

  Scenario: Adding a property-thick to border around the window.     # features/Decoratorfeature.feature:13
    Given an empty window                                            # features/steps/forDecorator.py:4
    And the border is given                                          # features/steps/forDecorator.py:8
    And a thick is given                                             # features/steps/forDecorator.py:12
    When wonderful work in a thick border                            # features/steps/forDecorator.py:24
    Then get the result <Thick><Border>Empty Window</Border></Thick> # features/steps/forDecorator.py:28

1 feature passed, 0 failed, 0 skipped
```

myfeature.feature



```
Terminal  Local
myfeature.feature : Имя "myfeature.feature" не распознано как имя командлета, функции, файла сценария или выполняемой программы. Проверьте пра
личие и правильность пути, после чего повторите попытку.
строка:1 знак:1
Feature: Point Conversion # features/myfeature.feature:1

  Scenario: Convert Cartesian coordinates to polar coordinates     # features/myfeature.feature:3
    Given a Cartesian point with x-coordinate 3 and y-coordinate 4 # features/steps/myfeaturesSteps.py:19
    When converting the Cartesian point to polar coordinates       # features/steps/myfeaturesSteps.py:24
    Then the resulting coordinates should be x: 3, y: 4            # features/steps/myfeaturesSteps.py:28

  Scenario: Convert polar coordinates to Cartesian coordinates                                      # features/myfeature.feature:8
    Given a polar point with radius 1 and angle 2 degrees                                           # features/steps/myfeaturesSteps.py:4
    When converting the polar point to Cartesian coordinates                                        # features/steps/myfeaturesSteps.py:9
    Then the resulting Cartesian coordinates should be x: -0.4161468365471424, y: 0.9092974268256817 # features/steps/myfeaturesSteps.py:13

1 feature passed, 0 failed, 0 skipped
2 scenarios passed, 0 failed, 0 skipped
6 steps passed, 0 failed, 0 skipped, 0 undefined
Took 0m0.003s
(lab_4) PS C:\Users\lerum\OneDrive\Desktop\PCPL\lab_4>
```