**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 |
| Кузнецов М.Д. |  |  |  | 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.](https://ru.wikipedia.org/wiki/%D0%A8%D0%B0%D0%B1%D0%BB%D0%BE%D0%BD_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F) Для сдачи лабораторной работы в минимальном варианте достаточно реализовать один паттерн.
2. В модульных тестах необходимо применить следующие технологии:
   1. TDD - фреймворк.
   2. BDD - фреймворк.
   3. 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 ycoordinate {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 ycoordinate 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

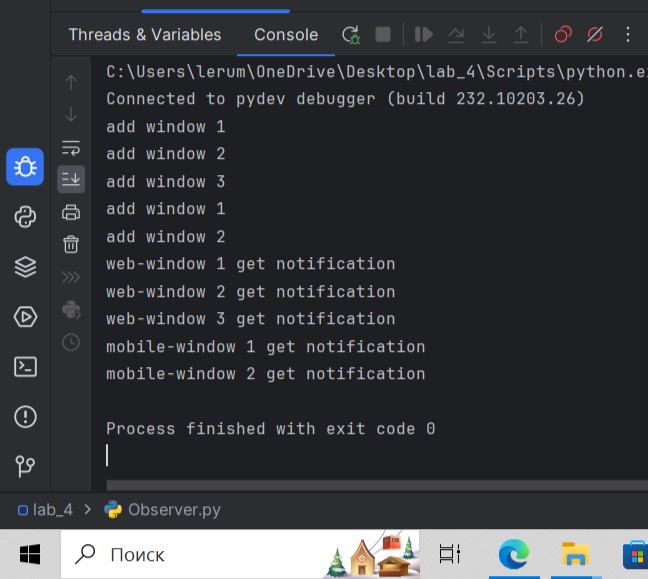
Given a polar point with radius 1 and angle 2 degrees

When converting the polar point to Cartesian coordinates

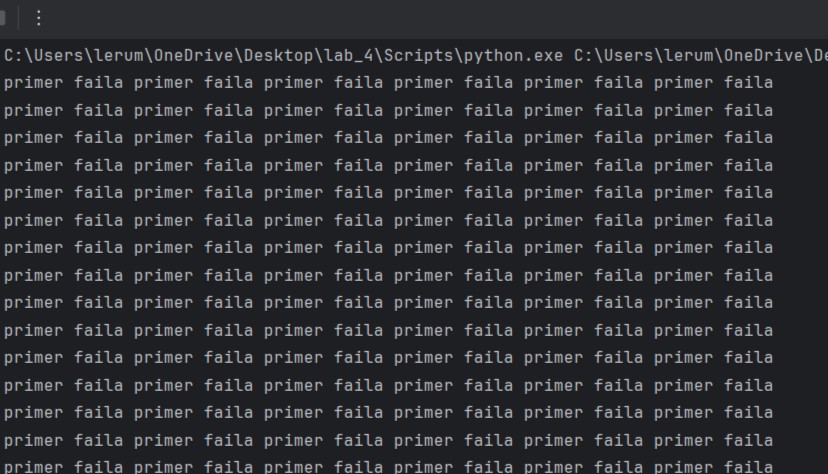
Then the resulting Cartesian coordinates should be x: -0.4161468365471424, y: 0.9092974268256817

**Выполнение программы**

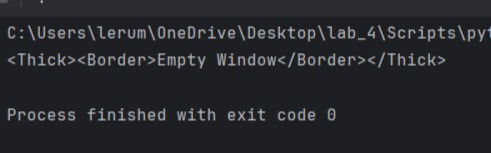
Файл Observer.py



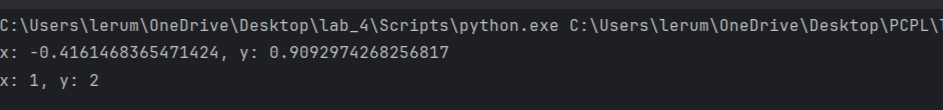
Файл formock.py



Файл Decorator.py

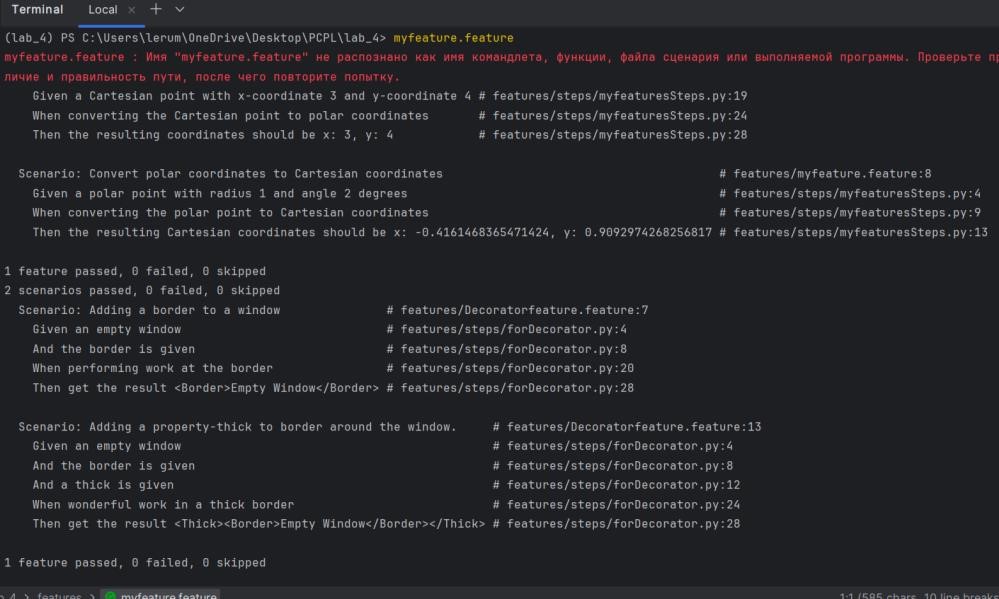


Файл Fabric\_method.py



Тесты:

**Decoratorfeature.feature**



## myfeature.feature

