

**Министерство образования и науки Российской Федерации**

**Федеральное государственное бюджетное образовательное учреждение**

**высшего профессионального образования**

**«Московский государственный технический университет имени Н.Э. Баумана»**

**(МГТУ им. Н.Э. Баумана)**

|  |  |
| --- | --- |
| **ФАКУЛЬТЕТ** | **«Информатики и систем управления»** |
| КАФЕДРА | ИУ5 |

Дисциплина «Разработка интернет-приложений»

Отчет по лабораторной работе №4

«Шаблоны проектирования и модульное тестирование в Python.»

|  |  |  |
| --- | --- | --- |
| Студент | группы ИУ5-52Б | Гришин Илья |
| Преподаватель |  | Гапанюк Ю.Е. |

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

1. **Задание**
2. Необходимо для произвольной предметной области реализовать три шаблона проектирования: один порождающий, один структурный и один поведенческий.
3. Для каждой реализации шаблона необходимо написать модульный тест. В модульных тестах необходимо применить следующие технологии:
   * TDD - фреймворк.
   * BDD - фреймворк.
4. **Текст программ**

**Порождающий паттерн проектирования «строитель»:**

stroitel.py

from \_\_future\_\_ import annotations

from abc import ABC, abstractmethod, abstractproperty

from typing import Any

class Builder(ABC):

    @abstractproperty

    def product(self) -> None:

        pass

    @abstractmethod

    def setEngine(self) -> None:

        pass

    @abstractmethod

    def setFuel(self) -> None:

        pass

    @abstractmethod

    def setMTransmission(self) -> None:

        pass

    @abstractmethod

    def setATransmission(self) -> None:

        pass

    @abstractmethod

    def setConditioner(self) -> None:

        pass

    @abstractmethod

    def setOnBoardComp(self) -> None:

        pass

    @abstractmethod

    def setFogLights(self) -> None:

        pass

class ConcreteBuilderCar(Builder):

    def \_\_init\_\_(self) -> None:

        self.reset()

    def reset(self) -> None:

        self.\_product = ProductCar()

    @property

    def product(self) -> ProductCar:

        product = self.\_product

        self.reset()

        return product

    def setEngine(self) -> None:

        self.\_product.add("Двигатель легкового автомобиля")

    def setFuel(self) -> None:

        self.\_product.add("Бензин")

    def setMTransmission(self) -> None:

        self.\_product.add("МКП")

        return True

    def setATransmission(self) -> None:

        self.\_product.add("АКП")

    def setConditioner(self) -> None:

        self.\_product.add("Кондиционер")

    def setOnBoardComp(self) -> None:

        self.\_product.add("Бортовой компьютер")

    def setFogLights(self) -> None:

        self.\_product.add("Противотуманные фары")

class ConcreteBuilderTruck(Builder):

    def \_\_init\_\_(self) -> None:

        self.reset()

    def reset(self) -> None:

        self.\_product = ProductTruck()

    @property

    def product(self) -> ProductTruck:

        product = self.\_product

        self.reset()

        return product

    def setEngine(self) -> None:

        self.\_product.add("Двигатель грузового автомобиля")

    def setFuel(self) -> None:

        self.\_product.add("Дизель")

    def setMTransmission(self) -> None:

        self.\_product.add("МКП")

        return True

    def setATransmission(self) -> None:

        self.\_product.add("АКП")

    def setConditioner(self) -> None:

        self.\_product.add("Кондиционер")

    def setOnBoardComp(self) -> None:

        self.\_product.add("Бортовой компьютер")

    def setFogLights(self) -> None:

        self.\_product.add("Противотуманные фары")

class ProductCar():

    def \_\_init\_\_(self) -> None:

        self.parts = []

    def add(self, part: Any) -> None:

        self.parts.append(part)

    def list\_parts(self) -> None:

        print(f"Комплектация легкового автомобиля: {', '.join(self.parts)}", end="")

        return f"{', '.join(self.parts)}"

class ProductTruck():

    def \_\_init\_\_(self) -> None:

        self.parts = []

    def add(self, part: Any) -> None:

        self.parts.append(part)

    def list\_parts(self) -> None:

        print(f"Комплектация грузового автомобиля: {', '.join(self.parts)}", end="")

        return f"{', '.join(self.parts)}"

class Director:

    def \_\_init\_\_(self) -> None:

        self.\_builder = None

    @property

    def builder(self) -> Builder:

        return self.\_builder

    @builder.setter

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

        self.\_builder = builder

    def build\_basic(self) -> None:

        print("Базовая конфигурация автомобиля (Basic): ")

        self.builder.setEngine()

        self.builder.setFuel()

        self.builder.setMTransmission()

    def build\_standart(self) -> None:

        print("Стандартная конфигурация автомобиля (Standart): ")

        self.builder.setEngine()

        self.builder.setFuel()

        self.builder.setATransmission()

        self.builder.setConditioner()

    def build\_advanced(self) -> None:

        print("Раcширенная конфигурация автомобиля (Advanced): ")

        self.builder.setEngine()

        self.builder.setFuel()

        self.builder.setATransmission()

        self.builder.setConditioner()

        self.builder.setOnBoardComp()

        self.builder.setFogLights()

if \_\_name\_\_ == "\_\_main\_\_":

    director = Director()

    builder = ConcreteBuilderCar()

    director.builder = builder

    # print("Базовая конфигурация автомобиля (Basic): ")

    director.build\_basic()

    builder.product.list\_parts()

    print("\n")

    # print("Стандартная конфигурация автомобиля (Standart): ")

    director.build\_standart()

    builder.product.list\_parts()

    print("\n")\

    # print("Раcширенная конфигурация автомобиля (Advanced): ")

    director.build\_advanced()

    builder.product.list\_parts()

    print("\n")

    builder = ConcreteBuilderTruck()

    director.builder = builder

    print("Собранная конфигурация автомобиля (Custom): ")

    builder.setEngine()

    builder.setFuel()

    builder.setMTransmission()

    builder.setConditioner()

    builder.setFogLights()

    builder.product.list\_parts()

test\_stroitel.py

import unittest

from stroitel import \*

class TestStringMethods(unittest.TestCase):

    def setUp(self):

        self.director=Director()

    def test\_setEngine(self):

        builder = ConcreteBuilderCar()

        director = Director()

        director.builder = builder

        builder.setEngine()

        self.assertEqual(builder.product.list\_parts(), "Двигатель легкового автомобиля")

        print("\n")

        builder = ConcreteBuilderTruck()

        director.builder = builder

        builder.setEngine()

        self.assertEqual(builder.product.list\_parts(), "Двигатель грузового автомобиля")

        print("\n")

    def test\_build\_config(self):

        director = Director()

        builder = ConcreteBuilderCar()

        director.builder = builder

        director.build\_standart()

        self.assertEqual(builder.product.list\_parts(), "Двигатель легкового автомобиля, Бензин, АКП, Кондиционер")

        print("\n")

        director.build\_basic()

        self.assertEqual(builder.product.list\_parts(), "Двигатель легкового автомобиля, Бензин, МКП")

        print("\n")

        director.build\_advanced()

        self.assertEqual(builder.product.list\_parts(), "Двигатель легкового автомобиля, Бензин, АКП, Кондиционер, Бортовой компьютер, Противотуманные фары")

        print("\n")

    def test\_setMTransmission(self):

        director = Director()

        builder = ConcreteBuilderCar()

        director.builder = builder

        self.assertTrue(builder.setMTransmission())

if \_\_name\_\_ == '\_\_main\_\_':

    unittest.main()

**Структурный паттерн проектирования «адаптер»:**

adapter.py

import codecs

class Target:

    def request(self,path) -> str:

        f = open(path)

        text=f.read()

        f.close()

        return f"Содержимое файла: "+text

class Adaptee:

    def specific\_request(self, path) -> str:

        f = open(path, 'r')

        text=f.read()

        f.close()

        return text

class Adapter(Target, Adaptee):

    def request(self, path) -> str:

        f=codecs.open(path, encoding='utf8')

        text=f.read()

        f.close()

        return f"Содержимое файла: "+text

def client\_code(path, target: "Target") -> None:

    print(target.request(path), end="")

    return target.request(path)

if \_\_name\_\_ == "\_\_main\_\_":

    path = 'C:\\2\\rip\\lab4\\testfiles\\file.txt'

    # path = 'C:\\2\\file.txt'

    print("\n")

    print("Попытка открытия файла без необходимости смены кодировки")

    target = Target()

    client\_code(path, target)

    print("\n\n")

    adaptee = Adaptee()

    print("Попытка открытия файла без смены кодировки")

    print(f"Содержимое файла: {adaptee.specific\_request(path)}", end="\n\n")

    print("Попытка открытия файла с кодировкой UTF-8")

    adapter = Adapter()

    client\_code(path, adapter)

test\_adapter.py

import unittest

from adapter import \*

import os

class TestStringMethods(unittest.TestCase):

    def test\_adapter(self):

        str1="Текстовый Файл --- "

        for index in 'абвгдеёжзийклмнопрстуфхцчшщъыьэюя':

            with self.subTest(index=index):

                s=str(index)

                testpath="C:\\2\\rip\\lab4\\testfiles\\file"+s+".txt"

                str1 = str1 + s

                with open(testpath, 'wb') as f:

                    f.write(str1.encode('utf-8'))

                f.close()

                print("\n")

                adaptee = Adaptee()

                print("Попытка открытия файла без смены кодировки")

                print(f"Содержимое файла: {adaptee.specific\_request(testpath)}", end="\n")

                f.close()

                print("Попытка открытия файла с кодировкой UTF-8")

                adapter = Adapter()

                str2='111'

                if s=='ж':

                    self.assertEqual(client\_code(testpath, adapter), "Содержимое файла: " + str2)

                else:

                    self.assertEqual(client\_code(testpath, adapter), "Содержимое файла: " + str1)

                # self.assertEqual(client\_code(testpath, adapter), "Содержимое файла: " + str1)

                os.remove(testpath)

if \_\_name\_\_ == '\_\_main\_\_':

    unittest.main()

**Поведенческий паттерн проектирования «стратегия»:**

strategy.py

from \_\_future\_\_ import annotations

from abc import ABC, abstractmethod

from typing import List

import json

from tabulate import tabulate

class Context():

    def \_\_init\_\_(self, strategy: Strategy) -> None:

        self.\_strategy = strategy

    @property

    def strategy(self) -> Strategy:

        return self.\_strategy

    @strategy.setter

    def strategy(self, strategy: Strategy) -> None:

        self.\_strategy = strategy

    def do\_some\_business\_logic(self,path) -> None:

        with open(path, encoding='utf-8') as f:

            data = json.load(f)

        result = self.\_strategy.do\_algorithm(data)

        best=[]

        for i in range(0,5):

            best.append(result[i])

        print ("\n        Лучший отель:")

        print ("Название:           "+result[0]['name'])

        print ("Стоимость за день:  "+str(result[0]['costperday']))

        print ("Звезды:             "+str(result[0]['star']))

        print ("Цена за звезду:     "+str(result[0]['starcost']))

        print ("\n             Топ лучших отелей:")

        print (tabulate(best, headers='keys', tablefmt="fancy\_grid"))

        return result[0]['name']

class Strategy(ABC):

    @abstractmethod

    def do\_algorithm(self, data):

        pass

class ConcreteStrategyA(Strategy):

    def do\_algorithm(self, data):

        return sorted(data, key=lambda k: k['star'], reverse = True)

class ConcreteStrategyB(Strategy):

    def do\_algorithm(self, data):

        return sorted(data, key=lambda k: k['costperday'])

class ConcreteStrategyC(Strategy):

    def do\_algorithm(self, data):

        return sorted(data, key=lambda k: k['starcost'])

if \_\_name\_\_ == "\_\_main\_\_":

    path = 'data\_light.json'

    context = Context(ConcreteStrategyA())

    print("\n\nКлиент: Лучший отель по звездам")

    context.do\_some\_business\_logic(path)

    print()

    print("\n\nКлиент: Лучший отель по цене")

    context.strategy = ConcreteStrategyB()

    context.do\_some\_business\_logic(path)

    print("\n\nКлиент: Лучший отель по цене за звезду")

    context.strategy = ConcreteStrategyC()

    context.do\_some\_business\_logic(path)

step.py

from strategy import \*

# -\*- coding: utf-8 -\*-

from radish import given, when, then

@given('"I have the path "{path}"')

def have\_path(step, path):

    step.context.path = path

@when('I am making a request')

def do\_request(step):

    context.strategy = ConcreteStrategyB()

    step.context.result = context.do\_some\_business\_logic(step.context.path)

@then('I expect the result to be "{result}"')

def expect\_result(step, result):

    assert step.context.result == result

test\_strategy.feature

Feature: My first feature file using radish

In order to test my awesome software

I need an awesome BDD tool like radish

to test my software.

Scenario: Test my strategy

Given I have the path "C:\2\rip\lab4\data\_light.json"

When I am making a request

Then I expect the result to be "Отель4"

data\_light.json

[

    {

        "name": "Отель1",

        "costperday": 1000,

        "star": 4,

        "starcost": 250

    },

    {

        "name": "Отель2",

        "costperday": 1000,

        "star": 5,

        "starcost": 200

    },

    {

        "name": "Отель3",

        "costperday": 1200,

        "star": 4,

        "starcost": 300

    },

    {

        "name": "Отель4",

        "costperday": 300,

        "star": 2,

        "starcost": 150

    },

    {

        "name": "Отель5",

        "costperday": 2000,

        "star": 5,

        "starcost": 400

    },

    {

        "name": "Отель6",

        "costperday": 1600,

        "star": 4,

        "starcost": 400

    },

    {

        "name": "Отель7",

        "costperday": 900,

        "star": 3,

        "starcost": 300

    },

    {

        "name": "Отель8",

        "costperday": 750,

        "star": 3,

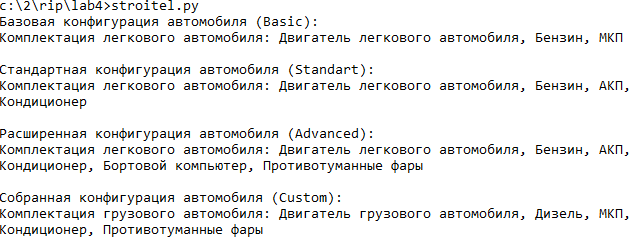
        "starcost": 250

    }

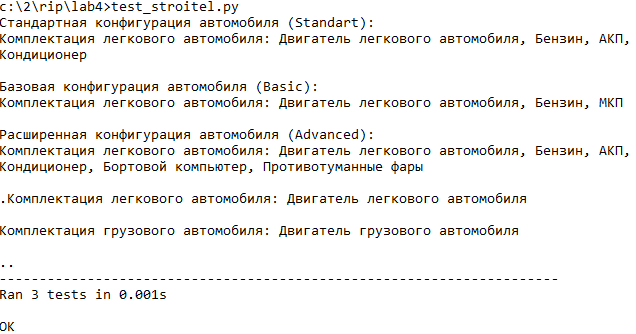
]

1. **Результаты**

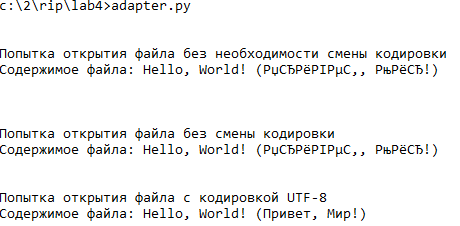
stroitel.py



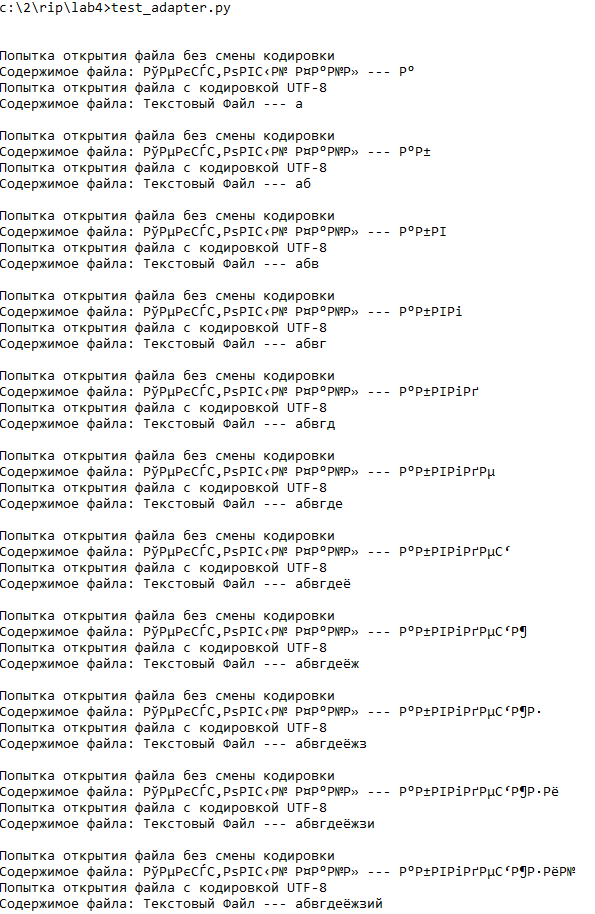
test\_stroitel.py

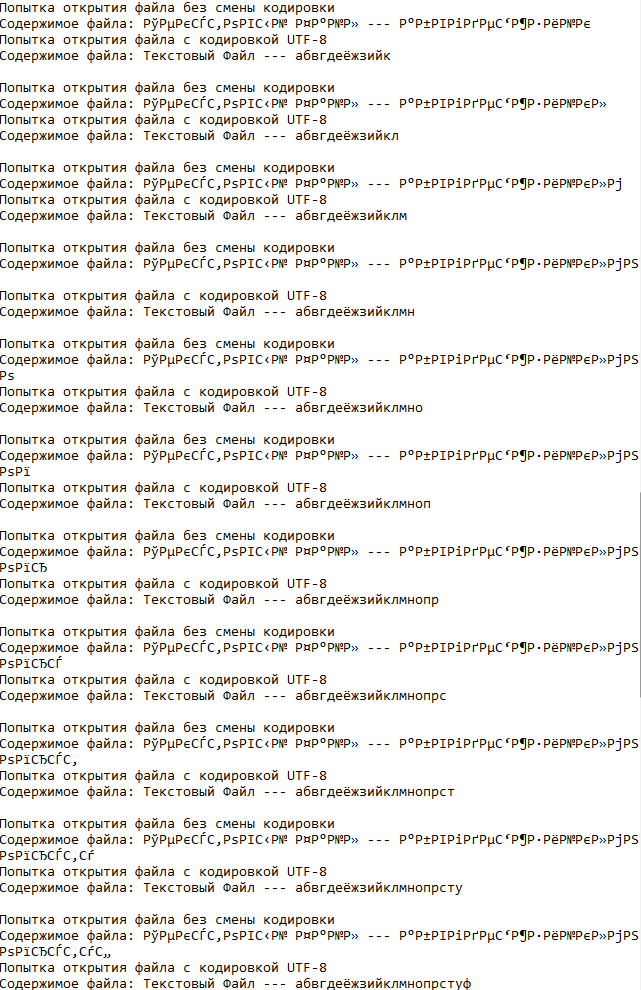


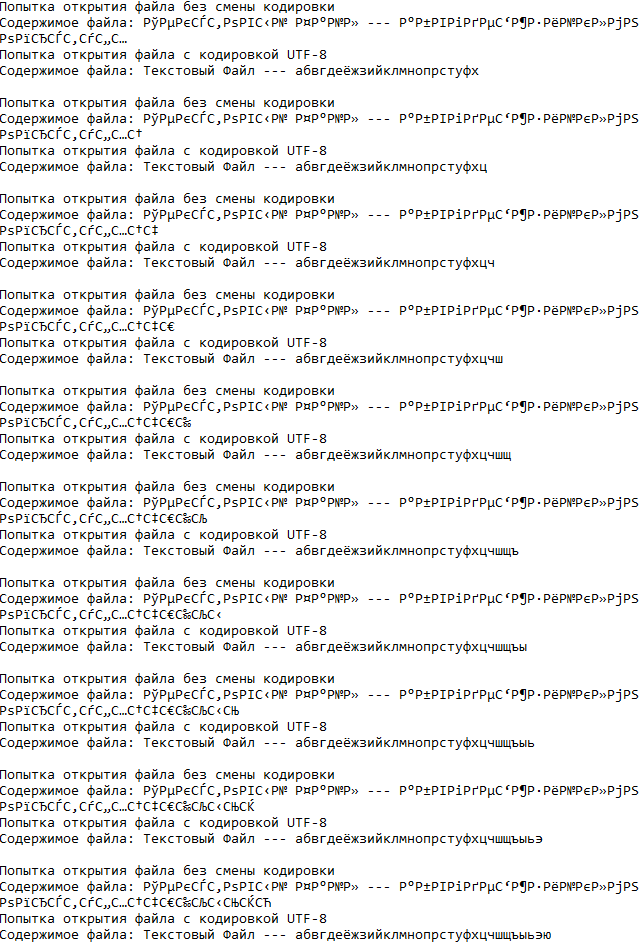
adapter.py

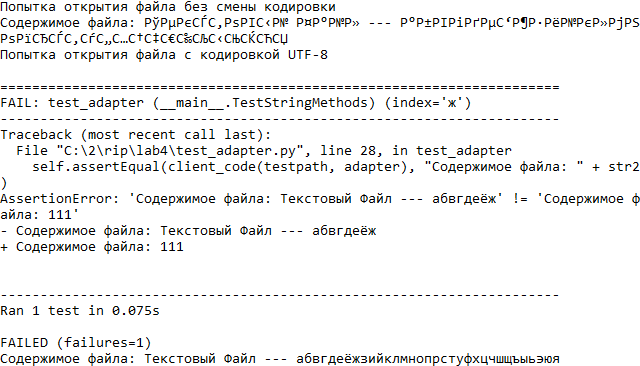


test\_adapter.py









strategy.py

