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»

ДИСЦИПЛИНА:

«БКИТ»

Лабораторная работа № 4

Студент Бабаян А.А. ИУ5Ц-52Б _____

(И.О. Фамилия) (Группа) (Подпись, дата)

Преподаватель Гапанюк Ю.Е. _____

(И.О. Фамилия) (Подпись, дата)

Описание задания

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. Вместо реализации паттерна Вы можете написать тесты для своей программы решения биквадратного уравнения. В этом случае, возможно, Вам потребуется доработать программу решения биквадратного уравнения, чтобы она была пригодна для модульного тестирования. В модульных тестах необходимо применить следующие технологии:

TDD - фреймворк.

BDD - фреймворк.

Создание Mock-объектов.

Файл **main.py:**

```
from abc import ABC, abstractmethod
from enum import Enum, auto


class Consoles_Brands(Enum):
    PlayStation = auto()
    XBOX = auto()
    Nintendo = auto()
    Steam = auto()

class Cpu_Type(Enum):
    cpu_PS = auto()
    cpu_Xbox = auto()
    cpu_Nvidia = auto()

class Size_Memory(Enum):
    S512 = auto()
    s825 = auto()
    s64 = auto()

class Gpu_Type(Enum):
    AMD_Radeon = auto()
    Intel_Celeron = auto()
    Nvidia_gpu = auto()
    Nvidia_gpu88 = auto()

class Consoles:
    def __init__(self, name):
        self.name = name
        self.brand = None
        self.cpu = None
        self.memory = None
        self.gpu = None
```

```
        self.cost = None

    def __str__(self):
        info: str = f"Название игровой консоли: {self.name} \n" \
                    f"{self.brand} \n" \
                    f"{self.cpu} \n" \
                    f"{self.memory} \n" \
                    f"{self.gpu} \n" \
                    f"Cost: {self.cost} рублей"

        return info


###################### ШАБЛОН ПРОЕКТИРОВАНИЯ - СТРОИТЕЛЬ
#########################
class Builder(ABC):

    @abstractmethod
    def add_brand(self) -> None: pass

    @abstractmethod
    def add_cpu(self) -> None: pass

    @abstractmethod
    def add_memory(self) -> None: pass

    @abstractmethod
    def add_gpu(self) -> None: pass

#########################################################################
####


class PlayStation5(Builder):

    def __init__(self):
        self.console = Consoles("PlayStation 5 / PS5")
        self.console.cost = 105000

    def add_brand(self) -> None:
        self.console.brand = Consoles_Brands.PlayStation

    def add_cpu(self) -> None:
        self.console.cpu = Cpu_Type.cpu_PS

    def add_memory(self) -> None:
        self.console.memory = Size_Memory.s825

    def add_gpu(self) -> None:
        self.console.gpu = Gpu_Type.AMD_Radeon

    def get_lap(self) -> Consoles:
        return self.console


class Xbox_series_X(Builder):

    def __init__(self):
        self.console = Consoles("Xbox series X / XSX")
        self.console.cost = 50000

    def add_brand(self) -> None:
        self.console.brand = Consoles_Brands.XBOX
```

```
    def add_cpu(self) -> None:
        self.console.cpu = Cpu_Type.cpu_Xbox

    def add_memory(self) -> None:
        self.console.memory = Size_Memory.s825

    def add_gpu(self) -> None:
        self.console.gpu = Gpu_Type.AMD_Radeon

    def get_lap(self) -> Consoles:
        return self.console


class Nintendo_Switch(Builder):

    def __init__(self):
        self.console = Consoles("Nintendo Switch / NS")
        self.console.cost = 38840

    def add_brand(self) -> None:
        self.console.brand = Consoles_Brands.Nintendo

    def add_cpu(self) -> None:
        self.console.cpu = Cpu_Type.cpu_Nvidia

    def add_memory(self) -> None:
        self.console.memory = Size_Memory.s64

    def add_gpu(self) -> None:
        self.console.gpu = Gpu_Type.Nvidia_gpu

    def get_lap(self) -> Consoles:
        return self.console


class Director:
    def __init__(self):
        self.builder = None

    def set_builder(self, builder: Builder):
        self.builder = builder

    def make_console(self):
        if not self.builder:
            raise ValueError("Builder didn't set")

        self.builder.add_brand()
        self.builder.add_cpu()
        self.builder.add_memory()
        self.builder.add_gpu()


def check_cost(name1):
    for it1 in (PlayStation5, Xbox_series_X, Nintendo_Switch):

        director1 = Director()
        builder1 = it1()

        director1.set_builder(builder1)

        director1.make_console()

        console1 = builder1.get_lap()
```

```
        if console1.name == name1:

            return console1.cost

def sum_cost(x):
    for it1 in (PlayStation5, Xbox_series_X, Nintendo_Switch):
        director1 = Director()
        builder1 = it1()
        director1.set_builder(builder1)
        director1.make_console()
        console1 = builder1.get_lap()
        x = x + console1.cost
    return x


if __name__ == "__main__":
    print("\nИгровые консоли:\n")
    director = Director()

    for it in (PlayStation5, Xbox_series_X, Nintendo_Switch):

        builder = it()
        director.set_builder(builder)

        director.make_console()
        console = builder.get_lap()

        print(console)
        print('\n/////////////////\n')

    name = "PS5"
    print(name, "Cost:", check_cost(name))

    x = 0
    print('Полная стоимость всех рассмотренных консолей = ', sum_cost(x),
'руб.')
```

Скрин:

```
C:\Users\79626\PycharmProjects\lab4\venv\Scripts\python.exe C:/Users/79626/PycharmProjects/lab4/main.py

Игровые консоли:

Название игровой консоли: PlayStation 5 / PS5
Consoles_Brands.PlayStation
Cpu_Type.cpu_PS
Size_Memory.s825
Gpu_Type.AMD_Radeon
Cost: 105000 рублей

//////////////////

Название игровой консоли: Xbox series X / XSX
Consoles_Brands.XBOX
Cpu_Type.cpu_Xbox
Size_Memory.s825
Gpu_Type.AMD_Radeon
Cost: 50000 рублей

//////////////////

Название игровой консоли: Nintendo Switch / NS
Consoles_Brands.Nintendo
Cpu_Type.cpu_Nvidia
Size_Memory.s64
Gpu_Type.Nvidia_gpu
Cost: 38840 рублей
```

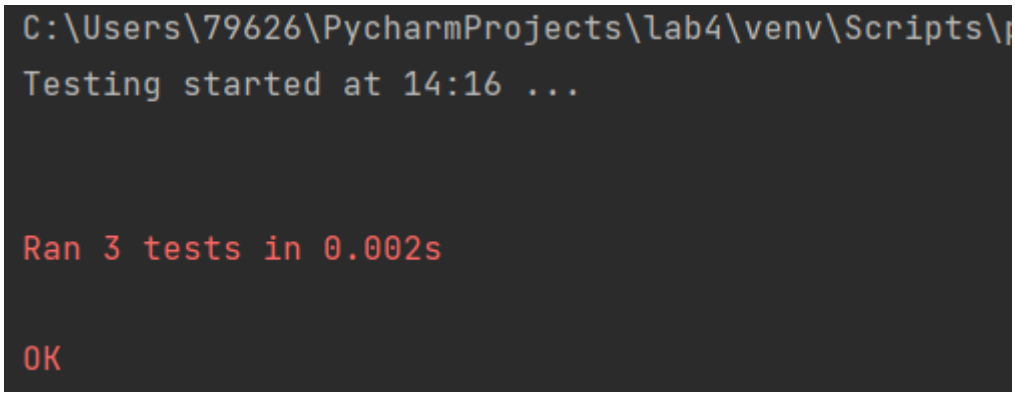
```
//////////////////

PS5 Cost: None
Полная стоимость всех рассмотренных консолей =  193840 руб.

Process finished with exit code 0
```

Папка Test**TDD.py:**

```python
import unittest
import sys, os

sys.path.append(os.getcwd())
from main import *

class TestCost(unittest.TestCase):

    def test_cost(self):
        self.assertEqual(check_cost("Xbox series X / XSX"), 50000)

    def test_cost1(self):
        self.assertEqual(check_cost("PlayStation 5 / PS5"), 105000)

    def test_cost2(self):
        self.assertEqual(check_cost("Nintendo Switch / NS"), 38840)
```

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

Скрин:



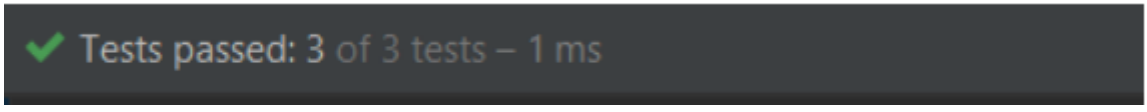
Папка **features\build.feature:**

```
Feature: Test
  Scenario: Test sum_cost
    Given I have sum = 0
    When I sum the cost
    Then I expect to get result = 193840
```

Папка **features\steps\steps.py:**

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

@given ('I have sum = {x:g}')
def step(context, x):
    context.x

@when ('I sum the cost')
def step(context):
    context.x = sum_cost(context.x)

@then('I expect to get result = {result:g}')
def step(context, result):
 assert context.x == result
```

Скрин:



Папка Test**MOCK.py:**

```
from main import *
from unittest import TestCase
from unittest.mock import patch

class TestCost(TestCase):
    @patch('main.sum_cost', return_value= 193840)
    def test_sum_cost(self, x):
        self.assertEqual(sum_cost(0), 193840)
```

Скрин:

```
Testing started at 14:38 ...
Launching unittests with arguments python



Ran 1 test in 0.002s
```