|  |  |
| --- | --- |
| 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)** |

Базовые компоненты интернет технологий

Отчет по лабораторной работе №4

Студент: Макеев В. А.

Группа: ИУ5Ц-54Б

Преподаватель: Гапанюк Ю. Е.

2021 г.

**Лабораторная работа №4**

**Задание**

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. 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.
3. В модульных тестах необходимо применить следующие технологии:
   * TDD - фреймворк.
   * BDD - фреймворк.
   * Создание Mock-объектов.

**Текст программы**

1. **builder.py**

from abc import ABC, abstractmethod

from enum import Enum, auto

class BrandType(Enum):

    MSI = auto()

    ASUS = auto()

    ACER = auto()

    HP = auto()

class CpuType(Enum):

    I5 = auto()

    I7 = auto()

    I9 = auto()

class RamType(Enum):

    RAM8 = auto()

    RAM16 = auto()

class CardType(Enum):

    GTX1650 = auto()

    GTX1650ti = auto()

    GTX1660ti = auto()

    RTX2060 = auto()

class Laptop:

    def \_\_init\_\_(self, name):

        self.name = name

        self.brand = None

        self.cpu = None

        self.ram = None

        self.card = None

        self.cost = None

    def \_\_str\_\_(self):

       info: str = f"Laptop name: {self.name} \n" \

                   f"{self.brand} \n" \

                   f"{self.cpu} \n" \

                   f"{self.ram} \n" \

                   f"{self.card} \n" \

                   f"Cost: {self.cost} rub"

       return info

class Builder(ABC):

    @abstractmethod

    def add\_brand(self) -> None: pass

    @abstractmethod

    def add\_cpu(self) -> None: pass

    @abstractmethod

    def add\_ram(self) -> None: pass

    @abstractmethod

    def add\_card(self) -> None: pass

class MSIGL62LapBuilder(Builder):

    def \_\_init\_\_(self):

        self.laptop = Laptop("MSI GL62")

        self.laptop.cost = 50000

    def add\_brand(self) -> None:

        self.laptop.brand = BrandType.MSI

    def add\_cpu(self) -> None:

        self.laptop.cpu = CpuType.I5

    def add\_ram(self) -> None:

        self.laptop.ram = RamType.RAM8

    def add\_card(self) -> None:

        self.laptop.card = CardType.GTX1650

    def get\_lap(self) -> Laptop:

        return self.laptop

class ASUSTUFLapBuilder(Builder):

    def \_\_init\_\_(self):

        self.laptop = Laptop("ASUS TUF")

        self.laptop.cost = 100000

    def add\_brand(self) -> None:

        self.laptop.brand = BrandType.ASUS

    def add\_cpu(self) -> None:

        self.laptop.cpu = CpuType.I7

    def add\_ram(self) -> None:

        self.laptop.ram = RamType.RAM16

    def add\_card(self) -> None:

        self.laptop.card = CardType.RTX2060

    def get\_lap(self) -> Laptop:

        return self.laptop

class Director:

    def \_\_init\_\_(self):

        self.builder = None

    def set\_builder(self, builder: Builder):

        self.builder = builder

    def make\_lap(self):

        if not self.builder:

            raise ValueError("Builder didn't set")

        self.builder.add\_brand()

        self.builder.add\_cpu()

        self.builder.add\_ram()

        self.builder.add\_card()

def check\_cost(name1):

    for it1 in (MSIGL62LapBuilder, ASUSTUFLapBuilder):

        director1 = Director()

        builder1 = it1()

        director1.set\_builder(builder1)

        director1.make\_lap()

        laptop1 = builder1.get\_lap()

        if laptop1.name == name1:

            return laptop1.cost

def sum\_cost(x):

    for it1 in (MSIGL62LapBuilder, ASUSTUFLapBuilder):

        director1 = Director()

        builder1 = it1()

        director1.set\_builder(builder1)

        director1.make\_lap()

        laptop1 = builder1.get\_lap()

        x = x + laptop1.cost

    return x

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

    print("Объекты:")

    director = Director()

    for it in (MSIGL62LapBuilder, ASUSTUFLapBuilder):

        builder = it()

        director.set\_builder(builder)

        director.make\_lap()

        laptop = builder.get\_lap()

        print(laptop)

        print('---------------')

    name = "ASUS TUF"

    print(name, "Cost:", check\_cost(name))

    x = 0

    print('sum = ', sum\_cost(x))

**2.** **mock.py**

from builder import \*

from unittest import TestCase

from unittest.mock import patch

class TestCost(TestCase):

    @patch('builder.sum\_cost', return\_value=150000)

    def test\_sum\_cost(self, x):

        self.assertEqual(sum\_cost(0), 150000)

**3. tdd.py**

import unittest

import sys, os

sys.path.append(os.getcwd())

from builder import \*

class TestCost(unittest.TestCase):

    def test\_cost(self):

        self.assertEqual(check\_cost("ASUS TUF"), 100000)

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

    unittest.main()

1. **steps.py**

from behave import given, when, then

from builder import \*

@given('I have sum = {x:g}')

def step(context, x):

    context.x = 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

1. **build.feature**

Feature: Test

  Scenario: Test sum\_cost

    Given  I have sum = 0

    When I sum the cost

    Then I expect to get result = 150000

**Тест программы**

