**Московский государственный технический**

**университет им. Н.Э. Баумана**

Факультет «Информатика и системы управления»

Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Базовые компоненты интернет-технологий»

Отчет по лабораторной работе №4

|  |  |  |
| --- | --- | --- |
| Выполнил: |  | Проверил: |
| студент группы ИУ5-31Б |  | преподаватель каф. ИУ5 |
| Коваленко  Алексей Викторович |  | Гапанюк Юрий Евгеньевич |
|  |  |  |
| Подпись:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Дата:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |  | Подпись:\_\_\_\_\_\_\_\_\_\_\_\_\_  Дата:\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  |  |  |
|  |  |  |

Москва, 2021 г.

**Лабораторная работа №1**

**Описание задания**

### 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.](https://refactoring.guru/ru/design-patterns/catalog) 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**

Папка patterns

Файл Bulder.py

from \_\_future\_\_ import annotations  
from abc import ABC, abstractmethod  
from typing import Any  
  
class Builder(ABC):  
  
 @property  
 @abstractmethod  
 def product(self) -> None:  
 pass  
  
 @abstractmethod  
 def produce\_hammer(self,amount: int) -> None:  
 pass  
  
 @abstractmethod  
 def produce\_screwdriver(self,amount: int) -> None:  
 pass  
  
 @abstractmethod  
 def produce\_screw(self,amount: int,type: string) -> None:  
 pass  
  
 @abstractmethod  
 def produce\_nail(self,amount: int,type: string) -> None:  
 pass  
  
 @abstractmethod  
 def produce\_drill(self,amount: int,model: string) -> None:  
 pass  
  
 @abstractmethod  
 def produce\_drill\_head(self,amount: int,type: string) -> None:  
 pass  
  
 @abstractmethod  
 def produce\_plank(self,amount: int, type: string) -> None:  
 pass  
  
  
  
class CartBuilder(Builder):  
  
 def \_\_init\_\_(self) -> None:  
 self.reset()  
  
 def reset(self) -> None:  
 self.\_product = Product1()  
  
 @property  
 def product(self) -> Product1:  
 return self.\_product  
  
 def produce\_hammer(self,amount: int) -> None:  
 self.\_product.add({'hammer':amount})  
  
 def produce\_screwdriver(self,amount: int) -> None:  
 self.\_product.add({'screwdriver':amount})  
  
 def produce\_screw(self,amount: int,type: string) -> None:  
 self.\_product.add({'screw':amount,'size':type})  
  
 def produce\_nail(self,amount: int,type: string) -> None:  
 self.\_product.add({'nail':amount,'size':type})  
  
 def produce\_drill(self,amount: int,model: string) -> None:  
 self.\_product.add({'drill':amount,'model':model})  
  
 def produce\_drill\_head(self,amount: int,type: string) -> None:  
 self.\_product.add({'drillhead':amount,'type':type})  
  
 def produce\_plank(self,amount: int, type:string) -> None:  
 self.\_product.add({'plank':amount,'material':type})  
  
  
  
  
class Product1():  
  
 def \_\_init\_\_(self) -> None:  
 self.size = 0  
 self.parts = []  
  
 def add(self, part: Any) -> None:  
 self.size+=1  
 self.parts.append(part)  
  
 def print\_info(self) -> None:  
 print(f"Размер корзины:{self.size} \nТовары корзины:")  
 for i in range(len(self.parts)):  
 print (', '.join([f'{key.capitalize()}: {value}' for key, value in self.parts[i].items()]))  
  
  
 def get\_amount(self):  
 return self.size  
  
 def get\_cart(self):  
 return 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\_hammer\_cart(self) -> None:  
 self.builder.produce\_hammer(1)  
 self.builder.produce\_nail(100,'medium')  
  
 def build\_drill\_cart(self) -> None:  
 self.builder.produce\_drill(1,'classic A1')  
 self.builder.produce\_drill\_head(2,'small')  
 self.builder.produce\_drill\_head(2,'medium')  
 self.builder.produce\_drill\_head(2,'large')  
  
 def build\_screw\_cart(self) -> None:  
 self.builder.produce\_screwdriver(2)  
 self.builder.produce\_screw(50,'small')  
 self.builder.produce\_screw(50,'medium')  
  
 def build\_starter\_kit(self) -> None:  
 self.builder.produce\_hammer(1)  
 self.builder.produce\_drill(1,'classic A1')  
 self.builder.produce\_drill\_head(1,'small')  
 self.builder.produce\_drill\_head(1,'medium')  
 self.builder.produce\_drill\_head(1,'large')  
 self.builder.produce\_screwdriver(2)  
  
  
  
def Auto\_create():  
 director=Director()  
 builder = CartBuilder()  
 director.builder=builder  
 print ("Список товаров: nail, drill, drill head, hammer, screwdriver, screw, plank.\n Такжи список готорвых наборов:/"  
 " starter kit, drill kit, hammer kit, screw kit.\n Для выхода введите 0.")  
 while (True):  
 item = str(input("Введите элемент который вы хотите добавить: "))  
 match item:  
 case "hammer":  
 amount = (int(input("Введите количество: ")))  
 builder.produce\_hammer(amount)  
 case "nail":  
 amount = (int(input("Введите количество: ")))  
 size = (str(input("Введите размер: ")))  
 builder.produce\_nail(amount,size)  
 case "screwdriver":  
 amount = (int(input("Введите количество: ")))  
 builder.produce\_screwdriver(amount)  
 case "screw":  
 amount = (int(input("Введите количество: ")))  
 size = (str(input("Введите размер: ")))  
 builder.produce\_screw(amount, size)  
 case "drill":  
 amount = (int(input("Введите количество: ")))  
 size = (str(input("Введите модель: ")))  
 builder.produce\_drill(amount, size)  
 case "drillhead":  
 amount = (int(input("Введите количество: ")))  
 size = (str(input("Введите размер: ")))  
 builder.produce\_drill\_head(amount, size)  
 case "plank":  
 amount = (int(input("Введите количество: ")))  
 size = (str(input("Введите древесину: ")))  
 builder.produce\_plank(amount, size)  
 case "starter kit":  
 director.build\_starter\_kit()  
 case "drill kit":  
 director.build\_drill\_cart()  
 case "hammer kit":  
 director.build\_hammer\_cart()  
 case "screw kit":  
 director.build\_screw\_cart()  
 case "0":  
 break  
 case \_:  
 print("Ошибка запроса, введите корректные данные!")  
  
 return builder.product  
  
def pars\_build(builder, director, name, \*args):  
 match name:  
 case "hammer":  
 amount = args[0]  
 builder.produce\_hammer(amount)  
 case "nail":  
 amount = args[0]  
 size = args[1]  
 builder.produce\_nail(amount,size)  
 case "screwdriver":  
 amount = args[0]  
 builder.produce\_screwdriver(amount)  
 case "screw":  
 amount = args[0]  
 size = args[1]  
 builder.produce\_screw(amount, size)  
 case "drill":  
 amount = args[0]  
 size = args[1]  
 builder.produce\_drill(amount, size)  
 case "drillhead":  
 amount = args[0]  
 size = args[1]  
 builder.produce\_drill\_head(amount, size)  
 case "plank":  
 amount = args[0]  
 size = args[1]  
 builder.produce\_plank(amount, size)  
 case "starter kit":  
 director.build\_starter\_kit()  
 case "drill kit":  
 director.build\_drill\_cart()  
 case "hammer kit":  
 director.build\_hammer\_cart()  
 case "screw kit":  
 director.build\_screw\_cart()  
 case \_:  
 return builder  
 return builder

Файл build tests.py

import Bulder  
  
builder1 = Bulder.CartBuilder()  
director = Bulder.Director()  
director.builder = builder1  
  
director.build\_hammer\_cart()  
result=builder1.product  
  
def test\_amount():  
  
 assert result.get\_amount() == 2  
  
def test\_cart():  
  
 assert result.get\_cart() == [{'hammer':1},{'nail':100,'size':'medium'}]

Файл mocktest.py

from Bulder import \*  
from unittest import TestCase  
from unittest.mock import patch  
  
class TestBuilder(TestCase):  
 @patch('Bulder.CartBuilder')  
 def test\_pars\_built(self, b):  
 secb = CartBuilder()  
 secb.produce\_hammer(3)  
 secb.produce\_screwdriver(2)  
 b.product.return\_value = secb.product  
 prod = Product1()  
 prod.size=2  
 prod.parts=[{'hammer':3},{'screwdriver':2}]  
 res =b.product()  
 self.assertIsNotNone(res)  
 self.assertIsInstance(res, Product1)  
 self.assertEqual(res.parts,prod.parts)

Папка Features

Файл build.feature

Feature: My first tests as Cart Builder tests.  
  
 Scenario: Build basic cart  
 Given I want to order **starter kit** and **4 oak plank**s  
 When I end my order  
 Then I expect to see cart with 7 items, which consist of 1 hammer, 1 drill, 1 small drill head, 1 medium drill head, 1 big drill head, 2 screwdrivers, 4 oak planks.

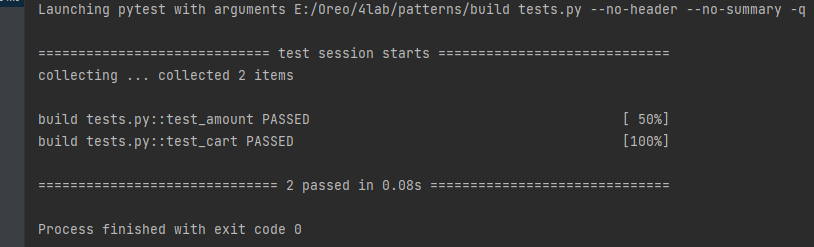
Папка Steps

Файл step.py

# -\*- coding: utf-8 -\*-  
from behave import given, when, then  
import patterns.Bulder  
import sys  
  
builder = patterns.Bulder.CartBuilder()  
director = patterns.Bulder.Director()  
director.builder = builder  
  
@given(u'I want to order {object1} and {amount} {material} {object2}s')  
def step\_impl(context, object1, amount, material, object2):  
 context.object1 = object1  
 context.amount = amount  
 context.material = material  
 context.object2 = object2  
  
@when(u'I end my order')  
def step\_impl(context):  
 context.cart=patterns.Bulder.pars\_build(builder,director,context.object1)  
 context.cart=patterns.Bulder.pars\_build(builder,director,context.object2,context.amount,context.material)  
  
  
  
@then(u'I expect to see cart with 7 items, which consist of {amount1} {object1}, {amount2} {object2}, {amount3} \  
{size1} {object3}, {amount4} {size2} {object4}, {amount5} {size3} {object5}, {amount6} {object6}s, {amount7} {material} {object7}s.')  
def step\_impl(context, amount1, object1, amount2, object2, amount3, size1, object3, amount4, size2, object4, amount5, size3, object5, amount6,\  
 object6, amount7, material, object7):  
 result = patterns.Bulder.pars\_build(builder,director,object1,amount1)  
 result = patterns.Bulder.pars\_build(builder,director,object2,amount2,'classic')  
 result = patterns.Bulder.pars\_build(builder,director,object3,amount3,size1)  
 result = patterns.Bulder.pars\_build(builder,director,object4,amount4,size2)  
 result = patterns.Bulder.pars\_build(builder,director,object5,amount5,size3)  
 result = patterns.Bulder.pars\_build(builder,director,object6,amount6)  
 result = patterns.Bulder.pars\_build(builder,director,object7,amount7,material)  
  
 assert context.cart == result

**Экранные формы с примерами выполнения программы**

TDD:



BDD:

Mock:

