**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»**

**Кафедра ИУ5 «Системы обработки информации и управления»**

Лабораторная работа №4

по дисциплине «Базовые компоненты интернет-технологий»

Выполнила:

студентка группы ИУ5-33Б

Пересыпкина М.А.

Проверил:

Канев А.И.

2021 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://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:

builder.py

from \_\_future\_\_ import annotations  
from abc import ABC, abstractmethod  
from typing import Any  
  
  
class Builder(ABC):  
  
 @property # property позволяет превратить метод класса в атрибут класса  
  
 @abstractmethod # Абстрактным называется объявленный, но не реализованный метод  
 def product(self) -> None:  
 pass  
  
 @abstractmethod # Абстрактным называется объявленный, но не реализованный метод  
 def tshirt(self) -> None: # футболка  
 pass  
  
 @abstractmethod  
 def pullover(self) -> None: # свитер  
 pass  
  
 @abstractmethod  
 def jeans(self) -> None: # джинсы  
 pass  
  
  
class Clothes\_Builder(Builder):  
  
 def \_\_init\_\_(self) -> None:  
 self.reset()  
  
 def reset(self) -> None:  
 self.\_product = Shop()  
  
 @property # property позволяет превратить метод класса в атрибут класса  
 def product(self) -> Shop:  
 product = self.\_product  
 self.reset()  
 return product  
  
 def tshirt(self) -> None:  
 self.\_product.add("футболка")  
  
 def pullover(self) -> None:  
 self.\_product.add("свитер")  
  
 def jeans(self) -> None:  
 self.\_product.add("джинсы")  
  
  
class Shop():  
  
 def \_\_init\_\_(self) -> None:  
 self.parts = []  
  
 def add(self, part: Any) -> None:  
 self.parts.append(part)  
  
 def list\_parts(self) -> None:  
 print(f"В магазине продаются: {', '.join(self.parts)}", end="")  
  
  
class Director:  
  
 def \_\_init\_\_(self) -> None:  
 self.\_builder = None  
  
 @property # property позволяет превратить метод класса в атрибут класса  
 def builder(self) -> Builder:  
 return self.\_builder  
  
 @builder.setter # применяется сеттер к методу builder, то есть делаем метод доступным для записи  
 def builder(self, builder: Builder) -> None:  
 self.\_builder = builder  
  
 def HM(self) -> None:  
 self.builder.pullover()  
 self.builder.tshirt()  
  
 def Bershka(self) -> None:  
 self.builder.jeans()  
 self.builder.pullover()  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 director = Director()  
 builder = Clothes\_Builder()  
 director.builder = builder  
  
 print("HM: ")  
 director.HM()  
 builder.product.list\_parts()  
  
 print("\n\nBershka: ")  
 director.Bershka()  
 builder.product.list\_parts()

TDD\_test.py

import unittest  
import sys, os  
from features.steps.builder import \*  
  
sys.path.append(os.getcwd())  
  
class Clothes\_Builder\_Test(unittest.TestCase):  
 director = Director()  
 builder = Clothes\_Builder()  
 director.builder = builder  
 def test\_Bershka(self):  
 print("Bershka: ")  
 self.director.Bershka()  
 self.builder.product.list\_parts()  
  
 def test\_HM(self):  
 print("\nHM: ")  
 self.director.HM()  
 self.builder.product.list\_parts()  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 unittest.main()

testing.feature

Feature: Test  
 Scenario: Test Builder  
 Given Furniture\_Builder  
 When test\_shatura\_builder return OK  
 And test\_lasurit\_builder return OK  
 Then Good job

BDD\_test.py

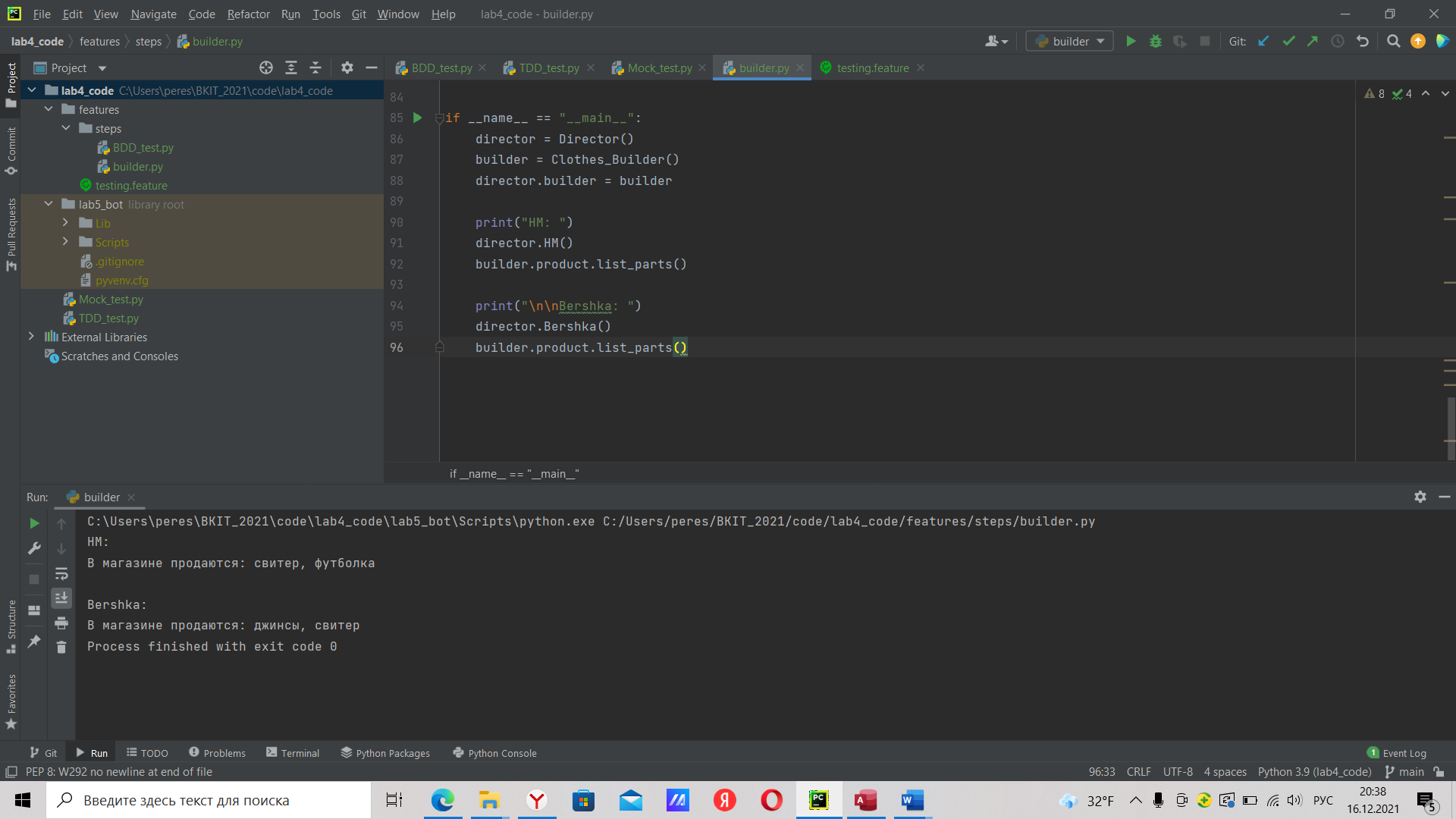
from behave import \*  
from TDD\_test import \*  
  
@given("Clothes\_Builder")  
def first\_step(context):  
 context.a = Clothes\_Builder\_Test()  
  
@when("test\_HM\_builder return OK")  
def test\_HM\_builder(context):  
 context.a.test\_HM()  
  
@when("test\_Bershka\_builder return OK")  
def test\_Bershka\_builder(context):  
 context.a.test\_Bershka()  
  
@then("Good job")  
def last\_step(context):  
 pass

Mock\_test.py

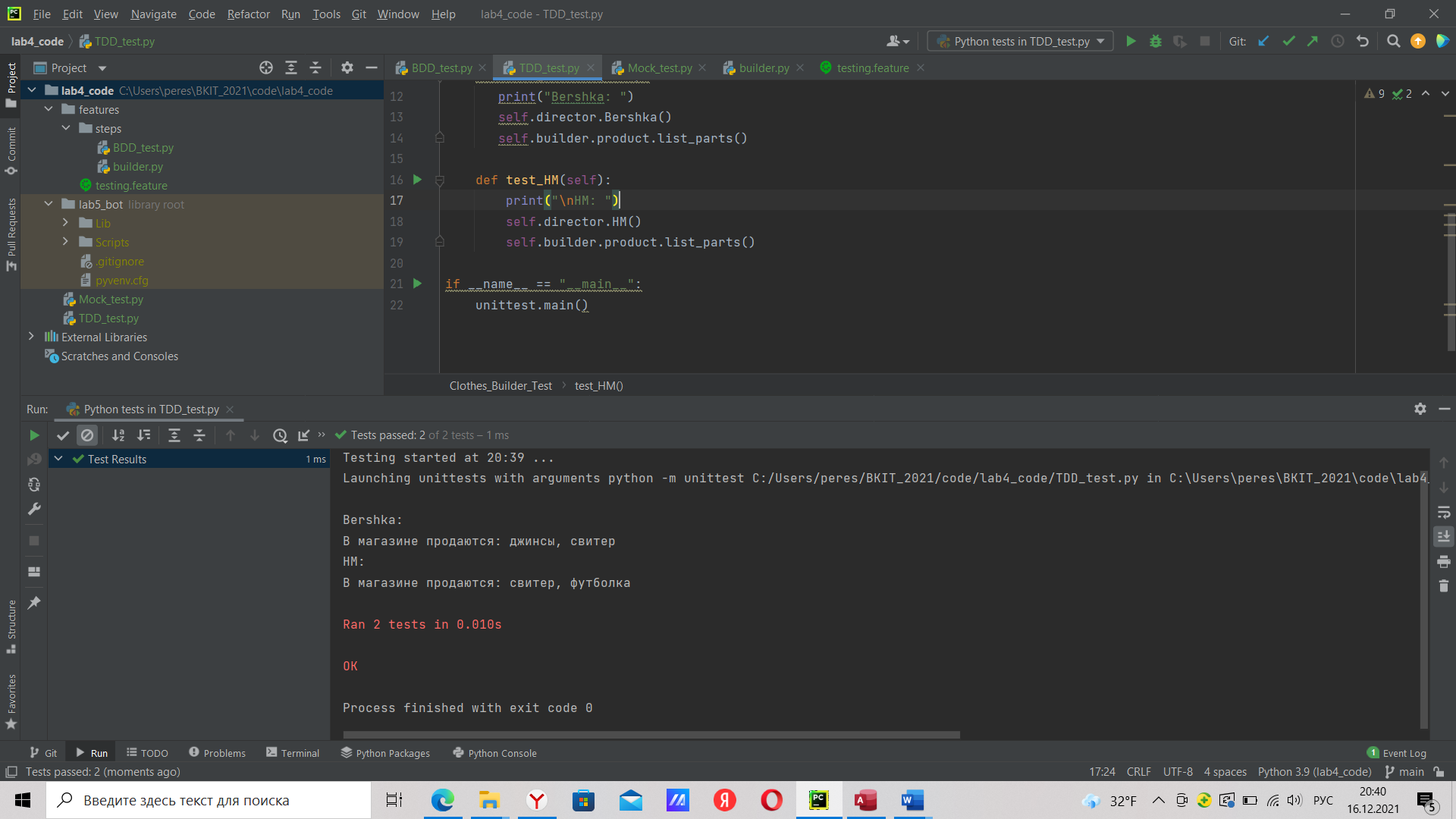
import unittest  
import sys, os  
from unittest.mock import patch  
  
from features.steps import builder  
  
sys.path.append(os.getcwd())  
  
from features.steps.builder import \*  
  
  
class Clothes\_Builder\_Test(unittest.TestCase):  
 @patch.object(builder.Clothes\_Builder(), 'pullover')  
 def test\_pullover(self, mock\_pullover):  
 mock\_pullover.return\_value = None  
 self.assertEqual(Clothes\_Builder().pullover(), None)

# Экранные формы с примерами выполнения программы:

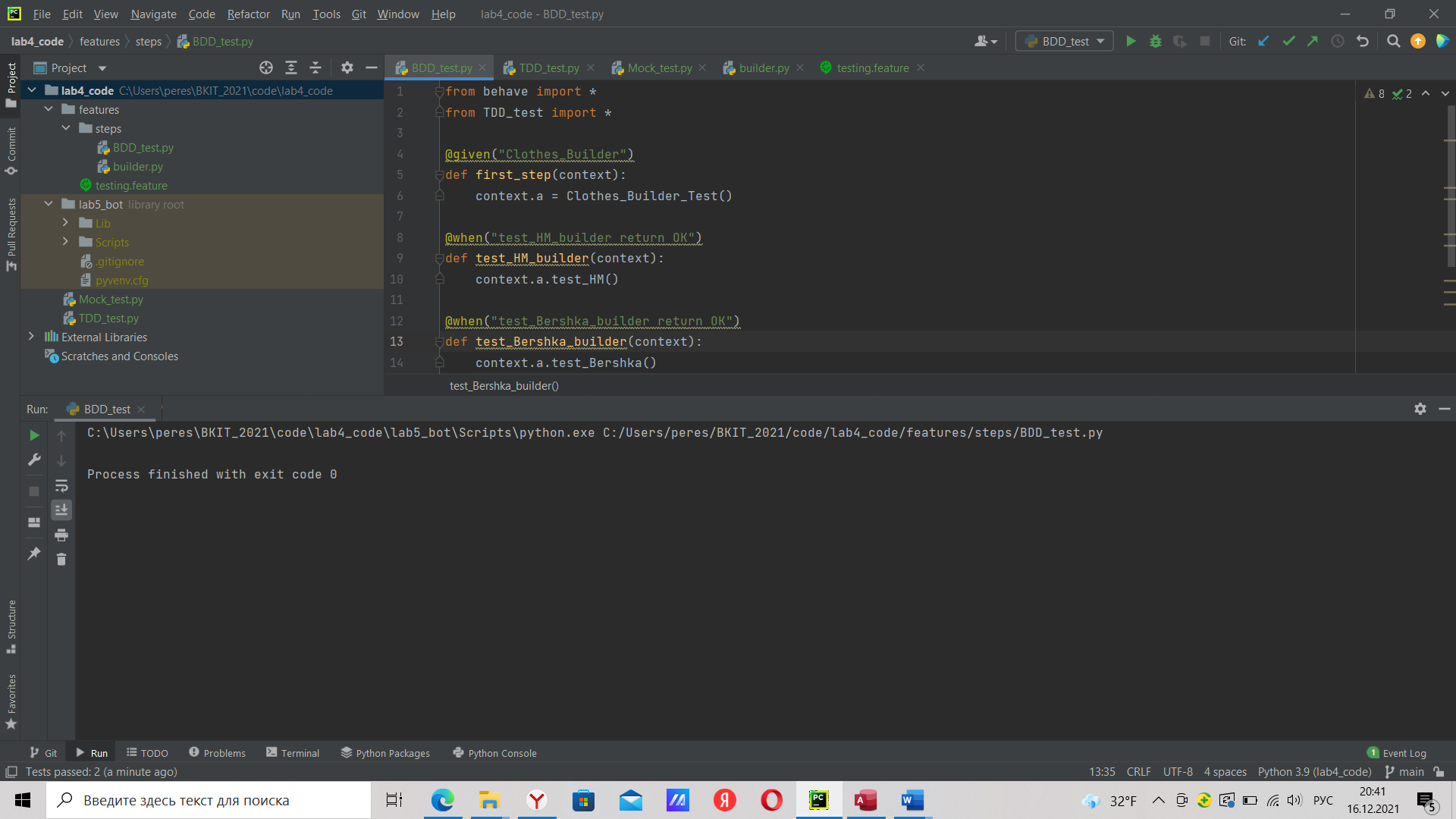
builder.py

****

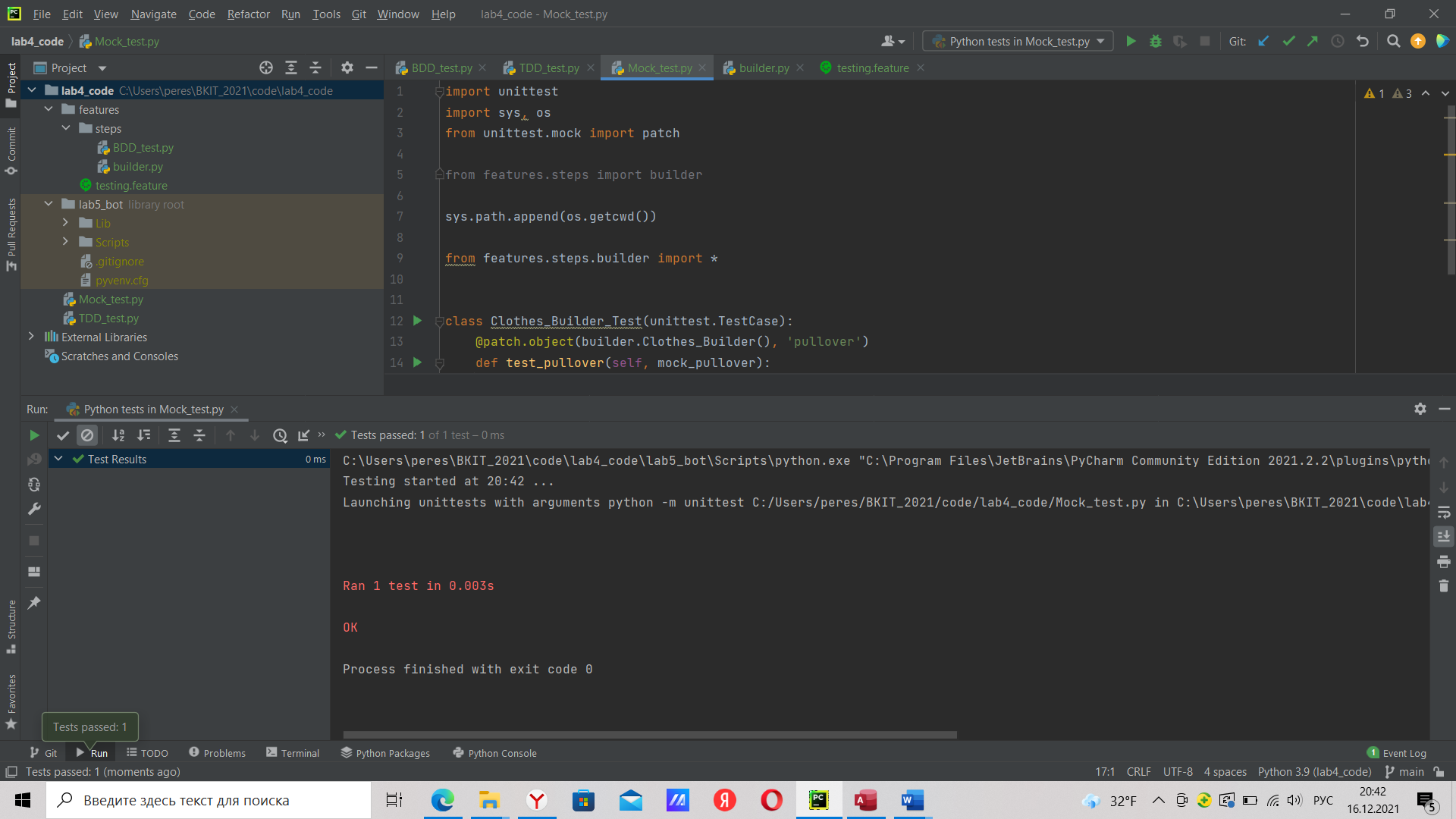
TDD\_test.py

****

BDD\_test.py

****

Mock\_test.py

****