



**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 «Системы обработки
информации и управления»**

Домашнее задание
по дисциплине «Базовые компоненты интернет технологий»

Выполнил:
студент группы ИУ5-35Б Герасимов А.Д

Задание:

1. Модифицируйте код лабораторной работы №6 таким образом, чтобы он был пригоден для модульного тестирования.
2. Используя материалы лабораторной работы №4 создайте модульные тесты с применением TDD - фреймворка (2 теста) и BDD - фреймворка (2 теста).

Текст программы

Botmain.py

```
import telebot
from telebot import types
import config
import dbworker
import unittest
import TDD

from calcfunc import Calculaion
bot = telebot.TeleBot(config.TOKEN)

class BotMain():

    @bot.message_handler(commands=['start'])
    def cmd_start(message):
        bot.send_message(message.chat.id, 'Я умею вычислять периметр и площадь фигуры!')
        dbworker.set(dbworker.make_key(message.chat.id, config.CURRENT_STATE), config.States.STATE_SELECT_SHAPE.value)
        bot.send_message(message.chat.id, 'Выберете или введите название фигуры!')
        itembtn1 = types.KeyboardButton('Прямоугольник')
        itembtn2 = types.KeyboardButton('Круг')
        itembtn3 = types.KeyboardButton('Квадрат')
        markup = types.ReplyKeyboardMarkup(row_width=3)
        markup.add(itembtn1, itembtn2, itembtn3)
        bot.send_message(message.chat.id, 'Введите название фигуры!', reply_markup=markup)

    # По команде /reset будем сбрасывать состояния, возвращаясь к началу диалога
    @bot.message_handler(commands=['reset'])
```

```

def cmd_reset(message):
    bot.send_message(message.chat.id, 'Сбрасываем результаты предыдущего
ввода.')
    dbworker.set(dbworker.make_key(message.chat.id,
config.CURRENT_STATE), config.States.STATE_SELECT_SHAPE.value)
    bot.send_message(message.chat.id, 'Выберете или введите название
фигуры!')
    itembtn1 = types.KeyboardButton('Прямоугольник')
    itembtn2 = types.KeyboardButton('Круг')
    itembtn3 = types.KeyboardButton('Квадрат')
    markup = types.ReplyKeyboardMarkup(row_width=3)
    markup.add(itembtn1, itembtn2, itembtn3)
    bot.send_message(message.chat.id, 'Введите название фигуры!',
reply_markup=markup)

# Обработка названия фигуры
@bot.message_handler(func=lambda message: dbworker.get(
    dbworker.make_key(message.chat.id, config.CURRENT_STATE)) ==
config.States.STATE_SELECT_SHAPE.value)
def select_shape(message):
    text = message.text
    print(text)
    if text == 'Круг' or text == 'круг' or text == 'circle':
        dbworker.set(dbworker.make_key(message.chat.id,
config.CURRENT_STATE),
            config.States.STATE_CIRCLE_RADIUS.value)
        bot.send_message(message.chat.id, 'Введите радиус круга!')

    elif text == 'Прямоугольник' or text == 'прямоугольник' or text ==
'Rectangle':
        dbworker.set(dbworker.make_key(message.chat.id,
config.CURRENT_STATE),
            config.States.STATE_RECTANGLE_HEIGHT.value)
        bot.send_message(message.chat.id, 'Введите высоту прямоугольника!')

    elif text == 'Квадрат' or text == 'квадрат' or text == 'square':
        dbworker.set(dbworker.make_key(message.chat.id,
config.CURRENT_STATE),
            config.States.STATE_SQUARE_SIDE.value)
        bot.send_message(message.chat.id, 'Введите размер стороны квадрата!')

@bot.message_handler(func=lambda message: dbworker.get(
    dbworker.make_key(message.chat.id, config.CURRENT_STATE)) ==
config.States.STATE_SQUARE_SIDE.value)
def square_side(message):

```

```

text = message.text
if not text.isdigit():
    # Состояние не изменяется, выводится сообщение об ошибке
    bot.send_message(message.chat.id, 'Пожалуйста введите размер стороны
    квадрата!')
    return
else:
    bot.send_message(message.chat.id, f'Вы ввели размер стороны квадрата:
    {text} ')
    # Меняем текущее состояние
    dbworker.set(dbworker.make_key(message.chat.id,
    config.CURRENT_STATE),
    config.States.STATE_CALCULATION_SQUARE.value)
    # Сохраняем первое число
    dbworker.set(dbworker.make_key(message.chat.id,
    config.States.STATE_SQUARE_SIDE.value), text)
    itembtn1 = types.KeyboardButton("Посчитать площадь.")
    itembtn2 = types.KeyboardButton("Посчитать периметр.")
    itembtn3 = types.KeyboardButton("Вычислить площадь и периметр.")
    markup = types.ReplyKeyboardMarkup(row_width=3)
    markup.add(itembtn1, itembtn2, itembtn3)
    bot.send_message(message.chat.id, 'Выберете или введите операцию!',
    reply_markup=markup)

@bot.message_handler(func=lambda message: dbworker.get(
    dbworker.make_key(message.chat.id, config.CURRENT_STATE)) ==
    config.States.STATE_CALCULATION_SQUARE.value)
def square_calculation(message):
    operation = message.text
    side = dbworker.get(dbworker.make_key(message.chat.id,
    config.States.STATE_SQUARE_SIDE.value))
    float_side = float(side)
    calculate_area = 0
    calculate_perimeter = 0
    multiplication = "*"
    markup = types.ReplyKeyboardRemove(selective=False)
    if operation == "Посчитать площадь.":

        bot.send_message(message.chat.id,
            f'Площадь квадрата равна:
            {float_side} {multiplication} {float_side} = {str(Calculaion.calculation_square_area(
            float_side))}',
            reply_markup=markup)
    elif operation == "Посчитать периметр.":

```

```

        bot.send_message(message.chat.id,
                          f'Периметр квадрата равен:
({float_side}+{float_side})*2={str(Calculaion.calculation_square_perimetr(float_
side))}',
                          reply_markup=markup)
    elif operation == "Вычислить площадь и периметр.":

        bot.send_message(message.chat.id,
                          f'Площадь квадрата равна:
{float_side}{multiplication}{float_side}={str(Calculaion.calculation_square_area(
float_side))}',
                          reply_markup=markup)
        calculate_perimeter = (float_side + float_side) * 2
        bot.send_message(message.chat.id,
                          f'Периметр квадрата равен:
({float_side}+{float_side})*2={str(Calculaion.calculation_square_perimetr(float_
side))}',
                          reply_markup=markup)
        bot.send_message(message.chat.id, 'Введите /reset !', reply_markup=markup)

@bot.message_handler(func=lambda message: dbworker.get(
    dbworker.make_key(message.chat.id, config.CURRENT_STATE)) ==
config.States.STATE_RECTANGLE_HEIGHT.value)
def rectangle_heiht(message):
    text = message.text
    if not text.isdigit():
        # Состояние не изменяется, выводится сообщение об ошибке
        bot.send_message(message.chat.id, 'Пожалуйста введите высоту
прямоугольника!')
        return
    else:
        bot.send_message(message.chat.id, f'Вы ввели высоту прямоугольника:
{text} ')
        # Меняем текущее состояние
        dbworker.set(dbworker.make_key(message.chat.id,
config.CURRENT_STATE),
                      config.States.STATE_RECTANGLE_WIGHT.value)
        # Сохраняем первое число
        dbworker.set(dbworker.make_key(message.chat.id,
config.States.STATE_RECTANGLE_HEIGHT.value), text)
        bot.send_message(message.chat.id, 'Введите ширину прямоугольника!')

@bot.message_handler(func=lambda message: dbworker.get(
    dbworker.make_key(message.chat.id, config.CURRENT_STATE)) ==

```

```

config.States.STATE_RECTANGLE_WIGHT.value)
    def rectangle_wight(message):
        text = message.text
        if not text.isdigit():
            # Состояние не изменяется, выводится сообщение об ошибке
            bot.send_message(message.chat.id, 'Пожалуйста введите ширину
прямоугольника!')
            return
        else:
            bot.send_message(message.chat.id, f'Вы ввели ширину прямоугольника:
{text} ')
            # Меняем текущее состояние
            dbworker.set(dbworker.make_key(message.chat.id,
config.CURRENT_STATE),
                        config.States.STATE_CALCULATING_RECTANGLE.value)
            # Сохраняем первое число
            dbworker.set(dbworker.make_key(message.chat.id,
config.States.STATE_RECTANGLE_WIGHT.value), text)
            itembtn1 = types.KeyboardButton("Посчитать площадь.")
            itembtn2 = types.KeyboardButton("Посчитать периметр.")
            itembtn3 = types.KeyboardButton("Вычислить площадь и периметр.")
            markup = types.ReplyKeyboardMarkup(row_width=3)
            markup.add(itembtn1, itembtn2, itembtn3)
            bot.send_message(message.chat.id, 'Выберете или введите операцию!',
reply_markup=markup)

    @bot.message_handler(func=lambda message: dbworker.get(
        dbworker.make_key(message.chat.id, config.CURRENT_STATE)) ==
config.States.STATE_CALCULATING_RECTANGLE.value)
        def rectangle_calculation(message):
            operation = message.text
            heiht = dbworker.get(dbworker.make_key(message.chat.id,
config.States.STATE_RECTANGLE_HEIGHT.value))
            wight = dbworker.get(dbworker.make_key(message.chat.id,
config.States.STATE_RECTANGLE_WIGHT.value))
            float_heiht = float(heiht)
            float_wight = float(wight)
            calculate_area = 0
            calculate_perimeter = 0
            multiplication = "*"
            markup = types.ReplyKeyboardRemove(selective=False)
            if operation == "Посчитать площадь.":
                bot.send_message(message.chat.id,
                                f'Площадь прямогольника равна:
{float_heiht} {multiplication} {float_wight}={str(Calculaion.calculate_rectangle_ar

```

```

ea(float_heiht, float_wight))}',
        reply_markup=markup)
    elif operation == "Посчитать периметр.":
        bot.send_message(message.chat.id,
            f'Периметр прямоугольника равен:
({float_heiht}+{float_wight})*2={str(Calculaion.calculation_rectangle_perimetr(f
loat_heiht, float_wight))}',
            reply_markup=markup)
    elif operation == "Вычислить площадь и периметр.":
        bot.send_message(message.chat.id,
            f'Площадь прямоугольника равна:
{float_heiht}{multiplication}{float_wight}={str(Calculaion.calculate_rectangle_ar
ea(float_heiht, float_wight))}',
            reply_markup=markup)
        bot.send_message(message.chat.id,
            f'Периметр прямоугольника равен:
({float_heiht}+{float_wight})*2={str(Calculaion.calculation_rectangle_perimetr(f
loat_heiht, float_wight))}',
            reply_markup=markup)
        dbworker.set(dbworker.make_key(message.chat.id,
config.CURRENT_STATE), config.States.STATE_SELECT_SHAPE.value)
        # Выводим сообщение
        bot.send_message(message.chat.id, 'Введите /reset !')

@bot.message_handler(func=lambda message: dbworker.get(
    dbworker.make_key(message.chat.id, config.CURRENT_STATE)) ==
config.States.STATE_CIRCLE_RADIUS.value)
def circle_radius(message):
    text = message.text
    if not text.isdigit():
        # Состояние не изменяется, выводится сообщение об ошибке
        bot.send_message(message.chat.id, 'Пожалуйста введите радиус круга!')
        return
    else:
        bot.send_message(message.chat.id, f'Вы ввели радиус круга: {text} ')
        # Меняем текущее состояние
        dbworker.set(dbworker.make_key(message.chat.id,
config.CURRENT_STATE),
            config.States.STATE_CALCULATING_CIRCLE.value)
        # Сохраняем первое число
        dbworker.set(dbworker.make_key(message.chat.id,
config.States.STATE_CIRCLE_RADIUS.value), text)
        itembtn1 = types.KeyboardButton("Посчитать площадь.")
        itembtn2 = types.KeyboardButton("Посчитать периметр.")
        itembtn3 = types.KeyboardButton("Вычислить площадь и периметр.")

```



```

markup = types.ReplyKeyboardMarkup(row_width=3)
markup.add(itembtn1, itembtn2, itembtn3)
bot.send_message(message.chat.id, 'Выберете или введите операцию!',
reply_markup=markup)

@bot.message_handler(func=lambda message: dbworker.get(
    dbworker.make_key(message.chat.id, config.CURRENT_STATE)) ==
config.States.STATE_CALCULATING_CIRCLE.value)
def calculating_circle(message):
    operation = message.text

    radius = dbworker.get(dbworker.make_key(message.chat.id,
config.States.STATE_CIRCLE_RADIUS.value))
    float_radius = float(radius)
    calculate_area = 0
    calculate_perimeter = 0
    multiplication = "*"
    markup = types.ReplyKeyboardRemove(selective=False)
    if operation == "Посчитать площадь.":
        bot.send_message(message.chat.id,
            f'Площадь круга равна:
{3.14}{multiplication}{float_radius}{multiplication}{float_radius}={str(Calculai
on.calculate_circle_area(float_radius))}',
            reply_markup=markup)
    elif operation == "Посчитать периметр.":
        bot.send_message(message.chat.id,
            f'Периметр круга равен:
{2}{multiplication}{3.14}{multiplication}{float_radius}={str(Calculaion.calculat
e_circle_perimetr(float_radius))}',
            reply_markup=markup)
    elif operation == "Вычислить площадь и периметр.":
        bot.send_message(message.chat.id,
            f'Площадь круга равна:
{3.14}{multiplication}{float_radius}{multiplication}{float_radius}={str(Calculai
on.calculate_circle_area(float_radius))}',
            reply_markup=markup)
        bot.send_message(message.chat.id,
            f'Периметр круга равен:
{2}{multiplication}{3.14}{multiplication}{float_radius}={str(Calculaion.calculat
e_circle_perimetr(float_radius))}',
            reply_markup=markup)

    dbworker.set(dbworker.make_key(message.chat.id,
config.CURRENT_STATE), config.States.STATE_SELECT_SHAPE.value)
    # Выводим сообщение

```



```
bot.send_message(message.chat.id, 'Введите /reset !', reply_markup=markup)

# class TestShapeFunctions(unittest.TestCase):

if __name__ == '__main__':
    bot.infinity_polling()
```

calcfunc.py

```
class Calculaion():
    def calculate_circle_area(float_radius):
        calculate_area = 0
        calculate_area = 3.14 * float_radius * float_radius
        return calculate_area

    def calculate_circle_perimetr(float_radius):
        calculate_perimeter = 0
        calculate_perimeter = 2 * 3.14 * float_radius
        return calculate_perimeter

    def calculate_rectangle_area(float_heiht, float_wight):
        calculate_area = 0
        calculate_area = float_heiht * float_wight
        return calculate_area

    def calculation_rectangle_perimetr(float_heiht, float_wight):
        calculate_perimeter = 0
        calculate_perimeter = (float_heiht + float_wight) * 2
        return calculate_perimeter

    def calculation_square_area(float_side):
        calculate_area = 0
        calculate_area = float_side * float_side
        return calculate_area

    def calculation_square_perimetr(float_side):
        calculate_perimeter = 0
        calculate_perimeter = (float_side + float_side) * 2
        return calculate_perimeter
```

config.py

```
from enum import Enum

# Токент бота
TOKEN = "PrivateToken"

# Файл базы данных Vedis
db_file = "db.vdb"

# Ключ записи в БД для текущего состояния
CURRENT_STATE = "CURRENT_STATE"

# Состояния автомата
class States(Enum):
    STATE_START = "STATE_START" # Начало нового диалога
    STATE_SELECT_SHAPE = "STATE_SELECT_SHAPE"
    STATE_RECTANGLE_HEIGHT = "STATE_RECTANGLE_HEIGHT"
    STATE_RECTANGLE_WIGHT = "STATE_RECTANGLE_WIGHT"
    STATE_CIRCLE_RADIUS = "STATE_CIRCLE_RADIUS"
    STATE_CALCULATING_CIRCLE = "STATE_CALCULATING_CIRCLE"
    STATE_SQUARE_SIDE = "STATE_SQUARE_SIDE"
    STATE_CALCULATING_RECTANGLE = "STATE_CALCULATING_RECTANGLE"
    STATE_CALCULATION_SQUARE = "STATE_CALCULATION_SQUARE"
```

dbworker.py

```
from vedis import Vedis
import config

# Чтение значения
def get(key):
    with Vedis(config.db_file) as db:
        try:
            return db[key].decode()
        except KeyError:
            # в случае ошибки значение по умолчанию - начало диалога
            return config.States.S_START.value

# Запись значения
def set(key, value):
    with Vedis(config.db_file) as db:
        try:
```

```

        db[key] = value
        return True
    except:
        # тут желательно как-то обработать ситуацию
        return False

# Создание ключа для записи и чтения
def make_key(chatid, keyid):
    res = str(chatid) + '__' + str(keyid)
    return res

```

TDD.py

```

from botmain import Calculaion
import unittest
classic = Calculaion
class TestShapeFunctions(unittest.TestCase):
    def test_calculate_area_of_rectangle(self):

        self.assertEqual(classic.calculate_rectangle_area(2,2), 4, "Should be 4")

    def test_calculate_perimeter_of_rectangle(self):

        self.assertEqual(classic.calculation_rectangle_perimetr(2,2), 8, "Should be 8")

    def test_calculate_area_of_Square(self):

        self.assertEqual(classic.calculation_square_area(2), 4, "Should be 4")

    def test_calculate_perimeter_of_Square(self):

        self.assertEqual(classic.calculation_square_perimetr(2), 8, "Should be 8")

    def test_calculate_area_of_circle(self):

        self.assertEqual(classic.calculate_circle_area(2), 12.56, "Should be 12.56")

    def test_calculate_perimeter_of_circle(self):

        self.assertEqual(classic.calculate_circle_perimetr(2), 12.56, "Should be
12.56")

suite = unittest.TestLoader().loadTestsFromTestCase(TestShapeFunctions)
unittest.TextTestRunner(verbosity=2).run(suite)

```

steps.py

```
from behave import given, when, then, step
import telebot
from telebot import types
from vedis import Vedis
from calcfunc import Calculaion

@given("Я хочу получить площадь прямоугольника")
def initial_impl(context):
    pass

@when("Стороны прямоугольника равны {float_heiht:d} {float_wight:d}")
def area_impl(context, float_heiht, float_wight):
    context.result = Calculaion.calculate_rectangle_area(float_heiht, float_wight)
    pass

@then("Получаю площадь, равную {result:d}")
def result_impl(context, result):
    assert context.result == result
    pass

# # Square

@given("Я хочу получить площадь квадрата")
def initial_impl(context):
    pass

@when("Сторона квадрата равна {float_side:d}")
def area_impl(context, float_side):
    context.result = Calculaion.calculation_square_area(float_side)
    pass

@then("Получаю площадь, равную {result:d}")
def result_impl(context, result):
    assert context.result == result
    pass
```

rectangle.feature

Feature: Rectangle

Scenario: Arguments for given, when, then

Given Я хочу получить площадь прямоугольника

When Стороны прямоугольника равны **2 3**

Then Получаю площадь, равную **6**

square.feature

Feature: square

Scenario: Arguments for given, when, then

Given Я хочу получить площадь квадрата

When Сторона квадрата равна **2**

Then Получаю площадь, равную **4**

Результаты тестирования:

TDD:

```
Run: botmain (1) ×
D:\Gerandden\FinalyBot\Scripts\python.exe D:/Gerandden/FinalyBot/botmain.py
test_calculate_area_of_Square (TDD.TestShapeFunctions) ... ok
test_calculate_area_of_circle (TDD.TestShapeFunctions) ... ok
test_calculate_area_of_rectangle (TDD.TestShapeFunctions) ... ok
test_calculate_perimeter_of_Square (TDD.TestShapeFunctions) ... ok
test_calculate_perimeter_of_circle (TDD.TestShapeFunctions) ... ok
test_calculate_perimeter_of_rectangle (TDD.TestShapeFunctions) ... ok

-----
Ran 6 tests in 0.000s

OK
```

BDD:

```
PS D:\Gerandden\FinalyBot> behave
Feature: Rectangle # features/rectangle.feature:1

  Scenario: Arguments for given, when, then # features/rectangle.feature:2
    Given Я хочу получить площадь прямоугольника # features/steps/steps.py:8
    When Стороны прямоугольника равны 2 3 # features/steps/steps.py:13
    Then Получаю площадь, равную 6 # features/steps/steps.py:19

Feature: square # features/square.feature:1

  Scenario: Arguments for given, when, then # features/square.feature:2
    Given Я хочу получить площадь квадрата # features/steps/steps.py:27
    When Сторона квадрата равна 2 # features/steps/steps.py:32
    Then Получаю площадь, равную 4 # features/steps/steps.py:19

2 features passed, 0 failed, 0 skipped
2 scenarios passed, 0 failed, 0 skipped
6 steps passed, 0 failed, 0 skipped, 0 undefined
Took 0m0.002s
```