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 домашнему заданию.

Выполнил:
студент группы ИУ5-31Б
Рысьева Е.А.

Подпись:_____

Дата:_____

Проверил:
преподаватель каф. ИУ5
Канев Антон Игоревич

Подпись:_____

Дата:_____

Москва, 2021 г.

Домашнее задание
Общее описание задания

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

- **Main.py**

```
import telebot
from telebot import types

TOKEN = '***'
bot = telebot.TeleBot(TOKEN)
global config
config = ['Первое число', 'Второе число', 'Действие', 'Посчитать']
global cases
cases = ['first', 'second', 'action', 'res']
global call
call = ''
global actions
actions = ['plus', 'minus', 'multi']


def sumoutput(config):
    config[3] = int(config[0]) + int(config[1])
    return f'{int(config[0])} + {int(config[1])} = {config[3]}'


def minusoutput(config):
    config[3] = int(config[0]) - int(config[1])
    return f'{int(config[0])} - {int(config[1])} = {config[3]}'


def multioutput(config):
    config[3] = int(config[0]) * int(config[1])
    return f'{int(config[0])} * {int(config[1])} = {config[3]}'


@bot.message_handler(commands='start')
def start(message):
    msg = 'Добро пожаловать.'
    markup = types.InlineKeyboardMarkup()
    btn = types.InlineKeyboardButton('Начать работу',
callback_data='work')
    markup.add(btn)
    bot.send_message(message.chat.id, msg, reply_markup=markup)


@bot.callback_query_handler(lambda message: message.data == 'work')
def work(message):
    msg = 'Введите данные'
    markup = types.InlineKeyboardMarkup(row_width=1)
    for i in range(4):
```

```
        btn = types.InlineKeyboardButton(f'{config[i]}',
callback_data=f'{cases[i]}')
        markup.add(btn)
    btn = types.InlineKeyboardButton('Сбросить', callback_data='reset')
    markup.add(btn)
    bot.send_message(message.message.chat.id, msg, reply_markup=markup)


@bot.callback_query_handler(lambda message: message.data == cases[0] or
message.data == cases[1])
def nums(message):
    global call
    call = cases[cases.index(message.data)]
    msg = 'Введите число'
    bot.send_message(message.message.chat.id, msg)


@bot.callback_query_handler(lambda message: message.data == cases[2])
def action(message):
    msg = 'Выберите действие'
    markup = types.InlineKeyboardMarkup(row_width=2)
    btn = types.InlineKeyboardButton('+', callback_data='plus')
    btn1 = types.InlineKeyboardButton('-', callback_data='minus')
    btn2 = types.InlineKeyboardButton('*', callback_data='multi')
    markup.add(btn, btn1, btn2)
    bot.send_message(message.message.chat.id, msg, reply_markup=markup)


@bot.callback_query_handler(lambda message: message.data in actions)
def act(message):
    if message.data == actions[0]:
        config[2] = actions[0]
    elif message.data == actions[1]:
        config[2] = actions[1]
    else:
        config[2] = actions[2]
    markup = types.InlineKeyboardMarkup(row_width=1)
    msg = 'Введите данные'
    for i in range(4):
        if not config[i].isdigit() and not config[i] in actions:
                btn = types.InlineKeyboardButton(f'{config[i]}',
callback_data=f'{cases[i]}')
                markup.add(btn)
    btn = types.InlineKeyboardButton('Сбросить', callback_data='reset')
    markup.add(btn)
    bot.send_message(message.message.chat.id, msg, reply_markup=markup)


@bot.callback_query_handler(lambda message: message.data == cases[3])
def res(message):
    if config[0].isdigit() and config[1].isdigit() and (config[2] in
actions):
        if config[2] == 'plus':
            msg = sumoutput(config)
        elif config[2] == 'minus':
            msg = minusoutput(config)
        else:
            msg = multioutput(config)
        markup = types.InlineKeyboardMarkup()
        btn = types.InlineKeyboardButton('Сбросить',
callback_data='reset')
        markup.add(btn)
        bot.send_message(message.message.chat.id, msg,
```

```
reply_markup=markup)
    else:
        msg = 'Недостаточно данных'
        markup = types.InlineKeyboardMarkup()
        for i in range(4):
            if not config[i].isdigit() and not config[i] in actions:
                btn = types.InlineKeyboardButton(f'{config[i]}',
callback_data=f'{cases[i]}')
                markup.add(btn)
        btn = types.InlineKeyboardButton('Сбросить',
callback_data='reset')
        markup.add(btn)
        bot.send_message(message.message.chat.id, msg,
reply_markup=markup)


@bot.callback_query_handler(lambda message: message.data == 'reset')
def reset(message):
    msg = 'Данные сброшены.'
    global config
    config = ['Первое число', 'Второе число', 'Действие', 'Посчитать']
    markup = types.InlineKeyboardMarkup()
    btn = types.InlineKeyboardButton('Продолжить', callback_data='work')
    markup.add(btn)
    bot.send_message(message.message.chat.id, msg, reply_markup=markup)


@bot.message_handler(content_types='text')
def text(message):
    if (call == cases[0] or call == cases[1]) and message.text.isdigit():
        if call == cases[0]:
            config[0] = message.text
            bot.send_message(message.chat.id, f'Вы ввели первое число
{int(message.text)}')
        elif call == cases[1]:
            config[1] = message.text
            bot.send_message(message.chat.id, f'Вы ввели второе число
{int(message.text)}')
        markup = types.InlineKeyboardMarkup(row_width=1)
        msg = 'Введите данные'
        for i in range(4):
            if not config[i].isdigit() and not config[i] in actions:
                btn = types.InlineKeyboardButton(f'{config[i]}',
callback_data=f'{cases[i]}')
                markup.add(btn)
        btn = types.InlineKeyboardButton('Сбросить',
callback_data='reset')
        markup.add(btn)
        bot.send_message(message.chat.id, msg, reply_markup=markup)
    if (call == cases[0] or call == cases[1]) and not
message.text.isdigit():
        msg = 'Ошибка, введите число'
        bot.send_message(message.chat.id, msg)
```

- # Tests.py

```
from main import sumoutput, multioutput
import unittest


class Tests(unittest.TestCase):

    def test_sum(self):
        msg = sumoutput(['4', '2', 'plus', 'Посчитать'])
```

```
            self.assertEqual('4 + 2 = 6', msg)

        def test_multi(self):
            msg = multioutput(['3', '4', 'multi', 'Посчитать'])
            self.assertEqual('3 * 4 = 12', msg)
```

- **Bddplus.feature**

```
Feature: plus
  Scenario: plus 9 and 4
    Given I have context for summing: ['9', '4', 'plus', 'Посчитать']
    When I call sumoutput
    Then I expect to get message with summing result: '9 + 4 = 13'
```

- **Bddmulti.feature**

```
Feature: multi
  Scenario: multi 3 and 4
    Given I have context for multi: ['3', '4', 'multi', 'Посчитать']
    When I call multioutput
    Then I expect to get message with multi result: '3 * 4 = 12'
```

- **Stepsmulti.py**

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


@given(u'I have context for multi: [\'{first}\', \'{second}\',
\'{action}\', \'{result}\']')
def step_multi(context, first: string, second: string, action: string,
result: string):
    context.first = first
    context.second = second
    context.action = action
    context.result = result


@when(u'I call multioutput')
def step_multi(context):
    context.msg = multioutput([context.first, context.second,
context.action, context.result])


@then(u'I expect to get message with multi result: \'{msg}\'')
def step_multi(context, msg: string):
    assert context.msg == msg
```

- **Stepsplus.py**

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


@given(u'I have context for summing: [\'{first}\', \'{second}\',
\'{action}\', \'{result}\']')
def step_plus(context, first: string, second: string, action: string,
result: string):
    context.first = first
    context.second = second
    context.action = action
    context.result = result
```

```
@when(u'I call sumoutput')
def step_plus(context):
    context.msg = sumoutput([context.first, context.second,
context.action, context.result])


@then(u'I expect to get message with summing result: \'{msg}\'')
def step_plus(context, msg: string):
    assert context.msg == msg
```

Экранные формы.