Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Департамент цифровых, робототехнических
систем и электроники института перспективной инженерии

**ОТЧЕТ**
**ПО ЛАБОРАТОРНОЙ РАБОТЕ №2**
**дисциплины «Объектно-ориентированное программирование»**

Выполнил:
Плугатырев Владислав Алексеевич
3 курс, группа ПИЖ-б-о-22-1,
09.03.04 «Программная инженерия»,
направленность (профиль) «Разработка и
сопровождение программного
обеспечения», очная форма обучения

_____
(подпись)

Доцент кафедры инфокоммуникаций:
Воронкин Роман Александрович

_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2024 г.

Тема: Элементы объектно-ориентированного программирования в языке Python.

Цель работы: приобретение навыков по работе с классами и объектами при написании программ с помощью языка программирования Python версии 3.x.

## Ход работы

1. Создал репозиторий.

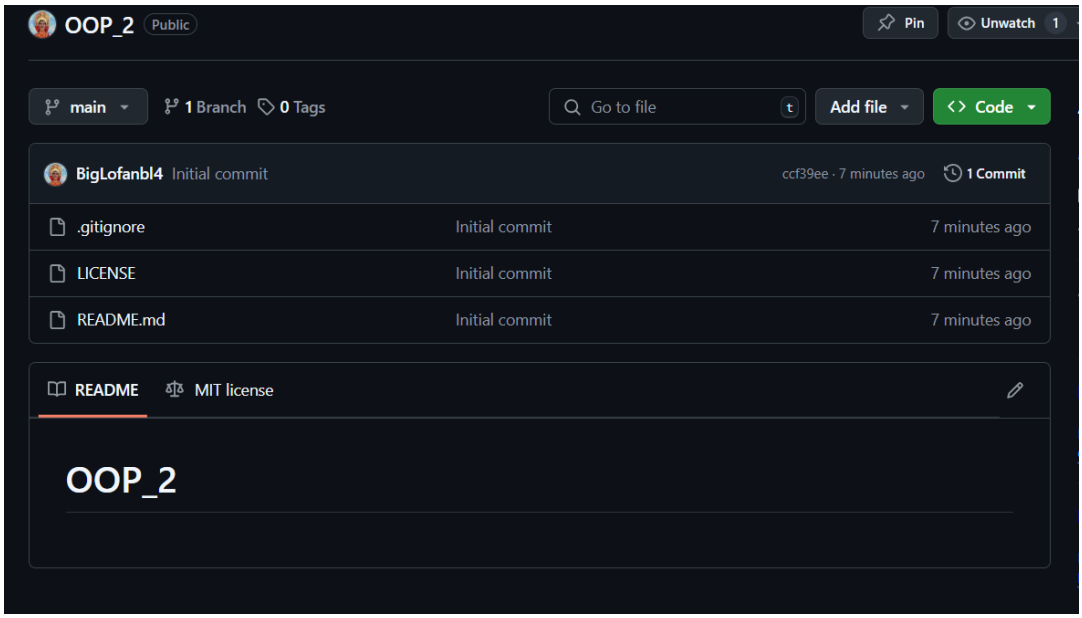


Рисунок 1.1 – Созданный репозиторий

2. Выполнение примеров из лабораторной работы.

```
if __name__ == "__main__":
    r1 = Rational(3, 4)
    r1.display()
    r2 = Rational()
    r2.read("Введите обыкновенную дробь: ")
    r2.display()
    r3 = r2.add(r1)
    r3.display()
    r4 = r2.sub(r1)
    r4.display()
    r5 = r2.mul(r1)
    r5.display()
    r6 = r2.div(r1)
    r6.display()
```

Рисунок 1.2 – Код примера

```
3/4
Введите обыкновенную дробь: 1/2
1/2
5/4
1/4
3/8
2/3
```

Рисунок 1.3 – Результат выполнения примера

3. Выполнение первого задания: поле first — дробное положительное число, катет прямоугольного треугольника; поле second — дробное положительное число, катет прямоугольного треугольника. Реализовать метод hypotenuse() — вычисление гипотенузы.

```python
import math


class RightTriangle:
    def __init__(self, first, second):
        if not isinstance(first, (int, float)) or not isinstance(
            second, (int, float)
        ):
            raise ValueError("The first and second inputs must be numbers")
        if first <= 0 or second <= 0:
            raise ValueError("The numbers must be > 0")
        self.first = first
        self.second = second

    def read(self):
        try:
            first = float(input("Enter first value > 0: "))
            second = float(input("Enter second value > 0: "))
            if first <= 0 or second <= 0:
                raise ValueError("The numbers must be > 0")
            self.first = first
            self.second = second
        except ValueError as error:
            print(error)
            self.read()

    def display(self):
        print(f"The catheter a={self.first}, the catheter b={self.second}")

    def hypotenuse(self):
        return math.sqrt(self.first**2 + self.second**2)


def make_right_triangle(first, second):
    try:
        return RightTriangle(first, second)
    except ValueError as error:
        print(error)


if __name__ == "__main__":
    triangle1 = make_right_triangle(3, 4)
    triangle1.display()
    print(triangle1.hypotenuse())

    triangle2 = make_right_triangle(3, 4)
    triangle2.read()
    triangle2.display()
    print(triangle2.hypotenuse())
```

Рисунок 1.4 – Код первого задания

```
The catheter a=3, the catheter b=4
5.0
Enter first value > 0: 4
Enter second value > 0: 6
The catheter a=4.0, the catheter b=6.0
7.211102550927978
```

Рисунок 1.5 – Результат выполнения

```python
# !/usr/bin/env python3
# -*- coding: utf-8 -*-

import unittest
from ind1 import RightTriangle


class RightTriangleTest(unittest.TestCase):
    def test_valid_initialization(self):
        triangle = RightTriangle(4, 6)
        self.assertEqual(triangle.first, 4.0)
        self.assertEqual(triangle.second, 6.0)

    def test_invalid_initialization(self):
        self.assertRaises(ValueError, RightTriangle, -1, 4)
        self.assertRaises(ValueError, RightTriangle, 4, -1)
        self.assertRaises(ValueError, RightTriangle, "bruh", 4)
        self.assertRaises(ValueError, RightTriangle, 4, "bruh")


    def test_hypotenuse(self):
        triangle = RightTriangle(3, 4)
        self.assertEqual(triangle.hypotenuse(), 5.0)

if __name__ == "__main__":
    unittest.main()
```

Рисунок 1.6 – Юниттест первого задания

```
...
----------------------------------------------------------------------
Ran 3 tests in 0.001s

OK
```

Рисунок1.7 – Результат тестирования

4. Выполнение второго задания: создать класс Goods (товар). В классе должны быть представлены поля: наименование товара, дата оформления, цена товара, количество единиц товара, номер накладной, по которой товар поступил на склад. Реализовать методы изменения цены товара, изменения количества товара (увеличения и уменьшения), вычисления стоимости товара.

```python
class Goods:
    def __init__(self, name, date, price, amount, invoice_number):
        self.name = str(name)
        self.date = self._validate_date(date)
        self.price = self._validate_price(price)
        self.amount = self._validate_amount(amount)
        self.invoice_number = str(invoice_number)

    def _validate_date(self, date_str):
        try:
            return datetime.strptime(date_str, "%Y-%m-%d")
        except ValueError:
            raise ValueError("Incorrect date")

    def _validate_price(self, value):
        value = float(value)
        if value < 0:
            raise ValueError("Price must be positive")
        return value

    def _validate_amount(self, value):
        value = int(value)
        if value < 0:
            raise ValueError("Amount must be positive")
        return value

    def read(self):
        self.name = input("Enter good's name: ")
        self.date = self._validate_date(input("Enter date (YYYY-MM-DD): "))
        self.price = self._validate_price(input("Enter price: "))
        self.amount = self._validate_amount(input("Enter amount: "))
        self.invoice_number = input("Enter invoice number: ")

    def display(self):
        print(f"Good's name: {self.name}")
        print(f"Date: {self.date}")
        print(f"Price: {self.price}")
        print(f"Amount: {self.amount}")
        print(f"Invoice number: {self.invoice_number}")

    def change_price(self, new_price):
        self.price = new_price

    def increase_amount(self, amount):
        self.amount += amount

    def decrease_amount(self, amount):
        if self.amount >= amount:
            self.amount -= amount
        else:
            print("Not enough goods")

    def calc_total_cost(self):
        return self.amount * self.price
```

Рисунок 1.8 – Код программы

```python
import unittest
from ind2 import Goods
from datetime import datetime


class GoodsTest(unittest.TestCase):
    def test_valid_initialization(self):
        item = Goods("Desktop", "2024-10-01", 50000.99, 10, "INV123")
        self.assertEqual(item.name, "Desktop")
        self.assertEqual(item.date, datetime.strptime("2024-10-01", "%Y-%m-%d"))
        self.assertEqual(item.price, 50000.99)
        self.assertEqual(item.amount, 10)
        self.assertEqual(item.invoice_number, "INV123")

    def test_invalid_initialization(self):
        self.assertRaises(ValueError, Goods, "Name", "2024.10.1", 100, 10, "INV123")
        self.assertRaises(ValueError, Goods, "Name", "2024-10-1", "bruh", 10, "INV123")
        self.assertRaises(ValueError, Goods, "Name", "2024-10-1", -100, 10, "INV123")
        self.assertRaises(ValueError, Goods, "Name", "2024-10-1", 100, "bruh", "INV123")
        self.assertRaises(ValueError, Goods, "Name", "2024-10-1", -00, -10, "INV123")

    def test_change_price(self):
        item = Goods("Desktop", "2024-10-01", 50000.99, 10, "INV123")
        item.change_price(10000.5)
        self.assertEqual(item.price, 10000.5)
        self.assertRaises(ValueError, item.change_price, -10)
        self.assertRaises(ValueError, item.change_price, "bruh")

    def test_increase_amount(self):
        item = Goods("Desktop", "2024-10-01", 50000.99, 10, "INV123")
        item.increase_amount(10)
        self.assertEqual(item.amount, 20)
        self.assertRaises(ValueError, item.increase_amount, -10)
        self.assertRaises(ValueError, item.increase_amount, "bruh")

    def test_decrease_amount(self):
        item = Goods("Desktop", "2024-10-01", 50000.99, 10, "INV123")
        item.decrease_amount(5)
        self.assertEqual(item.amount, 5)
        item.decrease_amount(10)
        self.assertEqual(item.amount, 0)

    def test_calc_total_cost(self):
        item = Goods("Desktop", "2024-10-01", 50000, 10, "INV123")
        self.assertEqual(item.calc_total_cost(), 50000 * 10)
```

Рисунок 1.9 – Код юниттеста

```
......
----------------------------------------------------------------------
Ran 6 tests in 0.008s

OK
PS C:\Users\MSI\Desktop\ООП\2\ООП_2> ^C
```

Рисунок 2.1 – Результат тестирования

```
Good's name: Desktop
Date: 2024-04-26 00:00:00
Price: 50000.99
Amount: 10
Invoice number: INV123
Good's name: Desktop
Date: 2024-04-26 00:00:00
Price: 55000.0
Amount: 10
Invoice number: INV123
550000.0
```

Рисунок 2.2 – Результат работы программы

Ответы на контрольные вопросы

1. Объявление класса в Python: Класс создаётся с использованием ключевого слова «class», за которым следует имя класса и двоеточие. В теле класса определяются методы и атрибуты.

2. Различие между атрибутами класса и атрибутами экземпляра: - Атрибуты класса объявляются внутри класса, но вне любых методов. Они являются общими для всех экземпляров класса. - Атрибуты экземпляра определяются внутри методов класса и принадлежат конкретному экземпляру класса. Для их определения используется «self», который указывает на текущий объект.

3. Назначение методов класса: Методы класса воздействуют на состояние класса в целом или используются для реализации поведения, характерного для всех экземпляров класса, не требуя создания экземпляра класса. Для того чтобы метод рассматривался как метод класса, его необходимо декорировать декоратором «@classmethod», и его первым параметром должен быть «cls», который ссылается на сам класс.

4. Назначение метода «__init__()»: Метод «__init__()» называют конструктором класса. Он вызывается автоматически при создании нового экземпляра класса. Этот метод используется для инициализации экземпляра, установки начальных значений атрибутов объекта. 5. Назначение «self»: «self» в методах класса представляет текущий экземпляр объекта. Он используется для доступа к атрибутам и другим методам этого объекта изнутри класса. «self» аналогичен «this» в других языках программирования.

6. Добавление атрибутов в класс:

Атрибуты класса добавляются путём объявления их в теле класса. Атрибуты экземпляра обычно добавляются в методе «__init__» или других методах экземпляра с использованием «self».

7. Управление доступом к методам и атрибутам: В Python управление доступом можно осуществлять с помощью указания префиксов для имён атрибутов и методов (одно подчеркивание для "защищенных" и двойное

подчеркивание для "приватных"). Однако это больше соглашения, нежели жесткие правила доступа, как в некоторых других языках.

8. Назначение функции «isinstance()»: Функция «isinstance()» проверяет, принадлежит ли объект к заданному классу или его подклассам. Если объект является экземпляром класса или наследуется от него, функция возвращает «True». Например, «isinstance(obj, MyClass)» вернет «True», если «obj» является экземпляром «MyClass» или его подкласса.