МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
“БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ”

Кафедра интеллектуальных информационных технологий

Отчёт по лабораторной работе №2
Специальность ПО-11

Выполнил:
А. А. Билялова
студент группы ПО11

Проверил:
А. А. Крощенко,
ст. преп. кафедры ИИТ,
22.02.2025

Брест 2025

Цель работы: закрепить навыки объектно-ориентированного программирования на языке Python.

Задание 1. Реализовать пользовательский класс по варианту.

Для каждого класса

• Создать атрибуты (поля) классов

• Создать методы классов

• Добавить необходимые свойства и сеттеры (по необходимости)

• Переопределить магические методы __str__ и __eq__

Прямоугольный треугольник, заданный длинами сторон – Предусмотреть возможность определения площади и периметра, а также логический метод, определяющий существует или такой треугольник. Конструктор должен позволять создавать объекты с начальной инициализацией. Переопределить метод __eq__, выполняющий сравнение объектов данного типа.

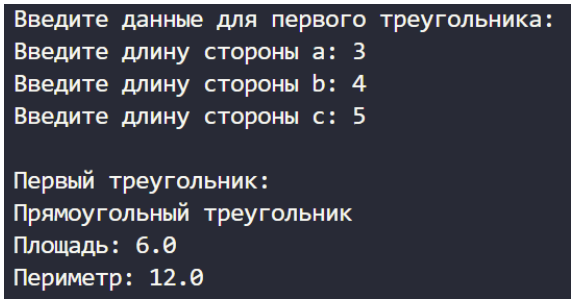
Выполнение:

Код программы:

```
import math

class RightTriangle:
    def __init__(self, a, b, c):
        self.a = a
        self.b = b
        self.c = c

    @property
    def a(self):
        return self._a
    @property
    def b(self):
        return self._b
    @property
    def c(self):
        return self._c
    @a.setter
    def a(self, value):
        if value <= 0:
            raise ValueError("Длина стороны должна быть положительной.")
        self._a = value
    @b.setter
    def b(self, value):
        if value <= 0:
            raise ValueError("Длина стороны должна быть положительной.")
        self._b = value
    @c.setter
    def c(self, value):
        if value <= 0:
            raise ValueError("Длина стороны должна быть положительной.")
        self._c = value
    def is_valid(self):
        sides = sorted([self.a, self.b, self.c])
        return math.isclose(sides[0]**2 + sides[1]**2, sides[2]**2, rel_tol=1e-9)
    def area(self):
        if not self.is_valid():
            raise ValueError("Треугольник с такими сторонами не существует.")
        return (self.a * self.b) / 2
    def perimeter(self):
        if not self.is_valid():
```

```
            raise ValueError("Треугольник с такими сторонами не существует.")
        return self.a + self.b + self.c
    def __str__(self):
        return "Прямоугольный треугольник"
    def __eq__(self, other):
        if not isinstance(other, RightTriangle):
            return False

        return sorted([self.a, self.b, self.c]) == sorted([other.a, other.b, other.c])
def input_triangle():
    try:
        a = float(input("Введите длину стороны a: "))
        b = float(input("Введите длину стороны b: "))
        c = float(input("Введите длину стороны c: "))
        return RightTriangle(a, b, c)
    except ValueError as e:
        print(f"Ошибка: {e}")
        return None

if __name__ == "__main__":
    print("Введите данные для первого треугольника:")
    triangle1 = input_triangle()

    if triangle1:
        print("\nПервый треугольник:")
        print(triangle1)
        if triangle1.is_valid():
            print("Площадь:", triangle1.area())
            print("Периметр:", triangle1.perimeter())
        else:
            print("Треугольник с такими сторонами не существует.")

    print("\nВведите данные для второго треугольника:")
    triangle2 = input_triangle()

    if triangle2:
        print("\nВторой треугольник:")
        print(triangle2)
        if triangle2.is_valid():
            print("Площадь:", triangle2.area())
            print("Периметр:", triangle2.perimeter())
        else:
            print("Треугольник с такими сторонами не существует.")

    if triangle1 and triangle2:
        print("\nСравнение треугольников:")
        print("Треугольник 1 и треугольник 2 равны?", triangle1 == triangle2)
```

Рисунки с результатами работы программы



```
Введите данные для второго треугольника:
Введите длину стороны a: 6
Введите длину стороны b: 8
Введите длину стороны c: 10

Второй треугольник:
Прямоугольный треугольник
Площадь: 24.0
Периметр: 24.0

Сравнение треугольников:
Треугольник 1 и треугольник 2 равны? False
```

```
Введите данные для первого треугольника:
Введите длину стороны a: 6
Введите длину стороны b: 8
Введите длину стороны c: 12

Первый треугольник:
Прямоугольный треугольник
Треугольник с такими сторонами не существует.
```

Задание 2. Построить модель программной системы с применением отношений (обобщения, агрегации, ассоциации, реализации) между классами. Задать атрибуты и методы классов. Реализовать (если необходимо) дополнительные классы. Продемонстрировать работу разработанной системы.

Система Больница. Пациенту назначается лечащий Врач. Врач может сделать назначение Пациенту (процедуры, лекарства, операции). Медсестра или другой Врач выполняют назначение. Пациент может быть выписан из Больницы по окончании лечения, при нарушении режима или иных обстоятельствах.

Выполнение:

Код программы:

```
from typing import List

class Person:
    def __init__(self, name: str, age: int):
        self.name = name
        self.age = age

    def __str__(self):
        return f"{self.name}, {self.age} лет"

class Patient(Person):
    def __init__(self, name: str, age: int, patient_id: str):
        super().__init__(name, age)
        self.patient_id = patient_id
        self.assigned_doctor = None
        self.status = "на лечении"
        self.treatments = []

    def assign_doctor(self, doctor):
        self.assigned_doctor = doctor
        print(f"Пациенту {self.name} назначен врач: {doctor.name}")

    def discharge(self, reason: str):
        self.status = f"выписан ({reason})"
```

```
        print(f"Пациент {self.name} выписан. Причина: {reason}")

    def __str__(self):
        return f"Пациент: {self.name}, ID: {self.patient_id}, Статус: {self.status}"

class Doctor(Person):
    def __init__(self, name: str, age: int, doctor_id: str, specialization: str):
        super().__init__(name, age)
        self.doctor_id = doctor_id
        self.specialization = specialization

    def prescribe_treatment(self, patient, treatment_type: str, description: str):
        treatment = Treatment(treatment_type, description)
        patient.treatments.append(treatment)
        print(f"Врач {self.name} назначил пациенту {patient.name}: {treatment}")

    def __str__(self):
        return f"Врач: {self.name}, Специализация: {self.specialization}"

class Nurse(Person):
    def __init__(self, name: str, age: int, nurse_id: str):
        super().__init__(name, age)
        self.nurse_id = nurse_id

    def perform_treatment(self, patient):
        if patient.treatments:
            treatment = patient.treatments.pop(0)
            print(f"Медсестра {self.name} выполнила назначение для пациента {patient.name}:
{treatment}")
        else:
            print(f"Для пациента {patient.name} нет назначений.")

    def __str__(self):
        return f"Медсестра: {self.name}, ID: {self.nurse_id}"

class Treatment:
    def __init__(self, treatment_type: str, description: str):
        self.treatment_type = treatment_type
        self.description = description

    def __str__(self):
        return f"{self.treatment_type}: {self.description}"

class Hospital:
    def __init__(self):
        self.patients: List[Patient] = []
        self.doctors: List[Doctor] = []
        self.nurses: List[Nurse] = []

    def add_patient(self, patient: Patient):
        self.patients.append(patient)
        print(f"Пациент {patient.name} добавлен в больницу.")

    def add_doctor(self, doctor: Doctor):
        self.doctors.append(doctor)
        print(f"Врач {doctor.name} добавлен в больницу.")

    def add_nurse(self, nurse: Nurse):
        self.nurses.append(nurse)
        print(f"Медсестра {nurse.name} добавлена в больницу.")

    def assign_doctor_to_patient(self, patient: Patient, doctor: Doctor):
        patient.assign_doctor(doctor)
        print(f"Врач {doctor.name} назначен пациенту {patient.name}.")

if __name__ == "__main__":
    # Создаем больницу
    hospital = Hospital()
```

```
# Создаем пациентов
patient1 = Patient("Иван Иванов", 30, "P001")
patient2 = Patient("Мария Петрова", 25, "P002")

# Создаем врачей
doctor1 = Doctor("Алексей Сидоров", 45, "D001", "Хирург")
doctor2 = Doctor("Елена Кузнецова", 40, "D002", "Терапевт")

# Создаем медсестру
nurse1 = Nurse("Ольга Смирнова", 35, "N001")

# Добавляем всех в больницу
hospital.add_patient(patient1)
hospital.add_patient(patient2)
hospital.add_doctor(doctor1)
hospital.add_doctor(doctor2)
hospital.add_nurse(nurse1)

# Назначаем врачей пациентам
hospital.assign_doctor_to_patient(patient1, doctor1)
hospital.assign_doctor_to_patient(patient2, doctor2)

# Врачи назначают лечение
doctor1.prescribe_treatment(patient1, "операция", "Удаление аппендицита")
doctor2.prescribe_treatment(patient2, "лекарство", "Антибиотики")

# Медсестра выполняет назначения
nurse1.perform_treatment(patient1)
nurse1.perform_treatment(patient2)

# Выписываем пациентов
patient1.discharge("окончание лечения")
patient2.discharge("нарушение режима")

print("\nИнформация о пациентах:")
print(patient1)
print(patient2)
```

Рисунки с результатами работы программы

```
Пациент Иван Иванов добавлен в больницу.
Пациент Мария Петрова добавлен в больницу.
Врач Алексей Сидоров добавлен в больницу.
Врач Елена Кузнецова добавлен в больницу.
Медсестра Ольга Смирнова добавлена в больницу.
Пациенту Иван Иванов назначен врач: Алексей Сидоров
Врач Алексей Сидоров назначен пациенту Иван Иванов.
Пациенту Мария Петрова назначен врач: Елена Кузнецова
Врач Елена Кузнецова назначен пациенту Мария Петрова.
Врач Алексей Сидоров назначил пациенту Иван Иванов: операция: Удаление аппендицита
Врач Елена Кузнецова назначил пациенту Мария Петрова: лекарство: Антибиотики
Медсестра Ольга Смирнова выполнила назначение для пациента Иван Иванов: операция: Удаление аппендицита
Медсестра Ольга Смирнова выполнила назначение для пациента Мария Петрова: лекарство: Антибиотики
Пациент Иван Иванов выписан. Причина: окончание лечения
Пациент Мария Петрова выписан. Причина: нарушение режима
Пациент Мария Петрова выписан. Причина: нарушение режима

Информация о пациентах:
Пациент: Иван Иванов, ID: P001, Статус: выписан (окончание лечения)
Пациент: Мария Петрова, ID: P002, Статус: выписан (нарушение режима)
```

Вывод: закрепила навыки объектно-ориентированного программирования на языке Python.