Министерство образования Республики Беларусь
Учреждение образования
«Брестский государственный технический университет»
Кафедра ИИТ

Лабораторная работа №2

По дисциплине «Современные платформы программирования»

Выполнил:
Студент 3 курса
Группы ПО-11
Микулич М. И.
Проверил:
Козик И. Д.

Брест 2025

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

Вариант 14

Задание 1

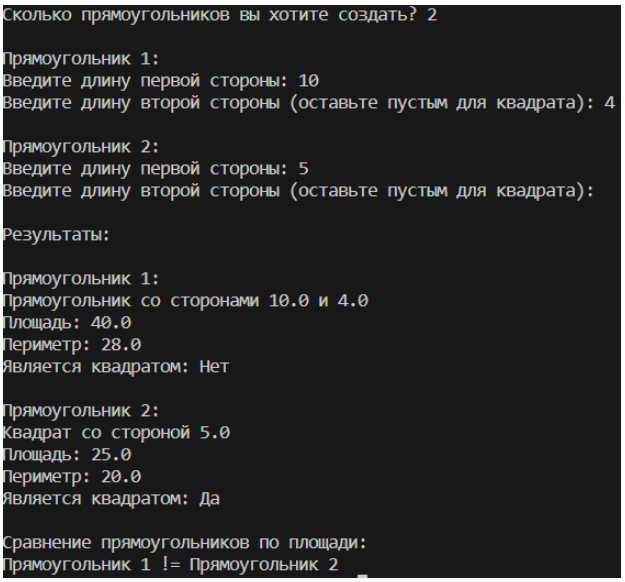
Прямоугольник, заданный длинами двух сторон – Предусмотреть возможность определения площади и периметра, а также логические методы,определяющие, является ли прямоугольник квадратом и существует ли такой прямоугольник. Конструктор должен позволять создавать объекты с начальной инициализацией. Переопределить метод __eq__, выполняющий сравнение объектов данного типа.

Код программы:

```
class Rectangle:
    def __init__(self, side1: float, side2: float = None):
        """
        Конструктор класса Rectangle.
        Позволяет создать прямоугольник с заданными сторонами.
        Если передана только одна сторона, создаётся квадрат.
        :param side1: Длина первой стороны
        :param side2: Длина второй стороны (необязательный, если None - квадрат)
        """
        self.side1 = side1
        self.side2 = side2 if side2 is not None else side1

    @property
    def side1(self) -> float:
        """Геттер для первой стороны"""
        return self._side1

    @side1.setter
    def side1(self, value: float):
        """Сеттер для первой стороны с проверкой на положительность"""
        if value <= 0:
            raise ValueError("Длина стороны должна быть положительной")
        self._side1 = value

    @property
    def side2(self) -> float:
        """Геттер для второй стороны"""
        return self._side2

    @side2.setter
    def side2(self, value: float):
        """Сеттер для второй стороны с проверкой на положительность"""
        if value <= 0:
            raise ValueError("Длина стороны должна быть положительной")
        self._side2 = value

    def area(self) -> float:
        """Вычисляет площадь прямоугольника"""
        return self.side1 * self.side2

    def perimeter(self) -> float:
        """Вычисляет периметр прямоугольника"""
        return 2 * (self.side1 + self.side2)

    def is_square(self) -> bool:
        """Проверяет, является ли прямоугольник квадратом"""
        return self.side1 == self.side2

    def exists(self) -> bool:
        """Проверяет, существует ли такой прямоугольник (стороны положительные)"""
        return self.side1 > 0 and self.side2 > 0

    def __str__(self) -> str:
        """Строковое представление объекта"""
        if self.is_square():
            return f"Квадрат со стороной {self.side1}"
        return f"Прямоугольник со сторонами {self.side1} и {self.side2}"

    def __eq__(self, other) -> bool:
        """Сравнение двух прямоугольников по площадям"""
        if not isinstance(other, Rectangle):
```

```
            return False
        return self.area() == other.area()

def input_rectangle() -> Rectangle:
    """Функция для ввода данных прямоугольника с клавиатуры"""
    while True:
        try:
            side1 = float(input("Введите длину первой стороны: "))
            side2 = input("Введите длину второй стороны (оставьте пустым для квадрата): ")
            if side2.strip() == "":
                return Rectangle(side1)
            else:
                return Rectangle(side1, float(side2))
        except ValueError as e:
            print(f"Ошибка: {e}. Попробуйте снова.")

def main():
    rectangles = []
    n = int(input("Сколько прямоугольников вы хотите создать? "))

    for i in range(n):
        print(f"\nПрямоугольник {i + 1}:")
        rect = input_rectangle()
        rectangles.append(rect)

    print("\nРезультаты:")
    for i, rect in enumerate(rectangles, 1):
        print(f"\nПрямоугольник {i}:")
        print(rect)
        print(f"Площадь: {rect.area()}")
        print(f"Периметр: {rect.perimeter()}")
        print(f"Является квадратом: {'Да' if rect.is_square() else 'Нет'}")

    # Сравнение всех прямоугольников попарно
    print("\nСравнение прямоугольников по площади:")
    for i in range(len(rectangles)):
        for j in range(i + 1, len(rectangles)):
            print(f"Прямоугольник {i + 1} {'==' if rectangles[i] == rectangles[j] else '!='} Прямоугольник {j + 1}")

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

Результат работы:



Задание 2:

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.

Код программы:

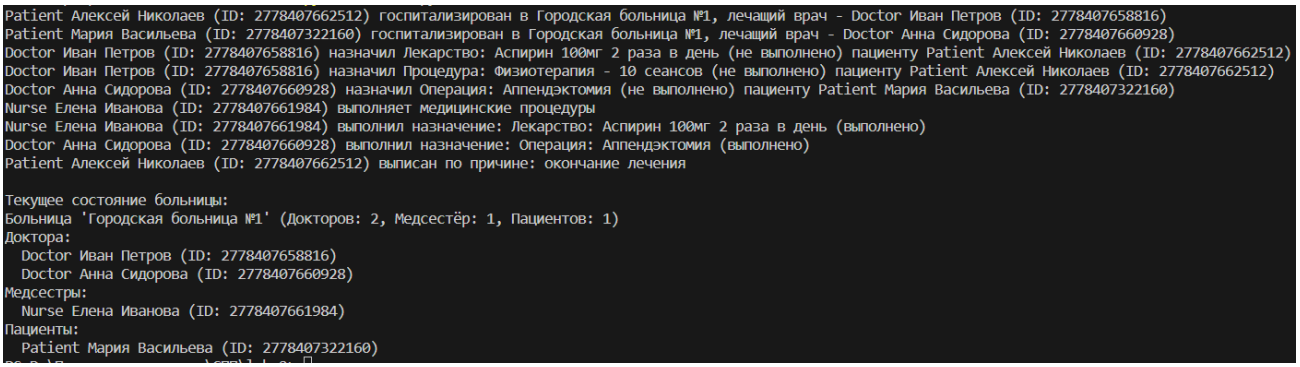
```
from abc import ABC, abstractmethod
```

```
from datetime import datetime
from typing import List, Optional

class Person:
    """Базовый класс для всех людей в системе"""
    def __init__(self, name: str, birth_date: str):
        self.name = name
        self.birth_date = birth_date
        self._id = id(self)

    def __str__(self):
        return f"{self.__class__.__name__} {self.name} (ID: {self._id})"

class MedicalStaff(Person, ABC):
    """Абстрактный класс для медицинского персонала"""
    def __init__(self, name: str, birth_date: str, position: str):
        super().__init__(name, birth_date)
        self.position = position
        self.assigned_patients: List['Patient'] = []

    @abstractmethod
    def perform_duty(self):
        pass

class Doctor(MedicalStaff):
    """Класс врача"""
    def __init__(self, name: str, birth_date: str, specialty: str):
        super().__init__(name, birth_date, "Врач")
        self.specialty = specialty

    def prescribe_treatment(self, patient: 'Patient', prescription: 'Prescription'):
        """Назначить лечение пациенту"""
        if patient not in self.assigned_patients:
            self.assigned_patients.append(patient)
            patient.attending_doctor = self

        patient.add_prescription(prescription)
        print(f"{self} назначил {prescription} пациенту {patient}")

    def perform_duty(self):
        print(f"{self} проводит осмотр пациентов")

class Nurse(MedicalStaff):
    """Класс медсестры"""
    def __init__(self, name: str, birth_date: str):
        super().__init__(name, birth_date, "Медсестра")

    def perform_duty(self):
        print(f"{self} выполняет медицинские процедуры")

class Patient(Person):
    """Класс пациента"""
    def __init__(self, name: str, birth_date: str, admission_date: str):
        super().__init__(name, birth_date)
        self.admission_date = admission_date
        self.discharge_date: Optional[str] = None
        self.discharge_reason: Optional[str] = None
        self.attending_doctor: Optional[Doctor] = None
        self.prescriptions: List['Prescription'] = []

    def add_prescription(self, prescription: 'Prescription'):
        self.prescriptions.append(prescription)

    def discharge(self, reason: str):
        """Выписать пациента"""
        self.discharge_date = datetime.now().strftime("%Y-%m-%d")
        self.discharge_reason = reason
        print(f"{self} выписан по причине: {reason}")

class Prescription:
    """Класс медицинского назначения"""
    def __init__(self, prescription_type: str, description: str, date: str):
        self.prescription_type = prescription_type  # процедура, лекарство, операция
        self.description = description
        self.date = date
        self.completed = False
        self.completed_by: Optional[MedicalStaff] = None

    def complete(self, staff: MedicalStaff):
        self.completed = True
        self.completed_by = staff
        print(f"{staff} выполнил назначение: {self}")
```

```
    def __str__(self):
        status = "выполнено" if self.completed else "не выполнено"
        return f"{self.prescription_type}: {self.description} ({status})"

class Hospital:
    """Класс больницы (агрегирует все остальные классы)"""
    def __init__(self, name: str):
        self.name = name
        self.doctors: List[Doctor] = []
        self.nurses: List[Nurse] = []
        self.patients: List[Patient] = []

    def add_doctor(self, doctor: Doctor):
        self.doctors.append(doctor)

    def add_nurse(self, nurse: Nurse):
        self.nurses.append(nurse)

    def admit_patient(self, patient: Patient, doctor: Doctor):
        self.patients.append(patient)
        doctor.assigned_patients.append(patient)
        patient.attending_doctor = doctor
        print(f"{patient} госпитализирован в {self.name}, лечащий врач - {doctor}")

    def discharge_patient(self, patient: Patient, reason: str):
        if patient in self.patients:
            patient.discharge(reason)
            self.patients.remove(patient)
            if patient.attending_doctor:
                patient.attending_doctor.assigned_patients.remove(patient)
        else:
            print(f"Пациент {patient} не найден в больнице")

    def __str__(self):
        return f"Больница '{self.name}' (Докторов: {len(self.doctors)}, Медсестёр: {len(self.nurses)}, Пациентов:
{len(self.patients)})"

def demonstrate_hospital_system():
    # Создаем больницу
    hospital = Hospital("Городская больница №1")

    # Нанимаем персонал
    dr_smith = Doctor("Иван Петров", "1980-05-15", "Кардиолог")
    dr_jones = Doctor("Анна Сидорова", "1975-11-22", "Хирург")
    nurse_emma = Nurse("Елена Иванова", "1990-03-10")

    hospital.add_doctor(dr_smith)
    hospital.add_doctor(dr_jones)
    hospital.add_nurse(nurse_emma)

    # Госпитализируем пациентов
    patient1 = Patient("Алексей Николаев", "1995-07-20", "2023-11-01")
    patient2 = Patient("Мария Васильева", "1988-02-14", "2023-11-02")

    hospital.admit_patient(patient1, dr_smith)
    hospital.admit_patient(patient2, dr_jones)

    # Врачи делают назначения
    prescription1 = Prescription("Лекарство", "Аспирин 100мг 2 раза в день", "2023-11-01")
    prescription2 = Prescription("Процедура", "Физиотерапия - 10 сеансов", "2023-11-02")
    prescription3 = Prescription("Операция", "Аппендэктомия", "2023-11-03")

    dr_smith.prescribe_treatment(patient1, prescription1)
    dr_smith.prescribe_treatment(patient1, prescription2)
    dr_jones.prescribe_treatment(patient2, prescription3)

    # Медперсонал выполняет назначения
    nurse_emma.perform_duty()
    prescription1.complete(nurse_emma)
    prescription3.complete(dr_jones)

    # Выписываем пациента
    hospital.discharge_patient(patient1, "окончание лечения")

    # Выводим информацию о больнице
    print("\nТекущее состояние больницы:")
    print(hospital)
    print("Доктора:", *hospital.doctors, sep="\n  ")
    print("Медсестры:", *hospital.nurses, sep="\n  ")
    print("Пациенты:", *hospital.patients, sep="\n  ")
```

```
if __name__ == "__main__":
    demonstrate_hospital_system()
```

Результат работы:



Вывод: были закреплены навыки объектно-ориентированного программирования на языке Python.