МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

“БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ”

ФАКУЛЬТЕТ ЭЛЕКТРОННО-ИНФОРМАЦИОННЫХ СИСТЕМ

Кафедра интеллектуальных информационных технологий

Отчет по лабораторной работе №2

Специальность ПО-13

Выполнил:

Литвичук А.М.

студент группы ПО-13

Проверил:

А. Д. Кулик

13.02.2026

Брест 2026

Цель работы:

Закрепить навыки объектно-ориентированного программирования на языке Python

Задание 1.

Реализовать простой класс
Требования к выполнению
• 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__
2) Равносторонний 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.

Код программы:

```
import math


class EquilateralTriangle:
    def __init__(self, a: float, b: float, c: float):
        # Инициализация сторон
        self._a = a
        self._b = b
        self._c = c

    # ---------- Свойства ----------

    @property
    def a(self):
        return self._a

    @a.setter
    def a(self, value):
        self._a = value

    @property
    def b(self):
        return self._b

    @b.setter
    def b(self, value):
        self._b = value

    @property
    def c(self):
        return self._c

    @c.setter
    def c(self, value):
        self._c = value
```

```
    def exists(self) -> bool:
        # Проверка существования равностороннего треугольника
        return self._a > 0 and self._a == self._b == self._c

    def perimeter(self) -> float:
        if not self.exists():
            raise ValueError("Треугольник не существует")
        return self._a + self._b + self._c

    def area(self) -> float:
        if not self.exists():
            raise ValueError("Треугольник не существует")
        return (self._a**2 * math.sqrt(3)) / 4

    #Магические методы

    def __str__(self):
        return f"Равносторонний треугольник со сторонами: {self._a}, {self._b},
{self._c}"

    def __eq__(self, other):
        if not isinstance(other, EquilateralTriangle):
            return NotImplemented
        return self._a == other._a and self._b == other._b and self._c == other._c


# Точка входа

if __name__ == "__main__":
    t1 = EquilateralTriangle(5, 5, 5)
    t2 = EquilateralTriangle(3, 3, 3)
    t3 = EquilateralTriangle(5, 5, 5)

    print(t1)
    print("Существует:", t1.exists())
    print("Периметр:", t1.perimeter())
    print("Площадь:", round(t1.area(), 2))

    print("t1 == t2:", t1 == t2)
    print("t1 == t3:", t1 == t3)
```

Спецификация ввода

Введите длину стороны треугольника: <число>

Пример

Введите длину стороны треугольника: 5

Спецификация вывода

Треугольник существует

Периметр: <число>

Площадь: <число>

Треугольник не существует

Пример

Введите длину стороны треугольника: 5

Рисунки с результатами работы программы

```
Равносторонний треугольник со сторонами: 5, 5, 5
Существует: True
Периметр: 15
Площадь: 10.83
t1 == t2: False
t1 == t3: True
```

Задание 2.

1) Система Факультатив. Преподаватель объявляет запись на Курс. Студент записывается на Курс, обучается и по окончании Преподаватель выставляет Оценку, которая сохраняется в Архиве. Студентов, Преподавателей и Курсов при обучении может быть несколько..

Код программы:

```python
from abc import ABC, abstractmethod

# Обобщение


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

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


class Student(Person):
    def __init__(self, name: str):
        super().__init__(name)
        self.courses = []

    def enroll(self, course):
        course.add_student(self)
        self.courses.append(course)

    def __str__(self):
        return f"Student: {self.name}"


class Teacher(Person):
    def assign_grade(self, student, course, grade, archive_obj):
        archive_obj.save_grade(student, course, grade)


# Реализация (интерфейс)


class Graded(ABC):
```

```python
    @abstractmethod
    def save_grade(self, student, course, grade):
        pass


# Агрегация


class Archive(Graded):
    def __init__(self):
        self.records = []

    def save_grade(self, student, course, grade):
        self.records.append((student.name, course.title, grade))

    def show_records(self):
        for record in self.records:
            print(f"Студент: {record[0]}, " f"Курс: {record[1]}, " f"Оценка:
{record[2]}")


class Course:
    def __init__(self, title: str, teacher: Teacher):
        self.title = title
        self.teacher = teacher
        self.students = []

    def add_student(self, student: Student):
        self.students.append(student)

    def __str__(self):
        return f"Курс: {self.title}, Преподаватель: {self.teacher.name}"


# Демонстрация


if __name__ == "__main__":
    archive_storage = Archive()

    teacher1 = Teacher("Иванов")
    teacher2 = Teacher("Петров")

    course1 = Course("Python", teacher1)
    course2 = Course("Databases", teacher2)

    student1 = Student("Алексей")
    student2 = Student("Мария")

    student1.enroll(course1)
    student2.enroll(course1)
    student2.enroll(course2)

    teacher1.assign_grade(student1, course1, 5, archive_storage)
    teacher1.assign_grade(student2, course1, 4, archive_storage)
    teacher2.assign_grade(student2, course2, 5, archive_storage)
```

```
archive_storage.show_records()
```

Спецификация ввода

Ввод данных о преподавателях

Введите количество преподавателей: **<число>**

Для каждого преподавателя: Введите имя преподавателя: **<строка>**

Ввод данных о курсах

Введите количество курсов: **<число>**

Для каждого курса: Введите название курса: **<строка>** Введите имя преподавателя, ведущего курс: **<строка>** *(должен совпадать с ранее введённым)*

Ввод данных о студентах

Введите количество студентов: **<число>**

Для каждого студента: Введите имя студента: **<строка>** Введите количество курсов, на которые он записывается: **<число>**

Для каждого курса: Введите название курса для записи: **<строка>**

Ввод оценок

Введите количество выставляемых оценок: **<число>**

Для каждой оценки: Введите имя преподавателя, выставляющего оценку: **<строка>** Введите имя студента: **<строка>** Введите название курса: **<строка>** Введите оценку: **<число>**

...
Введите имя члена экипажа n: <строка>
Введите должность члена экипажа n: <строка>

Введите действие:
1 — отменить рейс из-за погоды
2 — изменить аэропорт назначения
Введите номер действия: <число>

Если выбрано действие 1:
Введите аэропорт, в котором плохая погода: <строка>

Если выбрано действие 2:
Введите новый аэропорт назначения: <строка>
Введите город нового аэропорта: <строка>

Пример
Введите код рейса: SU123
Введите модель самолёта: Airbus A320
Введите вместимость самолёта: 180
Введите дальность полёта самолёта: 6100

Введите аэропорт вылета: Шереметьево
Введите город вылета: Москва

Введите аэропорт назначения: Минск-2
Введите город назначения: Минск

Введите количество членов экипажа: 3

Введите имя члена экипажа 1: Иванов И.И.
Введите должность члена экипажа 1: пилот

Введите имя члена экипажа 2: Петров П.П.
Введите должность члена экипажа 2: штурман

Введите имя члена экипажа 3: Анна А.А.
Введите должность члена экипажа 3: стюардесса

Введите номер действия: 2
Введите новый аэропорт назначения: Гомель
Введите город нового аэропорта: Гомель

Спецификация вывода

Рейс <код> отменён из-за погодных условий в аэропорту <аэропорт>.

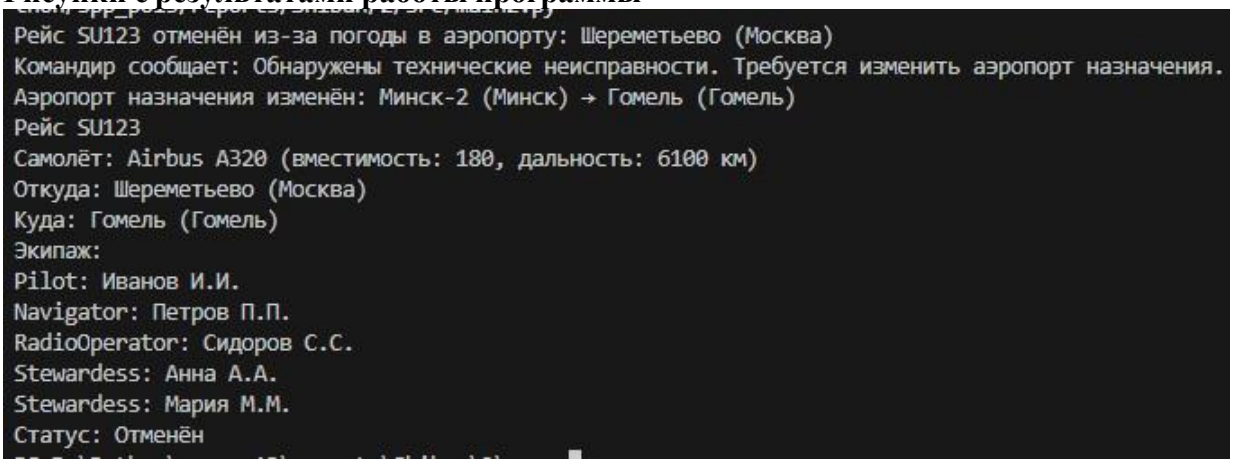
Пример

Командир сообщает: Обнаружены технические неисправности. Требуется изменить аэропорт назначения.
Аэропорт назначения изменён: Минск-2 (Минск) → Гомель (Гомель)

Рейс SU123
Самолёт: Airbus A320 (вместимость: 180, дальность: 6100 км)
Откуда: Шереметьево (Москва)
Куда: Гомель (Гомель)
Экипаж:
Pilot: Иванов И.И.
Navigator: Петров П.П.

Stewardess: Анна А.А.
Статус: Активен

Рисунки с результатами работы программы



Вывод: В процессе выполнения были применены ключевые принципы ООП: инкапсуляция, наследование, полиморфизм, агрегация, ассоциация и реализация интерфейсов. Создание собственных классов, определение их атрибутов и методов, переопределение магических методов и построение связей между объектами позволили сформировать целостное понимание того, как проектировать программные системы на основе объектов.