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:

Д. В. Шибун

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, side: float):
        self.side = side

    @property
    def side(self):
        return self._side

    @side.setter
    def side(self, value):
        if value <= 0:
            raise ValueError("Длина стороны должна быть положительным числом.")
        self._side = float(value)

    def exists(self) -> bool:
        return self.side > 0 # ну этого хватит, он же равносторонний

    @property
    def perimeter(self) -> float:
        return 3 * self.side

    @property
    def area(self) -> float:
        return (math.sqrt(3) / 4) * (self.side ** 2)

    def __eq__(self, other) -> bool:
        if not isinstance(other, EquilateralTriangle):
            return False
        return self.side == other.side

    def __str__(self):
        return (f"Равносторонний треугольник: сторона = {self.side}, "
                f"периметр = {self.perimeter:.2f}, площадь = {self.area:.2f}")

t1 = EquilateralTriangle(5)
t2 = EquilateralTriangle(5)
```

```
t3 = EquilateralTriangle(7)

print(t1)
print("t1 == t2:", t1 == t2)
print("t1 == t3:", t1 == t3)
print("Существует ли t1:", t1.exists())
```

Спецификация ввода

Введите длину стороны треугольника: <число>

Пример

Введите длину стороны треугольника: 5

Спецификация вывода

Треугольник существует

Периметр: <число>

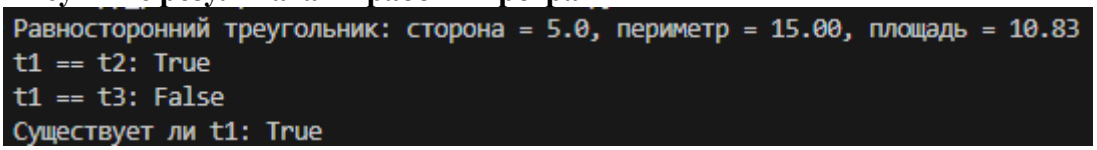
Площадь: <число>

Треугольник не существует

Пример

Введите длину стороны треугольника: 5

Рисунки с результатами работы программы



Задание 2.

11) Система Аэрофлот. Администратор формирует летную Бригаду (пилоты, штурман, радист, стюардессы) на Рейс. Каждый Рейс выполняется Самолетом с определенной вместимостью и дальностью полета. Рейс может быть отменен из-за погодных условий в Аэропорту отлета или назначения. Аэропорт назначения может быть изменен в полете из-за технических неисправностей, о которых сообщил командир.

Код программы:

```
from abc import ABC, abstractmethod

# реализация
class TechnicalReportProvider(ABC):
    @abstractmethod
    def report_issue(self) -> str:
        pass

# базовый класс члена экипажа
class CrewMember:
    def __init__(self, name: str):
        self.name = name

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

```
# наследники
class Pilot(CrewMember, TechnicalReportProvider):
    def report_issue(self) -> str:
        return "Обнаружены технические неисправности. Требуется изменить аэропорт назначения."

class Navigator(CrewMember):
    pass

class RadioOperator(CrewMember):
    pass

class Stewardess(CrewMember):
    pass

# самолёт
class Airplane:
    def __init__(self, model: str, capacity: int, range_km: int):
        self.model = model
        self.capacity = capacity
        self.range_km = range_km

    def __str__(self):
        return f"{self.model} (вместимость: {self.capacity}, дальность: {self.range_km} км)"

# аэропорт
class Airport:
    def __init__(self, name: str, city: str):
        self.name = name
        self.city = city

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

# агрегация
class FlightCrew:
    def __init__(self):
        self.members = []

    def add_member(self, member: CrewMember):
        self.members.append(member)

    def __str__(self):
        return "\n".join(str(m) for m in self.members)

# рейс
class Flight:
    def __init__(self, code: str, airplane: Airplane, departure: Airport, destination: Airport):
        self.code = code
        self.airplane = airplane
        self.departure = departure
        self.destination = destination
        self.crew = FlightCrew()
        self.cancelled = False

    def cancel_due_weather(self, airport: Airport):
        self.cancelled = True
        print(f"Рейс {self.code} отменён из-за погоды в аэропорту: {airport}")

    def change_destination(self, new_airport: Airport, pilot: Pilot):
        reason = pilot.report_issue()
```

```
        print(f"Командир сообщает: {reason}")
        print(f"Аэропорт назначения изменён: {self.destination} → {new_airport}")
        self.destination = new_airport

    def __str__(self):
        return (f"Рейс {self.code}\n"
                f"Самолёт: {self.airplane}\n"
                f"Откуда: {self.departure}\n"
                f"Куда: {self.destination}\n"
                f"Экипаж:\n{self.crew}\n"
                f"Статус: {'Отменён' if self.cancelled else 'Активен'}")

# аэропорты
moscow = Airport("Шереметьево", "Москва")
minsk = Airport("Минск-2", "Минск")
gomel = Airport("Гомель", "Гомель")

# самолёт
airbus = Airplane("Airbus A320", 180, 6100)

# рейс
flight = Flight("SU123", airbus, moscow, minsk)

# формирование экипажа
pilot = Pilot("Иванов И.И.")
flight.crew.add_member(pilot)
flight.crew.add_member(Navigator("Петров П.П."))
flight.crew.add_member(RadioOperator("Сидоров С.С."))
flight.crew.add_member(Stewardess("Анна А.А."))
flight.crew.add_member(Stewardess("Мария М.М."))

# отмена рейса из-за погоды
flight.cancel_due_weather(moscow)

# изменение аэропорта назначения из-за неисправностей
flight.change_destination(gomel, pilot)

# вывод информации о рейсе
print(flight)
```

Спецификация ввода

Введите код рейса: <строка>
Введите модель самолёта: <строка>
Введите вместимость самолёта: <число>
Введите дальность полёта самолёта: <число>

Введите аэропорт вылета: <строка>
Введите город вылета: <строка>

Введите аэропорт назначения: <строка>
Введите город назначения: <строка>

Введите количество членов экипажа: <число>

Введите имя члена экипажа 1: <строка>
Введите должность члена экипажа 1 (пилот/штурман/радист/стюардесса): <строка>

Введите имя члена экипажа 2: <строка>
Введите должность члена экипажа 2: <строка>

...
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>.

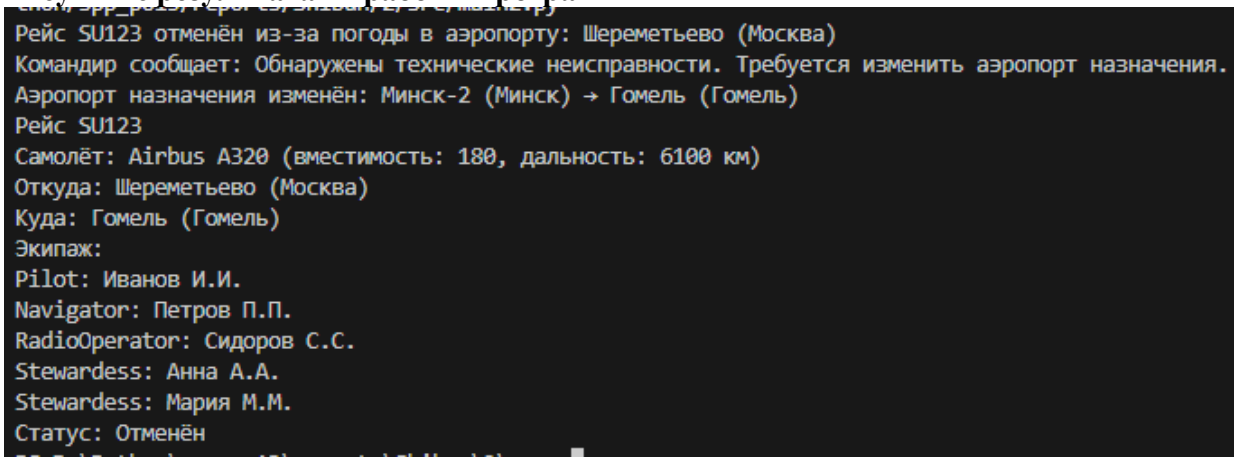
Пример

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.