

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

Институт перспективной инженерии  
Департамент цифровых, робототехнических систем и электроники

**ОТЧЕТ**  
**ПО ЛАБОРАТОРНОЙ РАБОТЕ №3**  
**дисциплины**  
**«Объектно-ориентированное программирование»**  
**Вариант № 3**

Выполнил:  
Левашев Тимур Рашидович  
3 курс, группа ИВТ-б-о-23-2,  
09.03.01 «Информатика и  
вычислительная техника»,  
направленность (профиль)  
«Программное обеспечение средств  
вычислительной техники и  
автоматизированных систем», очная  
форма обучения

---

(подпись)

Проверил:  
Доцент департамента цифровых,  
робототехнических систем и  
электроники института перспективной  
инженерии Воронкин Р.А

---

(подпись)

Отчет защищен с оценкой \_\_\_\_\_ Дата защиты \_\_\_\_\_

Ставрополь, 2025 г.

**Тема работы:** “Наследование и полиморфизм в языке Python”

**Цель работы:** Приобретение навыков по созданию иерархии классов при написании программ с помощью языка программирования Python.

**Порядок выполнения работы:**

**Ссылка на Git репозиторий:** [https://github.com/mazy99/oop\\_pract\\_3](https://github.com/mazy99/oop_pract_3)

1. Пример 1 использования абстрактного класса и абстрактного метода.

Листинг программы:

```
2.#!/usr/bin/env python3
3. # -*- coding: utf-8 -*-
4.
5. from abc import ABC, abstractmethod
6.
7. class Polygon(ABC):
8.
9.     @abstractmethod
10.    def noofsides(self) -> None:
11.        pass
12.
13. class Triangle(Polygon):
14.
15.    def noofsides(self) -> None:
16.        print("I have 3 sides")
17.
18. class Pentagon(Polygon):
19.
20.    def noofsides(self) -> None:
21.        print("I have 5 sides")
22.
23. class Hexagon(Polygon):
24.
25.    def noofsides(self) -> None:
26.        print("I have 6 sides")
27.
28. class Quadrilateral(Polygon):
29.
30.    def noofsides(self) -> None:
31.        print("I have 4 sides")
32.
33. R = Triangle()
34. R.noofsides()
35.
36. K = Quadrilateral()
```

```

37.K.noofsides()
38.
39.P = Pentagon()
40.P.noofsides()
41.
42.H = Hexagon()
43.H.noofsides()

```

```

I have 3 sides
I have 4 sides
I have 5 sides
I have 6 sides

```

Рисунок 1 – Пример вывода данных

2. Пример 2 использования абстрактного класса и абстрактного метода.

Листинг программы:

```

3.#!/usr/bin/env python3
4. # -*- coding: utf-8 -*-
5.
6. from abc import ABC, abstractmethod
7.
8. class Animal(ABC):
9.     @abstractmethod
10.    def move(self) -> None:
11.        pass
12.
13. class Human(Animal):
14.
15.    def move(self) -> None:
16.        print("I can walk and run")
17.
18. class Snake(Animal):
19.
20.    def move(self) -> None:
21.        print("I can crawl")
22.
23. class Dog(Animal):
24.
25.    def move(self) -> None:
26.        print("I can bark")
27.
28. class Lion(Animal):
29.
30.    def move(self) -> None:
31.        print("I can roar")
32.
33. H = Human()
34. H.move()

```

```

35.
36.S = Snake()
37.S.move()
38.
39.D = Dog()
40.D.move()
41.
42.K = Lion()
43.K.move()
44.

```

```

I can walk and run
I can crawl
I can bark
I can roar

```

Рисунок 2 – Пример вывода данных

3. Пример использования абстрактного класса с абстрактными свойствами.

Листинг программы:

```

4. #!/usr/bin/env python3
5. # -*- coding: utf-8 -*-
6.
7. from abc import ABC, abstractmethod
8.
9. class parent(ABC):
10.
11.     @property
12.     @abstractmethod
13.     def geeks(self) -> str:
14.         pass
15.
16. class child(parent):
17.
18.     @property
19.     def geeks(self) -> str:
20.         return "child class"
21.
22. try:
23.     r = parent() # type: ignore[abstract]
24.     print(r.geeks)
25. except Exception as err:
26.     print(err)
27.
28. r = child()
29. print(r.geeks)
30.

```

```
Can't instantiate abstract class parent without an implementation for abstract method 'geeks'  
child class
```

Рисунок 3 – Пример вывода данных

#### 4. Составить программу с использованием иерархии классов.

3. Создать класс Liquid (жидкость), имеющий поля названия и плотности. Определить методы переназначения и изменения плотности. Создать производный класс Alcohol (спирт), имеющий крепость. Определить методы переназначения и изменения крепости.

Рисунок 4 – Вариант задания

Листинг программы:

```
#!/usr/bin/env python3  
# -*- coding: utf-8 -*-  
  
class Liquid:  
  
    def __init__(self, names: str, dens: float) -> None:  
  
        self.__name = names  
        self.__dens = dens  
  
    def __str__(self) -> str:  
        return f"Жидкость: {self.name}, Плотность: {self.density} кг/м³"  
  
    @property  
    def name(self) -> str:  
        return self.__name  
  
    @property  
    def density(self) -> float:  
        return self.__dens  
  
    @name.setter  
    def name(self, new_name: str) -> None:  
        self.__name = new_name  
  
    @density.setter  
    def density(self, new_dens: float) -> None:  
        if new_dens <= 0:  
            raise ValueError("Плотность должна быть положительным числом")  
        self.__dens = new_dens
```

```

class Alcohol(Liquid):

    def __init__(self, names: str, dens: float, strength: float) -> None:
        super().__init__(names, dens)
        self.__strength = strength

    def __str__(self):
        return f"Алкогoль: {self.name},\
Плотность: {self.density} кг/м³, Крепость: {self.strength}%"

    @property
    def strength(self) -> float:
        return self.__strength

    @strength.setter
    def strength(self, new_strength: float):
        if new_strength < 0 or new_strength > 100:
            raise ValueError("Крепость должна быть в диапазоне от 0 до 100%")
        self.__strength = new_strength

```

Пример вызова программы:

```

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

from liquid import Alcohol, Liquid

if __name__ == "__main__":
    water = Liquid("Вoдa", 997)
    print(water)

    water.name = "Дистиллированная вода"
    water.density = 1000
    print(f"После изменений: {water}")
    print(f"Название: {water.name}, Плотность: {water.density}")

    print("\n" + "=" * 50 + "\n")

    vodka = Alcohol("Вoдкa", 940, 40)
    print(vodka)

    vodka.name = "Премиальная водка"
    vodka.density = 942
    vodka.strength = 45
    print(f"После переназначения: {vodka}")

    vodka.strength = vodka.strength - 5
    print(f"После уменьшения крепости: {vodka}")

```

```

print(f"Крепость напитка: {vodka.strength}%")

try:
    vodka.strength = 150
except ValueError as e:
    print(f"Ошибка: {e}")

try:
    water.density = -10
except ValueError as e:
    print(f"Ошибка: {e}")

```

```

Жидкость: Вода, Плотность: 997 кг/м³
После изменений: Жидкость: Дистиллированная вода, Плотность: 1000 кг/м³
Название: Дистиллированная вода, Плотность: 1000

=====

Алкоголь: Водка, Плотность: 940 кг/м³, Крепость: 40%
После переназначения: Алкоголь: Премиальная водка, Плотность: 942 кг/м³, Крепость: 45%
После уменьшения крепости: Алкоголь: Премиальная водка, Плотность: 942 кг/м³, Крепость: 40%
Крепость напитка: 40%
Ошибка: Крепость должна быть в диапазоне от 0 до 100%
Ошибка: Плотность должна быть положительным числом

```

Рисунок 5 – Пример вывода данных

```

tests/test_liquid.py::TestLiquid::test_create_liquid PASSED [ 8%]
tests/test_liquid.py::TestLiquid::test_liquid_str PASSED [ 16%]
tests/test_liquid.py::TestLiquid::test_change_name PASSED [ 25%]
tests/test_liquid.py::TestLiquid::test_change_density_valid PASSED [ 33%]
tests/test_liquid.py::TestLiquid::test_change_density_invalid PASSED [ 41%]
tests/test_liquid.py::TestAlcohol::test_create_alcohol PASSED [ 50%]
tests/test_liquid.py::TestAlcohol::test_alcohol_inheritance PASSED [ 58%]
tests/test_liquid.py::TestAlcohol::test_alcohol_str PASSED [ 66%]
tests/test_liquid.py::TestAlcohol::test_change_strength_valid PASSED [ 75%]
tests/test_liquid.py::TestAlcohol::test_change_strength_invalid PASSED [ 83%]
tests/test_liquid.py::TestAlcohol::test_inherited_properties PASSED [ 91%]
tests/test_liquid.py::test_multiple_objects PASSED [100%]

===== 12 passed in 0.02s =====

```

Рисунок 6 – Результаты тестов

5. Реализовать абстрактный базовый класс, определив в нем абстрактные методы и свойства. Эти методы определяются в производных классах. В базовых классах должны быть объявлены абстрактные методы ввода/вывода, которые реализуются в производных классах.

3. Создать абстрактный базовый класс Body (тело) с абстрактными функциями вычисления площади поверхности и объема. Создать производные классы: Parallelepiped (параллелепипед) и Ball (шар) со своими функциями площади поверхности и объема.

Рисунок 7 – Вариант задания

Листинг программы:

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

import math
from abc import ABC, abstractmethod

class Body(ABC):

    @abstractmethod
    def surface_area(self) -> float:
        pass

    @abstractmethod
    def volume(self) -> float:
        pass

    @abstractmethod
    def __str__(self) -> str:
        pass

    @classmethod
    @abstractmethod
    def from_input(cls) -> "Body":
        pass

    def virtual_output(self) -> None:
        print(f"Площадь поверхности: {self.surface_area():.2f}")
        print(f"Объем: {self.volume():.2f}")

class Parallelepiped(Body):

    def __init__(self, length: float = 0, width: float = 0, height: float = 0):
        self.__length = length
        self.__width = width
        self.__height = height

    @property
    def length(self) -> float:
        return self.__length

    @property
    def width(self) -> float:
        return self.__width

    @property
    def height(self) -> float:
        return self.__height
```

```

@length.setter
def length(self, new_length: float) -> None:
    if new_length <= 0:
        raise ValueError("Длина должна быть положительной")
    self.__length = new_length

@height.setter
def height(self, new_height: float) -> None:
    if new_height <= 0:
        raise ValueError("Высота должна быть положительной")
    self.__height = new_height

@width.setter
def width(self, new_width: float) -> None:
    if new_width <= 0:
        raise ValueError("Ширина должна быть положительной")
    self.__width = new_width

def surface_area(self) -> float:
    return 2 * (
        self.__length * self.__width
        + self.__length * self.__height
        + self.__width * self.__height
    )

def volume(self) -> float:
    return self.__length * self.__width * self.__height

def __str__(self) -> str:
    return (
        f"Параллелепипед:\n"
        f"  Длина: {self.__length}\n"
        f"  Ширина: {self.__width}\n"
        f"  Высота: {self.__height}\n"
        f"  Площадь поверхности: {self.surface_area():.2f}\n"
        f"  Объем: {self.volume():.2f}"
    )

@classmethod
def from_input(cls) -> "Parallelepiped":
    try:
        length = float(input("Введите длину параллелепипеда: "))
        width = float(input("Введите ширину параллелепипеда: "))
        height = float(input("Введите высоту параллелепипеда: "))
        return cls(length, width, height)
    except ValueError:
        print("Ошибка: введите числовые значения!")
        return cls()

def virtual_output(self) -> None:
    print(f"Площадь поверхности параллелепипеда: {self.surface_area():.2f}")

```

```

        print(f"Объем параллелепипеда: {self.volume():.2f}")

class Ball(Body):

    def __init__(self, radius: float = 0):
        self.__radius = radius

    @property
    def radius(self) -> float:
        return self.__radius

    @radius.setter
    def radius(self, new_radius: float) -> None:
        if new_radius >= 0:
            self.__radius = new_radius
        else:
            print("Ошибка: радиус не может быть отрицательным!")

    def surface_area(self) -> float:
        return 4 * math.pi * self.__radius**2

    def volume(self) -> float:
        return (4 / 3) * math.pi * self.__radius**3

    def __str__(self) -> str:
        return (
            f"Шар:\n"
            f"  Радиус: {self.__radius}\n"
            f"  Площадь поверхности: {self.surface_area():.2f}\n"
            f"  Объем: {self.volume():.2f}"
        )

    @classmethod
    def from_input(cls) -> "Ball":
        try:
            radius = float(input("Введите радиус шара: "))

            if radius < 0:
                print(
                    "Ошибка: радиус не может быть отрицательным! Установлен
радиус 0."
                )
                return cls(0)

            return cls(radius)
        except ValueError:
            print("Ошибка: введите числовое значение! Установлен радиус 0.")
            return cls(0)

    def virtual_output(self) -> None:

```

```
print(" Шар:")
print(f" Радиус: {self.__radius}")
print(f" Площадь поверхности: {self.surface_area():.2f}")
print(f" Объем: {self.volume():.2f}")
```

Пример вызова программы:

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

from body_package.body import Ball, Parallelepiped

if __name__ == "__main__":
    print("=" * 60)
    print("ДЕМОНСТРАЦИЯ ВСЕХ ПЕРЕОПРЕДЕЛЕННЫХ АБСТРАКТНЫХ МЕТОДОВ")
    print("=" * 60)

    print("\n1. СОЗДАНИЕ ОБЪЕКТОВ НАПРЯМУЮ:")
    print("-" * 40)

    print("\n--- Параллелепипед (прямое создание) ---")
    p1 = Parallelepiped(5, 3, 2)
    print(f"Создан: {p1}")

    print("\n--- Шар (прямое создание) ---")
    b1 = Ball(4)
    print(f"Создан: {b1}")

    print("\n\n2. СОЗДАНИЕ ОБЪЕКТОВ ЧЕРЕЗ from_input():")
    print("-" * 40)

    print("\n--- Создание параллелепипеда через ввод ---")

    import io
    import sys

    test_inputs = ["5", "3", "2"]
    original_stdin = sys.stdin
    sys.stdin = io.StringIO("\n".join(test_inputs))

    p2 = Parallelepiped.from_input()
    print(f"Создан через from_input(): {p2}")

    print("\n--- Создание шара через ввод ---")
    test_inputs = ["4"]
    sys.stdin = io.StringIO("\n".join(test_inputs))

    b2 = Ball.from_input()
```

```

print(f"Создан через from_input(): {b2}")

sys.stdin = original_stdin

print("\n\n3. ВЫЗОВ АБСТРАКТНЫХ МЕТОДОВ:")
print("-" * 40)

print("\n--- Методы параллелепипеда ---")
print(f"surface_area(): {p1.surface_area():.2f}")
print(f"volume(): {p1.volume():.2f}")
print(f"__str__(): {p1}")

print("\n--- Методы шара ---")
print(f"surface_area(): {b1.surface_area():.2f}")
print(f"volume(): {b1.volume():.2f}")
print(f"__str__(): {b1}")

print("\n\n4. ВИРТУАЛЬНЫЕ МЕТОДЫ:")
print("-" * 40)

print("\n--- virtual_output() параллелепипеда ---")
p1.virtual_output()

print("\n--- virtual_output() шара ---")
b1.virtual_output()

print("\n\n5. РАБОТА ЧЕРЕЗ БАЗОВЫЙ КЛАСС Body:")
print("-" * 40)

bodies = [p1, b1]

for i, body in enumerate(bodies, 1):
    print(f"\n--- Тело #{i} ---")
    print(f"Тип: {type(body).__name__}")
    print(f"Площадь поверхности: {body.surface_area():.2f}")
    print(f"Объем: {body.volume():.2f}")
    body.virtual_output()

print("\n\n6. РАБОТА СО СВОЙСТВАМИ:")
print("-" * 40)

print("\n--- Свойства параллелепипеда ---")
print(f"Длина: {p1.length}")
print(f"Ширина: {p1.width}")
print(f"Высота: {p1.height}")

p1.length = 6
p1.width = 4
p1.height = 3
print(f"После изменения: Длина={p1.length}, Ширина={p1.width},
Высота={p1.height}")

```

```

print(f"Новая площадь поверхности: {p1.surface_area():.2f}")
print(f"Новый объем: {p1.volume():.2f}")

print("\n--- Свойства шара ---")
print(f"Радиус: {b1.radius}")

b1.radius = 5
print(f"После изменения: Радиус={b1.radius}")
print(f"Новая площадь поверхности: {b1.surface_area():.2f}")
print(f"Новый объем: {b1.volume():.2f}")

print("\n\n7. ОБРАБОТКА ОШИБОК:")
print("-" * 40)

print("\n--- Попытка установки отрицательных значений ---")

try:
    p1.length = -1
except ValueError as e:
    print(f"Ошибка параллелепипеда: {e}")

print("Попытка установки отрицательного радиуса:")
b1.radius = -1

print("\n" + "=" * 60)
print("ДЕМОНСТРАЦИЯ ЗАВЕРШЕНА")
print("=" * 60)

```

```

1. СОЗДАНИЕ ОБЪЕКТОВ НАПРЯМУЮ:
-----

--- Параллелепипед (прямое создание) ---
Создан: Параллелепипед:
Длина: 5
Ширина: 3
Высота: 2
Площадь поверхности: 62.00
Объем: 30.00

--- Шар (прямое создание) ---
Создан: Шар:
Радиус: 4
Площадь поверхности: 201.06
Объем: 268.08

2. СОЗДАНИЕ ОБЪЕКТОВ ЧЕРЕЗ from_input():
-----

--- Создание параллелепипеда через ввод ---
Введите длину параллелепипеда: Введите ширину параллелепипеда: Введите высоту параллелепипеда: Создан через from_input(): Параллелепипед:
Длина: 5.0
Ширина: 3.0
Высота: 2.0
Площадь поверхности: 62.00
Объем: 30.00

```

Рисунок 8 – Пример вывода данных

```

--- Создание шара через ввод ---
Введите радиус шара: Создан через from_input(): Шар:
Радиус: 4.0
Площадь поверхности: 201.06
Объем: 268.08

3. ВЫВОД АБСТРАКТНЫХ МЕТОДОВ:
-----

--- Методы параллелепипеда ---
surface_area(): 62.00
volume(): 30.00
__str__(): Параллелепипед:
Длина: 5
Ширина: 3
Высота: 2
Площадь поверхности: 62.00
Объем: 30.00

--- Методы шара ---
surface_area(): 201.06
volume(): 268.08
__str__(): Шар:
Радиус: 4
Площадь поверхности: 201.06
Объем: 268.08

```

Рисунок 9 – Пример вывода данных

```

4. ВИРТУАЛЬНЫЕ МЕТОДЫ:
-----

--- virtual_output() параллелепипеда ---
Площадь поверхности параллелепипеда: 62.00
Объем параллелепипеда: 30.00

--- virtual_output() шара ---
Шар:
Радиус: 4
Площадь поверхности: 201.06
Объем: 268.08

5. РАБОТА ЧЕРЕЗ БАЗОВЫЙ КЛАСС Body:
-----

--- Тело #1 ---
Тип: Parallelepiped
Площадь поверхности: 62.00
Объем: 30.00
Площадь поверхности параллелепипеда: 62.00
Объем параллелепипеда: 30.00

```

Рисунок 10 – Пример вывода данных

```

--- Тело #2 ---
Тип: Ball
Площадь поверхности: 201.06
Объем: 268.08
Шар:
Радиус: 4
Площадь поверхности: 201.06
Объем: 268.08

6. РАБОТА СО СВОЙСТВАМИ:
-----

--- Свойства параллелепипеда ---
Длина: 5
Ширина: 3
Высота: 2
После изменения: Длина=6, Ширина=4, Высота=3
Новая площадь поверхности: 108.00
Новый объем: 72.00

```

Рисунок 11 – Пример вывода данных

```

6. РАБОТА СО СВОЙСТВАМИ:
-----

--- Свойства параллелепипеда ---
Длина: 5
Ширина: 3
Высота: 2
После изменения: Длина=6, Ширина=4, Высота=3
Новая площадь поверхности: 108.00
Новый объем: 72.00

--- Свойства шара ---
Радиус: 4
После изменения: Радиус=5
Новая площадь поверхности: 314.16
Новый объем: 523.60

7. ОБРАБОТКА ОШИБОК:
-----

--- Попытка установки отрицательных значений ---
Ошибка параллелепипеда: Длина должна быть положительной
Попытка установки отрицательного радиуса:
Ошибка: радиус не может быть отрицательным!

```

Рисунок 12 – Пример вывода данных

```

tests/test_body.py::TestParallelepiped::test_create_parallelepiped PASSED [ 5%]
tests/test_body.py::TestParallelepiped::test_surface_area PASSED [ 11%]
tests/test_body.py::TestParallelepiped::test_volume PASSED [ 16%]
tests/test_body.py::TestParallelepiped::test_setters_valid PASSED [ 22%]
tests/test_body.py::TestParallelepiped::test_setters_invalid PASSED [ 27%]
tests/test_body.py::TestParallelepiped::test_str_representation PASSED [ 33%]
tests/test_body.py::TestBall::test_create_ball PASSED [ 38%]
tests/test_body.py::TestBall::test_surface_area PASSED [ 44%]
tests/test_body.py::TestBall::test_volume PASSED [ 50%]
tests/test_body.py::TestBall::test_radius_setter_valid PASSED [ 55%]
tests/test_body.py::TestBall::test_radius_setter_invalid PASSED [ 61%]
tests/test_body.py::TestBall::test_str_representation PASSED [ 66%]
tests/test_body.py::TestInheritanceAndAbstract::test_inheritance PASSED [ 72%]
tests/test_body.py::TestInheritanceAndAbstract::test_abstract_methods_implementation PASSED [ 77%]
tests/test_body.py::TestVirtualMethods::test_virtual_output PASSED [ 83%]
tests/test_body.py::test_multiple_objects PASSED [ 88%]
tests/test_body.py::test_edge_cases PASSED [ 94%]
tests/test_body.py::test_calculation_accuracy PASSED [100%]

```

Рисунок 13 – Результат тестирования

**Вывод:** в ходе работы были приобретены навыки по созданию иерархии классов при написании программ с помощью языка программирования Python.