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:
М.А.Шумило
студент группы
ПО-13
Проверил:
А.Д.Кулик
27.02.2026

Брест 2026

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

Задание 1.

3) 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:

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

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

    @a.setter
    def a(self, value):
        if value <= 0:
            raise ValueError("Сторона должна быть положительным числом.")
        self._a = value

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

    @b.setter
    def b(self, value):
        if value <= 0:
            raise ValueError("Сторона должна быть положительным числом.")
        self._b = value

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

    @c.setter
    def c(self, value):
        if value <= 0:
            raise ValueError("Сторона должна быть положительным числом.")
        self._c = value

    # Логический метод существования прямоугольного треугольника
    def exists(self) -> bool:
        sides = sorted([self.a, self.b, self.c])
        return abs(sides[0]**2 + sides[1]**2 - sides[2]**2) < 1e-9

    # Площадь
    def area(self) -> float:
        if not self.exists():
            raise ValueError("Треугольник не существует.")
        sides = sorted([self.a, self.b, self.c])
```

```
        return (sides[0] * sides[1]) / 2

    # Периметр
    def perimeter(self) -> float:
        if not self.exists():
            raise ValueError("Треугольник не существует.")
        return self.a + self.b + self.c

    # Переопределение __str__
    def __str__(self):
        return (f"Прямоугольный треугольник со сторонами: "
                f"a={self.a}, b={self.b}, c={self.c}, "
                f"существует={self.exists()}")

    # Переопределение __eq__
    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])

t1 = RightTriangle(3, 4, 5)
t2 = RightTriangle(5, 3, 4)
t3 = RightTriangle(2, 3, 4)

print("t1 существует?", t1.exists())
print("t3 существует?", t3.exists())

print("Площадь t1:", t1.area())
print("Периметр t1:", t1.perimeter())

print("t1 == t2 ?", t1 == t2)
print("t1 == t3 ?", t1 == t3)

print(t1)
print(t3)
```

Рисунки с результатами работы программы

```
PS C:\Users\user\Desktop\СПП\spp_po13\reports\Shumilo\2\src> python 1.py
t1 существует? True
t3 существует? False
Площадь t1: 6.0
Периметр t1: 12
t1 == t2 ? True
t1 == t3 ? False
Прямоугольный треугольник со сторонами: a=3, b=4, c=5, существует=True
Прямоугольный треугольник со сторонами: a=2, b=3, c=4, существует=False
```

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

2) Система Платежи. Клиент имеет Счет в банке и Кредитную Карту (КК). Клиент может оплатить Заказ, сделать платеж на другой Счет, заблокировать КК и аннулировать Счет. Администратор может заблокировать КК за превышение кредита.

Выполнение:

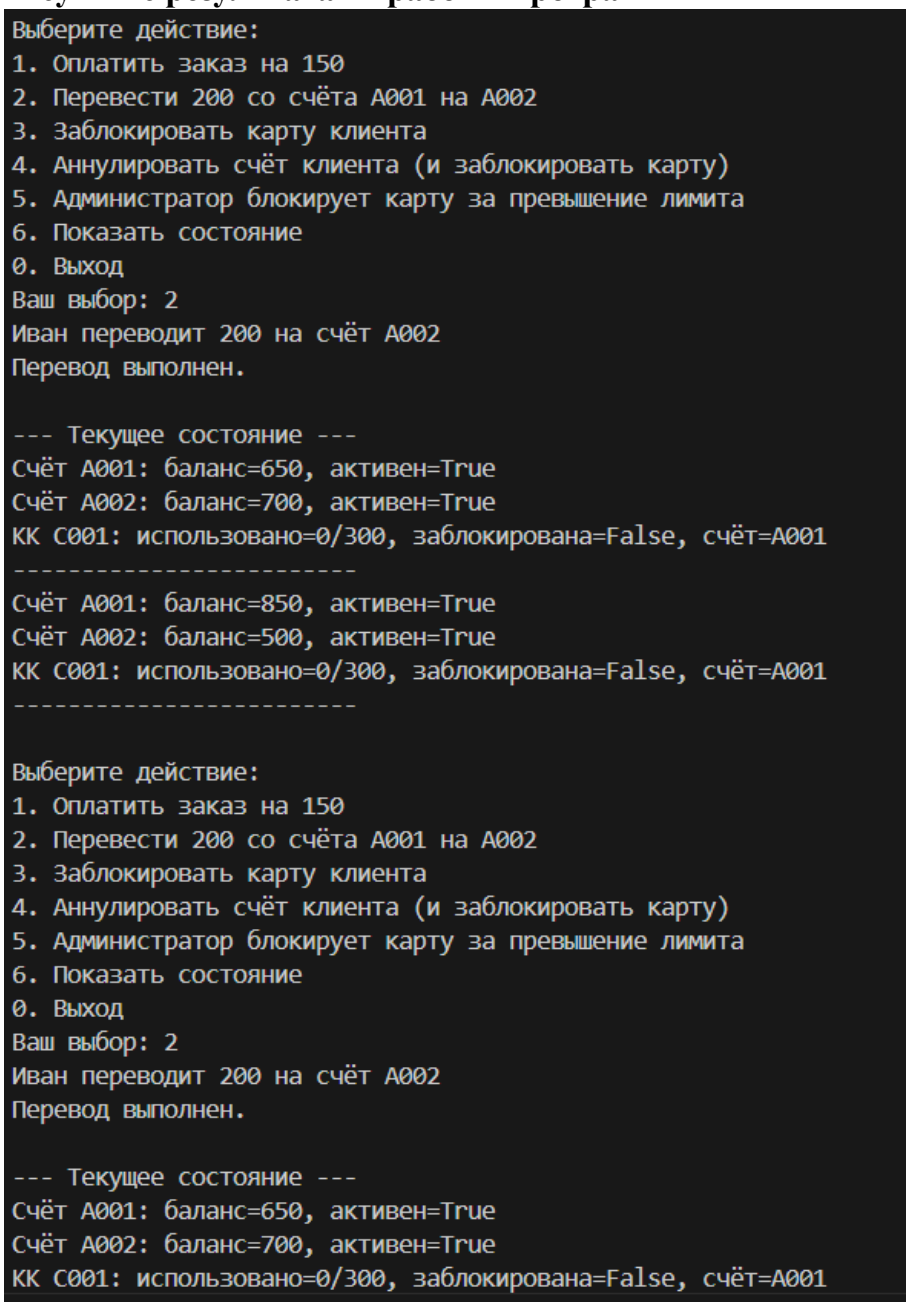
Код программы:

```
from abc import ABC, abstractmethod

# Интерфейс платежной системы
class PaymentSystem(ABC):
    @abstractmethod
    def pay_order(self, order):
        pass

    @abstractmethod
    def transfer(self, target_account, amount):
        pass

#  Базовый класс Person
class Person:
    def __init__(self, name):
        self.name = name

# Класс банковского счёта
class Account:
    def __init__(self, number, balance=0):
        self.number = number
        self.balance = balance
        self.active = True

    def deposit(self, amount):
        self.balance += amount

    def withdraw(self, amount):
        if not self.active:
            raise ValueError("Счёт аннулирован.")
        if amount > self.balance:
            raise ValueError("Недостаточно средств.")
        self.balance -= amount

    def close(self):
        self.active = False

    def __str__(self):
        return f"Счёт {self.number}: баланс={self.balance},
активен={self.active}"

# Кредитная карта, привязанная к счёту
class CreditCard:
    def __init__(self, number, credit_limit, account: Account):
        self.number = number
        self.credit_limit = credit_limit
        self.used = 0
```

```
        self.blocked = False
        self.account = account

    def spend(self, amount):

        if self.blocked:
            raise ValueError("Карта заблокирована.")

        if self.used + amount > self.credit_limit+self.account.balance:
            raise ValueError("Превышение кредитного лимита.")

        if self.account.balance >= amount:
            self.account.withdraw(amount)
        else:
            self.used += amount


    def block(self):
        self.blocked = True

    def __str__(self):
        return (f"КК {self.number}:
использовано={self.used}/{self.credit_limit}, "
                f"заблокирована={self.blocked}, счёт={self.account.number}")

class Order:
    def __init__(self, order_id, amount):
        self.order_id = order_id
        self.amount = amount

    def __str__(self):
        return f"Заказ {self.order_id}: сумма={self.amount}"

class Client(Person, PaymentSystem):
    def __init__(self, name, account: Account, card: CreditCard):
        super().__init__(name)
        self.account = account
        self.card = card

    def pay_order(self, order: Order):
        print(f"{self.name} оплачивает заказ {order.order_id}")
        self.card.spend(order.amount)

    def transfer(self, target_account: Account, amount):
        print(f"{self.name} переводит {amount} на счёт {target_account.number}")
        self.account.withdraw(amount)
        target_account.deposit(amount)

    def block_card(self):
        print(f"{self.name} блокирует свою карту.")
        self.card.block()

    def close_account(self):
        print(f"{self.name} аннулирует свой счёт.")
        self.account.close()
        self.card.block()

#  Администратор
class Administrator(Person):
    def block_card_for_limit(self, card: CreditCard):
        print(f"Администратор {self.name} блокирует карту {card.number} за
превышение лимита.")
        card.block()
```

```
acc1 = Account("A001", 1000)
acc2 = Account("A002", 500)

card1 = CreditCard("C001", 300, acc1)

client = Client("Иван", acc1, card1)
admin = Administrator("Мария")

def print_state():
    print("\n--- Текущее состояние ---")
    print(acc1)
    print(acc2)
    print(card1)
    print("--------------------------")


print_state()

while True:
    print("\nВыберите действие:")
    print("1. Оплатить заказ на 150")
    print("2. Перевести 200 со счёта A001 на A002")
    print("3. Заблокировать карту клиента")
    print("4. Аннулировать счёт клиента (и заблокировать карту)")
    print("5. Администратор блокирует карту за превышение лимита")
    print("6. Показать состояние")
    print("0. Выход")

    choice = input("Ваш выбор: ").strip()

    try:
        if choice == "1":
            order1 = Order("O100", 150)
            client.pay_order(order1)
            print("Заказ оплачен.")
        elif choice == "2":
            client.transfer(acc2, 200)
            print("Перевод выполнен.")
        elif choice == "3":
            client.block_card()
            print("Карта заблокирована.")
        elif choice == "4":
            client.close_account()
            print("Счёт аннулирован, карта заблокирована.")
        elif choice == "5":
            admin.block_card_for_limit(card1)
            print("Карта заблокирована администратором.")
        elif choice == "6":
            print_state()
            continue
        elif choice == "0":
            print("Выход.")
            break
        else:
            print("Неверный пункт.")
            continue
    except Exception as e:
        print(f"Ошибка: {e}")

    print_state()
```

Рисунки с результатами работы программы



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