МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
«БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ» ФАКУЛЬТЕТ ЭЛЕКТРОННО-ИНФОРМАЦИОННЫХ СИСТЕМ

Кафедра интеллектуальных информационных технологий

Отчёт по лабораторной работе №2

Специальность ПО11

Выполнил Н.
А. Антонюк
студент группы ПО11

Проверил
А. А. Крощенко ст.
преп. кафедры ИИТ,
05.04.2025 г.

Брест 2025

Цель работы: закрепить базовые знания языка программирования Python при решении практических задач

Задание 1. 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.

Выполнение:

Код программы:

```
import math

class Triangle:     def __init__(self,
a=1.0, b=1.0, c=1.0):
        """
        Конструктор класса Triangle с проверкой на равносторонность.
        :param a: длина первой стороны
    :param b: длина второй стороны
        :param c: длина третьей стороны
        """
        self.set_sides(a, b, c)

    def set_sides(self, a, b, c):
"""
        Устанавливает стороны треугольника с проверкой на равносторонность.
        :param a: длина первой стороны
    :param b: длина второй стороны
        :param c: длина третьей стороны
        """
        if not math.isclose(a, b, rel_tol=1e-9) or not math.isclose(a, c, rel_tol=1e-9):
            raise ValueError("Треугольник должен быть равносторонним (все стороны равны)")

        if a <= 0 or b <= 0 or c <= 0:
            raise ValueError("Длины сторон должны быть положительными числами")

        self._a = a
    self._b = b
        self._c = c

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

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

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

    def perimeter(self):
        """Вычисление периметра треугольника"""
return self._a + self._b + self._c

    def area(self):
        """Вычисление площади треугольника"""
        p = self.perimeter() / 2         return math.sqrt(p * (p - self._a)
* (p - self._b) * (p - self._c))

    def is_valid(self):
"""
        Проверка, соответствует ли треугольник неравенству треугольника
        Для равностороннего треугольника всегда True если стороны положительные
        """
        return (self._a + self._b > self._c and
self._a + self._c > self._b and                  self._b
+ self._c > self._a)

    def __str__(self):
        """Строковое представление объекта"""
        return f"Равносторонний треугольник со сторонами: a={self._a}, b={self._b}, c={self._c}"

    def __eq__(self, other):
        """
        Сравнение двух треугольников на равенство сторон
        :param other: другой объект Triangle
        :return: True, если треугольники равны (по сторонам), иначе False
        """         if not isinstance(other,
Triangle):
            return False         return
(math.isclose(self._a, other._a) and
math.isclose(self._b, other._b) and
math.isclose(self._c, other._c))


def validate_input(prompt):
    """Функция для валидации ввода чисел"""
while True:         try:
```

```
            value = float(input(prompt))
    if value <= 0:
                print("Длина стороны должна быть положительной. Попробуйте снова.")
    continue            return value        except ValueError:
            print("Пожалуйста, введите корректное число.")


    def main():
        print("Введите длины сторон равностороннего треугольника:")
    while True:         try:
            a = validate_input("Введите длину стороны a: ")
    b = validate_input("Введите длину стороны b: ")
    c = validate_input("Введите длину стороны c: ")
    triangle = Triangle(a, b, c)
            break       except ValueError as e:             print(f"Ошибка: {e}")
    print("Пожалуйста, введите равные длины для всех трех сторон.\n")

        print("\nИнформация о треугольнике:")
    print(triangle)
        print(f"Периметр: {triangle.perimeter()}")     print(f"Площадь:
    {triangle.area():.2f}")     print(f"Треугольник существует: {'да' if
    triangle.is_valid() else 'нет'}")

        # Создадим второй треугольник для сравнения
    print("\nСоздадим второй треугольник для сравнения:")
    while True:         try:
            a2 = validate_input("Введите длину стороны a для второго треугольника: ")
    b2 = validate_input("Введите длину стороны b для второго треугольника: ")
    c2 = validate_input("Введите длину стороны c для второго треугольника: ")

            triangle2 = Triangle(a2, b2, c2)            break       except ValueError as
    e:          print(f"Ошибка: {e}")           print("Пожалуйста, введите равные
    длины для всех трех сторон.\n")

        print(f"\nСравнение треугольников: {'равны' if triangle == triangle2 else 'не равны'}")


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

Спецификация ввода:

<Длина 1ой стороны 1ого равностороннего треугольника>…< Длина 3ей стороны 1ого равностороннего треугольника> <Длина 1ой стороны 2ого равностороннего треугольника>…< Длина 3ей стороны 2ого равностороннего треугольника>

Пример:

5 5 5 3 3 3

Пример ввода:

Введите длины сторон равностороннего треугольника:
Введите длину стороны a: 5
Введите длину стороны b: 5 Введите
длину стороны c: 5

Введите длину стороны a для второго треугольника: 3
Введите длину стороны b для второго треугольника: 3 Введите
длину стороны c для второго треугольника: 3
Рисунок с результатом работы программы:

```
PS C:\Users\Nikita> & C:/Users/Nikita/AppData/Local/Programs/Python/Python311-32/python.exe c:/Users/Nikita/Desktop/SPP_Lab2_Task1.py
Введите длины сторон равностороннего треугольника:
Введите длину стороны a: 5
Введите длину стороны b: 5
Введите длину стороны c: 5

Информация о треугольнике:
Равносторонний треугольник со сторонами: a=5.0, b=5.0, c=5.0
Периметр: 15.0
Площадь: 10.83
Треугольник существует: да

Создадим второй треугольник для сравнения:
Введите длину стороны a для второго треугольника: 3
Введите длину стороны b для второго треугольника: 3
Введите длину стороны c для второго треугольника: 3

Сравнение треугольников: не равны
PS C:\Users\Nikita>
```

Задание 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: import
sys

```
class BankAccount:     def __init__(self,
account_number, balance=0.0):
        self._account_number = account_number
        self._balance = balance
        self._is_active = True

    @property     def
account_number(self):
        return self._account_number

    @property     def
balance(self):
```

```
        return self._balance

    @property     def
is_active(self):
        return self._is_active

    def deposit(self, amount):
if amount <= 0:
            raise ValueError("Сумма должна быть положительной")
if not self._is_active:
            raise ValueError("Счет не активен")
self._balance += amount

    def withdraw(self, amount):
        if amount <= 0:
                raise ValueError("Сумма должна быть
    положительной")     if not self._is_active:
            raise ValueError("Счет не активен")
        if amount > self._balance:          raise
    ValueError("Недостаточно средств")
    self._balance -= amount

    def close_account(self):
self._is_active = False
        self._balance = 0

    def __str__(self):
        status = "активен" if self._is_active else "неактивен"          return f"Счет
№{self._account_number}, баланс: {self._balance:.2f}, статус: {status}"

class CreditCard:     def __init__(self, card_number,
credit_limit=10000.0):
        self._card_number = card_number
self._credit_limit = credit_limit          self._current_credit
= 0.0
        self._is_blocked = False

    @property     def
card_number(self):
        return self._card_number

    @property     def
credit_limit(self):
        return self._credit_limit

    @property     def
current_credit(self):
```

```
        return self._current_credit

    @property     def
is_blocked(self):
        return self._is_blocked

    def make_payment(self, amount):         if amount <= 0:
raise ValueError("Сумма должна быть положительной")         if
self._is_blocked:
            raise ValueError("Карта заблокирована")         if
amount > (self._credit_limit - self._current_credit):
raise ValueError("Превышен кредитный лимит")
self._current_credit += amount

   def repay_credit(self, amount):
if amount <= 0:
            raise ValueError("Сумма должна быть положительной")
        if amount > self._current_credit:
    amount = self._current_credit
    self._current_credit -= amount def
    block_card(self):
        self._is_blocked = True

    def unblock_card(self):        if
    self._current_credit     <=     0:
    self._is_blocked = False

    def __str__(self):
        status = "заблокирована" if self._is_blocked else "активна"         return
f"Карта №{self._card_number}, лимит: {self._credit_limit:.2f}, кредит:
{self._current_credit:.2f}, статус: {status}"

class Client:     def __init__(self, name, account,
credit_card):
        self._name = name
self._account = account
        self._credit_card = credit_card

    @property
def name(self):
        return self._name

    @property     def
account(self):
        return self._account
```

```
    @property     def
credit_card(self):
        return self._credit_card

    def pay_order(self, merchant_name, amount, use_credit=False):
if use_credit:
            try:
                self._credit_card.make_payment(amount)
                print(f"Оплачено {amount:.2f} с кредитной карты для {merchant_name}")
except ValueError as e:
                print(f"Ошибка оплаты: {e}")
else:               try:
                self._account.withdraw(amount)
                print(f"Оплачено {amount:.2f} со счета для {merchant_name}")
except ValueError as e:
                print(f"Ошибка оплаты: {e}")

   def transfer_to_account(self, target_account_number, amount):
try:
            target_account = None

            # В реальной системе здесь был бы поиск счета в базе данных
            # Для демонстрации просто создаем новый счет
  target_account = BankAccount(target_account_number)

            self._account.withdraw(amount)
            target_account.deposit(amount)
             print(f"Переведено {amount:.2f} на счет {target_account_number}")
        except ValueError as e:
          print(f"Ошибка перевода: {e}")

    def block_credit_card(self):
        self._credit_card.block_card()      print("Кредитная
   карта заблокирована клиентом")

    def close_account(self):
        self._account.close_account()
        print("Счет аннулирован")

    def __str__(self):
        return f"Клиент: {self._name}\n{self._account}\n{self._credit_card}"

class     Administrator:                    @staticmethod                    def
block_card_for_excess(client):        if client.credit_card.current_credit >
client.credit_card.credit_limit:
            client.credit_card.block_card()
```

```
            print("\n[АДМИНИСТРАТОР] Карта заблокирована за превышение кредитного лимита!")
return True
        return False

def input_float(prompt):
while True:         try:
            value = float(input(prompt))
if value <= 0:
                print("Значение должно быть положительным!")
continue             return value         except ValueError:
            print("Пожалуйста, введите число!")

def input_int(prompt):
while True:         try:
            return int(input(prompt))
except ValueError:
            print("Пожалуйста, введите целое число!")

def create_client():
    print("\n=== СОЗДАНИЕ НОВОГО КЛИЕНТА ===")    name =
 input("Введите имя клиента: ")    account_num = input("Введите
 номер счета (цифры): ")    balance = input_float("Введите
 начальный баланс счета: ")    card_num = input("Введите номер
 кредитной карты (16 цифр): ")    credit_limit =
 input_float("Введите кредитный лимит: ")

    account = BankAccount(account_num, balance)
    card = CreditCard(card_num, credit_limit)
    return Client(name, account, card)
def client_operations(client):
while True:
        print("\n=== ОПЕРАЦИИ С КЛИЕНТОМ ===")
        print("1. Оплатить заказ")         print("2.
Перевести на другой счет")         print("3.
Блокировать кредитную карту")         print("4.
Аннулировать счет")         print("5. Показать
информацию")         print("6. Вернуться в
меню")         print("7. Выход")

        choice = input("Выберите действие (1-7): ")

        if choice == '1':
            merchant = input("Введите название магазина/услуги: ")
amount = input_float("Введите сумму оплаты: ")
            use_credit = input("Использовать кредитную карту? (y/n): ").lower() == 'y'
client.pay_order(merchant, amount, use_credit)

        elif choice == '2':
```

```
            target = input("Введите номер целевого счета: ")
amount = input_float("Введите сумму перевода: ")
client.transfer_to_account(target, amount)

        elif choice == '3':
            client.block_credit_card()

        elif choice == '4':
            confirm = input("Вы уверены? Счет будет аннулирован! (y/n): ").lower() == 'y'
if confirm:
                client.close_account()
                return  # Возврат в меню после аннулирования

        elif choice == '5':
            print("\n=== ИНФОРМАЦИЯ О КЛИЕНТЕ ===")
print(client)

        elif choice == '6':
return

        elif choice == '7':
            print("\nВыход из системы...")
sys.exit()

def main():
  print("\n=== БАНКОВСКАЯ СИСТЕМА 'ПЛАТЕЖИ' ===")
clients = []    admin = Administrator()

  while True:
      print("\n=== ГЛАВНОЕ МЕНЮ ===")
  print("1. Создать нового клиента")     print("2.
  Выбрать клиента")
       print("3. Административные функции")
    print("4. Выход")

    choice = input("Выберите действие (1-4): ")

       if choice == '1':
            clients.append(create_client())
            print("\nКлиент успешно создан!")
print(clients[-1])

        elif choice == '2':
if not clients:
                print("Нет зарегистрированных клиентов!")
continue
```

```
        print("\nСписок клиентов:")
for i, client in enumerate(clients, 1):
            print(f"{i}. {client.name}")

try:
            selected = int(input("Выберите клиента (номер): ")) - 1
if 0 <= selected < len(clients):
                client_operations(clients[selected])

                # Проверка на превышение лимита после каждой операции
admin.block_card_for_excess(clients[selected])              else:
                print("Неверный номер клиента!")
except ValueError:
            print("Пожалуйста, введите число!")

      elif choice == '3':
if not clients:
            print("Нет зарегистрированных клиентов!")
continue

        print("\n=== АДМИНИСТРАТИВНЫЕ ФУНКЦИИ ===")
for i, client in enumerate(clients, 1):
            print(f"{i}. {client.name}")

try:
            selected = int(input("Выберите клиента для проверки (номер): ")) - 1
if 0 <= selected < len(clients):                  if
admin.block_card_for_excess(clients[selected]):
                 print(f"Карта клиента {clients[selected].name} была заблокирована!")
else:
                print(f"Кредитный лимит клиента {clients[selected].name} не превышен")
else:
              print("Неверный номер клиента!")
except ValueError:
            print("Пожалуйста, введите число!")
         elif choice ==
  '4':
        print("\nВыход из системы...")
        sys.exit()

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

Пример:

1

Лесько Максим Игоревич

111122223334444
1000
12345678
150

Пример ввода:

Введите имя клиента: Лесько Максим Игоревич
Введите номер счета: 1112223334444
Введите начальный баланс: 1000
Введите номер кредитной карты: 12345678
Введите кредитный лимит: 150

Рисунок с результатом работы программы:

```
=== БАНКОВСКАЯ СИСТЕМА ===
1. Создать нового клиента
2. Операции с клиентом
3. Административные функции
4. Показать всех клиентов
5. Выход
Выберите действие (1-5): 1

=== СОЗДАНИЕ НОВОГО КЛИЕНТА ===
Введите имя клиента: Лесько Максим Игоревич
Введите номер счета: 1112223334444
Введите начальный баланс: 1000
Введите номер кредитной карты: 12345678
Введите кредитный лимит: 150

Клиент успешно создан!
Клиент: Лесько Максим Игоревич
Счет №1112223334444, баланс: 1000.00, статус: активен
Карта №12345678, лимит: 150.00, кредит: 0.00, статус: активна

=== БАНКОВСКАЯ СИСТЕМА ===
1. Создать нового клиента
2. Операции с клиентом
3. Административные функции
4. Показать всех клиентов
5. Выход
Выберите действие (1-5): 2
```

```
=== БАНКОВСКАЯ СИСТЕМА ===
1. Создать нового клиента
2. Операции с клиентом
3. Административные функции
4. Показать всех клиентов
5. Выход
Выберите действие (1-5): 2

=== ВЫБОР КЛИЕНТА ===
1. Лесько Максим Игоревич
Выберите клиента (номер): 1

=== ИНФОРМАЦИЯ О КЛИЕНТЕ ===
Клиент: Лесько Максим Игоревич
Счет №1112223334444, баланс: 1000.00, статус: активен
Карта №12345678, лимит: 150.00, кредит: 0.00, статус: активна

=== ОПЕРАЦИИ С КЛИЕНТОМ ===
1. Оплатить заказ
2. Сделать перевод
3. Блокировать карту
4. Аннулировать счет
5. Вернуться в главное меню
Выберите действие (1-5): 1
Введите сумму оплаты: 500
Использовать кредитную карту? (y/n): n
Оплачено 500.00 со счета
```