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
И. А. Гурин
студент группы ПО11

Проверил
А. А. Крощенко
ст. преп. кафедры ИИТ,
01.03.2025 г.

Брест 2025

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__

Множество целых чисел переменной мощности – Предусмотреть возможность пересечения двух множеств, вывода на консоль элементов множества, а также метод, определяющий, принадлежит ли указанное значение множеству. Класс должен содержать методы, позволяющие добавлять и удалять элемент в/из множества. Конструктор должен позволять создавать объекты с начальной инициализацией. Реализацию множества осуществить на базе списка. Переопределить метод __eq__, выполняющий сравнение объектов данного типа.

Выполнение:

Код программы:

```
from task_1_clases import IntegerSet, Choice, choiceValues

def show_elements_action() -> None:
    while True:
        try:
            user_input: int = int(input("Enter the set number: "))
            if user_input == 1:
                print(set_1)
            elif user_input == 2:
                print(set_2)
            else:
                raise ValueError()
            break
        except ValueError:
            print("Invalid input. Please try again.")

def show_intersections_action() -> None:
    print(set_1.intersection(set_2))

def add_element_action() -> None:
    while True:
        try:
            user_input: int = int(input("Enter the set number: "))
            value: int = int(input("Enter the element value: "))
            if user_input == 1:
                set_1.add(value)
            elif user_input == 2:
                set_2.add(value)
            else:
                raise ValueError()
            break
        except ValueError:
            print("Invalid input. Please try again.")

def remove_element_action() -> None:
    while True:
        try:
            user_input: int = int(input("Enter the set number: "))
            value: int = int(input("Enter the element value: "))
            if user_input == 1:
                set_1.remove(value)
            elif user_input == 2:
                set_2.remove(value)
            else:
                raise ValueError()
            break
        except ValueError:
            print("Invalid input. Please try again.")

def check_element_action() -> None:
```

```
        while True:
            try:
                user_input: int = int(input("Enter the set number: "))
                value: int = int(input("Enter the element value: "))
                if user_input == 1:
                    print(set_1.contains(value))
                elif user_input == 2:
                    print(set_2.contains(value))
                else:
                    raise ValueError()
                break
            except ValueError:
                print("Invalid input. Please try again.")

    actions =  {
        Choice.SHOW_ELEMENTS.value: show_elements_action,
        Choice.SHOW_INTERSECTIONS.value: show_intersections_action,
        Choice.ADD_ELEMENT.value: add_element_action,
        Choice.REMOVE_ELEMENT.value: remove_element_action,
        Choice.CHECK_ELEMENT.value: check_element_action
    }

    def get_user_input() -> IntegerSet:
        while True:
            try:
                user_input: str = input("Enter the elements of the set: ")
                user_list = list(map(int, user_input.split()))
                return IntegerSet(user_list)
            except ValueError:
                print("Invalid input. Please try again.")

    def menu() -> None:
        while True:
            print("Menu:")
            for choice in Choice:
                print(f"{choice.value}. {choiceValues[choice.value]}", end="   ")
            print("")
            user_choice: int = int(input("Enter your choice: "))
            if user_choice == Choice.EXIT.value:
                break
            actions[user_choice]()

    if __name__ == "__main__":
        set_1 = get_user_input()
        set_2 = get_user_input()
        menu()
```

task_1_clases.py

```
    import enum

    class IntegerSet:
        def __init__(self, initial_elements=None):
            if initial_elements is None:
                self.elements = []
            else:
                self.elements = list(set([x for x in initial_elements if isinstance(x, int)]))

        def size(self):
            return len(self.elements)

        def add(self, element: int) -> None:
            if isinstance(element, int) and element not in self.elements:
                self.elements.append(element)

        def remove(self, element: int) -> None:
            if isinstance(element, int) and element in self.elements:
                self.elements.remove(element)

        def contains(self, element: int) -> bool:
            if isinstance(element, int) and element in self.elements:
```

```
                return True
            else:
                return False

        def intersection(self, other_set: "IntegerSet") -> "IntegerSet":
            intersection_set = IntegerSet()
            for element in self.elements:
                if element in other_set.elements:
                    intersection_set.add(element)
            return intersection_set

        def __str__(self):
            return "{" + ", ".join(map(str, self.elements)) + "}"

        def __eq__(self, other):
            return set(self.elements) == set(other.elements)

    class Choice(enum.Enum):
        SHOW_ELEMENTS = 1
        SHOW_INTERSECTIONS = 2
        ADD_ELEMENT = 3
        REMOVE_ELEMENT = 4
        CHECK_ELEMENT = 5
        EXIT = 6

    choiceValues = {
        Choice.SHOW_ELEMENTS.value: "Show elements",
        Choice.SHOW_INTERSECTIONS.value: "Show intersections",
        Choice.ADD_ELEMENT.value: "Add element",
        Choice.REMOVE_ELEMENT.value: "Remove element",
        Choice.CHECK_ELEMENT.value: "Check element",
        Choice.EXIT.value: "Exit",
    }
```

Рисунки с результатами работы программы:

```
Enter the elements of the set: 1 2 3
Enter the elements of the set: 3 4 5
Menu:
1. Show elements   2. Show intersections   3. Add element   4. Remove element   5. Check element   6. Exit
Enter your choice: 2
{3}
Menu:
1. Show elements   2. Show intersections   3. Add element   4. Remove element   5. Check element   6. Exit
Enter your choice: 5
Enter the set number: 2
Enter the element value: 5
True
```

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. Система Интернет-магазин. Администратор добавляет информацию о Товаре. Клиент делает и оплачивает Заказ на Товары. Администратор регистрирует Продажу и может занести неплательщиков в «черный список».

Выполнение:

Код программы:

```
class User:
    def __init__(self, user_id, name, email):
        self.id = user_id
        self.name = name
        self.email = email

    def get_info(self):
        return f"ID: {self.id}\nName: {self.name}\nEmail: {self.email}"
```

```
    def __str__(self):
        return f"User ID: {self.id}, Name: {self.name}, Email: {self.email}"

class Admin(User):
    def __init__(self, user_id, name, email):
        super().__init__(user_id, name, email)
        self.blacklist = []

    def add_product(self, product, products_list):
        products_list.append(product)

    def register_sale(self, sale, sales_list):
        sales_list.append(sale)

    def blacklist_customer(self, customer):
        if customer not in self.blacklist:
            self.blacklist.append(customer)

    def __str__(self):
        return f"Admin ID: {self.id}, Name: {self.name}, Email: {self.email}, Blacklist:
{len(self.blacklist)}"

class Customer(User):
    def __init__(self, user_id, name, email):
        super().__init__(user_id, name, email)
        self.orders = []

    def place_order(self, order):
        self.orders.append(order)

    def pay_order(self, order):
        order.status = "Paid"

    def __str__(self):
        return f"Customer ID: {self.id}, Name: {self.name}, Email: {self.email}, Orders:
{len(self.orders)}"

class Product:
    def __init__(self, product_id, name, price, quantity):
        self.id = product_id
        self.name = name
        self.price = price
        self.quantity = quantity

    def __str__(self):
        return f"Product ID: {self.id}, Name: {self.name}, Price: {self.price}, Quantity:
{self.quantity}"

class Order:
    def __init__(self, order_id, customer, products):
        self.id = order_id
        self.customer = customer
        self.products = products
        self.total_price = sum([p.price for p in products])
        self.status = "Unpaid"

    def pay(self):
        self.status = "Paid"

    def __str__(self):
        products_str = "\n".join([str(p) for p in self.products])
        return f"Order ID: {self.id}, Customer: {self.customer.name}, Products:\n{products_str}\nTotal
Price: {self.total_price}, Status: {self.status}"

class Sale:
    def __init__(self, sale_id, order, date):
        self.id = sale_id
        self.order = order
        self.date = date

    def __str__(self):
```

```
        return f"Sale ID: {self.id}, Order ID: {self.order.id}, Date: {self.date}"

if __name__ == "__main__":
    # Создаем администратора и клиента
    admin = Admin(1, "Admin", "admin@example.com")
    customer = Customer(2, "Customer", "customer@example.com")

    # Создаем товары
    product1 = Product(1, "Laptop", 1000.0, 10)
    product2 = Product(2, "Phone", 500.0, 20)

    # Администратор добавляет товары
    products_list = []
    admin.add_product(product1, products_list)
    admin.add_product(product2, products_list)

    # Клиент делает заказ
    order = Order(1, customer, [product1, product2])
    customer.place_order(order)

    # Клиент оплачивает заказ
    customer.pay_order(order)

    # Администратор регистрирует продажу
    sale = Sale(1, order, "2025-02-22")
    sales_list = []
    admin.register_sale(sale, sales_list)

    # Администратор заносит клиента в черный список
    admin.blacklist_customer(customer)

    print("Admin:", admin)
    print("Customer:", customer)
    print("Products List:")
    for product in products_list:
        print(product)
    print("Customer Orders:")
    for order in customer.orders:
        print(order)
    print("Sales List:")
    for sale in sales_list:
        print(sale)
    print("Admin Blacklist:")
    for customer in admin.blacklist:
        print(customer)
```

Рисунки с результатами работы программы:

```
Admin: Admin ID: 1, Name: Admin, Email: admin@example.com, Blacklist: 1
Customer: Customer ID: 2, Name: Customer, Email: customer@example.com, Orders: 1
Products List:
Product ID: 1, Name: Laptop, Price: 1000.0, Quantity: 10
Product ID: 2, Name: Phone, Price: 500.0, Quantity: 20
Customer Orders:
Order ID: 1, Customer: Customer, Products:
Product ID: 1, Name: Laptop, Price: 1000.0, Quantity: 10
Product ID: 2, Name: Phone, Price: 500.0, Quantity: 20
Total Price: 1500.0, Status: Paid
Sales List:
Sale ID: 1, Order ID: 1, Date: 2025-02-22
Admin Blacklist:
Customer ID: 2, Name: Customer, Email: customer@example.com, Orders: 1
```

Вывод: закрепил базовые знания Python при решении практических задач