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

С. С. Жватель

студент группы ПО11

Проверил

А. А. Крощенко

ст. преп. кафедры ИИТ,

12.04.2025 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 вещественных чисел переменной мощности – Предусмотреть возможность объединения 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:**

"""Module for working with sets of real numbers."""  
  
from typing import List, Optional, Union  
  
  
class RealNumberSet:  
 """A set of real numbers supporting operations like add, remove, union, and comparison."""  
  
 def \_\_init\_\_(self, initial\_elements: Optional[Union[List, set]] = None) -> None:  
 """Initialize a set with optional initial elements.  
  
 Args:  
 initial\_elements: An iterable of numbers (int or float) to initialize the set.  
 Non-numeric values are ignored with a warning.  
 """  
 self.elements: List[float] = []  
 if initial\_elements is not None:  
 for item in initial\_elements:  
 try:  
 self.add\_element(float(item))  
 except (ValueError, TypeError):  
 print(f"Пропущен неверный элемент: {item}")  
  
 def add\_element(self, element\_: Union[int, float]) -> bool:  
 """Add an element to the set if it doesn't already exist.  
  
 Args:  
 element\_: The number to add (int or float).  
  
 Returns:  
 bool: True if the element was added, False if it already exists.  
 """  
 try:  
 element\_float = float(element\_)  
 if element\_float not in self.elements:  
 self.elements.append(element\_float)  
 return True  
 return False  
 except (ValueError, TypeError):  
 print(f"Ошибка: '{element\_}' не является числом.")  
 return False  
  
 def remove\_element(self, element\_: Union[int, float]) -> bool:  
 """Remove an element from the set if it exists.  
  
 Args:  
 element\_: The number to remove (int or float).  
  
 Returns:  
 bool: True if the element was removed, False if it was not found.  
 """  
 try:  
 element\_float = float(element\_)  
 if element\_float in self.elements:  
 self.elements.remove(element\_float)  
 return True  
 return False  
 except (ValueError, TypeError):  
 print(f"Ошибка: '{element\_}' не является числом.")  
 return False  
  
 def union(self, other\_set: "RealNumberSet") -> "RealNumberSet":  
 """Create a new set containing all elements from this set and another.  
  
 Args:  
 other\_set: Another RealNumberSet to unite with.  
  
 Returns:  
 RealNumberSet: A new set with all unique elements.  
  
 Raises:  
 ValueError: If other\_set is not a RealNumberSet.  
 """  
 if not isinstance(other\_set, RealNumberSet):  
 raise ValueError("Аргумент должен быть объектом RealNumberSet")  
 result = RealNumberSet(self.elements)  
 for elem in other\_set.elements:  
 result.add\_element(elem)  
 return result  
  
 def contains(self, element\_: Union[int, float]) -> bool:  
 """Check if an element exists in the set.  
  
 Args:  
 element\_: The number to check (int or float).  
  
 Returns:  
 bool: True if the element is in the set, False otherwise.  
 """  
 try:  
 return float(element\_) in self.elements  
 except (ValueError, TypeError):  
 return False  
  
 def print\_elements(self) -> None:  
 """Print the elements of the set in sorted order."""  
 print("Множество:", sorted(self.elements))  
  
 def \_\_str\_\_(self) -> str:  
 """Return a string representation of the set.  
  
 Returns:  
 str: String of sorted elements.  
 """  
 return f"RealNumberSet({sorted(self.elements)})"  
  
 def \_\_eq\_\_(self, other: object) -> bool:  
 """Check if two sets are equal.  
  
 Args:  
 other: Another object to compare with.  
  
 Returns:  
 bool: True if the sets have the same elements, False otherwise.  
 """  
 if not isinstance(other, RealNumberSet):  
 return False  
 return sorted(self.elements) == sorted(other.elements)  
  
  
def create\_set\_interactively(set\_name: str) -> RealNumberSet:  
 """Interactively create a set by prompting for elements.  
  
 Args:  
 set\_name: Name of the set for user prompts (e.g., 'первого множества').  
  
 Returns:  
 RealNumberSet: A new set with user-provided elements.  
 """  
 elements = []  
 print(f"Создание {set\_name} (введите числа, 'q' для завершения):")  
 while True:  
 user\_input = input("Элемент: ")  
 if user\_input.lower() == "q":  
 break  
 try:  
 elements.append(float(user\_input))  
 except ValueError:  
 print("Ошибка: введите вещественное число.")  
 return RealNumberSet(elements)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 print("=== Задание 1: Работа с множествами ===")  
  
 # Create first set  
 set1 = create\_set\_interactively("первого множества")  
  
 # Create second set  
 set2 = create\_set\_interactively("второго множества")  
  
 # Interactive menu  
 while True:  
 print("\nВыберите действие:")  
 print("1. Вывести множества")  
 print("2. Добавить элемент")  
 print("3. Удалить элемент")  
 print("4. Проверить принадлежность элемента")  
 print("5. Объединить множества")  
 print("6. Сравнить множества")  
 print("7. Выход")  
 choice = input("Ваш выбор (1-7): ")  
  
 if choice == "1":  
 print("Первое множество:")  
 set1.print\_elements()  
 print("Второе множество:")  
 set2.print\_elements()  
  
 elif choice == "2":  
 print("\nВыберите множество:")  
 print("1. Первое множество")  
 print("2. Второе множество")  
 sub\_choice = input("Ваш выбор (1-2): ")  
 new\_input = input("Введите элемент для добавления: ")  
 try:  
 element = float(new\_input)  
 if sub\_choice == "1":  
 if set1.add\_element(element):  
 print("Элемент добавлен в первое множество.")  
 else:  
 print("Элемент уже есть в первом множестве.")  
 elif sub\_choice == "2":  
 if set2.add\_element(element):  
 print("Элемент добавлен во второе множество.")  
 else:  
 print("Элемент уже есть во втором множестве.")  
 else:  
 print("Неверный выбор множества.")  
 except ValueError:  
 print("Ошибка: введите вещественное число.")  
  
 elif choice == "3":  
 print("\nВыберите множество:")  
 print("1. Первое множество")  
 print("2. Второе множество")  
 sub\_choice = input("Ваш выбор (1-2): ")  
 new\_input = input("Введите элемент для удаления: ")  
 try:  
 element = float(new\_input)  
 if sub\_choice == "1":  
 if set1.remove\_element(element):  
 print("Элемент удален из первого множества.")  
 else:  
 print("Элемент не найден в первом множестве.")  
 elif sub\_choice == "2":  
 if set2.remove\_element(element):  
 print("Элемент удален из второго множества.")  
 else:  
 print("Элемент не найден во втором множестве.")  
 else:  
 print("Неверный выбор множества.")  
 except ValueError:  
 print("Ошибка: введите вещественное число.")  
  
 elif choice == "4":  
 print("\nВыберите множество:")  
 print("1. Первое множество")  
 print("2. Второе множество")  
 sub\_choice = input("Ваш выбор (1-2): ")  
 new\_input = input("Введите элемент для проверки: ")  
 try:  
 element = float(new\_input)  
 if sub\_choice == "1":  
 if set1.contains(element):  
 print(f"Элемент {element} принадлежит первому множеству.")  
 else:  
 print(f"Элемент {element} не принадлежит первому множеству.")  
 elif sub\_choice == "2":  
 if set2.contains(element):  
 print(f"Элемент {element} принадлежит второму множеству.")  
 else:  
 print(f"Элемент {element} не принадлежит второму множеству.")  
 else:  
 print("Неверный выбор множества.")  
 except ValueError:  
 print("Ошибка: введите вещественное число.")  
  
 elif choice == "5":  
 union\_set = set1.union(set2)  
 print("Результат объединения:")  
 union\_set.print\_elements()  
  
 elif choice == "6":  
 print("Множества равны" if set1 == set2 else "Множества не равны.")  
  
 elif choice == "7":  
 print("Выход из программы.")  
 break  
  
 else:  
 print("Неверный выбор. Попробуйте снова.")

**Спецификация ввода:**

<элементы первого множества>

<элементы второго множества>

<выбор действия из меню>

<дополнительные данные для операций>

**Пример:**

=== Задание 1: Работа с множествами ===

Создание первого множества (введите числа, 'q' для завершения):

Элемент: 1.5

Элемент: 2.0

Элемент: q

Создание второго множества (введите числа, 'q' для завершения):

Элемент: 2.0

Элемент: 3.7

Элемент: q

Выберите действие:

1. Вывести множества

2. Добавить элемент

3. Удалить элемент

4. Проверить принадлежность элемента

5. Объединить множества

6. Сравнить множества

7. Выход

Ваш выбор (1-7): 1

**Спецификация вывода:**

Множество: [<отсортированные элементы>]

Элемент добавлен в <первое/второе> множество.

Элемент уже есть в <первом/втором> множестве.

Элемент удален из <первого/второго> множества.

Элемент не найден в <первом/втором> множестве.

Элемент <значение> принадлежит <первому/второму> множеству.

Элемент <значение> не принадлежит <первому/второму> множеству.

Результат объединения:

Множество: [<отсортированные элементы>]

Множества равны

Множества не равны

Ошибка: введите вещественное число.

Неверный выбор множества.

Неверный выбор. Попробуйте снова.

Выход из программы.

**Пример:**

=== Задание 1: Работа с множествами ===

Создание первого множества (введите числа, 'q' для завершения):

Элемент: 1.5

Элемент: 2.0

Элемент: q

Создание второго множества (введите числа, 'q' для завершения):

Элемент: 2.0

Элемент: 3.7

Элемент: q

Выберите действие:

1. Вывести множества

2. Добавить элемент

3. Удалить элемент

4. Проверить принадлежность элемента

5. Объединить множества

6. Сравнить множества

7. Выход

Ваш выбор (1-7): 1

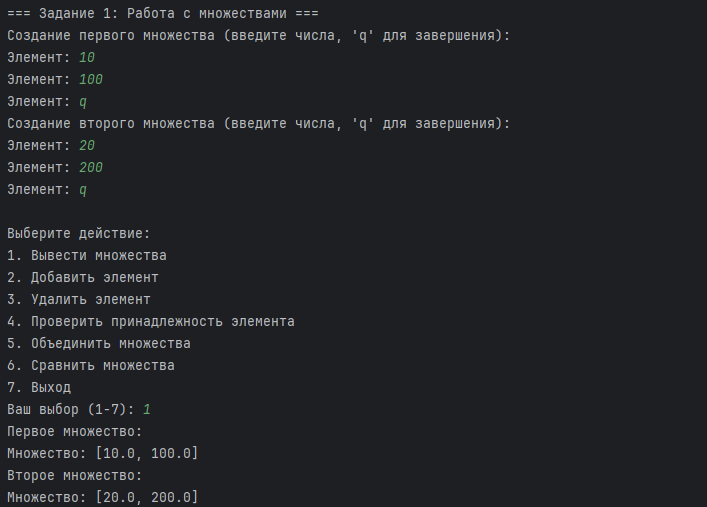
Первое множество:

Множество: [1.5, 2.0]

Второе множество:

Множество: [2.0, 3.7]

**Рисунки с результатами работы программы:**

****

**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 Железнодорожная касса. Пассажир делает Заявку на станцию назначения, время и дату поездки. Система регистрирует Заявку и осуществляет поиск подходящего Поезда. Пассажир делает выбор Поезда и получает Счет на Оплату. Администратор вводит номера Поездов, промежуточные и конечные станции, цены.**

Выполнение:

**Код программы:**

"""Module for managing railway system with trains, requests, and invoices."""  
  
from datetime import datetime  
from typing import List, Optional, Tuple  
  
  
class Train:  
 """Represents a train with details like ID, stations, price."""  
  
 def \_\_init\_\_(  
 self,  
 train\_id\_: str,  
 route\_: Tuple[str, str], # Tuple of (start\_station, end\_station)  
 price\_: float = 0.0,  
 middle\_stations\_: Optional[List[str]] = None,  
 ) -> None:  
 """Initialize a train with given attributes."""  
 self.train\_id = train\_id\_  
 self.route = route\_ # Stores start and end stations as a tuple  
 self.middle\_stations = middle\_stations\_ if middle\_stations\_ is not None else []  
 self.price = price\_  
  
 def \_\_str\_\_(self) -> str:  
 """Return a string representation of the train."""  
 return f"Поезд {self.train\_id}: {self.route[0]} -> {self.route[1]}, " f"Цена: {self.price}"  
  
 def get\_route(self) -> List[str]:  
 """Return the full route including all stations."""  
 return [self.route[0]] + self.middle\_stations + [self.route[1]]  
  
  
class Request:  
 """Represents a passenger's travel request."""  
  
 def \_\_init\_\_(self, p\_name: str, destination\_: str, date: datetime, time: str) -> None:  
 """Initialize a request with passenger details."""  
 self.p\_name = p\_name  
 self.destination = destination\_  
 self.date = date  
 self.time = time # Time added for passenger request  
 self.selected\_train: Optional[Train] = None  
  
 def \_\_str\_\_(self) -> str:  
 """Return a string representation of the request."""  
 return f"{self.p\_name} - {self.destination} " f"Дата: {self.date.date()} Время: {self.time}"  
  
 def is\_valid(self) -> bool:  
 """Check if the request has valid attributes."""  
 return bool(self.p\_name and self.destination and self.date and self.time)  
  
  
class Invoice:  
 """Represents an invoice for a passenger's train ticket."""  
  
 def \_\_init\_\_(self, request\_: Request, train\_: Train) -> None:  
 """Initialize an invoice with request and train details."""  
 self.request = request\_  
 self.train = train\_  
 self.amount = train\_.price  
  
 def \_\_str\_\_(self) -> str:  
 """Return a string representation of the invoice."""  
 return f"Счет {self.request.p\_name}: Поезд {self.train.train\_id}, " f"Сумма: {self.amount}"  
  
 def get\_details(self) -> dict:  
 """Return detailed information about the invoice."""  
 return {  
 "passenger": self.request.p\_name,  
 "train\_id": self.train.train\_id,  
 "route": self.train.get\_route(),  
 "price": self.amount,  
 "date": self.request.date, # Use request date since train no longer has time  
 }  
  
  
class RailwaySystem:  
 """Manages trains, passenger requests, and ticketing operations."""  
  
 def \_\_init\_\_(self) -> None:  
 """Initialize an empty railway system."""  
 self.trains: List[Train] = []  
 self.requests: List[Request] = []  
  
 def add\_train(self, train\_: Train) -> None:  
 """Add a train to the system (for administrators)."""  
 self.trains.append(train\_)  
  
 def create\_request(self, p\_name: str, destination\_: str, date: datetime, time: str) -> Request:  
 """Create and store a passenger request."""  
 new\_request = Request(p\_name, destination\_, date, time)  
 self.requests.append(new\_request)  
 return new\_request  
  
 def find\_trains(self, request\_: Request) -> List[Train]:  
 """Find trains matching the request's destination."""  
 matching\_trains = []  
 for transport in self.trains:  
 if request\_.destination in [transport.route[1]] + transport.middle\_stations:  
 matching\_trains.append(transport)  
 return matching\_trains  
  
 def select\_train(self, request\_: Request, train\_: Train) -> None:  
 """Assign a train to a request (for passengers)."""  
 request\_.selected\_train = train\_  
  
 def create\_invoice(self, request\_: Request) -> Invoice:  
 """Create an invoice for a request with a selected train."""  
 if request\_.selected\_train is None:  
 raise ValueError("Поезд не выбран")  
 return Invoice(request\_, request\_.selected\_train)  
  
  
def add\_train\_interactively(railway: RailwaySystem) -> None:  
 """Interactively add a train to the railway system (for administrators)."""  
 print("=== Добавление нового поезда (для администратора) ===")  
 train\_id = input("Введите номер поезда: ")  
 start\_station = input("Введите начальную станцию: ")  
 end\_station = input("Введите конечную станцию: ")  
 print("Введите промежуточные станции (пустая строка для завершения):")  
 stations = []  
 while True:  
 station = input("Промежуточная станция: ")  
 if station == "":  
 break  
 stations.append(station)  
 while True:  
 try:  
 price = float(input("Введите цену билета: "))  
 if price < 0:  
 raise ValueError("Цена не может быть отрицательной")  
 break  
 except ValueError as e:  
 print(f"Ошибка: {e if str(e) else 'введите число'}")  
 new\_train = Train(train\_id, (start\_station, end\_station), price, stations)  
 railway.add\_train(new\_train)  
 print("Поезд добавлен.")  
  
  
def create\_request\_interactively(railway: RailwaySystem) -> None:  
 """Interactively create a passenger request and process it (for passengers)."""  
 print("=== Создание заявки (для пассажира) ===")  
 passenger = input("Введите ваше имя: ")  
 destination = input("Введите станцию назначения: ")  
 while True:  
 try:  
 travel\_date = input("Введите дату поездки (ГГГГ-ММ-ДД): ")  
 date = datetime.strptime(travel\_date, "%Y-%m-%d")  
 break  
 except ValueError:  
 print("Ошибка: введите дату в формате ГГГГ-ММ-ДД")  
 while True:  
 travel\_time = input("Введите время поездки (ЧЧ:ММ): ")  
 try:  
 datetime.strptime(travel\_time, "%H:%M")  
 break  
 except ValueError:  
 print("Ошибка: введите время в формате ЧЧ:ММ")  
 new\_request = railway.create\_request(passenger, destination, date, travel\_time)  
 print("Ваша заявка создана:", new\_request)  
  
 matching\_trains = railway.find\_trains(new\_request)  
 if not matching\_trains:  
 print("Подходящих поездов не найдено.")  
 return  
 print("\nДоступные поезда:")  
 for i, transport in enumerate(matching\_trains, 1):  
 print(f"{i}. {transport}")  
  
 while True:  
 try:  
 train\_choice = int(input("Выберите id поезда (0 для отмены): "))  
 if train\_choice == 0:  
 print("Выбор отменен.")  
 break  
 if 1 <= train\_choice <= len(matching\_trains):  
 railway.select\_train(new\_request, matching\_trains[train\_choice - 1])  
 invoice = railway.create\_invoice(new\_request)  
 print("\nВаш счет на оплату создан:")  
 print(invoice)  
 break  
 print("Неверный номер поезда.")  
 except ValueError:  
 print("Ошибка: введите число.")  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 print("=== Железнодорожная касса ===")  
 system = RailwaySystem()  
  
 while True:  
 print("\n1. Добавить поезд (для администратора)")  
 print("2. Создать заявку и выбрать поезд (для пассажира)")  
 print("3. Показать все поезда")  
 print("4. Показать все заявки")  
 print("5. Выход")  
 choice = input("Ваш выбор (1-5): ")  
  
 if choice == "1":  
 add\_train\_interactively(system)  
  
 elif choice == "2":  
 create\_request\_interactively(system)  
  
 elif choice == "3":  
 if not system.trains:  
 print("Поездов нет.")  
 else:  
 print("\nСписок всех поездов:")  
 for train in system.trains:  
 print(train)  
  
 elif choice == "4":  
 if not system.requests:  
 print("Заявок нет.")  
 else:  
 print("\nСписок всех заявок:")  
 for request in system.requests:  
 print(request)  
  
 elif choice == "5":  
 print("Выход из программы.")  
 break  
  
 else:  
 print("Неверный выбор. Попробуйте снова.")

**Спецификация ввода:**

<выбор действия из меню>

<данные для выбранной операции>

**Пример:**

=== Железнодорожная касса ===

1. Добавить поезд (для администратора)

2. Создать заявку и выбрать поезд (для пассажира)

3. Показать все поезда

4. Показать все заявки

5. Выход

Ваш выбор (1-5): 1

=== Добавление нового поезда (для администратора) ===

Введите номер поезда: T1

Введите начальную станцию: Минск

Введите конечную станцию: Брест

Введите промежуточные станции (пустая строка для завершения):

Промежуточная станция: Барановичи

Промежуточная станция:

Введите цену билета: 25.5

**Спецификация вывода:**

Список всех поездов:

Поезд <id>: <начальная станция> -> <конечная станция>, Цена: <цена>

Поездов нет.

Список всех заявок:

<имя> - <станция назначения> Дата: <дата> Время: <время>

Заявок нет.

Выход из программы.

Неверный выбор. Попробуйте снова.

**Пример:**

Ваш выбор (1-5): 3

Список всех поездов:

Поезд T1: Минск -> Брест, Цена: 25.5

Ваш выбор (1-5): 4

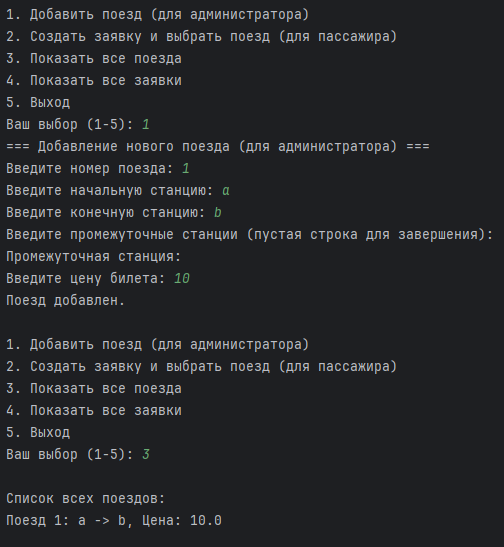
Список всех заявок:

Иван - Барановичи Дата: 2025-05-01 Время: 10:30

Ваш выбор (1-5): 5

Выход из программы.

**Рисунки с результатами работы программы:**

****

**Вывод:** закрепил базовые знания Python при решении практических задач