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
Гулевич Е.А.
студент группы ПО11

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

Брест 2025

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

Задание 1. Множество символов ограниченной 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. Мощность множества задается при создании объекта. Реализацию множества осуществить на базе списка. Переопределить метод __eq__, выполняющий сравнение объектов данного типа.

Выполнение:

Код программы:

```
class LimitedCharSet:
    def __init__(self, capacity, initial_elements=None):
        self.capacity = capacity
        self._elements = []
        if initial_elements:
            for elem in initial_elements:
                self.add(elem)

    def add(self, char):
        if char in self._elements:
            print(f"'{char}' уже есть в множестве.")
        elif len(self._elements) >= self.capacity:
            print(f"Невозможно добавить '{char}': превышена мощность множества.")
        else:
            self._elements.append(char)

    def remove(self, char):
        if char in self._elements:
            self._elements.remove(char)
        else:
            print(f"'{char}' нет в множестве.")

    def contains(self, char):
        return char in self._elements

    def __str__(self):
        return "{" + ", ".join(self._elements) + "}"

    def __eq__(self, other):
        if not isinstance(other, LimitedCharSet):
            return False
        return sorted(self._elements) == sorted(other._elements)

    def union(self, other):
        if not isinstance(other, LimitedCharSet):
```

```
            raise TypeError("Можно объединять только с объектом LimitedCharSet")

        new_capacity = max(self.capacity, other.capacity)
        new_set = LimitedCharSet(new_capacity)

        for char in self._elements + other._elements:
            new_set.add(char)

        return new_set

    @property
    def elements(self):
        return list(self._elements)

    @property
    def size(self):
        return len(self._elements)

    @property
    def max_size(self):
        return self.capacity

def main():
    print("== Работа с ограниченным множеством символов ==")
    capacity = int(input("Введите мощность множества: "))
    initial = input("Введите начальные символы (например: abc): ")

    my_set = LimitedCharSet(capacity, list(initial))

    while True:
        print("\nМеню:")
        print("1. Добавить символ")
        print("2. Удалить символ")
        print("3. Проверить наличие символа")
        print("4. Вывести множество")
        print("5. Создать второе множество и объединить")
        print("6. Проверить равенство с другим множеством")
        print("7. Завершить")

        choice = input("Выберите действие: ")

        if choice == "1":
            ch = input("Введите символ для добавления: ")
            my_set.add(ch)

        elif choice == "2":
            ch = input("Введите символ для удаления: ")
            my_set.remove(ch)

        elif choice == "3":
            ch = input("Введите символ для проверки: ")
```

```
            print(f"'{ch}' в множестве? ->", my_set.contains(ch))

        elif choice == "4":
            print("Текущее множество:", my_set)

        elif choice == "5":
            print("== Создание второго множества для объединения ==")
            cap2 = int(input("Введите мощность второго множества: "))
            init2 = input("Введите символы второго множества: ")
            other_set = LimitedCharSet(cap2, list(init2))
            combined = my_set.union(other_set)
            print("Результат объединения:", combined)

        elif choice == "6":
            print("== Создание второго множества для сравнения ==")
            cap2 = int(input("Введите мощность второго множества: "))
            init2 = input("Введите символы второго множества: ")
            other_set = LimitedCharSet(cap2, list(init2))
            print("Множества равны?" , my_set == other_set)

        elif choice == "7":
            print("Выход из программы.")
            break

        else:
            print("Неверный выбор. Попробуйте снова.")

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

Спецификация ввода:

Введите мощность множества: <1-ый элемент>
Введите начальные символы (например: abc): <начальные символы>
Меню:
1. Добавить символ
2. Удалить символ
3. Проверить наличие символа
4. Вывести множество
5. Создать второе множество и объединить
6. Проверить равенство с другим множеством
7. Завершить
Выберите действие: <значение от 1 до 7>

Пример:

Введите мощность множества: 123
Введите начальные символы (например: abc): abc
Меню:
1. Добавить символ
2. Удалить символ

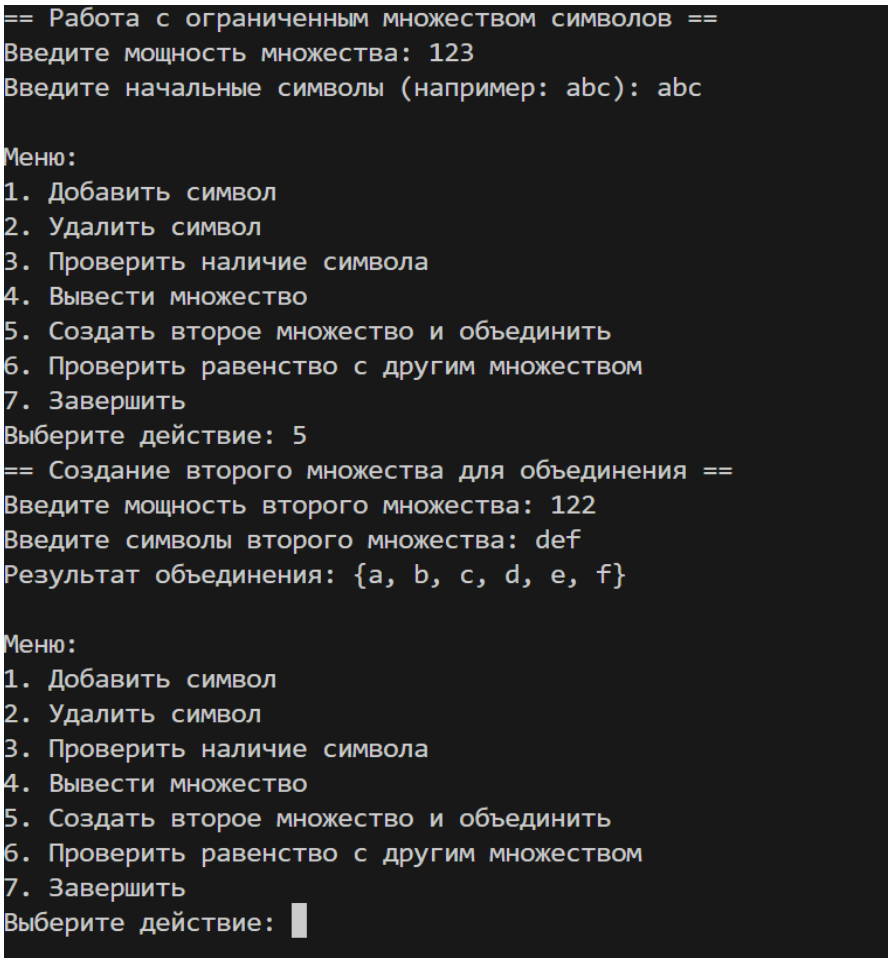
3. Проверить наличие символа
4. Вывести множество
5. Создать второе множество и объединить
6. Проверить равенство с другим множеством
7. Завершить

Выберите действие: 5

Спецификация вывода:

== Создание второго множества для объединения ==
Введите мощность второго множества: <мощность 2-го множества>
Введите символы второго множества: <символы 2-го множества>
Результат объединения: {<объединённое множество>}

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

Выполнение:

Код программы:

```
class Driver:
    def __init__(self, name: str):
        self.name = name
        self.assigned_trips = []
        self.is_active = True

    def assign_trip(self, trip):
        if self.is_active:
            self.assigned_trips.append(trip)
            trip.assign_driver(self)
        else:
            print(f"Водитель {self.name} отстранён и не может быть назначен на рейс.")

    def report_trip_completion(self, trip, vehicle_condition: str):
        if trip in self.assigned_trips:
            trip.complete_trip(vehicle_condition)
        else:
            print("Рейс не найден среди назначенных.")

    def request_repair(self, dispatcher, vehicle, reason: str):
        dispatcher.create_repair_request(vehicle, self, reason)

    def __str__(self):
        return f"Водитель: {self.name}, Статус: {'Активен' if self.is_active else 'Отстранён'}"

class Vehicle:
    def __init__(self, model: str):
        self.model = model
        self.is_available = True

    def __str__(self):
        return f"Автомобиль: {self.model}, Статус: {'Доступен' if self.is_available else 'Недоступен'}"

class Trip:
    def __init__(self, destination: str):
        self.destination = destination
        self.driver = None
        self.vehicle = None
        self.completed = False
        self.vehicle_condition = None

    def assign_driver(self, driver):
        self.driver = driver

    def assign_vehicle(self, vehicle):
        self.vehicle = vehicle
        vehicle.is_available = False
```

```
        def complete_trip(self, condition: str):
            self.completed = True
            self.vehicle_condition = condition
            self.vehicle.is_available = True

        def __str__(self):
            status = "Завершен" if self.completed else "В процессе"
            return f"Рейс в {self.destination}, Статус: {status}"

    class RepairRequest:
        def __init__(self, vehicle, driver, reason: str):
            self.vehicle = vehicle
            self.driver = driver
            self.reason = reason

        def __str__(self):
            return f"Заявка на ремонт: {self.vehicle.model} от {self.driver.name}, причина: {self.reason}"

    class Dispatcher:
        def __init__(self, name: str):
            self.name = name
            self.repair_requests = []

        def assign_trip(self, driver, vehicle, trip):
            if driver.is_active and vehicle.is_available:
                driver.assign_trip(trip)
                trip.assign_vehicle(vehicle)
                print(f"Рейс в {trip.destination} назначен водителю {driver.name} на автомобиле
{vehicle.model}")
            else:
                print("Невозможно назначить рейс. Проверьте статус водителя и автомобиля.")

        def create_repair_request(self, vehicle, driver, reason: str):
            request = RepairRequest(vehicle, driver, reason)
            self.repair_requests.append(request)
            print(f"Создана заявка на ремонт: {request}")

        def suspend_driver(self, driver):
            driver.is_active = False
            print(f"Водитель {driver.name} отстранён от работы.")

        def __str__(self):
            return f"Диспетчер: {self.name}"

    def autobase_system():
        dispatcher = Dispatcher(input("Введите имя диспетчера: "))

        drivers = []
        vehicles = []
        trips = []
```

```
    while True:
        name = input("Введите имя водителя (или 'стоп'): ")
        if name.lower() == 'стоп':
            break
        drivers.append(Driver(name))

    while True:
        model = input("Введите модель автомобиля (или 'стоп'): ")
        if model.lower() == 'стоп':
            break
        vehicles.append(Vehicle(model))

    while True:
        dest = input("Введите пункт назначения рейса (или 'стоп'): ")
        if dest.lower() == 'стоп':
            break
        trips.append(Trip(dest))

    while True:
        print("\n1. Назначить рейс\n2. Заявка на ремонт\n3. Отстранить водителя\n4. Завершить
рейс\n5. Показать заявки на ремонт\n0. Выход")
        choice = input("Выберите действие: ")

        if choice == "1":
            for i, d in enumerate(drivers): print(f"{i+1}. {d}")
            driver = drivers[int(input("Выберите водителя: ")) - 1]
            for i, v in enumerate(vehicles): print(f"{i+1}. {v}")
            vehicle = vehicles[int(input("Выберите автомобиль: ")) - 1]
            for i, t in enumerate(trips): print(f"{i+1}. {t}")
            trip = trips[int(input("Выберите рейс: ")) - 1]
            dispatcher.assign_trip(driver, vehicle, trip)

        elif choice == "2":
            for i, d in enumerate(drivers): print(f"{i+1}. {d}")
            driver = drivers[int(input("Выберите водителя: ")) - 1]
            for i, v in enumerate(vehicles): print(f"{i+1}. {v}")
            vehicle = vehicles[int(input("Выберите автомобиль: ")) - 1]
            reason = input("Причина ремонта: ")
            driver.request_repair(dispatcher, vehicle, reason)

        elif choice == "3":
            for i, d in enumerate(drivers): print(f"{i+1}. {d}")
            driver = drivers[int(input("Выберите водителя: ")) - 1]
            dispatcher.suspend_driver(driver)

        elif choice == "4":
            for i, t in enumerate(trips): print(f"{i+1}. {t}")
            trip = trips[int(input("Выберите завершённый рейс: ")) - 1]
            condition = input("Состояние автомобиля после рейса: ")
            trip.driver.report_trip_completion(trip, condition)
```

```
        elif choice == "5":
            print("Заявки на ремонт:")
            for r in dispatcher.repair_requests:
                print(r)

        elif choice == "0":
            break

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

Спецификация ввода:

Введите имя диспетчера: <имя диспетчера>
Введите имя водителя (или 'стоп'): <имя водителя>
Введите модель автомобиля (или 'стоп'): <модель авто>
Введите пункт назначения рейса (или 'стоп'): <пункт назначения>

Пример:

Введите имя диспетчера: Егор
Введите имя водителя (или 'стоп'): Илья
Введите имя водителя (или 'стоп'): стоп
Введите модель автомобиля (или 'стоп'): Форд
Введите модель автомобиля (или 'стоп'): стоп
Введите пункт назначения рейса (или 'стоп'): А
Введите пункт назначения рейса (или 'стоп'): Б
Введите пункт назначения рейса (или 'стоп'): стоп

Спецификация вывода:

1. Назначить рейс
2. Заявка на ремонт
3. Отстранить водителя
4. Завершить рейс
5. Показать заявки на ремонт
0. Выход

Выберите действие: <значение от 0 до 5>

Пример:

Введите имя диспетчера: Егор
Введите имя водителя (или 'стоп'): Илья
Введите имя водителя (или 'стоп'): стоп
Введите модель автомобиля (или 'стоп'): Форд
Введите модель автомобиля (или 'стоп'): стоп
Введите пункт назначения рейса (или 'стоп'): А
Введите пункт назначения рейса (или 'стоп'): Б
Введите пункт назначения рейса (или 'стоп'): стоп

1. Назначить рейс
2. Заявка на ремонт
3. Отстранить водителя
4. Завершить рейс
5. Показать заявки на ремонт
0. Выход

Выберите действие: 1
1. Водитель: Илья, Статус: Активен

Рисунки с результатами работы программы:

```
PS D:\pythonvs> & d:/python/python.exe d:/pythonvs/lab2.2.py
Введите имя диспетчера: Антон
Введите имя водителя (или 'стоп'): Илья
Введите имя водителя (или 'стоп'): Егор
Введите имя водителя (или 'стоп'): Стас
Введите имя водителя (или 'стоп'): стоп
Введите модель автомобиля (или 'стоп'): Форд
Введите модель автомобиля (или 'стоп'): стоп
Введите пункт назначения рейса (или 'стоп'): А
Введите пункт назначения рейса (или 'стоп'): Б
Введите пункт назначения рейса (или 'стоп'): стоп

1. Назначить рейс
2. Заявка на ремонт
3. Отстранить водителя
4. Завершить рейс
5. Показать заявки на ремонт
0. Выход
Выберите действие: 1
1. Водитель: Илья, Статус: Активен
2. Водитель: Егор, Статус: Активен
3. Водитель: Стас, Статус: Активен
Выберите водителя: 2
1. Автомобиль: Форд, Статус: Доступен
Выберите автомобиль: 1
1. Рейс в А, Статус: В процессе
2. Рейс в Б, Статус: В процессе
Выберите рейс: 2
Рейс в Б назначен водителю Егор на автомобиле Форд
```

Вывод: закрепил навыки объектно-ориентированного программирования на языке Python