

Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт перспективной инженерии
Департамент цифровых, робототехнических систем и электроники

ОТЧЕТ
По лабораторной работе №6
Дисциплины «Объектно-ориентированное программирование»

Выполнил:

Пустяков Андрей Сергеевич

3 курс, группа ИВТ-б-о-22-1,

09.03.01 «Информатика и
вычислительная техника (профиль)
«Программное обеспечение средств
вычислительной
техники и автоматизированных
систем», очная форма обучения

(подпись)

Руководитель практики:

Воронкин Р. А., доцент департамента
цифровых и робототехнических
систем и электроники института
перспективной инженерии

(подпись)

Отчет защищен с оценкой _____ Дата защиты _____

Ставрополь, 2024 г.

Тема: Классы данных в Python.

Цель: приобрести навыки по работе с классами данных при написании программ с помощью языка программирования Python версии 3.x.

Ход работы:

Проработка примеров лабораторной работы:

Пример 1.

Для примера лабораторной работы 4.5 необходимо добавить возможность работы с классами данных, а также сохранения и чтения данных в формате XML. Код примера с приведенными классами данных:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

from dataclasses import dataclass, field
from datetime import date
import sys
from typing import List
import xml.etree.ElementTree as ET

@dataclass(frozen=True)
class Worker:
    name: str
    post: str
    year: int

@dataclass
class Staff:
    workers: List[Worker] = field(default_factory=lambda: [])

    def add(self, name, post, year):
        self.workers.append(
            Worker(
                name=name,
                post=post,
                year=year
            )
        )

        self.workers.sort(key=lambda worker: worker.name)

    def __str__(self):
        # Заголовок таблицы.
        table = []
        line = '+-{}-+-{}-+-{}-+-{}-+'.format(
            '-' * 4,
            '-' * 30,
            '-' * 20,
            '-' * 8
        )
        table.append(line)
        table.append(
            '| {:^4} | {:^30} | {:^20} | {:^8} |'.format(
                "№",

```

```

        "Ф.И.О.",
        "Должность",
        "Год"
    )
)
table.append(line)

# Вывести данные о всех сотрудниках.
for idx, worker in enumerate(self.workers, 1):
    table.append(
        '| {:>4} | {:<30} | {:<20} | {:>8} |'.format(
            idx,
            worker.name,
            worker.post,
            worker.year
        )
    )

table.append(line)

return '\n'.join(table)

def select(self, period):
    # Получить текущую дату.
    today = date.today()

    result = []
    for worker in self.workers:
        if today.year - worker.year >= period:
            result.append(worker)

    return result

def load(self, filename):
    with open(filename, 'r', encoding='utf8') as fin:
        xml = fin.read()

    parser = ET.XMLParser(encoding="utf8")
    tree = ET.fromstring(xml, parser=parser)

    self.workers = []
    for worker_element in tree:
        name, post, year = None, None, None

        for element in worker_element:
            if element.tag == 'name':
                name = element.text
            elif element.tag == 'post':
                post = element.text
            elif element.tag == 'year':
                year = int(element.text)

        if name is not None and post is not None \
            and year is not None:
            self.workers.append(
                Worker(
                    name=name,
                    post=post,
                    year=year
                )
            )

def save(self, filename):
    root = ET.Element('workers')
```

```

        for worker in self.workers:
            worker_element = ET.Element('worker')

            name_element = ET.SubElement(worker_element, 'name')
            name_element.text = worker.name

            post_element = ET.SubElement(worker_element, 'post')
            post_element.text = worker.post

            year_element = ET.SubElement(worker_element, 'year')
            year_element.text = str(worker.year)

            root.append(worker_element)

        tree = ET.ElementTree(root)
        with open(filename, 'wb') as fout:
            tree.write(fout, encoding='utf8', xml_declaration=True)

if __name__ == '__main__':
    # Список работников.
    staff = Staff()

    # Организовать бесконечный цикл запроса команд.
    while True:
        # Запросить команду из терминала.
        command = input(">>> ").lower()

        # Выполнить действие в соответствии с командой.
        if command == 'exit':
            break

        elif command == 'add':
            # Запросить данные о работнике.
            name = input("Фамилия и инициалы? ")
            post = input("Должность? ")
            year = int(input("Год поступления? "))

            # Добавить работника.
            staff.add(name, post, year)

        elif command == 'list':
            # Вывести список.
            print(staff)

        elif command.startswith('select '):
            # Разбить команду на части для выделения номера года.
            parts = command.split(maxsplit=1)
            # Запросить работников.
            selected = staff.select(int(parts[1]))

            # Вывести результаты запроса.
            if selected:
                for idx, worker in enumerate(selected, 1):
                    print(
                        '{:>4}: {}'.format(idx, worker.name)
                    )
            else:
                print("Работники с заданным стажем не найдены.")

        elif command.startswith('load '):
            # Разбить команду на части для имени файла.
            parts = command.split(maxsplit=1)
            # Загрузить данные из файла.

```

```

        staff.load(parts[1])

    elif command.startswith('save '):
        # Разбить команду на части для имени файла.
        parts = command.split(maxsplit=1)
        # Сохранить данные в файл.
        staff.save(parts[1])

    elif command == 'help':
        # Вывести справку о работе с программой.
        print("Список команд:\n")
        print("add - добавить работника;")
        print("list - вывести список работников;")
        print("select <стаж> - запросить работников со стажем;")
        print("load <имя_файла> - загрузить данные из файла;")
        print("save <имя_файла> - сохранить данные в файл;")
        print("help - отобразить справку;")
        print("exit - завершить работу с программой.")

    else:
        print(f"Неизвестная команда {command}", file=sys.stderr)

```

Результаты работы программы примера 1. Для примера были созданы несколько работников с некоторым стажем, выбраны работники с некоторым стажем и сохранены данные в файл (рис. 1).

```

>>> help
Список команд:

add - добавить работника;
list - вывести список работников;
select <стаж> - запросить работников со стажем;
load <имя_файла> - загрузить данные из файла;
save <имя_файла> - сохранить данные в файл;
help - отобразить справку;
exit - завершить работу с программой.
>>> add
Фамилия и инициалы? Петров В. Ф.
Должность? Директор
Год поступления? 2004
>>> add
Фамилия и инициалы? Белов Р. И.
Должность? Менеджер
Год поступления? 2007
>>> add
Фамилия и инициалы? Румянцев П. Н.
Должность? Бухгалтер
Год поступления? 2008

```

```

>>> select 10
1: Белов Р. И.
2: Петров В. Ф.
3: Румянцев П. Н.
>>> list
+-----+-----+-----+-----+
| № | Ф.И.О. | Должность | Год |
+-----+-----+-----+-----+
| 1 | Белов Р. И. | Менеджер | 2007 |
| 2 | Петров В. Ф. | Директор | 2004 |
| 3 | Румянцев П. Н. | Бухгалтер | 2008 |
+-----+-----+-----+-----+
>>> save предприятие
>>> exit

```

Рисунок 1 – Результаты работы программы примера 1 с классами данных

После сохранения списка работников был создан файл формата XML с этим списком (рис. 2).

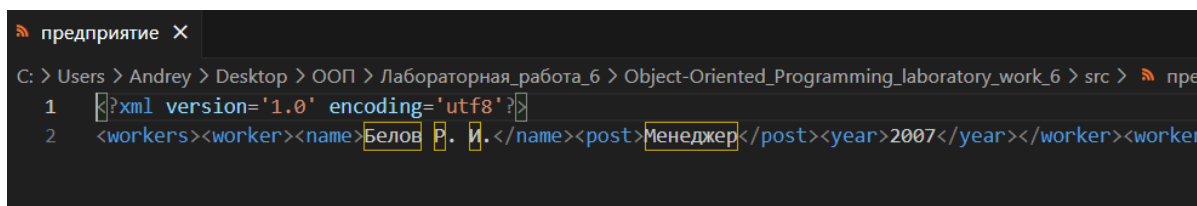


Рисунок 2 – Содержимое файла формата XML

Выполнение индивидуальных заданий:

Вариант 25

Задание 1.

Необходимо выполнить индивидуальное задание лабораторной работы 4.5, используя классы данных, а также загрузку и сохранение данных в формат XML.

Лабораторная работа 2.19:

Необходимо использовать словарь, содержащий следующие ключи:

- название пункта назначения;
- номер поезда;
- время отправления.

Написать программу, выполняющую следующие действия: ввод с клавиатуры данных в список, состоящий из словарей заданной структуры; записи должны быть упорядочены по времени отправления поезда; вывод на экран информации о поездах, направляющихся в пункт, название которого

введено с клавиатуры; если таких поездов нет, выдать на дисплей соответствующее сообщение (Вариант 26 (7), работа 2.8).

Код программы индивидуального задания 1 с использованием классов данных и сохранением файла в формате XML:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import logging
import os
import sys
import xml.etree.ElementTree as ET
from dataclasses import dataclass, field
from typing import List

class UnknownCommandError(Exception):
    """
    Класс пользовательского исключения в случае,
    если введенная команда является недопустимой.
    Сообщения об этой ошибке записываются в журнал (лог-файл) "trains.log".
    """

    def __init__(self, command: str, message: str = "Неизвестная команда") ->
None:
        """
        Конструктор класса пользовательского исключения.
        При создании экземпляра исключения возможна запись в лог
        для сохранения информации об ошибочной команде.

        :param command: Неверная команда;
        :param message: Сообщение об ошибке.
        """
        self.command = command
        self.message = message
        super().__init__(message)

    def __str__(self) -> str:
        """
        Метод вывода сообщения об ошибке (магический метод).

        :return: Неверная команда и сообщение об ошибке.
        """
        return f"{self.command} -> {self.message}"

@dataclass
class Train:
    """
    Датакласс, описывающий информацию о поезде.
    """

    departure_point: str
    number_train: str
    time_departure: str
    destination: str

class TrainManager:
```

Класс для управления списком поездов. Предоставляет методы для:

- добавления поезда (add_train),
- отображения всех поездов (list_trains),
- выборки поездов по пункту назначения (select_trains),
- загрузки поездов из XML (load_from_xml),
- сохранения поездов в XML (save_to_xml).

Все основные действия (добавление, загрузка, сохранение) сопровождаются записью

в журнал (лог-файл) "trains.log".

"""

```
def __init__(self) -> None:
```

"""

Инициализировать пустой список поездов.

"""

```
self.trains: List[Train] = field(default_factory=list)
```

```
# Прямое присвоение, поскольку dataclasses.field() нельзя просто так  
# использовать внутри __init__ без дополнительных приёмов:
```

```
self.trains = []
```

```
def add_train(
```

```
self, departure_point: str, number_train: str,
```

```
time_departure: str, destination: str
```

```
) -> None:
```

"""

Добавить информацию о поезде в список.

Добавленный поезд – это объект dataclass Train:

departure_point, number_train, time_departure, destination.

После добавления список поездов упорядочивается по времени отправления.

:param departure_point: Пункт отправления;

:param number_train: Номер поезда;

:param time_departure: Время отправления;

:param destination: Пункт назначения.

После успешного добавления поезда информация вносится в лог-файл.

"""

```
new_train = Train(
```

```
departure_point=departure_point,
```

```
number_train=number_train,
```

```
time_departure=time_departure,
```

```
destination=destination,
```

```
)
```

```
self.trains.append(new_train)
```

```
self.trains.sort(key=lambda t: t.time_departure)
```

```
logging.info(
```

```
f"Добавлен поезд: пункт отправления={departure_point}, "
```

```
f"№={number_train}, время={time_departure}, "
```

```
f"пункт назначения={destination}"
```

```
)
```

```
def list_trains(self) -> List[Train]:
```

"""

Вернуть текущий список поездов.

:return: Список поездов.

"""

```
return self.trains
```

```
def select_trains(self, point_user: str) -> List[Train]:
```



```

"""
Выбрать поезда, пункт назначения которых совпадает с point_user.
Результат выборки логируется (указывается, сколько найдено поездов).

:param point_user: Пункт назначения (введенный пользователем).
:return: Список таких поездов (может быть пустым).
"""
point_user_lower = point_user.lower()
selected = [
    train
    for train in self.trains
    if train.destination.lower() == point_user_lower
]
logging.info(
    f"Выполнен поиск поездов по пункту назначения='{point_user}'. "
    f"Найдено {len(selected)} поезд(а).")
)
return selected

def load_from_xml(self, filename: str) -> None:
    """
    Загрузить список поездов из указанного файла в формате XML.
    Если файл отсутствует, список остаётся пустым или прежним.
    При успешной загрузке записывается соответствующее сообщение в лог.
    Если файл не существует, в лог также добавляется предупреждение.

    :param filename: Имя файла для загрузки.
    """
    if os.path.exists(filename):
        with open(filename, "r", encoding="utf-8") as f:
            xml_data = f.read()

        parser = ET.XMLParser(encoding="utf-8")
        root = ET.fromstring(xml_data, parser=parser)

        self.trains = []
        for train_element in root.findall("train"):
            dp_el = train_element.find("departure_point")
            nt_el = train_element.find("number_train")
            td_el = train_element.find("time_departure")
            ds_el = train_element.find("destination")

            if not (dp_el and nt_el and td_el and ds_el):
                continue

            new_train = Train(
                departure_point=dp_el.text if dp_el.text else "",
                number_train=nt_el.text if nt_el.text else "",
                time_departure=td_el.text if td_el.text else "",
                destination=ds_el.text if ds_el.text else "",
            )
            self.trains.append(new_train)

        logging.info(f"Данные успешно загружены из файла: {filename}.")
    else:
        logging.warning(f"Файл {filename} не найден. Загрузка не
        выполнена.")

def save_to_xml(self, filename: str) -> None:
    """
    Сохранить текущий список поездов в указанный файл в формате XML.
    Если файл отсутствует, создается новый с таким же именем.
    При успешном сохранении выполняется запись в лог-файл.

```

```

        :param filename: Имя файла для сохранения.
        """
        # Если расширение не .xml, то добавим.
        if not filename.endswith(".xml"):
            filename += ".xml"

        root = ET.Element("trains")
        for train in self.trains:
            train_el = ET.SubElement(root, "train")

            dp_el = ET.SubElement(train_el, "departure_point")
            dp_el.text = train.departure_point

            nt_el = ET.SubElement(train_el, "number_train")
            nt_el.text = train.number_train

            td_el = ET.SubElement(train_el, "time_departure")
            td_el.text = train.time_departure

            ds_el = ET.SubElement(train_el, "destination")
            ds_el.text = train.destination

        tree = ET.ElementTree(root)
        with open(filename, "wb") as fout:
            tree.write(fout, encoding="utf-8", xml_declaration=True)

        logging.info(f"Данные сохранены в файл: {filename}.")

def print_trains(trains: List[Train]) -> None:
    """
    Напечатать таблицу поездов в табличном формате.
    Если список пуст, выводится сообщение о пустом списке.

    :param trains: Список поездов.
    """

    if not trains:
        print("Список поездов пуст или ничего не найдено.")
        return

    line = "+-{}-+-{}-+-{}-+-{}-+-{}-+ ".format("-" * 4, "-" * 20, "-" * 13,
        "-" * 18, "-" * 20)
    print(line)
    print(
        "| {:^4} | {:^20} | {:^13} | {:^18} | {:^20} | ".format(
            "№", "Пункт отправления", "№ поезда", "Время отправления", "Пункт
назначения"
        )
    )
    print(line)
    for idx, train in enumerate(trains, 1):
        print(
            "| {:>4} | {:<20} | {:<13} | {:>18} | {:<20} | ".format(
                idx,
                train.departure_point,
                train.number_train,
                train.time_departure,
                train.destination,
            )
        )
    print(line)

```

```

def main() -> None:
    """
    Главная функция, организующая цикл взаимодействия с пользователем.
    Также решается проблема `теневого имен` (одинаковые имена в основном коде
    и методах).
    Все введённые пользователем команды, а также возникшие ошибки,
    регистрируются в лог-файле "trains.log".
    """

    logging.basicConfig(
        filename="trains.log",
        level=logging.INFO,
        format="%(asctime)s [%(levelname)s] %(message)s",
    )

    manager = TrainManager()
    logging.info("Программа запущена.")

    while True:
        try:
            command = input(">>> ").strip().lower()
            logging.info(f"Введена команда: '{command}'")

            if command == "exit":
                logging.info("Программа завершена по команде 'exit'.")
                print("Программа завершена.")
                break

            elif command == "add":
                departure_point = input("Пункт отправления? ")
                number_train = input("Номер поезда? ")
                time_departure = input("Время отправления? ")
                destination = input("Пункт назначения? ")
                manager.add_train(departure_point, number_train,
time_departure, destination)
                print("Поезд добавлен.")

            elif command == "list":
                trains_list = manager.list_trains()
                print_trains(trains_list)
                logging.info(f"Выведен список из {len(trains_list)}
поезд(ов).")

            elif command.startswith("select "):
                parts = command.split(maxsplit=1)
                point_user = parts[1]
                selected = manager.select_trains(point_user)
                print_trains(selected)

            elif command.startswith("load "):
                parts = command.split(maxsplit=1)
                filename = parts[1]
                # Теперь грузим из XML.
                manager.load_from_xml(filename)
                print(f"Данные загружены из файла {filename}.")

            elif command.startswith("save "):
                parts = command.split(maxsplit=1)
                filename = parts[1]
                # Теперь сохраняем в XML.
                manager.save_to_xml(filename)
                print(f"Данные сохранены в файл {filename}.")

            elif command == "help":

```

```

        print("Список доступных команд:")
        print("add - добавить поезд;")
        print("list - вывести список всех поездов;")
        print("select <пункт_назначения> - вывести поезда по пункту
назначения;")
        print("load <имя_файла> - загрузить данные из файла XML;")
        print("save <имя_файла> - сохранить данные в файл XML;")
        print("help - показать справку;")
        print("exit - завершить работу.")

    else:
        raise UnknownCommandError(command)

except Exception as exc:
    logging.error(f"Произошла ошибка: {exc}")
    print(f"Ошибка: {exc}", file=sys.stderr)

if __name__ == "__main__":
    main()

```

Для проверки работоспособности программы были созданы несколько поездов, выведена информация о них, выбраны поезда по пункту назначения, данные о поездах сохранены в файл формата XML, введена неверная команда и завершена работа программы (рис. 3).

```

>>> help
Список доступных команд:
add - добавить поезд;
list - вывести список всех поездов;
select <пункт_назначения> - вывести поезда по пункту назначения;
load <имя_файла> - загрузить данные из файла XML;
save <имя_файла> - сохранить данные в файл XML;
help - показать справку;
exit - завершить работу.
>>> add
Пункт отправления? Ставрополь
Номер поезда? 001
Время отправления? 10:00
Пункт назначения? Ростов
Поезд добавлен.
>>> add
Пункт отправления? Михайловск
Номер поезда? 002
Время отправления? 11:40
Пункт назначения? Грозный
Поезд добавлен.
>>> add
Пункт отправления? Омск
Номер поезда? 003
Время отправления? 22:00
Пункт назначения? Москва
Поезд добавлен.

```

```
>>> list
+-----+-----+-----+-----+-----+
| № | Пункт отправления | № поезда | Время отправления | Пункт назначения |
+-----+-----+-----+-----+-----+
| 1 | Ставрополь | 001 | 10:00 | Ростов |
+-----+-----+-----+-----+-----+
| 2 | Михайловск | 002 | 11:40 | Грозный |
+-----+-----+-----+-----+-----+
| 3 | Омск | 003 | 22:00 | Москва |
+-----+-----+-----+-----+-----+

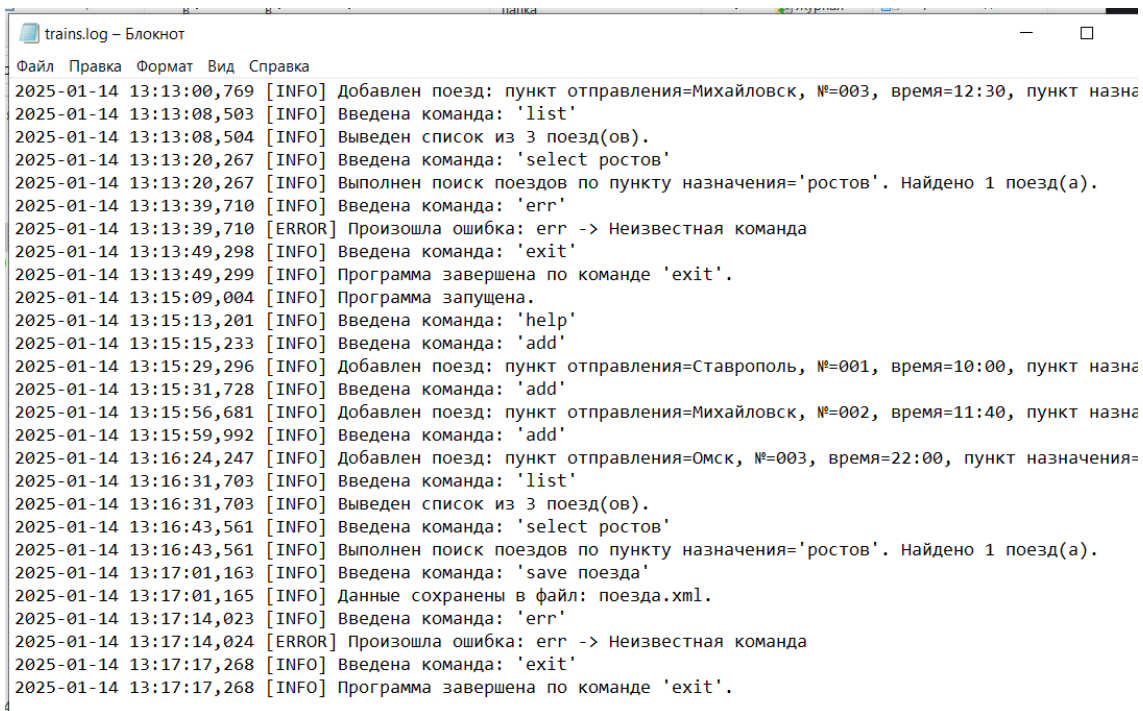
>>> select Ростов
+-----+-----+-----+-----+-----+
| № | Пункт отправления | № поезда | Время отправления | Пункт назначения |
+-----+-----+-----+-----+-----+
| 1 | Ставрополь | 001 | 10:00 | Ростов |
+-----+-----+-----+-----+-----+

>>> save поезда
Данные сохранены в файл поезда.

>>> err
>>> Ошибка: err -> Неизвестная команда
exit
Программа завершена.

Process finished with exit code 0
```

Рисунок 3 – Результаты работы программы



```
trains.log - Блокнот
Файл Правка Формат Вид Справка
2025-01-14 13:13:00,769 [INFO] Добавлен поезд: пункт отправления=Михайловск, №=003, время=12:30, пункт назна
2025-01-14 13:13:08,503 [INFO] Введена команда: 'list'
2025-01-14 13:13:08,504 [INFO] Выведен список из 3 поезд(ов).
2025-01-14 13:13:20,267 [INFO] Введена команда: 'select ростов'
2025-01-14 13:13:20,267 [INFO] Выполнен поиск поездов по пункту назначения='ростов'. Найдено 1 поезд(а).
2025-01-14 13:13:39,710 [INFO] Введена команда: 'err'
2025-01-14 13:13:39,710 [ERROR] Произошла ошибка: err -> Неизвестная команда
2025-01-14 13:13:49,298 [INFO] Введена команда: 'exit'
2025-01-14 13:13:49,299 [INFO] Программа завершена по команде 'exit'.
2025-01-14 13:15:09,004 [INFO] Программа запущена.
2025-01-14 13:15:13,201 [INFO] Введена команда: 'help'
2025-01-14 13:15:15,233 [INFO] Введена команда: 'add'
2025-01-14 13:15:29,296 [INFO] Добавлен поезд: пункт отправления=Ставрополь, №=001, время=10:00, пункт назна
2025-01-14 13:15:31,728 [INFO] Введена команда: 'add'
2025-01-14 13:15:56,681 [INFO] Добавлен поезд: пункт отправления=Михайловск, №=002, время=11:40, пункт назна
2025-01-14 13:15:59,992 [INFO] Введена команда: 'add'
2025-01-14 13:16:24,247 [INFO] Добавлен поезд: пункт отправления=Омск, №=003, время=22:00, пункт назначения=
2025-01-14 13:16:31,703 [INFO] Введена команда: 'list'
2025-01-14 13:16:31,703 [INFO] Выведен список из 3 поезд(ов).
2025-01-14 13:16:43,561 [INFO] Введена команда: 'select ростов'
2025-01-14 13:16:43,561 [INFO] Выполнен поиск поездов по пункту назначения='ростов'. Найдено 1 поезд(а).
2025-01-14 13:17:01,163 [INFO] Введена команда: 'save поезда'
2025-01-14 13:17:01,165 [INFO] Данные сохранены в файл: поезда.xml.
2025-01-14 13:17:14,023 [INFO] Введена команда: 'err'
2025-01-14 13:17:14,024 [ERROR] Произошла ошибка: err -> Неизвестная команда
2025-01-14 13:17:17,268 [INFO] Введена команда: 'exit'
2025-01-14 13:17:17,268 [INFO] Программа завершена по команде 'exit'.
```

Рисунок 4 – Содержимое лог-файла

```

<!-- XML has been converted to have any style information -->
</xml>

<trains>
  <train>
    <departure_point>Ставрополь</departure_point>
    <number_train>001</number_train>
    <time_departure>10:00</time_departure>
    <destination>Ростов</destination>
  </train>
  <train>
    <departure_point>Михайловск</departure_point>
    <number_train>002</number_train>
    <time_departure>11:40</time_departure>
    <destination>Грозный</destination>
  </train>
  <train>
    <departure_point>Омск</departure_point>
    <number_train>003</number_train>
    <time_departure>22:00</time_departure>
    <destination>Москва</destination>
  </train>
</trains>

```

Рисунок 5 – Содержимое файла формата XML

Данная программа была проверена на соответствие типов (аннотация типов) с помощью утилиты «myru» (рис. 6).

```

(lab-oop-6-py3.12) PS C:\Users\Andrey\Desktop\ООП\Лабораторная_работа_6\Object-Oriented_Programming_laboratory_work_6> mypy src/individual_1.py
Success: no issues found in 1 source file
(lab-oop-6-py3.12) PS C:\Users\Andrey\Desktop\ООП\Лабораторная_работа_6\Object-Oriented_Programming_laboratory_work_6>

```

Рисунок 6 – Использование утилиты «myru»

Для проверки работоспособности некоторых методов класса и выпадения исключений в программе был создан unit-тест. Результаты тестирования программы индивидуального задания 1 (рис. 7).

```

(lab-oop-6-py3.12) PS C:\Users\Andrey\Desktop\ООП\Лабораторная_работа_6\Object-Oriented_Programming_laboratory_work_6> pytest -v
===== test session starts =====
platform win32 -- Python 3.12.8, pytest-8.3.4, pluggy-1.5.0 -- C:\Users\Andrey\AppData\Local\pypoetry\Cache\virtualenvs\lab-oop-6-7aBZN_TU-py3.12\Scripts\python.exe
cachedir: .pytest_cache
rootdir: C:\Users\Andrey\Desktop\ООП\Лабораторная_работа_6\Object-Oriented_Programming_laboratory_work_6
configfile: pytest.ini
collected 3 items

tests/test_individual_1.py::TestTrainManager::test_add_and_list_trains PASSED [ 33%]
tests/test_individual_1.py::TestTrainManager::test_select_trains PASSED [ 66%]
tests/test_individual_1.py::TestTrainManager::test_unknown_command_error PASSED [100%]

===== 3 passed in 0.05s =====

```

Рисунок 7 – Результаты тестирования программы

Ссылка на репозиторий данной лабораторной работы:

https://github.com/AndreyPust/Object-Oriented_Programming_laboratory_work_6.git

Ответы на контрольные вопросы:

1. Как создать класс данных в языке Python?

Классы данных (или датаклассы) в Python можно создать с помощью декоратора «@dataclass» из модуля «dataclasses». Декоратор «@dataclass»

автоматически добавляет полезные методы в класс, такие как «init», «repr», «eq», и другие, что облегчает работу с данными.

2. Какие методы по умолчанию реализует класс данных?

Декоратор «@dataclass» автоматически добавляет следующие методы, если они не были определены вручную:

- init : Конструктор, который инициализирует поля, используя аргументы, переданные при создании объекта.
- repr : Представление объекта в виде строки, удобно для его просмотра в консоли.
- eq : Метод сравнения, позволяющий сравнивать два объекта на равенство по значению их полей.
- lt , le , gt , ge : Методы для выполнения операций сравнения <, <=, >, >=, если в @dataclass указана опция order=True.
- hash : Если класс помечен как неизменяемый, также будет добавлен метод hash для использования объектов в качестве ключей словаря или элементов множества.

3. Как создать неизменяемый класс данных?

Чтобы создать неизменяемый класс данных, нужно использовать параметр «frozen=True» в декораторе «@dataclass». Этот параметр запрещает изменение полей после создания объекта, делая его неизменяемым (immutable).

Вывод: в ходе выполнения лабораторной работы были приобретены навыки по работе с классами данных при написании программ с помощью языка программирования Python версии 3.x.