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»
Вариант 1

Выполнил:
Бабенко Артём Тимофеевич
3 курс, группа ИВТ-б-о-22-1,
09.03.01 «Информатика и вычислительная техника»,
направленность (профиль) «Программное обеспечение средств вычислительной техники и автоматизированных систем», очная форма обучения

_____
(подпись)

Проверил:
Ассистент департамента цифровых, робототехнических систем и электроники Богданов С.С

_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2024 г.

Тема: Классы данных в Python

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

Порядок выполнения работы:

Пример 1.

Результат работы программы:

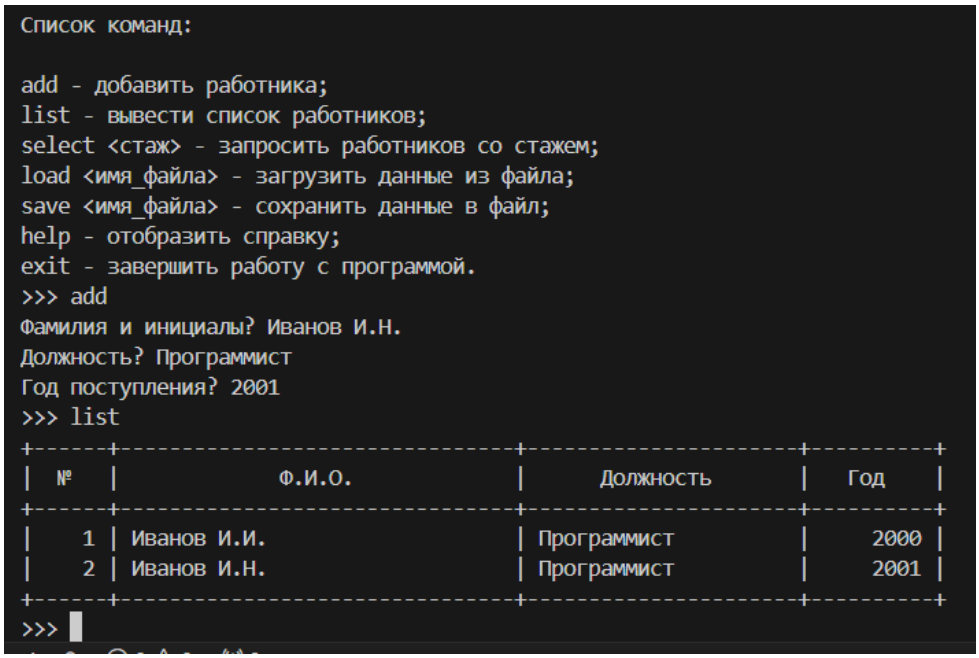

Рисунок 1 – Результат работы программы

Код программы:

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


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

# Класс пользовательского исключения в случае, если неверно
# введен номер года.

class IllegalYearError(Exception):
    def __init__(self, year, message="Illegal year number"):
        self.year = year
        self.message = message
        super(IllegalYearError, self).__init__(message)
```

```
    def __str__(self):
        return f"{self.year} -> {self.message}"


# Класс пользовательского исключения в случае, если введенная
# команда является недопустимой.
class UnknownCommandError(Exception):
    def __init__(self, command, message="Unknown command"):
        self.command = command
        self.message = message
        super(UnknownCommandError, self).__init__(message)
    def __str__(self):
        return f"{self.command} -> {self.message}"


@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: str, post: str, year: int) -> None:
      # Получить текущую дату.
      today = date.today()
      if year < 0 or year > today.year:
          raise IllegalYearError(year)
      self.workers.append(
          Worker(
              name=name,
              post=post,
              year=year
          )
      )
      self.workers.sort(key=lambda worker: worker.name)
  def __str__(self) -> str:
      # 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: int) -> List[Worker]:
    # Получить текущую дату.
    today = date.today()
    result: List[Worker] = []
    for worker in self.workers:
        if today.year - worker.year >= period:
            result.append(worker)
    return result
def load(self, filename: str) -> None:
  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: str) -> None:
  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__':
        # Выполнить настройку логгера.
        logging.basicConfig(
            filename='workers4.log',
            level=logging.INFO
        )
        # Список работников.
        staff = Staff()
        # Организовать бесконечный цикл запроса команд.
        while True:
            try:
                # Запросить команду из терминала.
                command = input(">>> ").lower()
                # Выполнить действие в соответствие 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)
                    logging.info(
                        f"Добавлен сотрудник: {name}, {post}, "
                        f"поступивший в {year} году."
                    )
                elif command == 'list':
                    # Вывести список.
                    print(staff)
                    logging.info("Отображен список сотрудников.")
                elif command.startswith('select '):
                    # Разбить команду на части для выделения номера года.
                    parts = command.split(maxsplit=1)
                    # Запросить работников.
                    selected = staff.select(parts[1])
```

```
            # Вывести результаты запроса.
            if selected:
                for idx, worker in enumerate(selected, 1):
                    print(
                        '{:>4}: {}'.format(idx, worker.name)
                    )
                    logging.info(
                    f"Найдено {len(selected)} работников со "
                    f"стажем более {parts[1]} лет."
                )
            else:
                print("Работники с заданным стажем не найдены.")
                logging.warning(
                    f"Работники со стажем более {parts[1]} лет 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])
            logging.info(f"Загружены данные из файла {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])
            logging.info(f"Сохранены данные в файл {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:
            raise UnknownCommandError(command)
    except Exception as exc:
        logging.error(f"Ошибка: {exc}")
        print(exc, file=sys.stderr)
```

Задание 1. Выполнить индивидуальное задание лабораторной работы 4.5, использовав классы данных, а также загрузку и сохранение данных в формат XML.

Результат работы программы:

```
D:\Gitlab\0-op\0-op6\.venv\Scripts\python.exe D:\Gitlab\0-op\0-op6\individ.py
b1:         10101010
b2:         10101010
Size b1:    8
b1[2]:      1
AND:        00000000
OR:         00000000
XOR:        00000000
NOT b1:     00000000
Shift L 2:  00000000
Shift R 2:  00000000
After count=6: 101010

XML representation:
<BitString><size>8</size><count>6</count><bits>101010</bits></BitString>
Loaded from XML: 00000000

Process finished with exit code 0
```

Рисунок 2 – Результат работы программы

Код программы:

```
from dataclasses import dataclass, field
from typing import List
import xml.etree.ElementTree as ET
from io import StringIO

@dataclass
class BitString:
    """Класс для работы с битовыми строками"""
    MAX_SIZE: int = field(default=100, init=False)  # Максимальный размер как константа
    size: int = 0                                   # Максимальный размер строки
    bits: List[int] = field(default_factory=list)   # Список бит
    count: int = 0                                  # Текущее количество элементов

    def __post_init__(self) -> None:
        """Инициализация после создания объекта"""
        if isinstance(self.size, int) and self.size > 0:
            if self.size > self.MAX_SIZE:
                raise ValueError(f"Размер должен быть от 1 до {self.MAX_SIZE}")
```

```
            self.bits = [0] * self.size
            self.count = self.size
        elif isinstance(self.size, str):
            bit_string = self.size
            if len(bit_string) > self.MAX_SIZE:
                raise ValueError(f"Длина строки не должна превышать {self.MAX_SIZE}")
            for char in bit_string:
                if char not in '01':
                    raise ValueError("Строка должна содержать только 0 и 1")
            self.size = len(bit_string)
            self.bits = [int(char) for char in bit_string]
            self.count = self.size

    def __str__(self) -> str:
        """Строковое представление битовой строки"""
        return ''.join(str(bit) for bit in self.bits[:self.count])

    def __getitem__(self, index: int) -> int:
        """Чтение по индексу"""
        if not isinstance(index, int):
            raise TypeError("Индекс должен быть целым числом")
        if index < 0 or index >= self.count:
            raise IndexError(f"Индекс должен быть от 0 до {self.count - 1}")
        return self.bits[index]

    def __setitem__(self, index: int, value: int) -> None:
        """Запись по индексу"""
        if not isinstance(index, int):
            raise TypeError("Индекс должен быть целым числом")
        if index < 0 or index >= self.count:
            raise IndexError(f"Индекс должен быть от 0 до {self.count - 1}")
        if value not in (0, 1):
            raise ValueError("Значение должно быть 0 или 1")
        self.bits[index] = value

    def get_size(self) -> int:
        """Возвращает максимальный размер"""
        return self.size

    def set_count(self, new_count: int) -> None:
        """Установка текущего количества элементов"""
        if not isinstance(new_count, int):
            raise TypeError("Count должен быть целым числом")
        if new_count < 0 or new_count > self.size:
            raise ValueError(f"Count должен быть от 0 до {self.size}")
        if new_count > self.count:
            self.bits[self.count:new_count] = [0] * (new_count - self.count)
        self.count = new_count

    def __and__(self, other: 'BitString') -> 'BitString':
        """Операция AND"""
        if not isinstance(other, BitString) or self.count != other.count:
```

```
            raise ValueError("Операнды должны иметь одинаковое количество элементов")
        return BitString(self.count, [a & b for a, b in zip(self.bits, other.bits)], self.count)

    def __or__(self, other: 'BitString') -> 'BitString':
        """Операция OR"""
        if not isinstance(other, BitString) or self.count != other.count:
            raise ValueError("Операнды должны иметь одинаковое количество элементов")
        return BitString(self.count, [a | b for a, b in zip(self.bits, other.bits)], self.count)

    def __xor__(self, other: 'BitString') -> 'BitString':
        """Операция XOR"""
        if not isinstance(other, BitString) or self.count != other.count:
            raise ValueError("Операнды должны иметь одинаковое количество элементов")
        return BitString(self.count, [a ^ b for a, b in zip(self.bits, other.bits)], self.count)

    def __invert__(self) -> 'BitString':
        """Операция NOT"""
        return BitString(self.size, [1 - bit for bit in self.bits[:self.count]], self.count)

    def shift_left(self, n: int) -> 'BitString':
        """Сдвиг влево"""
        if not isinstance(n, int) or n < 0:
            raise ValueError("Сдвиг должен быть неотрицательным целым числом")
        if n >= self.count:
            return BitString(self.size, [0] * self.count, self.count)
        return BitString(self.size, self.bits[n:self.count] + [0] * n, self.count)

    def shift_right(self, n: int) -> 'BitString':
        """Сдвиг вправо"""
        if not isinstance(n, int) or n < 0:
            raise ValueError("Сдвиг должен быть неотрицательным целым числом")
        if n >= self.count:
            return BitString(self.size, [0] * self.count, self.count)
        return BitString(self.size, [0] * n + self.bits[:self.count - n], self.count)

    def to_xml(self) -> str:
        """Сохранение в XML-строку"""
        root = ET.Element("BitString")
        ET.SubElement(root, "size").text = str(self.size)
        ET.SubElement(root, "count").text = str(self.count)
        bits_elem = ET.SubElement(root, "bits")
        bits_elem.text = ''.join(str(bit) for bit in self.bits[:self.count])
        return ET.tostring(root, encoding='unicode', method='xml')

    @classmethod
    def from_xml(cls, xml_str: str) -> 'BitString':
        """Загрузка из XML-строки"""
        root = ET.fromstring(xml_str)
        size = int(root.find("size").text)
        count = int(root.find("count").text)
        bits_str = root.find("bits").text
        bits = [int(bit) for bit in bits_str]
```

```
        if len(bits) != count:
            raise ValueError("Несоответствие count и длины bits")
        if size < count:
            raise ValueError("Size не может быть меньше count")
        return cls(size, bits, count)


# Пример использования
if __name__ == "__main__":
    # Создание объектов
    b1 = BitString(8)          # Через размер
    b2 = BitString("10101010")  # Через строку

    # Установка значений
    b1[0] = 1
    b1[2] = 1
    b1[4] = 1
    b1[6] = 1

    # Демонстрация операций
    print(f"b1:         {b1}")
    print(f"b2:         {b2}")
    print(f"Size b1:    {b1.get_size()}")
    print(f"b1[2]:      {b1[2]}")
    print(f"AND:        {b1 & b2}")
    print(f"OR:         {b1 | b2}")
    print(f"XOR:        {b1 ^ b2}")
    print(f"NOT b1:     {~b1}")
    print(f"Shift L 2:  {b1.shift_left(2)}")
    print(f"Shift R 2:  {b1.shift_right(2)}")

    # Изменение count
    b1.set_count(6)
    print(f"After count=6: {b1}")

    # Сохранение в XML
    xml_data = b1.to_xml()
    print("\nXML representation:")
    print(xml_data)

    # Загрузка из XML
    b3 = BitString.from_xml(xml_data)
    print(f"Loaded from XML: {b3}")
```

Ответы на контрольные вопросы:

1. Как создать класс данных в языке Python?

В Python для создания класса данных используется конструкция @dataclass из модуля dataclasses.

Пример:

```
from dataclasses import dataclass
@dataclass
class Point:
    x: int
    y: int
```

2. Какие методы по умолчанию реализует класс данных?

Класс данных автоматически реализует следующие методы:

__init__() — конструктор для инициализации полей.

__repr__() — строковое представление объекта.

__eq__() — метод сравнения объектов на равенство.

__hash__() — метод для получения хэш-значения (если класс изменяемый).

3. Как создать неизменяемый класс данных?

Для создания неизменяемого класса данных, необходимо установить параметр frozen=True в декораторе @dataclass.

Пример:

```
from dataclasses import dataclass
@dataclass(frozen=True)
class Point:
    x: int
    y: int
```

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