Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования «Ульяновский государственный технический университет»

Кафедра «Вычислительная техника»

**Отчет по лабораторной работе №4**

Дисциплина: «Разработка профессиональных приложений»

Классы

Вариант №8

Выполнил:

студент группы ИВТАСбд-21

Зюзин Г.А.

Проверил:

преподаватель кафедры

«Вычислительная техника»

Исхаков И.И.

Ульяновск, 2023

**Задание по варианту.**

Необходимо переделать лабораторную работу №3 с использованием классов, описывающих предметную область, заданную вариантом, с реализацией следующих особенностей:  
1. Класс должен содержать итератор;  
2. Должна быть реализована перегрузка стандартных операций (repr, например);  
3. Должно быть реализовано наследование;  
4. Запись значений в свойства - только через \_\_setattr\_\_;  
5. Возможность доступа к элементам коллекции по индексу (\_\_getitem\_\_);  
6. Должны быть реализованы статические методы;  
7. Должны быть реализованы генераторы.

В 3-й лабораторной работе было следующее: История посещений поликлиники: №, ФИО пациента, ФИО врача, причина обращения, длительность.

**Описание реализации.**

В рамках данной лабораторной работы была разработана программа, которая предоставляет функциональность для работы с историей посещений поликлиники. Для этого были созданы два класса - Record и RecordHospital.

Класс Record является базовым классом, содержащим атрибут idx. Он также определяет методы для работы с записями.

Класс RecordHospital наследуется от класса Record и расширяет его, добавляя дополнительные атрибуты, такие как name\_patient, name\_doctor, reason и diraction. Класс RecordHospital также определяет статические методы для преобразования времени посещения в различные форматы и обратно.

Класс Data предоставляет функциональность для работы с данными. Он содержит атрибуты path, table и item, а также определяет методы для чтения данных из файла, добавления новой записи, сортировки данных, доступа к элементам коллекции по индексу и другие операции. Класс Data также реализует генераторы и итератор для перебора записей.

**Описание возникших затруднений**.

В процессе выполнения работы возникли некоторые затруднения при понимании принципов работы итераторов и генераторов, а также при определении модели наследования для классов.

**Описание альтернативных способов решения.**

В процессе выполнения лабораторной работы не было найдено альтернативных способов решения задачи. Использованный подход с обходом матрицы является достаточно.

**Листинг кода.**

import csv  
  
class Record():  
 idx = 0  
  
 def \_\_init\_\_(self, idx: int):  
 self.\_\_setattr\_\_("idx", idx)  
  
# Класс наследуемый от класс Record, который содержить в себе атрибут Record и имеет свои атрибуты  
class RecordHospital(Record):  
 idx = 0  
 name\_patient = ""  
 name\_doctor = ""  
 reason = ""  
 diraction = 0  
  
 # Атрибуты иницилизируются только с помощью \_\_setatter\_\_  
 def \_\_init\_\_(self, idx: int, name\_patient: str, name\_doctor: str, reason: str, diraction: int):  
 super().\_\_init\_\_(idx)  
 self.\_\_setattr\_\_("name\_patient", name\_patient)  
 self.\_\_setattr\_\_("name\_doctor", name\_doctor)  
 self.\_\_setattr\_\_("reason", reason)  
 self.\_\_setattr\_\_("diraction", self.get\_format(diraction))  
  
 # Налсожение ограничений на \_\_setatter\_\_, чтобы не создавать лишние атрибуты  
 def \_\_setattr\_\_(self, key, value):  
 if not(key in ['idx', 'name\_patient', 'name\_doctor', 'reason', 'diraction']):  
 raise KeyError("Не существет такого атрибута")  
 else:  
 self.\_\_dict\_\_[key] = value  
  
 # Пример распределинея стандартного метода  
 def \_\_repr\_\_(self):  
 return f"RecordHospital(idx={self.idx}, name\_patient={self.name\_patient}, " \  
 f"name\_doctor={self.name\_doctor}, reason={self.reason}, diraction={self.diraction})"  
  
 @staticmethod  
 # Функция для преобразовния секунд в формат минута:секунда  
 def get\_format(seconds: int):  
 minutes = str(seconds // 60)  
 seconds = str(seconds % 60)  
  
 if int(minutes) < 10:  
 minutes = '0' + minutes  
 if int(seconds) < 10:  
 seconds = '0' + seconds  
  
 return f"{minutes}:{seconds}"  
  
 @staticmethod  
 # Функция преобразования формата в секунды  
 def get\_seconds(format):  
 minutes, seconds = map(int, format.split(":"))  
 seconds += minutes \* 60  
  
 return seconds  
  
class Data():  
 path = ""  
 table = []  
 item = 0  
  
 def \_\_init\_\_(self, path):  
 self.path = path  
 self.table = self.parse(self.path)  
  
 def \_\_iter\_\_(self):  
 return self  
  
 # Функция класса для добавление новой записи в таблицу. Всё то же, что и в 3 работе  
 def add\_new\_record(self):  
 new\_record = RecordHospital(len(self.table), input("Имя пациента: "), input("Имя врача: "),  
 input("Причина обращения: "), int(input("Время приёма: ")))  
 with open(self.path, "a") as f:  
 writer = csv.DictWriter(f, fieldnames=['idx', 'name\_patient', 'name\_doctor', 'reason', 'diraction'])  
 writer.writerow(new\_record.\_\_dict\_\_)  
 f.close()  
  
 self.table = self.parse(self.path)  
  
 # Реализация генератора  
 def my\_generator(self):  
 self.item = 0  
  
 while self.item < len(self.table):  
 yield self.table[self.item]  
 self.item += 1  
  
 def \_\_repr\_\_(self):  
 return f"Data({[repr(rm) for rm in self.table]})"  
  
 # Реализация функции \_\_next\_\_  
 def \_\_next\_\_(self):  
 if self.item >= len(self.table):  
 self.item = 0  
 raise StopIteration  
 else:  
 self.item += 1  
 return self.table[self.item - 1]  
  
 def \_\_getitem\_\_(self, item):  
 if not isinstance(item, int):  
 raise TypeError("Индекс должен быть целым числом.")  
  
 if 0 <= item < len(self.table):  
 return self.table[item]  
 else:  
 raise IndexError("Выход за границы списка.")  
  
 def \_\_str\_\_(self):  
 table\_str = '\n'.join([str(rh) for rh in self.table])  
 return f"Записи:\n{table\_str}"  
  
 @staticmethod  
 # Функция для разбиения таблицы на записи  
 def parse(path):  
 res = []  
 with open(path, "r") as f:  
 reader = csv.DictReader(f, fieldnames=['idx', 'name\_patient', 'name\_doctor', 'reason', 'diraction'])  
 for record in reader:  
 res.append(dict(record))  
 f.close()  
  
 return res  
  
 # Сортировка различные варианты  
 def my\_sort(self, way\_sort):  
 if way\_sort == 0:  
 self.table.sort(key=lambda x: int(x['id'])) # Сортировка по id  
 elif way\_sort == 1:  
 self.table.sort(key=lambda x: x['name\_patient']) # Сортировка по имени пациента  
 elif way\_sort == 2:  
 self.table.sort(key=lambda x: x['name\_doctor']) # Сортировка по врача  
 elif way\_sort == 3:  
 self.table.sort(key=lambda x: x['reason']) # Сортировка по причине обращения  
 elif way\_sort == 4:  
 self.table.sort(key=lambda x: x['diraction']) # Сортировка по времени приёма (по возрастанию)  
 else:  
 print("Такого метода сортировки нет")  
  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 d = Data("lab4.csv")  
  
 print("1. Класс должен содержать итератор:\n")  
 for i in d.\_\_iter\_\_():  
 print(i)  
  
 print("\n2. Должна быть реализована перегрузка стандартных операций: \n")  
 print(d.\_\_repr\_\_())  
 print(d.\_\_str\_\_())  
  
 print("\n3. Должно быть реализовано наследование:\nКласс RecordHospital наследуется от Record\n")  
  
 print("\n4. Запись значений в свойства - только через \_\_setattr\_\_:\nСмотреть выше в коде\n")  
  
 print("\n5. Возможность доступа к элементам коллекции по индексу (\_\_getitem\_\_):\n")  
 print(d.\_\_getitem\_\_(1))  
  
 print("\n6. Должны быть реализованы статические методы: \nСмотреть выше в коде\n")  
  
 print("\n7. Должны быть реализованы генераторы\n")  
 for i in d.my\_generator():  
 print(i)  
  
 WRITE\_RECORD = False # Сделать запись в csv файл  
 OUTPUT\_RECORDS = True # Вывести записи, выбрав сбособ сортировки  
 SELECT\_RECORDS = True # Вывести записи по одному из критериев  
  
 if WRITE\_RECORD:  
 d.add\_new\_record()  
  
 if OUTPUT\_RECORDS:  
 print("Отсоритровать записпи по\n0 - id\n1 - ФИО пациента\n"  
 "2 - ФИО врача\n3 - причине обращения\n4 - длительности посещения")  
 way = int(input())  
 d.my\_sort(way)  
  
 for line in d.table:  
 print(line)  
  
 if SELECT\_RECORDS:  
 for line in d.table:  
 if (line['reason'] == "Temperature"):  
 print(line)

**Вывод.**

В результате работы была разработана программа, основанная на концепциях объектно-ориентированного программирования. Были созданы классы Record и RecordHospital, расширяющие базовый класс Record. Применены принципы наследования, перегрузки операторов и использования методов доступа к свойствам. Класс Data обеспечивает работу с данными и реализует генераторы и итераторы. Выводом является функциональная и эффективная программа для работы с историей посещений поликлиники.