Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Ульяновский государственный технический университет»

Кафедра «Вычислительная техника»

Разработка профессиональных приложений

(название дисциплины)

**Отчёт по лабораторной работе №4**

«Классы»

(название (тема) работы)

**Вариант №4**

Выполнил:

студент группы ИВТАПбд-21

Галацков И.А.

Проверил(а):

преподаватель кафедры «ВТ»

Исхаков И. И.

Ульяновск

2023

**Постановка задачи**

Необходимо переделать лабораторную работу №3 с использованием классов, описывающих предметную область, заданную вариантом, с реализацией следующих особенностей (вполне возможно, что предлагаемое в 3 лабе задание для этого нужно будет расширить):

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

**Описание реализации**

За основу был взят исходный код лабораторной работы №3. Был реализован родительский класс человек, в котором реализованы метод инициализации экземпляров класса, методы сеттер для получения имени, пола и возраста, а также методы геттер, для изменения этих свойств. Был реализован дочерний класс пользователь, наследованный от родительского класса человек. В этом классе был реализован переопределенный метод вывода информации об объекте класса пользователь, реализован генератор для перебора списка дат совершения операций. Также бал реализован класс кастомного итератора для класса пользователь. Далее по аналогии с третьей лабораторной работой, открываем csv-файл для чтения, далее с помощью класса DictReader модуля csv создается объект, который работает как обычный reader(), но отображает информацию в каждой строке в качестве словаря dict. Далее создаем объект класса пользователя и добавляем его в список, сортируем список при помощи лямбда-функции. Создаем файл для записи и заносим туда результат.

**Исходный код реализации**

import csv # Библиотека импорта и экспорта для электронных таблиц и баз данных  
import datetime  
  
  
# Родительский класс "Человек"  
class Person:  
 # Метод инициализации экземпляров класса после их создания  
 def \_\_init\_\_(self, name, age, gender):  
 self.\_\_name = name  
 self.\_\_age(age)  
 self.\_\_gender = gender  
  
 def \_\_age(self, value):  
 if value <= 120:  
 self.\_\_age = value  
  
 # Статические метод проверки на совершеннолетие  
 @staticmethod  
 def is\_adult(age):  
 return age >= 18  
  
 # Метод (сеттер) для получения имени человека  
 def setName(self, newName):  
 self.\_\_name = newName  
  
 # Метод (сеттер) для получения возраста человека  
 def setAge(self, newAge):  
 if newAge <= 120:  
 self.\_\_age = newAge  
  
 # Метод (сеттер) для получения пола человека  
 def setGender(self, newGender):  
 if (newGender == "Male") or (newGender == "Female"):  
 self.\_\_gender = newGender  
  
 # Метод (геттер) для изменения имени человека  
 def getName(self):  
 return self.\_\_name  
  
 # Метод (геттер) для изменения возраста человека  
 def getAge(self):  
 return self.\_\_age  
  
 # Метод (геттер) для изменения пола человека  
 def getGender(self):  
 return self.\_\_gender  
  
  
# Дочерний класс "Пользователь", наследованный от родительского класса "Человек"  
class User(Person):  
 def \_\_init\_\_(self, name, age, gender, n, date, time, sum, assessment\_description):  
 Person.\_\_init\_\_(self, name, age, gender)  
 self.\_\_n = n  
 self.\_\_dates = []  
 self.login(date)  
 self.\_\_time = time  
 self.\_\_sum = sum  
 self.\_\_assessment\_description = assessment\_description  
  
 # Метод, который вызывается при соверщении операции  
 def login(self, date):  
 self.\_\_dates.append(date)  
  
 # Кастомный итератор класса (описание ниже)  
 def \_\_iter\_\_(self):  
 return UserDateIterator(self.\_\_dates)  
  
 # Переопределённый метод вывода информации об объекте класса "Пользователь"  
 def \_\_repr\_\_(self):  
 # Если сегодня пользователь совершал операцию  
  
 if (datetime.datetime.now().strftime("%d.%m.%Y") == self.\_\_dates[len(self.\_\_dates) - 1]):  
 # Значит он онлайн (время не учитывается)  
 return f"{self.\_\_n}. Name = {self.getName()}, age = {self.getAge()}, Time = {self.\_\_time}, Sum = {self.\_\_sum}"  
 else:  
 # Иначе он оффлайн (выводим, когда пользователь совершал операцию последний раз)  
 return f"{self.\_\_n}. Name = {self.getName()}, age = {self.getAge()} Sum = {self.\_\_sum}, Date = {self.\_\_dates[len(self.\_\_dates) - 1]}, {self.\_\_time})"  
  
 # Генератор для перебора списка дат совершения операций  
 def generator(self, i):  
 while i < len(self.\_\_dates):  
 yield self.\_\_dates[i] # Ключевое слово yield возвращает нас обратно в этот метод, запоминая состояние выхода  
 i += 1  
 else:  
 yield 0  
  
 # Метод присвоения значения value какому-либо атрибуту key объекта класса "Пользователь"  
 def \_\_setattr\_\_(self, key, value):  
 self.\_\_dict\_\_[key] = value  
  
 # Метод позволяет получить значение какого-либо атрибута объекта класса "Пользователь" по его индексу  
 def \_\_getitem\_\_(self, item):  
 s = "\_User\_\_" + item  
 return self.\_\_dict\_\_[s]  
  
  
# Класс кастомного итератора для класса "Пользователь"  
class UserDateIterator():  
 # При создании итератора, получаем список дат выполненныйх опреаций и инициализируем счётчик для его обхода  
 def \_\_init\_\_(self, dates):  
 self.dates = dates  
 self.i = 0  
  
 # Метод возвращает следующий элемент из списка посещений  
 def \_\_next\_\_(self):  
 if self.i < len(self.dates):  
 date = self.dates[self.i]  
 self.i += 1  
 return date  
 else:  
 return 0  
  
  
fI = open("C:\\Users\\Molniya\\Desktop\\data.csv", 'r') # Открываем файл для чтения (read)  
# Класс DictReader модуля csv создаёт объект, который работает как обычный reader(), но отображает информацию о каждой строке в качестве словаря dict  
reader = csv.DictReader(fI, fieldnames = None, restkey = None, restval = None, dialect = "excel")  
  
l = []  
  
for row in reader:  
 # Создаём объект класса пользователя и добавляем его в список  
 a = User(row["Name"], int(row["Age"]), row["Gender"], int(row['N']), row["Date"], row["Time"], int(row["Sum"]), row["Assessment\_description"])  
 l.append(a)  
  
print("Исходные данные: ")  
print(l)  
  
l[2].setAge(17)  
  
l[2].login("17.02.2023")  
  
print(l)  
print()  
  
# Сортируем список при помощи лямбда-функции по столбцу с именем из таблицы  
sSL = sorted(l, key = lambda d: d.getName())  
# Сортируем список при помощи лямбда-функции по столбцу с номером клиента из таблицы и переворачиваем список (reverse)  
nSL = sorted(l, key = lambda d: d['n'], reverse = True)  
  
print("Сортировка по строковому полю (имени): ")  
print(sSL)  
print("Сортировка по числовому полю (номеру посетителя) в обратном порядке (reverse): ")  
print(nSL)  
  
newL = []  
  
# Отбор пользователей по совершеннолетию (только те, кому больше 17 лет)  
newL = list(filter(lambda x: x.is\_adult(x.getAge()), l))  
print("Совершеннолетние пользователи: ")  
print(newL)  
  
# Создаём файл для записи  
with open("C:\\Users\\Molniya\\Desktop\\data.csv", 'w', newline = '') as csvfile:  
 writer = csv.DictWriter(csvfile, fieldnames = reader.fieldnames) # Класс writer служит для записи данных в файл  
 writer.writeheader() # Записываем заголовок таблицы  
  
 for row in sSL:  
 d = {}  
 d['N'] = row['n'] # Доступ к элементам коллекции по индексу осуществляется при помощи метода \_\_getitem\_\_  
 d['Date'] = row['dates'][-1]  
 d['Time'] = row['time']  
 d['Sum'] = row['sum']  
 d['Assessment\_description'] = row['assessment\_description']  
 d['Gender'] = row.getGender()  
 d['Age'] = row.getAge()  
 d['Name'] = row.getName()  
  
 # Метод writerow записывает словарь со значениями в файл  
 writer.writerow(d)  
  
 for row in nSL:  
 d = {}  
 d['N'] = row['n']  
  
 # Запись даты через итератор класса  
 iterator = iter(row)  
 b = next(iterator)  
 d['Date'] = str(b)  
  
 while (b):  
 b = next(iterator)  
  
 if (b != 0):  
 d['Date'] = str(b)  
 #  
  
 d['Time'] = row['time']  
 d['Sum'] = row['sum']  
 d['Assessment\_description'] = row['assessment\_description']  
 d['Gender'] = row.getGender()  
 d['Age'] = row.getAge()  
 d['Name'] = row.getName()  
  
 writer.writerow(d)  
  
 for row in newL:  
 d = {}  
 d['N'] = row['n']  
  
 # Запись даты через генератор класса  
 i = 0  
  
 generator = row.generator(i)  
 b = next(generator)  
 d['Date'] = str(b)  
  
 while (b):  
 b = next(generator)  
  
 if (b != 0):  
 d['Date'] = str(b)  
 #  
  
 d['Time'] = row['time']  
 d['Sum'] = row['sum']  
 d['Assessment\_description'] = row['assessment\_description']  
 d['Gender'] = row.getGender()  
 d['Age'] = row.getAge()  
 d['Name'] = row.getName()  
  
 writer.writerow(d)  
  
print()  
print("Пожалуйста, проверьте выходной файл")

**Тестирование**

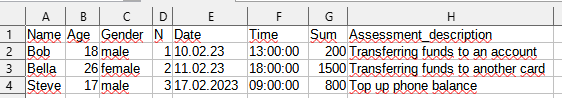


Рис. 1 Файл для чтения

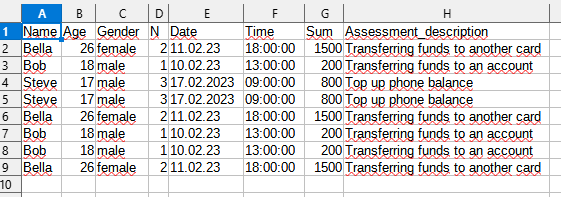


Рис. 2 Выходной файл

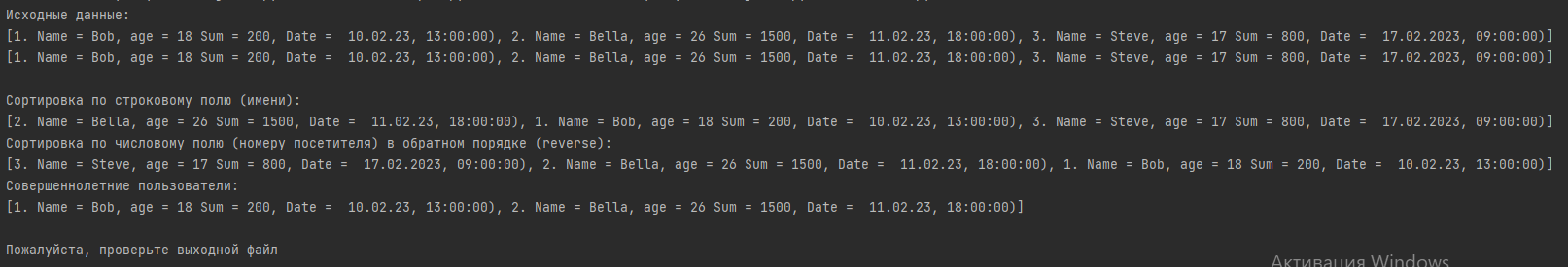


Рис. 3 Вывод в консоль

**Описание возникших затруднений**

При выполнении данной лабораторной работы возникли затруднения, так как я редко использую ООП, мне нужно было основательно повторить материал. Также я не знал, что такое генераторы и с нуля изучал данную тему.

**Вывод**

В ходе выполнения лабораторной работы, были улучшены навыки работы с классами, приобретен опыт работы с ООП на языке программирования Python. Данную работу можно считать выполненной, так как все требования были выполнены, написанная программа работает исправно и выполняет все поставленные задачи.