**Лабораторна робота №5**

**Введення-виведення(I/O). Файли. Виключення**

**Тема**: Розробка та реалізація програм із використанням класів для зберігання даних

та файлового введення-виведення

**Мета роботи**: отримати навички створення та реалізації програм, що реалізують

операції введення-виведення із файлами.

**Завдання**

1. На основі завдання з лабораторної роботи №3 (частина1), створити абстрактний

клас, в якому визначити абстрактні поля, що характеризують об’єкт, наприклад,

одиниці вимірювання, кількість в упаковці та інше.

2. Визначити метод, що створює порожній масив об’єктів, з максимальною

кількістю 100 елементів.

2. Реалізувати зберігання даних у текстовий файл та зчитування з текстового

файлу. Враховувати можливість виникнення виключень, та обробляти їх, виводячи

відповідні повідомлення для користувача.

3. Створити інтерактивне меню, за допомогою якого надати можливість

користувачеві виконувати додавання нових та вилучення існуючих даних, файлові

операції введення-виведення та запити відповідно варіанту завдання.

**Короткі теоретичні відомості**

Сохранение объектов в текстовом формате

Рассмотрим пример программы, сохраняющей массив записей типа Employee в

текстовом файле. Каждая запись сохраняется в отдельной строке. Поля экземпляра

отделяются друг от друга разделителем. В качестве разделителя в данном примере

используется знак вертикальной черты (|). Другим распространенным разделителем

является знак двоеточия (:).

Мы пока что не будем касаться того, что может произойти, если символ |

встретится непосредственно в одной из сохраняемых символьных строк. Ниже

приведен образец сохраняемых записей:

Harry Hacker|35500|1989|10|1

Carl Cracker|75000|1987|12|15

Tony Tester|38000|1990|3|15

Процесс записи происходит очень просто. Для этой цели применяется класс

PrintWriter, поскольку запись выполняется в текстовый файл. Все поля просто

записываются в файл, завершаясь символом |, а если это последнее поле, то

комбинацией символов \n. Весь процесс записи совершается в теле приведенного

ниже метода writeData(), который описан в классе Employee.

public void writeData(PrintWriter out) throws IOException

{

GregorianCalendar calendar = new GregorianCalendar();

calendar.setTime(hireDay);

out.println(name + "|"

+ salary + "|"

+ calendar.get(Calendar.YEAR) + "|"

+ (calendar.get(Calendar.MONTH) + 1) + "|"

+ calendar.get(Calendar.DAY\_OF\_MONTH));

}

Файлы

Приведенный выше пример целесообразно применять к текстовыми файлами умеренной длины. Если же файл крупный или двоичный, то для обращения с ним можно воспользоваться потоками ввода-вывода или чтения и записи

данных, как показано ниже. Их методы избавляют от необходимости обращаться непосредственно к классам FilelnputStream, FileOutputStream, BufferedReader или

BufferedWriter.

*Розв’язання*

1. **Текст програми**

package com.labs.lab5.ELib.models.storage;

import java.io.\*;

import java.lang.reflect.Array;

import java.util.\*;

import java.util.function.Predicate;

import java.util.stream.Collectors;

/\*\*

\* Класс для хранения данных на основании текстового файла

\* @param <T> класс хранимых данных

\*/

public class BinaryStorage<T> implements IStorage<T> {

//Текстовый файл для хранения данных

private File dataFile;

// Класс хранимих данных

private Class<T> dataClass;

// Массив хранимых данных

private ArrayList<T> data = new ArrayList<>();

public BinaryStorage(String url, Class<T> dataClass) throws IOException {

setDataClass(dataClass);

initFile(url);

load();

}

@Override

public void add(T item) throws IOException {

data.add(item);

save();

}

@Override

public void addAll(T[] items) throws IOException {

data.addAll(Arrays.asList(items));

save();

}

@Override

public ArrayList<T> getData() {

return data;

}

@Override

public ArrayList<T> getData(Predicate<T> filter) {

return data.stream().filter(filter)

.collect(Collectors.toCollection(ArrayList::new));

}

@Override

public T[] getArrOfData() {

return data.toArray(\_getTArray(data.size()));

}

@Override

public T[] getArrOfData(Predicate<T> filter) {

return data.stream().filter(filter).toArray(this::\_getTArray);

}

@Override

public void remove(T item) throws IOException {

remove(el -> el.equals(item));

}

@Override

public void remove(Predicate<T> isRemoved) throws IOException {

data = data.stream().filter(item -> !isRemoved.test(item))

.collect(Collectors.toCollection(ArrayList::new));

save();

}

@Override

public void replace(T prevItem, T newItem) throws IOException {

Collections.replaceAll(data, prevItem, newItem);

save();

}

/\*\*

\* Сохраняет переданные элементы в бинарном файле

\*/

private void save() throws IOException {

try (FileOutputStream fos = new FileOutputStream(dataFile);

ObjectOutputStream out = new ObjectOutputStream(fos)) {

out.writeObject(data);

}

}

/\*\*

\* Загружает данные из текстового файла в массив данных

\*/

private void load() throws IOException {

dataFile.createNewFile();

try(FileInputStream fis = new FileInputStream(dataFile);

ObjectInputStream ois = new ObjectInputStream(fis)) {

data = (ArrayList<T>)ois.readObject();

} catch (EOFException err) {

data = new ArrayList<>();

} catch (ClassNotFoundException err) {

err.printStackTrace();

}

}

private void initFile(String url) {

dataFile = new File(url);

// Создает родительские каталоги

dataFile.getParentFile().mkdirs();

}

/\*\*

\* Создает массив new T[len]

\*

\* @param len длина массива

\* @return массив данных

\* <p>

\*/

private T[] \_getTArray(int len) {

@SuppressWarnings("unchecked")

var arr = (T[]) Array.newInstance(dataClass, len);

return arr;

}

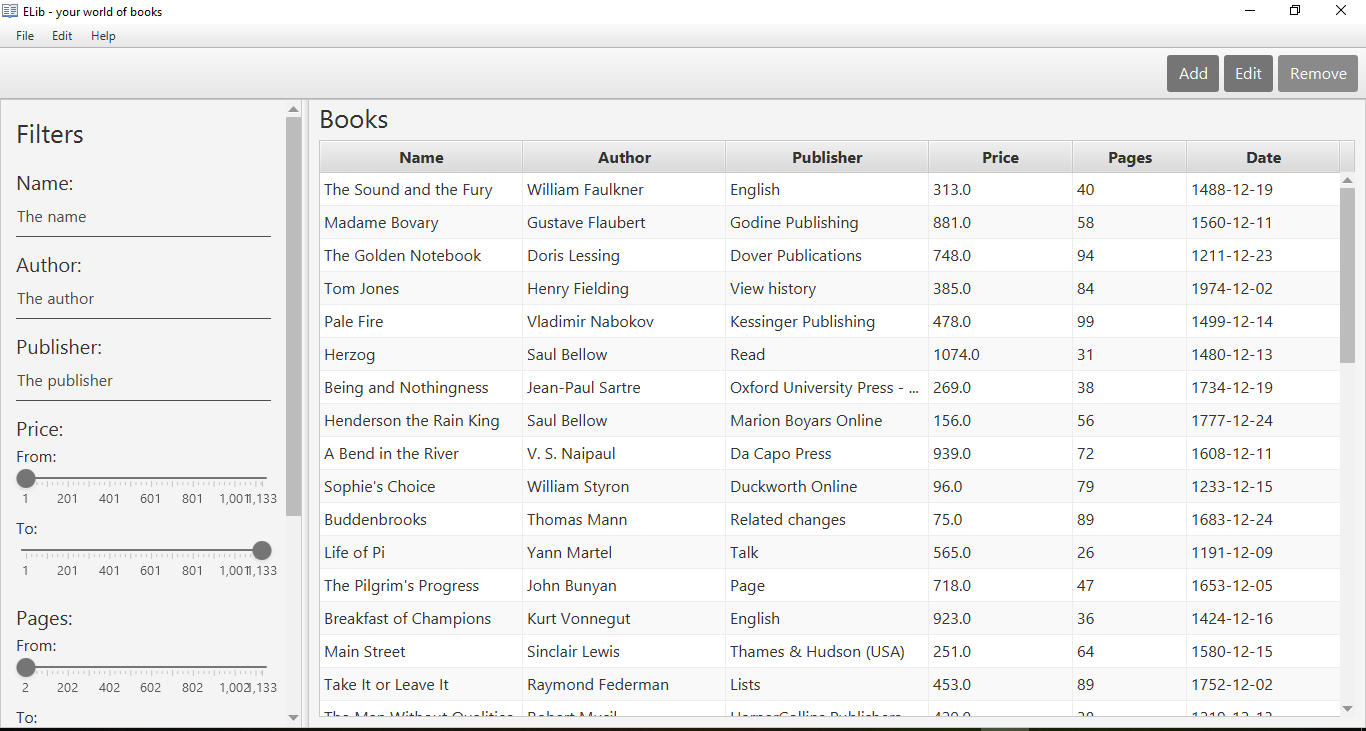
private void setDataClass(Class<T> dataClass) {

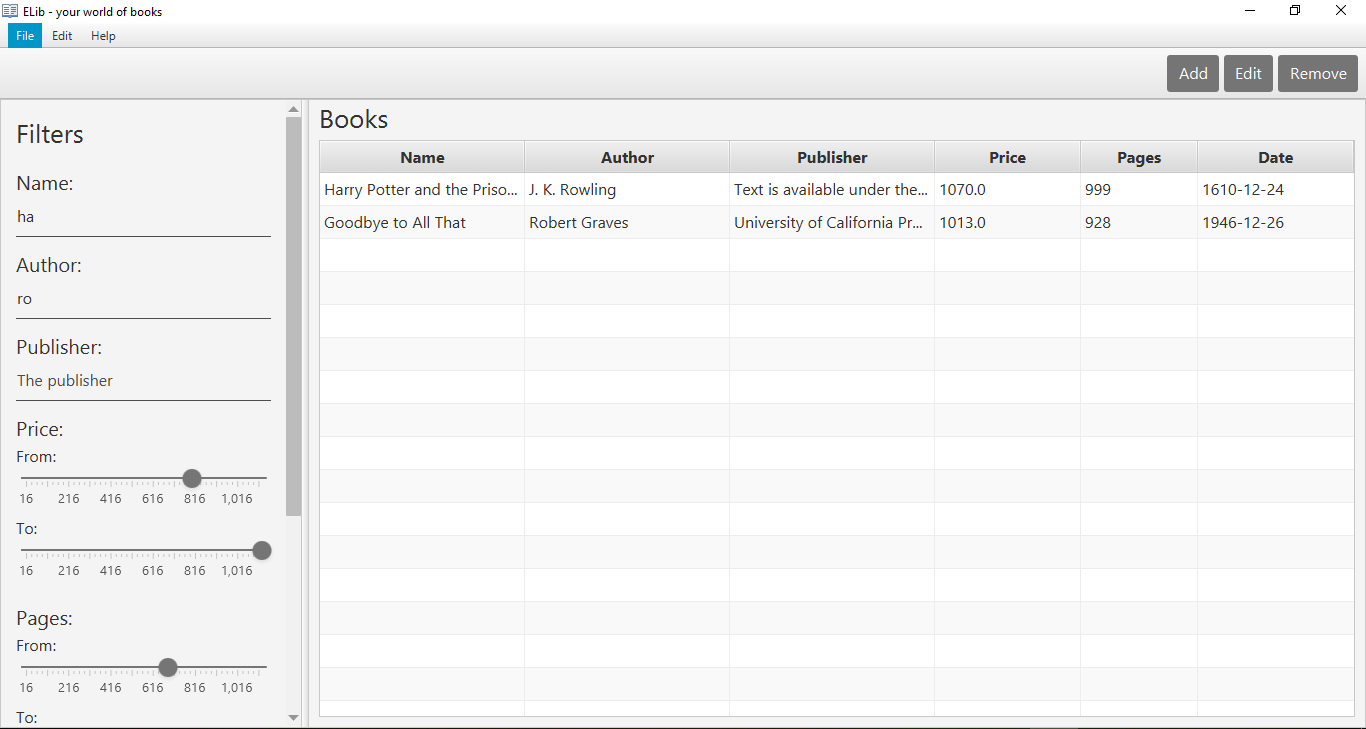
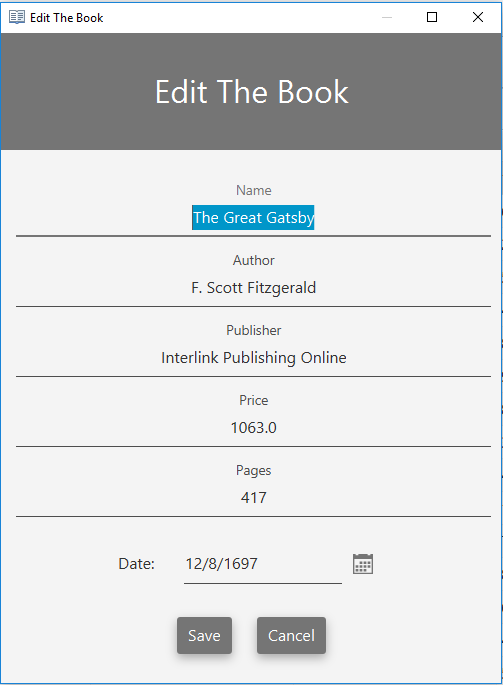
this.dataClass = dataClass;

}

}

1. Результат виконання програми:



**Висновок:** на цій лабораторній роботі я отримав навички створення та реалізації програм, що реалізують операції введення-виведення із файлами.