Лабораторная работа №4

Ввод и вывод информации в файлы

Цель работы: изучение методов работы с текстовыми и бинарными файлами.

Файловые потоки

FileInputStream

Обобщенное понятие источника ввода относится к различным способам получения информации: к чтению дискового файла, символов с клавиатуры, либо получению данных из сети. Аналогично, под обобщенным понятием вывода также могут пониматься дисковые файлы, сетевое соединение и т.п. Эти абстракции дают удобную возможность для работы с вводом-выводом (I/O), не требуя при этом, чтобы каждая часть вашего кода понимала разницу между, скажем, клавиатурой и сетью. В Java эта абстракция называется потоком (**stream**) и реализована в нескольких классах пакета *java.io*. Ввод инкапсулирован в классе **InputStream**, вывод — в **OutputStream**. В Java есть несколько специализаций этих абстрактных классов, учитывающих различия при работе с дисковыми файлами, сетевыми соединениями и даже с буферами в памяти.

InputStream — абстрактный класс, задающий используемую в Java модель входных потоков. Все методы этого класса при возникновении ошибки генерируют исключение *IOException*. Ниже приведен краткий обзор методов класса **InputStream**.

Класс **FileInputStream** используется для ввода данных из файлов. Для создания объекта **FileInputStream** мы можем использовать ряд конструкторов. Наиболее используемая версия конструктора в качестве параметра принимает путь к считываемому файлу:

```
FileInputStream(String fileName) throws FileNotFoundException
```

Если файл не может быть открыт, например, по указанному пути такого файла не существует, то генерируется исключение **FileNotFoundException**.

При завершении работы с потоком его надо закрыть с помощью метода **close()**, который определен в интерфейсе **Closeable**. Метод **close()** имеет следующее определение:

```
void close() throws IOException
```

Этот интерфейс уже реализуется в классах **InputStream** и **OutputStream**, а через них и во всех классах потоков.

При закрытии потока освобождаются все выделенные для него ресурсы, например, файл. В случае, если поток окажется не закрыт, может происходить утечка памяти.

Начиная с Java 7 можно использовать еще один способ, который автоматически вызывает метод **close**. Этот способ заключается в использовании конструкции **try-with-resources** (try-с-ресурсами). Данная конструкция работает с объектами, которые реализуют интерфейс **AutoCloseable**. Так как все классы потоков реализуют интерфейс

Closeable, который в свою очередь наследуется от **AutoCloseable**, то их также можно использовать в данной конструкции.

Синтаксис конструкции следующий:

try*(название_класса имя_переменной = конструктор_класса).*

Данная конструкция также не исключает использования блоков **catch**. После окончания работы в блоке **try** у ресурса (в данном случае у объекта **FileInputStream**) автоматически вызывается метод **close()**.

Если нам надо использовать несколько потоков, которые после выполнения надо закрыть, то мы можем указать объекты потоков через точку с запятой

```
import java.io.FileInputStream;
import java.io.IOException;

public class FileTest {

public static void main(String[] args) {
  try(FileInputStream inFile = new FileInputStream("D://Dir//notes.txt")){
    int i = -1;
    while((i = inFile.read()) != -1){
      System.out.print((char)i);
    }
  }
  catch(IOException ex){
    System.out.println(ex.getMessage());
  }
}
```

Запись файлов и класс FileOutputStream

Класс **FileOutputStream** предназначен для записи байтов в файл. Он является производным от класса **OutputStream**, поэтому наследует всю его функциональность.

Через конструктор класса **FileOutputStream** задается файл, в который производится запись. Класс поддерживает несколько конструкторов:

```
FileOutputStream(String filePath)
FileOutputStream(File fileObj)
FileOutputStream(String filePath, boolean append)
FileOutputStream(File fileObj, boolean append)
```

Файл задается либо через строковый путь, либо через объект File. Второй параметр - append задает способ записи: если он равен true, то данные дозаписываются в конец файла, а при false - файл полностью перезаписывается

```
public static void main(String[] args)  {
  String text = "Hello world!";
  try(FileOutputStream fileOut = new FileOutputStream("D://Dir//notes.txt")) {
    byte[] buffer = text.getBytes();
    fileOut.write(buffer, 0, buffer.length);
  }
  catch(IOException ex){
    System.out.println(ex.getMessage());
  }
  System.out.println("The file has been written");
}
```

Для создания объекта **FileOutputStream** используется конструктор, принимающий в качестве параметра путь к файлу для записи. Если такого файла нет, то он автоматически создается при записи. Так как здесь записываем строку, то ее надо сначала перевести в массив байтов. И с помощью метода **write** строка записывается в файл.

Совместим оба класса и выполним чтение из одного и запись в другой файл

```
try(FileInputStream fin = new FileInputStream("D://Dir//notes.txt");
      FileOutputStream fos=new FileOutputStream("D://Dir//notes_new.txt")){
  byte[] buffer = new byte[fin.available()];
  fin.read(buffer, 0, buffer.length);
  fos.write(buffer, 0, buffer.length);
}
catch(IOException ex){
  System.out.println(ex.getMessage());
}
```

Классы **FileInputStream** и **FileOutputStream** предназначены прежде всего для записи двоичных файлов, то есть для записи и чтения байтов. И хотя они также могут использоваться для работы с текстовыми файлами, но все же для этой задачи больше подходят другие классы, которые являются наследниками абстрактных классов **Reader** и **Writer**.

Запись файлов. Класс FileWriter

Класс FileWriter является производным от класса Writer. Он используется для записи текстовых файлов.

Чтобы создать объект FileWriter, можно использовать один из следующих конструкторов:

```
FileWriter(File file)
FileWriter(File file, boolean append)
FileWriter(String fileName)
FileWriter(String fileName, boolean append)
```

Так, в конструктор передается либо путь к файлу в виде строки, либо объект File, который ссылается на конкретный текстовый файл. Параметр append указывает, должны ли данные дозаписываться в конец файла (если параметр равен true), либо файл должен перезаписываться. В конструкторе использовался параметр **append** со значением **false** – то есть файл будет перезаписываться. Затем с помощью методов, определенных в базовом классе **Writer** производится запись данных.

Запишем в файл какой-нибудь текст:

```
try(FileWriter writer = new FileWriter("notes3.txt", false)){
  String text = "Hello World!";
  writer.write(text);
  writer.append('\n');
  writer.append("Hello students");
  writer.flush();
}
catch(IOException ex){
  System.out.println(ex.getMessage());
}
```

Чтение файлов. Класс FileReader

Класс FileReader наследуется от абстрактного класса Reader и предоставляет функциональность для чтения текстовых файлов.

Для создания объекта FileReader мы можем использовать один из его конструкторов:

```
FileReader(String fileName)
FileReader(File file)
```

А используя методы, определенные в базом классе Reader, произвести чтение файла:

```
try(FileReader reader = new FileReader("notes3.txt")){
  int c;
  while((c=reader.read()) != -1){
    System.out.print((char)c);
  }
}
```

```
catch(IOException ex){
  System.out.println(ex.getMessage());
}
```

Также мы можем считывать в промежуточный буфер из массива символов:

```
try(FileReader reader = new FileReader("notes3.txt")){
  char[] buf = new char[256];
  int c;
  while((c = reader.read(buf))>0){
    if(c < 256){
      buf = Arrays.copyOf(buf, c);
    }
    System.out.print(buf);
  }
}
catch(IOException ex){
  System.out.println(ex.getMessage());
}
```

В данном случае считываем последовательно символы из файла в массив из 256 символов, пока не дойдем до конца файла в этом случае метод **read** возвратит число -1.

Поскольку считанная порция файла может быть меньше 256 символов (например, в файле всего 73 символа), и если количество считанных данных меньше размера буфера (256), то выполняем копирование массива с помощью метода **Arrays.copy**. То есть фактически обрезаем массив buf, оставляя в нем только те символы, которые считаны из файла.

Задание на лабораторную работу:

Разработать программу работы с данными, хранящимися в текстовом файле.

Организовать чтение и обработку данных из файла в соответствии с индивидуальным заданием. Сохранить полученные результаты в новый текстовый файл.

В таблице представлена структура одной записи в файле. Одной записи соответствует одна строка файла. Таких записей в файле может быть несколько.

Вариант	**Задание**
1 «Человек»	*фамилия; имя; отчество; пол; национальность; рост; вес; дата рождения (год, месяц число); номер телефона; домашний адрес (почтовый индекс, страна, область, район, город, улица, дом, квартира).*

	Вывести сведения о самом молодом человеке.
2 «Школьник»	*фамилия; имя; отчество; пол; национальность; рост; вес; дата рождения (год, месяц число); номер телефона; домашний адрес (почтовый индекс, страна, область, район, город, улица, дом, квартира); школа; класс.* Вывести сведения про всех учеников пятых классов.
3 «Студент»	*фамилия; имя; отчество; пол; национальность; рост; вес; дата рождения (год, месяц число); номер телефона; домашний адрес (почтовый индекс, страна, область, район, город, улица, дом, квартира); ВУЗ; курс; группа; средний бал; специальность.* Вывести сведения про всех студентов у которых средний балл ниже 70 баллов.
4 «Покупатель»	*фамилия; имя; отчество; пол; национальность; рост; вес; дата рождения (год, месяц число); номер телефона; домашний адрес (почтовый индекс, страна, область, район, город, улица, дом, квартира); номер кредитной карточки; банковского счета.* Вывести данные о покупателях с города Муром.
5 «Пациент»	*фамилия; имя; отчество; пол; национальность; рост; вес; дата рождения (год, месяц число); номер телефона; домашний адрес (почтовый индекс, страна, область, район, город, улица, дом, квартира); номер больницы; отделение; номер медицинской карты; диагноз; группа крови.* Вывести данные про пациентов с 18 отделения.
6 «Владелец автомобиля»	*фамилия; имя; отчество; номер телефона; домашний адрес (почтовый индекс, страна, область, район, город, улица, дом, квартира) марка автомобиля; номер автомобиля; номер техпаспорта.* Вывести данные про автомобили марки ”Ваз”.

7 «Военнослужащий»	*фамилия; имя; отчество; домашний адрес (почтовый индекс, страна, область, район, город, улица, дом, квартира); национальность; дата рождения (год, месяц число); должность; звание.* Вывести данные про военнослужащих в звании “лейтенант”.
8 «Рабочий»	*фамилия; имя; отчество; домашний адрес (почтовый индекс, страна, область, район, город, улица, дом, квартира); национальность; дата рождения (год, месяц число); No цеха; табельный номер; образование; год поступления на работу.* Вывести данные про рабочих, поступивших на работу в 2010 году.
9 «Владелец телефона»	*фамилия; имя; отчество; домашний адрес (почтовый индекс, страна, область, район, город, улица, дом, квартира); No телефона.* Вывести данные про владельцев телефона номер, которого начинается на 720.
10 «Абитуриент»	*фамилия; имя; отчество; пол; национальность; дата рождения (год, месяц число); домашний адрес (почтовый индекс, страна, область, район, город, улица, дом, квартира); оценки по экзаменам; проходной балл.* Вывести данные про абитуриентов, проходной балл которых равен больше 4 .
11 «Государство»	*название страны; столица; государственный язык; население; площадь территории; денежная единица; государственный строй; глава государства.* Вывести данные про государства, население которых больше 20 млн жителей.
12 «Автомобиль»	*марка; цвет; серийный номер; регистрационный номер; год выпуска; год техосмотра; цена.*

	Вывести данные про автомобили, которым больше 2 лет.