# Потоки ввода-вывода

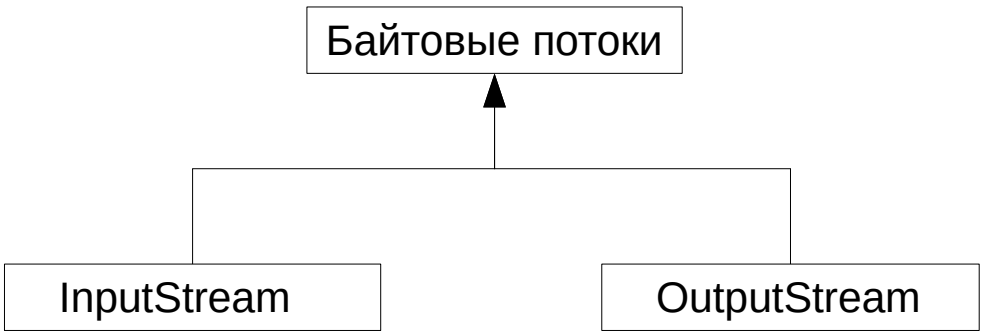
# Определение потока ввода-вывода

Поток ввода-вывода - абстракция, которая либо порождает либо принимает информацию. Поток связан с физическим устройством при помощи системы ввода-вывода. Например может быть определен поток, который связан с файлом и через который можно вести чтение или запись файла. Это также может быть поток, связанный с сетевым сокетом, с помощью которого можно получить или отправить данные в сети. Все эти задачи: чтение и запись различных файлов, обмен информацией по сети, ввод-ввывод в консоли решаются в Java с помощью потоков.

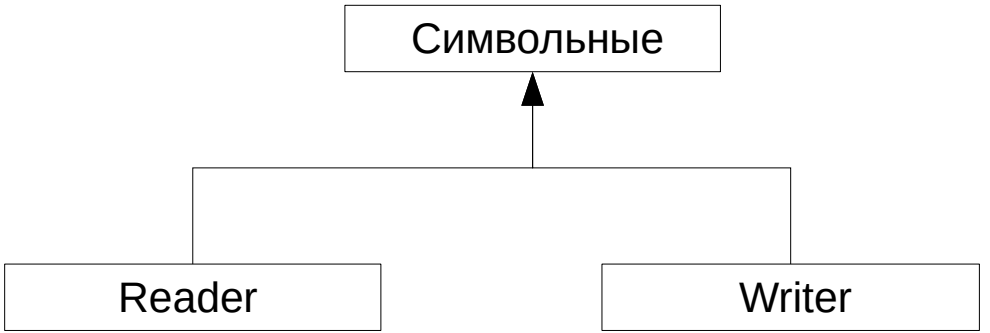
Объект, из которого можно считать данные, называется потоком ввода, а объект, в который можно записывать данные, - потоком вывода. Например, если надо считать содержание файла, то применяется поток ввода, а если надо записать в файл - то поток вывода.

Классы используемые для работы с потоками расположены в пакете java.io

# Два типа потоков в Java




Чтение (InputStream) и запись(OutputStream) данных в виде байт




Чтение (Reader) и запись(Writer) данных в виде символов

# Методы InputStream

| Метод | Описание |
|---|---|
| int available() | Вернет оценку какое количество байт может быть прочитано(или пропущено) из потока без блокирования (если для чтения нужно блокировать, то вернет 0). Если достигнут конец потока вернет 0. |
| void close() | Закрывает источник ввода и освобождает системные ресурсы связанные с потоком. |
| void mark( int readlimit) | Отмечает текущую позицию в этом входном потоке. |
| boolean markSupported() | Возвращает true если методы mark(), reset() поддерживаются вызывающим потоком. |
| abstract int read() | Возвращает представление следующего доступного байта в потоке (значение 0-255). При достижении конца потока возвращает значение -1. |
| int read(byte[] b) | Читает данные из потока в массив байт переданный в качестве параметра. Возвращает сколько было прочитано байт. Возвращает -1 при достижении конца потока. |
| int  read(byte[] b, int off, int len) | Действие аналогично предыдущему методу, однако байты записываются в массив со смещением. Параметр len указывает сколько именно байт требуется считать. |
| byte[] readAllBytes() | Считает все доступные байты из потока. Вернет массив с прочитанными данными. |
| int readNBytes(byte[] b, int off, int len) | Считывает len байт в массив со смещением of. Вернет количество прочитанных байт. Вернет -1 при достижении конца потока. |
| byte[] readNBytes(int len) | Считает len байт из потока и вернет считанные данные в виде массива. |
| void reset() | Сбрасывает входной указатель в ранее установленную метку. |
| long skip (long n) | Игнорирует n байт потока, возвращает количество действительно проигнорированных байтов. |
| long transferTo(OutputStream out) | Читает данные из входного потока и отправляет в выходной. Вернет сколько байт было вычитано и записано. |
| static InputStream nullInputStream() | Вернет поток чтения который не читает данные. |

# Методы OutputStream

| Метод | Описание |
|---|---|
| void close() | Закрывает источник ввода и освобождает системные ресурсы связанные с потоком. |
| void flush() | Указывает выполнить запись в буфере (если поддерживается буферизация). Внимание касается только классов Java. Для методов ОС гарантии нет. |
| abstract void write(int b) | Записывается младшие 8 бит параметра этого метода в поток. |
| void write(byte[] b) | Записывает в поток все содержимое массива байт. |
| void write(byte[] b, int off, int len) | Записывает в поток len байт из массива b начиная с индекса off. |
| static OutputStream nullOutputStream() | Вернет OutputStream который отбрасывает все записываемые данные. |

# Реализация InputStream для чтения из файла

Для чтения байтовой последовательности из файла используется класс FileInputStream.

Конструкторы этого класса следующие:

- FileInputStream(File file)
- FileInputStream(String name)
- FileInputStream(FileDescriptor fdObj)

При работе не забывайте закрыть файл. Для этого используйте метод close() (в блоке finally) или блок try с ресурсами (рекомендованный способ).

# Пример использования FileInputStream

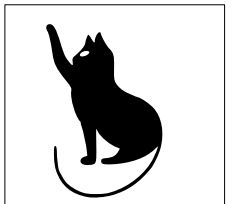
Для примера вычитаем первые 100 байт из файла с изображением. Для этого возьмем файл cat6.svg (векторное изображение кота) и FileInputStream для чтения.

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

public class Main {

	public static void main(String[] args) {
		File file = new File("cat6.svg");

		byte[] bytes = new byte[100];

		try (InputStream is = new FileInputStream(file)) {

			int readByte = is.read(bytes);
			System.out.println(readByte + " bytes read");

		} catch (IOException e) {
			e.printStackTrace();
		}
	}
}
```

Адрес файла для чтения

Массив для хранения вычитанных данных

Вычитка байт и запись их в массив



cat6.svg

# Реализация OutputStream для записи в файл

Для записи данных в файл (данные должны быть представимы в виде последовательности байт) используется класс FileOutputStream.

Конструкторы этого класса:

- FileOutputStream(File file)
- FileOutputStream(String name)
- FileOutputStream(FileDescriptor fdObj)

- FileOutputStream(File file, boolean append)
- FileOutputStream(String name, boolean append)

◄ Режим «добавления» данных в файл

При работе не забывайте закрыть файл. Для этого используйте метод close() (в блоке finally) или блок try с ресурсами (рекомендованный способ).

# Пример использования FileOutputStream

```
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.OutputStream;

public class Main {

     public static void main(String[] args) {

          String text = "Hello world";

          File file = new File("hello.txt");

          try (OutputStream os = new FileOutputStream(file)) {

               byte[] bytes = text.getBytes();

               os.write(bytes);

          } catch (IOException e) {
               e.printStackTrace();
          }
     }
}
```

Адрес файла куда производить запись

Массив байт для записи

Запись массива байт в файл

# Копирование файлов с использованием потоков ввода-вывода

При необходимости можно реализовать копирование файлов используя потоки ввода вывода. Для этого нужно создать FileInputStream для чтения данных из файла оригинала, и FileOutputStream для записи данных в файл копию. Реализация их совместной работы довольно проста, данные вычитанные с помощью FileInputStream передаются для записи в FileInputStream.

# Пример копирования используя метод transferTo (Java 9 и позже)

```
public class FileCopyService {

     public static void copyFile(File in, File out) throws IOException {

          try (InputStream is = new FileInputStream(in);
               OutputStream os = new FileOutputStream(out)) {

               long copyBytes = is.transferTo(os);

               System.out.println(copyBytes + " bytes copied");

          } catch (IOException e) {
               throw e;
          }

     }
}
```

Передача считанных данных из InputStream в OutputStream

Данный метод скопирует данные из одного файла (in) в другой (out) произведя копирование файлов. Для считывания данных используется FileInputStream для записи FileOutputStream.

# Копирование файла используя массив байт и методы read и write. Что делать когда пишешь на Java 8 и ниже.

В случае если используется Java 8 и ниже. Можно использовать следующий подход.

1) Создать объекты классов FileInputStream и FileOutputStream для копируемого файла и копии соответственно.
2) Объявить буферный массив byte. Например размером 10 Мбайт. (Размер можно менять в любую сторону на свое усмотрение)
3) В цикле сначала вычитывать байты в этот массив из файла, потом записывать в копию, и так до конца файла.

# Пример копирования с использованием методов read и write

```
public class FileCopyService {
     public static void copyFile(File in, File out) throws IOException {

          byte[] buffer = new byte[10_000_000];
          int readBytes = 0;
          long totalCopyByte = 0;

          try (InputStream is = new FileInputStream(in);
               OutputStream os = new FileOutputStream(out)) {

               for (;;) {
                    readBytes = is.read(buffer);
                    if (readBytes <= 0) {
                         break;
                    }
                    os.write(buffer, 0, readBytes);
                    totalCopyByte += readBytes;
               }
               System.out.println(totalCopyByte + " bytes copied");

          } catch (IOException e) {
               throw e;
          }
     }
}
```

Массив играющий роль буфера

Читаем данные из файла оригинала в буфер

Пишем данные в файл копию из буфера

# Буферизация байтовых потоков ввода вывода

Буферизация (от англ. buffer) — способ организации обмена, в частности, ввода и вывода данных в компьютерах и других вычислительных устройствах, который подразумевает использование буфера для временного хранения данных. При вводе данных одни устройства или процессы производят запись данных в буфер, а другие - чтение из него, при выводе - наоборот. Процесс, выполнивший запись в буфер, может немедленно продолжать работу, не ожидая, пока данные будут обработаны другим процессом, которому они предназначены. В свою очередь, процесс, обработавший некоторую порцию данных, может немедленно прочитать из буфера следующую порцию. Таким образом, буферизация позволяет процессам, производящим ввод, вывод и обработку данных, выполняться параллельно, не ожидая, пока другой процесс выполнит свою часть работы.

Применение буферизации означает что обращение к функциям ОС будет выполняться реже (меньше итоговых блокировок), что ускорит выполнение программы. К недостаткам можно отнести повышенный (относительно) расход памяти. Чаще всего роль подобного буфера играет массив.

# Классы реализующие буферизацию данных

В Java существует несколько классов реализующих буферизацию. Для байтовых потоков это классы BufferedInputStream и BufferedOutputStream. Эти классы в качестве полей содержат массив байт который и играет роль буфера. Поток данные которого вы хотите буферизировать нужно передать как параметр конструктора.

Конструкторы BufferedInputStream:

- BufferedInputStream(InputStream in)
- BufferedInputStream(InputStream in, int size)

Конструкторы BufferedOutputStream:

- BufferedOutputStream(OutputStream out)
- BufferedOutputStream(OutputStream out, int size)

# Пример использования буферизации

```
public static void main(String[] args) {
     File file = new File("utf8_sample.txt");
     byte[] startBytes = new byte[] { (byte) 0xEF, (byte) 0xBB, (byte) 0xBF };
     try {
          System.out.println(hasStartByte(startBytes, file));
     } catch (IOException e) {
          e.printStackTrace();
     }
}

public static boolean hasStartByte(byte[] startBytes, File file) throws IOException {

try (BufferedInputStream in = new BufferedInputStream(new FileInputStream(file))) {
          for (int i = 0; i < startBytes.length; i++) {
               byte b = (byte) in.read();
               if (startBytes[i] != b) {
                    return false;
               }
          }
          return true;
          } catch (IOException e) {
               throw e;
          }
     }
}
```

Передаем InputStream в конструктор

# Считывание запись значений примитивных типов в байтовые потоки

В ряде случаев возникает необходимость в чтении или записи из байтовых потоков значений примитивных типов (int, long, double и т. д.). Для этого можно использовать классы DataOutputStream и DataInputStream. Основной функционал этих классов заключается в конвертации значений примитивного типа в массивы байт с последующей записью в байтовый поток и вычитка данный из байтового потока и создание на их основе значений примитивных типов соответственно.

Конструктор DataOutputStream:

- DataOutputStream(OutputStream out)

Конструктор DataInputStream:

- DataInputStream(InputStream in)

В обеих случаях нужно передать в конструктор байтовый поток для записи или чтения в конструктор класса.

# Методы DateOutputStream

| Метод | Описание |
|---|---|
| void writeBoolean(boolean v) | Сохранение значения типа boolean. Займет 1 байт. |
| void writeByte(int v) | Сохранение значения типа boolean. Займет 1 байт. |
| void writeBytes(String s) | Сохранение строки как последовательности байт. При записи старшие 8 бит представления символа отбрасываются. |
| void writeChar(int v) | Сохранение значения типа char. Займет 2 байт. Порядок байт little endian. |
| void writeChars(String s) | Сохранение строки как последовательности символов. Т.е. каждый символ 2 байта. |
| void writeDouble(double v) | Сохранение значения типа double. Займет 8 байт. Порядок байт little endian. |
| void writeFloat(float v) | Сохранение значения типа float. Займет 4 байт. Порядок байт little endian. |
| void writeInt(int v) | Сохранение значения типа int. Займет 4 байт. Порядок байт little endian. |
| void writeLong(long v) | Сохранение значения типа long. Займет 8 байт. Порядок байт little endian. |
| void writeShort(int v) | Сохранение значения типа short. Займет 2 байт. Порядок байт little endian. |
| void writeUTF(String str) | Сохранение строки в последовательность байт в кодировке UTF-8. |
| void write(int b) | Запись int представления 1 байта. Записываются младшие 8 бит |
| void write(byte[] b, int off, int len) | Записывает в поток len байт из массива b начиная с индекса off. |
| int size() | Вернет количество реально записанных байт. |
| void flush() | Запросит сохранение буфера OutputStream |

# Методы DataInputStream

| Метод | Описание |
|---|---|
| boolean readBoolean() | Считает значение типа boolean. |
| byte readByte() | Считает значение типа byte. |
| char readChar() | Считает значение типа char. |
| double readDouble() | Считает значение типа double. |
| float readFloat() | Считает значение типа float. |
| int readInt() | Считает значение типа int. |
| long readLong() | Считает значение типа long. |
| short readShort( | Считает значение типа short. |
| int readUnsignedByte( | Считает значение типа без знаковый байт. Результат младшие 8 бит. |
| int readUnsignedShort() | Считает значение типа без знаковый short. Результат младшие 16 бит. |
| String readLine() | **Устарело.** Считает строку. 2-х байтовая кодировка. |
| String readUTF() | Считает строку. Кодировка UTF-8. |
| int read(byte[] b) | Считывает данные из потока в массив. |
| int read(byte[] b, int off, int len) | Считывает len байт из потока в массив со смещением off. |
| int skipBytes(int n) | Пропуск n байт из потока. |
| void readFully(byte[] b) | Считывание байт из потока и запись их в массив. |
| void readFully(byte[] b, int off, int len) | Считывание len байт из потока и запись их в массив со смещением of. |

# Пример использования DataOutputStream и DataInputStream

```
public static void saveIntArrayToFile(int[] array, File file) throws IOException {
     try (DataOutputStream dos = new DataOutputStream(new FileOutputStream(file))) {
          for (int i = 0; i < array.length; i++) {
               dos.writeInt(array[i]);
          }
     } catch (IOException e) {
          throw e;
     }
}

public static int[] loadArrayFromFIle(File file) throws IOException {
     try (DataInputStream dis = new DataInputStream(new FileInputStream(file))) {
          int arraySize = (int) (file.length() / 4);
          int[] result = new int[arraySize];
          for (int i = 0; i < result.length; i++) {
               result[i] = dis.readInt();
          }
          return result;
     } catch (IOException e) {
          throw e;
     }
}
```

Для записи int значений

Для чтения int значений

# Использование методов показанных ранее

```
public static void main(String[] args) {

     File file = new File("array storage.dat");

     int[] array1 = new int[] { 11, -5, 63 };

     try {
          saveIntArrayToFile(array1, file);
     } catch (IOException e) {
          e.printStackTrace();
     }

     int[] array2 = null;

     try {
          array2 = loadArrayFromFIle(file);
     } catch (IOException e) {
          e.printStackTrace();
     }

     System.out.println(Arrays.toString(array2));

}
```

Запись массива в файл

Вычитка массива из файла

# Символьные потоки ввода вывода

Для работы с данными представимыми в символьном виде используются символьные потоки. В их основе лежит два абстрактных класс Reader(чтение) и Writer(запись). Таким образом если нужно работать со строковыми данными проще использовать реализации этих потоков.

# Методы Reader

| Метод | Описание |
|---|---|
| abstract void close() | Закрывает источник ввода и освобождает системные ресурсы связанные с потоком. |
| void mark( int readlimit) | Отмечает текущую позицию в этом входном потоке. Возможно вернуться. |
| boolean markSupported() | Возвращает true если методы mark(), reset() поддерживаются вызывающим потоком. |
| int read() | Возвращает int представление следующего доступного символа в потоке. При достижении конца потока возвращает значение -1. |
| int read(char[] b) | Читает данные из потока в массив символов переданный в качестве параметра. Возвращает сколько было прочитано символов. Возвращает -1 при достижении конца потока. |
| abstract int read(char[] cbuf, int off, int len) | Действие аналогично предыдущему методу, однако символы записываются в массив со смещением. Параметр len указывает сколько именно символов требуется считать. |
| int read(CharBuffer target) | Считывает символы с потока в CharBuffer. Java NIO пока еще не изучали. |
| boolean ready() | Возможно ли провести чтение |
| void reset() | Сбрасывает входной указатель в ранее установленную метку. |
| long skip (long n) | Игнорирует n символов потока, возвращает количество действительно проигнорированных символов. |
| long transferTo(Writer out) | Читает данные из входного потока и отправляет в выходной. Вернет сколько символов было вычитано и записано. |
| static Reader nullReader() | Вернет поток чтения который не читает данные. |

# Методы Writer

| Метод | Описание |
|---|---|
| abstract void close() | Закрывает источник ввода и освобождает системные ресурсы связанные с потоком. |
| abstract void flush() | Указывает выполнить запись в буфере (если поддерживается буферизация). Внимание касается только классов Java. Для методов ОС гарантии нет. |
| void write(int b) | Записывается представление символа в виде int |
| void write(char[] b) | Записывает в поток все содержимое массива символов. |
| abstract void write(char[] b, int off, int len) | Записывает в поток len символов из массива b начиная с индекса off. |
| static Writer nullWriter() | Вернет Writer который отбрасывает все записываемые данные. |
| void write(String str) | Записывает строку. |
| void write(String str, int off, int len) | Записывает len символов из строки str начиная с индекса off. |
| Writer append(char c) | Добавит символ к Writer. Можно использовать для накопления. |
| Writer append(CharSequence csq) | Добавит последовательность символов к Writer. Можно использовать для накопления. |
| Writer append(CharSequence csq, int start, int end) | Добавит последовательность символов  со смещением к Writer. Можно использовать для накопления. |

# Реализация Writer для записи в файл

Для записи данных в файл (данные должны быть представимы в виде последовательности символов) используется класс FileWriter.

Конструкторы этого класса:

- FileWriter(File file)
- FileWriter(String fileName)
- FileWriter(FileDescriptor fd)
- FileWriter(File file, Charset charset)
- FileWriter(String fileName, Charset charset)

- FileWriter(File file, boolean append)
- FileWriter(String fileName, boolean append)
- FileWriter(File file, Charset charset, boolean append)
- FileWriter(String fileName, Charset charset, boolean append)

◄ Режим «добавления» данных в файл

При работе не забывайте закрыть файл. Для этого используйте метод close() (в блоке finally) или блок try с ресурсами (рекомендованный способ).

# Пример использования FileWriter

```
public class Main {
     public static void main(String[] args) {

          File file = new File("hello world.txt");

          String text = "Hello world";

          try {
               saveStringToFile(text, file);
          } catch (IOException e) {
               e.printStackTrace();
          }
     }

public static void saveStringToFile(String text, File file) throws IOException {
     try (Writer writer = new FileWriter(file)) {

          writer.write(text);

     } catch (IOException e) {
          throw e;
     }
}
```

Запись строки в файл

# Реализация Reader для чтения из файла

Для чтения символьной последовательности из файла используется класс FileReader.

Конструкторы этого класса следующие:

- FileReader(File file)
- FileReader(String fileName)
- FileReader(FileDescriptor fd)

- FileReader(File file, Charset charset)
- FileReader(String fileName, Charset charset)

◄ Для явного указания кодировки

При работе не забывайте закрыть файл. Для этого используйте метод close() (в блоке finally) или блок try с ресурсами (рекомендованный способ).

# Пример использования FileReader

```
public static void main(String[] args) {
	File file = new File("hello world.txt");
	try {
		String text = loadStringFromFile(file);
		System.out.println(text);
	} catch (IOException e) {
		e.printStackTrace();
	}
}

public static String loadStringFromFile(File file) throws IOException {
	String result = "";
	try (Reader read = new FileReader(file)) {
		char[] chars = new char[1000];
		int readChars = 0;
		for (;;) {
			readChars = read.read(chars);
			if (readChars <= 0) {
				break;
			}
			result += new String(chars, 0, readChars);
		}
		return result;
	} catch (IOException e) {
		throw e;
	}
}
```

Вычитка данных из файла и запись в массив символов

# Классы реализующие буферизацию данных для символьных потоков

Для символьных потоков буферизацию реализуют классы BufferedReader и BufferedWriter. Эти классы в качестве полей содержат массив символов который играет роль буфера. Поток данные которого вы хотите буферизировать нужно передать как параметр конструктора.

Конструкторы BufferedReader:

- BufferedReader(Reader in)
- BufferedReader(Reader in, int sz)

Конструкторы BufferedWriter:

- BufferedWriter(Writer out)
- BufferedWriter(Writer out, int sz)

# Пример работы с BufferedReader

```
public static void main(String[] args) {
      File file = new File("hello world.txt");
      try {
            String text = loadStringFromFile(file);
            System.out.println(text);
      } catch (IOException e) {
            e.printStackTrace();
      }
}

public static String loadStringFromFile(File file) throws IOException {
      try (BufferedReader br = new BufferedReader(new FileReader(file))) {
            String result = "";
            String temp = "";
            for (;;) {
                  temp = br.readLine();
                  if (temp == null) {
                        break;
                  }
                  result += temp + System.lineSeparator();
            }
            return result.substring(0, result.length() - 1);
      } catch (IOException e) {
            throw e;
      }
}
```

Читаем построчно

# Преобразование байтовых потоков в символьные

В случае когда поток байтовый, но нужно работать с ним как с символьным (например сетевые соединения по своей природе байтовые потоки) можно использовать готовые классы которые выполняют подобные преобразования.

InputStreamReader — класс выполняющий преобразование данных из потока байт и возвращающий результат в виде потока символов. Для работы нужно передать поток байт и кодировку(не обязательно, может быть использована кодировка по умолчанию).

Конструкторы:

- InputStreamReader(InputStream in)
- InputStreamReader(InputStream in, String charsetName)
- InputStreamReader(InputStream in, Charset cs)
- InputStreamReader(InputStream in, CharsetDecoder dec)

Для ускорения чтения рекомендуют использовать буферизацию например BufferedReader.

# Пример использования InputStreamReader

```
public static String loadStringFromFile(File file) throws IOException {

     try (InputStream is = new FileInputStream(file)) {

          BufferedReader br = new BufferedReader(new InputStreamReader(is));
          String result = "";
          String temp = "";
          for (;;) {
               temp = br.readLine();
               if (temp == null) {
                    break;
               }
               result += temp + System.lineSeparator();
          }
          return result.substring(0, result.length() - 1);
     } catch (IOException e) {
          throw e;
     }
}
```

Байтовый поток

Конвертация в символьный

# Задание для самостоятельной проработки. Основной уровень.

1) Напишите программу, которая скопирует все файлы с заранее определенным расширением (например, только doc) из одного каталога в другой.
2) Реализуйте отдельный класс GroupFileStorage в котором будут следующие методы:
   - void saveGroupToCSV(Group gr) — запись группы в CSV файл
   - Group loadGroupFromCSV(File file) — вычитка и возврат группы из файла
   - File findFileByGroupName(String groupName, File workFolder) — поиск файла в рабочем каталоге (workFolder). Название файла определяется названием группы в нем сохраненной.

# Задание для самостоятельной проработки. Продвинутый уровень.

1) Реализуйте сервис для сравнения файлов на идентичность. Файлы считаются идентичными если они побайтово равны. Файлы разной длинны, или файлы в которых хотя бы один байт отличен считаются разными.
2) Дополните полученный сервис возможностью передачи адресов файлов в ключевом режиме при запуске приложения из командной строки.

# Список литературы

1) Герберт Шилдт Java 8. Полное руководство 9-е издание ISBN 978-5-8459-1918-2
2) Кей Хорстман, Гари Корнелл Библиотека профессионала Java Том 2.Расширенные средства.
3) Брюс Эккель Философия Java. 4-е издание 2009 г.