InputStream // это потоки, они ничего не знают о источниках

OutputStream // это потоки, они ничего не знают о источниках

Reader // это потоки, они ничего не знают о источниках

Writer // это потоки, они ничего не знают о источниках

InputStream

```
This abstract class is the superclass of all classes representing
an input stream of bytes
public abstract class InputStream implements Closeable {}

Reads the next byte of data from the input stream.
public abstract int read() throws IOException;
```

OutputStream

```
This abstract class is the superclass of all classes representing
an output stream of bytes. An output stream accepts output bytes
and sends them to some sink.
public abstract class OutputStream implements Closeable,
Flushable {}
```

Считывание текстовой информации из файла. Краткое описание кода.

#Создаем объкт, который будет содержать информацию о нашем текстовом файле.

```
File f = new File("resources/sampler.txt");
```

#Открываем поток, который превратит всю информацию из файла в поток байт. Передаем в данный поток наш файл в виде объекта f.

```
InputStream is= new FileInputStream(f);
```

#Определим оценочное количество байт информации, имеющейся в нашем файле(с учетом специальных символов, например, символ новой строки). Важно помнить, что символы согласно таблицы ASCII могут быть представлены различным количеством байт.

```
int avaliable = is.available();
System.out.println("Bytes avaliable: "+avaliable);
```

#Создаем массив byte[]. Каждый елемент этого массива будет хранить код отдельного символа согласно таблицы ASCII. #Пример. Символ 'а' русского алфавита будет представлен одним байтом в виде числа 48. Символ 'ё' будет представлен двумя байтами в виде двух чисел:-47 и -111. Длина массива равна количеству байт информации. Не все наследники класса возвращают полное количество байт. Нужно быть внимательным с установкой размера массива.

```
byte[] data = new byte[avaliable];
```

#Заполняем массив данными из потока `FileInputStream`.Выводим массив на печать.

```
is.read(data); System.out.println(new String(data));
```

#Закрываем поток `FileInputStream`.

```
is.close();
```

Считывание текстовой информации из файла.
Код.

```
public class MainInputStream {

	public static void main(String[] args) {
		File f = new File("resources/sampler.txt");
		if(!f.exists()) {
			try {
				f.createNewFile();
			} catch (IOException e) {
				e.printStackTrace();
			}
		}
		try(InputStream is= new FileInputStream(f) ) {
			int avaliable = is.available();
			System.out.println("Bytes avaliable: "+avaliable);
			byte[] data = new byte[avaliable];
			is.read(data);
			System.out.println(new String(data));
		} catch (FileNotFoundException e) {
			e.printStackTrace();
		} catch (IOException e) {
			e.printStackTrace();
		}
	}
}
```

Запись текстовой информации в файл.

Краткое описание кода.

#Создаем объект, который будет содержать информацию о файле, куда будет производиться запись текста

```
File f = new File("resources/sampler.txt");
```

#Открываем поток, который будет хранить информацию, которую необходимо поместить в файл, в виде потока byte. В конструкток класса передаем наш файл в виде объекта f и параметр true, который позволяет дозаписывать информацию в конец файла.

```
OutputStream os= new FileOutputStream(f,true);
```

#Даем указание на запись информации в файл. Информацию передаем в виде массива byte[].

```
os.write("Hello person1\n".getBytes());
```

#Закрываем поток `FileOutputStream`

```
os.close();
```

Считывание текстовой информации из файла.
Код.

```
public class MainOutputStream {

	public static void main(String[] args) {
		File f = new File("resources/sampler.txt");
		if(!f.exists()) {
			try {
				f.createNewFile();
			} catch (IOException e) {
				// TODO Auto-generated catch block
				e.printStackTrace();
			}
		}
		try(OutputStream os= new FileOutputStream(f,true) ) {
			os.write("Hello person2\n".getBytes());
		} catch (FileNotFoundException e) {
			e.printStackTrace();
		} catch (IOException e) {
			e.printStackTrace();
		}
	}

}
```

```
Reader
Abstract class for reading character streams.  The only methods that a
subclass must implement are read(char[], int, int) and close().
public abstract class Reader implements Readable, Closeable {}

@see BufferedReader, LineNumberReader, CharArrayReader,
InputStreamReader, FileReader, FilterReader, PushbackReader,
PipedReader, StringReader

In general, each read request made of a Reader causes a corresponding
read request to be made of the underlying character or byte stream.  It
is therefore advisable to wrap a BufferedReader around any Reader whose
read() operations may be costly, such as FileReaders and
InputStreamReaders.

BufferedReader
Reads text from a character-input stream, buffering characters so as to
provide for the efficient reading of characters, arrays, and lines.
The buffer size may be specified, or the default size may be used.  The
default is large enough for most purposes.
public class BufferedReader extends Reader {}
```

Считывание текстовой информации из файла. Краткое описание кода.

#Создаем объкт, который будет содержать информацию о нашем текстовом файле.

```
File f = new File("resources/sampler.txt");
```

#Открываем поток `FileReader`, который превратит всю информацию из файла в поток символов char. Передаем в данный поток наш файл в виде объекта f.

```
Reader reader = new FileReader(f);
```

#Вариант 1.Класс FileReader не позволяет определить количество символов в файле, поэтому резервируем массив для считывания символов из потока большей длины.

```
char[] cbuf = new char[10];
Arrays.fill(cbuf, '@');
```

#Считываем содержимое в данный массив. И далее извлекаем информацию из массива

```
reader.read(cbuf);
```

#Вариант 2. Для более эффективной работы с символами, массивами символов и строками необходимо обернуть объкт класса Reader в объект класса BufferedReader.

```
BufferedReader br = new BufferedReader(reader);
```

#Теперь мы можем построчно считать информацию из потока.

```
String str = null;
while((str=br.readLine())!= null) {
      System.out.println(str);
}
```

#Закрываем потоки

```
br.close();    reader.close();
```

Считывание текстовой информации из файла.
Код.

```
public class MainMainReader {

	public static void main(String[] args) {

		File f = new File("resources/sampler.txt");
		 if(!f.exists()) {
			 System.out.println("create new file");
			 try {
				f.createNewFile();
			} catch (IOException e) {
				e.printStackTrace();
			}
		 }
		try (Reader reader = new FileReader(f);
		     BufferedReader br = new BufferedReader(reader) ) {
			String str = null;
			while((str=br.readLine())!= null) {
				System.out.println(str);
			}
		}catch(IOException e) {
			e.printStackTrace();
		}
	}
}
```

Writer
Abstract class for writing to character streams. The only methods that a subclass must implement are write(char[], int, int), flush(), and close().Most subclasses, however, will override some of the methods defined here in order to provide higher efficiency, additional functionality, or both.

```
public abstract class Writer implements Appendable, Closeable, Flushable {}
```

@see BufferedWriter, CharArrayWriter, FilterWriter, OutputStreamWriter, FileWriter, PipedWriter, PrintWriter, StringWriter

In general, a Writer sends its output immediately to the underlying character or byte stream. Unless prompt output is required, it is advisable to wrap a BufferedWriter around any Writer whose write() operations may be costly, such as FileWriters and OutputStreamWriters.

BufferedWriter
Writes text to a character-output stream, buffering characters so as toprovide for the efficient writing of single characters, arrays, and strings.

```
public class BufferedWriter extends Writer {}
```

Запись текстовой информации в файл. Краткое описание кода.

#Создаем объкт, который будет содержать информацию о нашем текстовом файле.

```
File f = new File("resources/sampler.txt");
```

#Открываем поток FileReader, который превратит всю информацию из файла в поток символов char. Передаем в данный поток наш файл в виде объекта f.

```
Writer writer = new FileWriter(f);
```

#Для более эффективной работы с символами, массивами символов и строками необходимо обернуть объкт класса Writer в объект класса BufferedWriter.

```
BufferedWriter bw = new BufferedWriter(writer);
```

#Записываем необходимую информацию в файл.

```
bw.write("If you have no money I have no time\n");
```

#Закрываем потоки.

```
bw.close();    writer.close();
```

Запись текстовой информации в файл. Код.

```
public class MainWriter {

	public static void main(String[] args) {
		File f = new File("resources/sampler.txt");
		if(!f.exists()) {
			System.out.println("New file needs to be created.");
		}
		try {
			f.createNewFile();
		} catch (IOException e) {
			e.printStackTrace();
		}

		try(Writer writer = new FileWriter(f, true);
		    BufferedWriter bw = new BufferedWriter(writer) ) {
			bw.write("If you have no money\n");
		} catch (IOException e) {
			e.printStackTrace();
		}
	}
}
```