Вопросы по теме Потоки ввода / вывода:

1. Разница между NIO / IO?
2. Что такое канал, селектор?
3. Основные классы потоков ввода / вывода?
4. Можно ли перенаправить потоки стандартного ввода / вывода?
5. Что такое символьная ссылка?
6. Что такое Wrapper Classes (Классы обертки)?

Потоки ввода / вывода.

Поток – абстракция, означающая процесс чтения / записи информации между программой и реальными физическими компонентами устройства, с которым данная программа взаимодействует (например, для компьютера – жесткий диск при чтении / записи в файл).

Потоки делятся на 2 основные категории:

1. По направлению потока:
   Потоки ввод – предназначены для считывания данных программой.
   Потоки вывода – предназначены для записи данных во внешнее устройство.
2. По типу данных:
   Потоки для работы с байтами.
   Потоки для работы с символами.

	Поток ввода	Поток вывода
Работает с байтами	**InputStream**	**OutputStream**
Работает с символами	**Reader**	**Writer**

В основе всех классов, работающих с байтами лежат абстрактные классы InputStream и OutputStream, а работающих с символами – абстрактные классы Reader и Writer. Все остальные классы, работающие с потоками, являются наследниками этих классов. Данные классы расположены в пакете java.io.*:

InputStream	OutputStream	Reader	Writer
FileInputStream	FileOutputStream	FileReader	FileWriter
BufferedInputStream	BufferedOutputStream	BufferedReader	BufferedWriter
ByteArrayInputStream	ByteArrayOutputStream	CharArrayReader	CharArrayWriter
FilterInputStream	FilterOutputStream	FilterReader	FilterWriter
DataInputStream	DataOutputStream		
ObjectInputStream	ObjectOutputStream		

Классы-обертки.

Класс-обертка – это надстройка над базовым классом, которая расширяет его функционал не изменяя при этом сам базовый класс. Такие классы реализуют паттерн проектирования Wrapper или Decorator.

Виталий Жуков
unicraft1989@gmail.com
Ссылка на курс Stepik

```java
public class Man {
    private String name;

    public Man(String name) { this.name = name; }

    public String getName() { return name; }

    public String say() { return "My name is " + name + "."; }
}
```

Базовый класс

```java
public class Superman extends Man {
    private Man man;

    public Superman(Man man) {
        super(man.getName());
        this.man = man;
    }

    public String say() {
        return "I'm not " + man.getName()
                + ". I'm Superman.";
    }
}
```

Класс-обертка

В примере выше мы расширили логику метода say() в классе Man с помощью класса-обертки Superman. Так же происходит и с остальными классами пакета java.io – они расширяют 4 основных абстрактных класса.

Чтобы создать свой класс-обертку нужно:

1. наследоваться от базового класса;
2. принять через конструктор объект базового класса и инициализировать его через super();
3. сохранить экземпляр базового класса;
4. изменить нужный метод через сохраненный экземпляр.

Данный подход позволяет из нескольких классов собрать необходимую нам функциональность. Например,

BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));

Объект reader умеет считывать поток байт, предварительно их буферизируя, и преобразовывать его в символы. В итоге у нас получается матрешка из оберток:

– класс InputStream представляет собой входящий поток байт;

– класс InputStreamReader является оберткой для класс InputStream и позволяет считывать байты и на выходе преобразовывает их в символы;

– класс BufferedReader является оберткой для класса InputStreamReader и позволяет буферизировать символы.

Java NIO (new/non-blocking io).

В Java существует еще один подход к организации обмена информацией между программой и внешними источниками. Он представлен пакетом java.nio.*

IO	NIO
Потокоориентированный	Буфер-ориентированный
Блокирующий (синхронный) ввод/вывод	Неблокирующий (асинхронный) ввод/вывод
	Селекторы

Виталий Жуков
unicraft1989@gmail.com
Ссылка на курс Stepik

Отличие NIO подхода от IO:

1. **IO – потоко-ориентированный ввод / вывод информации** (т.е. данные передаются сплошным потоком от передающей сущности к принимающей)
   NIO – буферо-ориентированный ввод / вывод информации (т.е. данные предварительно собираются в контейнер фиксированного размера определенного примитивного типа данных; по этому контейнеру можно передвигать и считывать с него информацию, что дает более гибкое возможности обработки данных)
   Интерфейс Buffer – базовый интерфейс для классов пакета java.nio.*, которые предоставляют возможность буферизации
2. **IO – является блокирующим** (т.е. если поток вызывает операцию чтения (метод read()) то, он блокируется до тех пор, пока не будет считана вся информация; в это время операция записи (метод write()) недоступна; и наоборот, при записи недоступно чтение)
   NIO – является неблокирующим (т.е. возможно одновременное чтение и запись информации, что позволяет экономить время)

   При NIO подходе вводятся следующие понятия и сущности:
3. **канал** – соединение между разными участниками обмена данными (файл / консоль / сетевой сокет и текущая программа), через которое осуществляется ввод / вывод информации
   Интерфейс Channel – базовый интерфейс для классов пакета java.nio.channels.*, которые предоставляют возможность управления каналами
4. **селектор** – позволяет управлять работой нескольких зарегистрированных каналов (не дает каналам простаивать, приостанавливает / запускает передачу данных по необходимости и т.д.)
   Селекторы представлены в пакете java.nio.channels.* классом Selector и его наследниками.
5. **кодировщики и декодеры** – перевод байт в Unicode и обратно
   Кодировщики и декодеры размещены в пакете java.nio.charset.*

Вопрос: Что такое абсолютный / относительный путь. Символьная ссылка.

Ответ: Абсолютный путь – путь в файловой системе, который и ведет к одному и тому же месту вне зависимости от текущей директории.

Относительный путь – путь в файловой системе, который указывается относительно текущей директории.

Символьная ссылка – специальный файл в файловой системе, в котором, вместо пользовательских данных, содержится путь к файлу, который должен быть открыт при попытке обратиться к данной ссылке (файлу).

Вопрос: Можно ли перенаправить потоки стандартного ввода / вывода?

Ответ: Можно, с помощью вызова статического метода класса System.

setIn(InputStream) – для ввода

setOut(PrintStream) – для вывода

setErr(PrintStream) – для ошибок