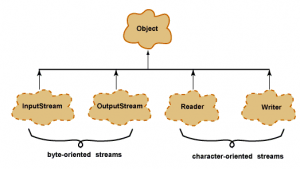
1. Какие существуют виды потоков ввода/вывода?  
2. Назовите основные предки потоков ввода/вывода.  
3. Что общего и чем отличаются следующие потоки: InputStream, OutputStream, Reader, Writer?  
4. Что вы знаете о RandomAccessFile?  
5. Какие есть режимы доступа к файлу?  
6. В каких пакетах лежат классы-потоки?  
7. Что вы знаете о классах-надстройках?  
8. Какой класс-надстройка позволяет читать данные из входного байтового потока в формате примитивных типов данных?  
9. Какой класс-надстройка позволяет ускорить чтение/запись за счет использования буфера?  
10. Какие классы позволяют преобразовать байтовые потоки в символьные и обратно?  
11. Какой класс предназначен для работы с элементами файловой системы (ЭФС)?  
12. Какой символ является разделителем при указании пути к ЭФС?  
13. Как выбрать все ЭФС определенного каталога по критерию (например, с определенным расширением)?  
14. Что вы знаете об интерфейсе FilenameFilter?  
15. Что такое сериализация?  
16. Какие условия “благополучной” сериализации объекта?  
17. Какие классы позволяют архивировать объекты?

**1. Какие существуют виды потоков ввода/вывода?**

Разделяют два вида потоков ввода/вывода: байтовые и символьные.

[](https://javastudy.ru/wp-content/uploads/2016/01/ioHierarchy.png)

Система ввода/вывода: http://developer.alexanderklimov.ru/android/java/io.php

Oracle Lesson: Basic I/O tutorial: https://docs.oracle.com/javase/tutorial/essential/io/

**2. Назовите основные предки потоков ввода/вывода.**

Байтовые: java.io.InputStream, java.io.OutputStream;

Символьные: java.io.Reader, java.io.Writer;

**3. Что общего и чем отличаются следующие потоки: InputStream, OutputStream, Reader, Writer?**

Общего:

* это всё абстрактные классы;
* они все предназначены для работы со стримами(потоками данных!);
* InpSandReader как и OutpSandWriterимеют большинство одинаковых методов.

Базовый класс **InputStream** представляет классы, которые получают данные из различных источников:  
— массив байтов  
— строка (String)  
— файл  
— канал (pipe): данные помещаются с одного конца и извлекаются с другого  
— последовательность различных потоков, которые можно объединить в одном потоке  
— другие источники (например, подключение к интернету)

Класс **OutputStream** — это абстрактный класс, определяющий потоковый байтовый вывод. В этой категории находятся классы, определяющие, куда направляются ваши данные: в массив байтов (но не напрямую в String; предполагается что вы сможете создать их из массива байтов), в файл или канал.

Символьные потоки имеют два основных абстрактных класса **Reader** и **Writer**, управляющие потоками символов Unicode. Класс **Reader** — абстрактный класс, определяющий символьный потоковый ввод. Класс **Writer** — абстрактный класс, определяющий символьный потоковый вывод. В случае ошибок все методы класса передают исключение **IOException**.

**4. Что вы знаете о RandomAccessFile?**

*Класс RandomAccessFile наследуется напрямую от Object и не наследуется от вышеприведенных базовых классов ввода\вывода. Предназначен для работы с файлами, поддерживая произвольный доступ к их содержимому.*

*Работа с классом****RandomAccessFile****напоминает использование совмещенных в одном классе потоков****DataInputStream****и****DataOutputStream****(они реализуют те же интерфейсы****DataInput****и****DataOutput****). Кроме того, метод****seek()****позволяет переместиться к определенной позиции и изменить хранящееся там значение.*

*При использовании****RandomAccessFile****необходимо знать структуру файла. Класс****RandomAccessFile****содержит методы для чтения и записи примитивов и строк UTF-8.*

**5. Какие есть режимы доступа к файлу?**

**RandomAccessFile** может открываться в режиме чтения («r») или чтения/записи («rw»). Также есть режим «rws», когда файл открывается для операций чтения-записи и каждое изменение данных файла немедленно записывается на физическое устройство.

**6. В каких пакетах лежат классы-потоки?**

Классы потоков ввода\вывода лежат в java.io;

С JDK 7 добавлен более современный способ работы с потоками — Java NIO. Классы лежат в java.nio. Для работы с архивами используются классы из пакета java.util.

**7. Что вы знаете о классах-надстройках?**

Классы-надстройки наделяют существующий поток дополнительными свойствами. Примеры классов: BufferedOutputStream, BufferedInputStream, BufferedWriter — буферизируют поток и повышают производительность.

**8. Какой класс-надстройка позволяет читать данные из входного байтового потока в формате примитивных типов данных?**

Для чтения байтовых данных (не строк) применяется класс **DataInputStream**. В этом случае необходимо использовать классы из группы **InputStream**.

Для преобразования строки в массив байтов, пригодный для помещения в поток **ByteArrayInputStream**, в классе **String** предусмотрен метод **getBytes()**. Полученный **ByteArrayInputStream** представляет собой поток **InputStream**, подходящий для передачи **DataInputStream**.

При побайтовом чтении символов из форматированного потока **DataInputStream** методом **readByte()** любое полученное значение будет считаться действительным, поэтому возвращаемое значение неприменимо для идентификации конца потока. Вместо этого можно использовать метод **available()**, который сообщает, сколько еще осталось символов.

Класс **DataInputStream** позволяет читать элементарные данные из потока через интерфейс **DataInput**, который определяет методы, преобразующие элементарные значения в форму последовательности байтов. Такие потоки облегчают сохранение в файле двоичных данных.

Конструктор: DataInputStream(InputStream stream)  
Методы: readDouble(), readBoolean(), readInt()

**9. Какой класс-надстройка позволяет ускорить чтение/запись за счет использования буфера?**

Дляэтогоиспользуютсяклассы, позволяющиебуферизироватьпоток:  
java.io.BufferedInputStream(InputStream in) || BufferedInputStream(InputStream in, int size),  
java.io.BufferedOutputStream(OutputStream out) || BufferedOutputStream(OutputStream out, int size),  
java.io.BufferedReader(Reader r) || BufferedReader(Reader in, intsz),  
java.io.BufferedWriter(Writer out) || BufferedWriter(Writer out, intsz)

**10. Какие классы позволяют преобразовать байтовые потоки в символьные и обратно?**

OutputStreamWriter — мост между классом OutputStream и классом Writer. Символы, записанные в поток, преобразовываются в байты.

|  |  |
| --- | --- |
| 1  2  3  4  6 | OutputStreamoutputStream       = new FileOutputStream("c:\\data\\output.txt");  Writer       outputStreamWriter = new OutputStreamWriter(outputStream, "UTF-8");    outputStreamWriter.write("Hello World");   outputStreamWriter.close(); |

InputStreamReader — аналог для чтения. При помощи методов класса Reader читаются байты из потока InputStream и далее преобразуются в символы.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | InputStreaminputStream       = new FileInputStream("c:\\data\\input.txt");  Reader      inputStreamReader = new InputStreamReader(inputStream, "UTF-8");    int data = inputStreamReader.read();  while(data != -1){      char theChar = (char) data;      data = inputStreamReader.read();  }    inputStreamReader.close(); |

**11. Какой класс предназначен для работы с элементами файловой системы (ЭФС)?**

В отличие от большинства классов ввода/вывода, класс **File** работает не с потоками, а непосредственно с файлами. Данный класс позволяет получить информацию о файле: права доступа, время и дата создания, путь к каталогу. А также осуществлять навигацию по иерархиям подкаталогов. Класс **java.io.File** может представлять имя определённого файла, а также имена группы файлов, находящихся в каталоге. Если класс представляет каталог, то его метод **list()** возвращает массив строк с именами всех файлов.

Для создания объектов класса **File** можно использовать один из следующих конструкторов:  
File(Filedir, Stringname) — указывается объект класса File (каталог) и имя файла  
File(Stringpath) — указывается путь к файлу без указания имени файла  
File(StringdirPath, Stringname) — указывается путь к файлу и имя файла  
File(URI uri) — указывается объекта URI, описывающий файл

**12. Какой символ является разделителем при указании пути к ЭФС?**

Для различных систем символ разделителя различается. Вытащить его можно используя file.separator, а так же в статическом поле File.separator.  Для Windows это ‘\’.

**13. Как выбрать все ЭФС определенного каталога по критерию (например, с определенным расширением)?**

Метод File.listFiles() возвращает массив объектов File, содержащихся в каталоге. Метод может принимать в качестве параметра объект класса, реализующего FileFilter. Это позволяет включить список только те элементы, для которых метод accept() возвращает true (критерием может быть длина имени файла или его расширение).

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | public class FileDemo {     public static void main(String[] args) {          File f = null;        File[] paths;          try{           // create new file           f = new File("c:/test");             // returns files metadata that matches the accepted filter.           paths = f.listFiles(file->file.getName().endsWith(“.txt”));             // for each pathname in pathname array           for(File path:paths)           {              // prints file and directory paths              System.out.println(path);           }        }catch(Exception e){           // if any error occurs           e.printStackTrace();        }     }  } |

**14. Что вы знаете об интерфейсе FilenameFilter?**

Интерфейс FilenameFilter применяется для проверки попадает ли объект File под некоторое условие. Этот интерфейс содержит единственный метод Boolean accept(File dir, String name). Этот метод необходимо переопределить и реализовать. Например:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | public boolean accept(File file) {      if (file.isDirectory()) {        return true;      } else {        String path = file.getAbsolutePath().toLowerCase();        for (inti = 0, n = extensions.length; i< n; i++) {          String extension = extensions[i];          if ((path.endsWith(extension) && (path.charAt(path.length()                    - extension.length() - 1)) == '.')) {            return true;          }        }      }      return false;  }  //OR  String yourPath = "insert here your path..";  File directory = new File(yourPath);*//при new File ФИЗИТЧЕСКИ НЕ СОЗДАЁТСЯ ФАЙЛ и НЕ ПРОВЕРЯЕТСЯ НА ЕГО НАЛИЧИЕ(т.е.создаются метаданные, но не создается файл в системе!!!! )!*  String[] myFiles = directory.list(new FilenameFilter() {      public boolean accept(File directory, String fileName) {          returnfileName.endsWith(".txt");      }  }); |

**15. Что такое сериализация?**

*Сериализация* это процесс сохранения состояния объекта в последовательность байт;

*десериализация* это процесс восстановления объекта, из этих байт. JavaSerialization API предоставляет стандартный механизм для создания сериализуемых объектов.

Для удобства используются более читабельные для человека форматы .Другими словами, *Сериализация* – перевод описания объекта из частного(конкретный язык программирования) в общий синтаксис(json, xml), понятный всем языкам программирования.(см. \_Core\_part\_3\_\_javastudy\_ru.docx)

Сериализация в Java: http://habrahabr.ru/post/60317/

Изучитесекреты Java Serialization API: http://www.ccfit.nsu.ru/~deviv/courses/oop/java\_ser\_rus.html

Как работает сериализация в Java (info.javarush.ru):  <http://goo.gl/K8GzJl>

**16. Какие условия “благополучной” сериализации объекта?**

Чтобы обладать способностью к сериализации, класс должен реализовывать интерфейс-метку Serializable. Так же все атрибуты и подтипы сериализуемого класса должны быть сериализуемы. Если класс предок был несереализуемым, то этот суперкласс должен содержать доступный (public, protected) конструктор без параметров для инициализации полей.

Также важен формат и имена полей сериализуемого объекта, например в Java все поля в camelCase, а в JSON – kebabCase(все слова с маленькой буквы и разделитель “–“) или snakeCase(все слова с маленькой буквы и разделитель “\_“). см. ПРИМЕР 1 (camelCase to kebabCase)

Каждое не статическое поле должно быть помечено соответствующей аннотацией и «приведено к соотв.виду» в строковом литерале.

Если полей много и они должны быть отображены в одном формате, то лучше использовать аннотацию над классом. См. ПРИМЕР 2 (camelCase to kebabCase)

Пример 1

Обр.внимание, class DisbursementCreateAttributes НЕ РЕАЛИЗУЕТ SERIALIZABLE,т.к. используется аннотированная конфигурация сериализуемого объекта из библиотеки Jackson,которая это позволяет!!!

@JsonInclude(JsonInclude.Include.NON\_NULL)  
public class DisbursementCreateAttributes {  
 @JsonProperty("account-id")  
 private String accountId;  
  
 @JsonProperty("contact-id")  
 private String contactId;  
  
 @JsonProperty("funds-transfer-method-id")  
 private String fundsTransferMethodId;  
  
 @JsonProperty("amount")  
 private String amount;  
  
 @JsonProperty("currency-type")  
 private String currencyType;  
  
 @JsonProperty("reference")  
 private String reference;

Пример 2(аналогично Примеру 1)

@JsonInclude(JsonInclude.Include.NON\_NULL)  
@JsonNaming(PropertyNamingStrategy.KebabCaseStrategy.class)  
public class FundsTransferAttributes {  
  
 private LocalDateTime contingenciesClearedAt;  
 private String cancellationDetails;  
 private LocalDateTime cancelledAt;  
 private String fundsTransferType;  
 private LocalDateTime reversedAt;  
 private String settlementDetails;  
 private LocalDateTime createdAt;  
 private LocalDateTime settledAt;  
 private LocalDateTime updatedAt;  
 private String fundsSourceName;  
 private String reversalDetails;  
 private double amountExpected;  
 private Double reversedAmount;  
 private String currencyType;  
 private String reference;  
 private double amount;  
 private String status;

**17. Какие классы позволяют архивировать объекты?**

DeflaterOutputStream, InflaterInputStream, ZipInputStream, ZipOutputStream,  GZIPInputStream, GZIPOutputStream.

Пример из http://www.concretepage.com/java/how\_to\_zip\_file\_java.

Java

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42 | package com.concretepage;  import java.io.File;  import java.io.FileInputStream;  import java.io.FileOutputStream;  import java.io.IOException;  import java.util.zip.ZipEntry;  import java.util.zip.ZipOutputStream;  public class ZipOutputStreamTest {      public static void main(String... args) throws IOException {          String source="D:/page/file.txt";          File sfile= new File(source);          String dest="D:/page/file.zip";          File dfile= new File(dest);          FileInputStream fis= new FileInputStream(sfile);          FileOutputStream fos= new FileOutputStream(dfile);          ZipOutputStream zos= new ZipOutputStream(fos);          ZipEntry ze= new ZipEntry(source);          //begins writing a new zip file and sets the position to the start of data          zos.putNextEntry(ze);          byte[] buf = new byte[1024];          intlen;          while((len=fis.read(buf))>0){              zos.write(buf, 0, len);          }          System.out.println("File created:"+dest);          fis.close();          zos.close();      }  } |