**Ввод-вывод.**

**1) Основы :**

Ключевым понятием здесь является понятие потока. Хотя понятие "поток" в программировании довольно перегружено и может обозначать множество различных концепций. В данном случае применительно к работе с файлами и вводом-выводом мы будем говорить о потоке (stream), как об абстракции, которая используется для чтения или записи информации (файлов, сокетов, текста консоли и т.д.)

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.

В основе всех классов, управляющих потоками байтов, находятся два абстрактных класса: InputStream (представляющий потоки ввода) и OutputStream (представляющий потоки вывода).

Но поскольку работать с байтами не очень удобно, то для работы с потоками символов были добавлены абстрактные классы Reader (для чтения потоков символов) и Writer (для записи потоков символов).

Все остальные классы, работающие с потоками, являются наследниками этих абстрактных классов.

**2)Класс InputStream:**

Класс InputStream является базовым для всех классов, управляющих байтовыми потоками ввода. Рассмотрим его основные методы:

* int available(): возвращает количество байтов, доступных для чтения в потоке
* void close(): закрывает поток
* int read(): возвращает целочисленное представление следующего байта в потоке. Когда в потоке не останется доступных для чтения байтов, данный метод возвратит число -1
* int read(byte[] buffer): считывает байты из потока в массив buffer. После чтения возвращает число считанных байтов. Если ни одного байта не было считано, то возвращается число -1
* int read(byte[] buffer, int offset, int length): считывает некоторое количество байтов, равное length, из потока в массив buffer. При этом считанные байты помещаются в массиве, начиная со смещения offset, то есть с элемента buffer[offset]. Метод возвращает число успешно прочитанных байтов.
* Long skip(long number): пропускает в потоке при чтении некоторое количество байт, которое равно number

**3)Класс OutputStream:**

Класс OutputStream является базовым классом для всех классов, которые работают с бинарными потоками записи. Свою функциональность он реализует через следующие методы:

* void close(): закрывает поток
* void flush(): очищает буфер вывода, записывая все его содержимое
* void write(int b): записывает в выходной поток один байт, который представлен целочисленным параметром b
* void write(byte[] buffer): записывает в выходной поток массив байтов buffer.
* void write(byte[] buffer, int offset, int length): записывает в выходной поток некоторое число байтов, равное length, из массива buffer, начиная со смещения offset, то есть с элемента buffer[offset].

**4) 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: eсли он равен true, то данные дозаписываются в конец файла, а при false - файл полностью перезаписывается

**5) Класс FileNotFoundException:**

Чтение файлов и класс FileInputStream

Для считывания данных из файла предназначен класс FileInputStream, который является наследником класса InputStream и поэтому реализует все его методы.

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 **F**ileNotFoundException.

**6) Класс BufferedInputStream:**

Класс BufferedInputStream накапливает вводимые данные в специальном буфере без постоянного обращения к устройству ввода. Класс BufferedInputStream определяет два конструктора:

|  |  |
| --- | --- |
|  | * BufferedInputStream(InputStream inputStream) * BufferedInputStream(InputStream inputStream, int bufSize) |

Первый параметр - это поток ввода, с которого данные будут считываться в буфер. Второй параметр - размер буфера.

**7) Класс BufferedOutputStream:**

Класс BufferedOutputStream аналогично создает буфер для потоков вывода. Этот буфер накапливает выводимые байты без постоянного обращения к устройству. И когда буфер заполнен, производится запись данных.BufferedOutputStream определяет два конструктора:

|  |  |
| --- | --- |
|  | * BufferedOutputStream(OutputStream outputStream) * BufferedOutputStream(OutputStream outputStream, int bufSize) |

Первый параметр - это поток вывода, который унаследован от OutputStream, а второй параметр - размер буфера.

**8)Класс Reader:**

Абстрактный класс Reader предоставляет функционал для чтения текстовой информации. Рассмотрим его основные методы:

* absract void close(): закрывает поток ввода
* int read(): возвращает целочисленное представление следующего символа в потоке. Если таких символов нет, и достигнут конец файла, то возвращается число -1
* int read(char[] buffer): считывает в массив buffer из потока символы, количество которых равно длине массива buffer. Возвращает количество успешно считанных символов. При достижении конца файла возвращает -1
* int read(CharBuffer buffer): считывает в объект CharBuffer из потока символы. Возвращает количество успешно считанных символов. При достижении конца файла возвращает -1
* absract int read(char[] buffer, int offset, int count): считывает в массив buffer, начиная со смещения offset, из потока символы, количество которых равно count
* long skip(long count): пропускает количество символов, равное count. Возвращает число успешно пропущенных символов

**9)Класс Writer:**

Класс Writer определяет функционал для всех символьных потоков вывода. Его основные методы:

* Writer append(char c): добавляет в конец выходного потока символ c. Возвращает объект Writer
* Writer append(CharSequence chars): добавляет в конец выходного потока набор символов chars. Возвращает объект Writer
* abstract void close(): закрывает поток
* abstract void flush(): очищает буферы потока
* void write(int c): записывает в поток один символ, который имеет целочисленное представление
* void write(char[] buffer): записывает в поток массив символов
* absract void write(char[] buffer, int off, int len) : записывает в поток только несколько символов из массива buffer. Причем количество символов равно len, а отбор символов из массива начинается с индекса off
* void write(String str): записывает в поток строку
* void write(String str, int off, int len): записывает в поток из строки некоторое количество символов, которое равно len, причем отбор символов из строки начинается с индекса off

**10) 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(FileDescriptor fd) * 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.

**11) 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) * FileReader(FileDescriptor fd) |

**12)Класс BufferedWriter :**

Класс BufferedWriter записывает текст в поток, предварительно буферизируя записываемые символы, тем самым снижая количество обращений к физическому носителю для записи данных.

Класс BufferedWriter имеет следующие конструкторы:

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

В качестве параметра он принимает поток вывода, в который надо осуществить запись. Второй параметр указывает на размер буфера.

**13) Класс** **BufferedReader :**

Класс BufferedReader считывает текст из символьного потока ввода, буферизируя прочитанные символы. Использование буфера призвано увеличить производительность чтения данных из потока.

Класс BufferedReader имеет следующие конструкторы:

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

Второй конструктор, кроме потока ввода, из которого производится чтение, также определяет размер буфера, в который будут считываться символы.

Так как BufferedReader наследуется от класса Reader, то он может использовать все те методы для чтения из потока, которые определены в Reader. И также BufferedReader определяет свой собственный метод readLine(), который позволяет считывать из потока построчно.

**14) Класс DataOutputStream :**

Класс DataOutputStream представляет поток вывода и предназначен для записи данных примитивных типов, таких, как int, double и т.д. Для записи каждого из примитивных типов предназначен свой метод:

* **writeBoolean(boolean v)**: записывает в поток булевое однобайтовое значение
* **writeByte(int v)**: записывает в поток 1 байт, которые представлен в виде целочисленного значения
* **writeChar(int v)**: записывает 2-байтовое значение char
* **writeDouble(double v)**: записывает в поток 8-байтовое значение double
* **writeFloat(float v)**: записывает в поток 4-байтовое значение float
* **writeInt(int v)**: записывает в поток целочисленное значение int
* **writeLong(long v)**: записывает в поток значение long
* **writeShort(int v)**: записывает в поток значение short
* **writeUTF(String str)**: записывает в поток строку в кодировке UTF-8

**15) Класс DataInputStreaм :**

Класс DataInputStream действует противоположным образом - он считывает из потока данные примитивных типов. Соответственно для каждого примитивного типа определен свой метод для считывания:

* **boolean readBoolean()**: считывает из потока булевое однобайтовое значение
* **byte readByte()**: считывает из потока 1 байт
* **char readChar()**: считывает из потока значение char
* **double readDouble()**: считывает из потока 8-байтовое значение double
* **float readFloat()**: считывает из потока 4-байтовое значение float
* **int readInt()**: считывает из потока целочисленное значение int
* **long readLong()**: считывает из потока значение long
* **short readShort()**: считывает значение short
* **String readUTF()**: считывает из потока строку в кодировке UTF-8
* **int skipBytes(int n)**: пропускает при чтении из потока n байтов

**16) Закрытие потока:**

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.

**17) Класс RandomAccessFile :**Класс RandomAccessFile позволяет отыскивать или записывать данные где угодно в файле. Файлы на дисках всегда имеют возможность для произвольного доступа, тогда как потоки ввода-вывода данных из сети такой возможности не имеют. Файл с произвольным доступом может открываться только для чтения или же как для чтения, так и для записи. Требуемый режим доступа задается указанием во втором параметре конструктора символьной строки "г” или символьной строки "rw" соответственно, как показано ниже. Если существующий файл открывается как объект типа RandomAccessFile, он не удаляется.

* RandomAccessFile in = new RandomAccessFile("employee.dat", "r");
* RandomAccessFile inOut = new RandomAccessFile("employee.dat", "rw");

У любого файла с произвольным доступом имеется так называемый указатель файла, обозначающий позицию следующего байта, который будет считываться или записываться. Метод seek () устанавливает этот указатель на произвольную байтовую позицию в файле. Этому методу в качестве аргумента может передаваться целочисленное значение типа long в пределах от нуля до числового значения, обозначающего длину файла в байтах. А метод getFilePointer () возвращает текущую позицию указателя файла.

Класс RandomAccessFile реализует как интерфейс Datalnput, так и интерфейс DataOutput. Для чтения данных и записи данных в файл с произвольным доступом применяются методы readlnt () и writelnt (), а также методы readChar () и writeChar (), обсуждавшиеся в предыдущем разделе.

**18) Сохранение и загрузка сериализируемых объектов :**

Для сохранения данных объектов необходимо прежде всего открыть поток вывода объектов типа ObjectOutputStream следующим образом:

* ObjectOutputStream out = new ObjectOutputStream(new FileOutputStream("employee.dat"));

Далее для сохранения объекта остается лишь вызвать метод writeObject () из класса ObjectOutputStream, как показано в приведенном ниже фрагменте кода.

* Employee harry = new Employee("Harry Hacker", 50000, 1989, 10, 1);
* Manager boss = new Manager("Carl Cracker", 80000, 1987, 12, 15);
* out.writeObject(harry);
* out.writeObject(boss);

А для того чтобы прочитать данные объектов обратно, нужно сначала получить объект типа ObjectInputStream, т.е. поток ввода объектов, следующим образом:

* ObjectInputStream in = new ObjectlnputStream(new FilelnputStream("employee.dat"));

И затем извлечь объекты в том порядке, в каком они записывались, вызвав метод readObject (), как показано ниже.

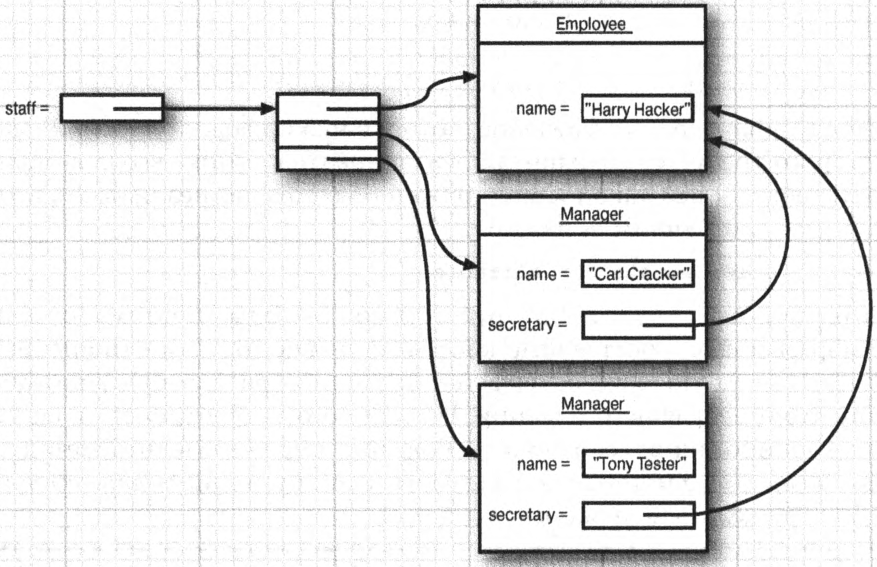
* Employee el = (Employee) in.readObject();
* Employee e2 = (Employee) in.readObject();

Имеется, однако, одно изменение, которое нужно внести в любой класс, объекты которого требуется сохранить и восстановить в потоке ввода-вывода объектов, а именно: каждый такой класс должен обязательно реализовать интерфейс Serializable следующим образом:

* class Employee implements Serializable { . . . }

У интерфейса Serializable отсутствуют методы, поэтому изменять каким-то образом свои собственные классы не нужно. В этом отношении интерфейс Serializable подобен интерфейсу Cloneable. Но для того чтобы сделать класс пригодным для клонирования, все равно требовалось переопределить метод clone () из класса Object. А для того, чтобы сделать класс пригодным для сериализации, ничего больше делать не нужно.

**19) Сериализация:**

****

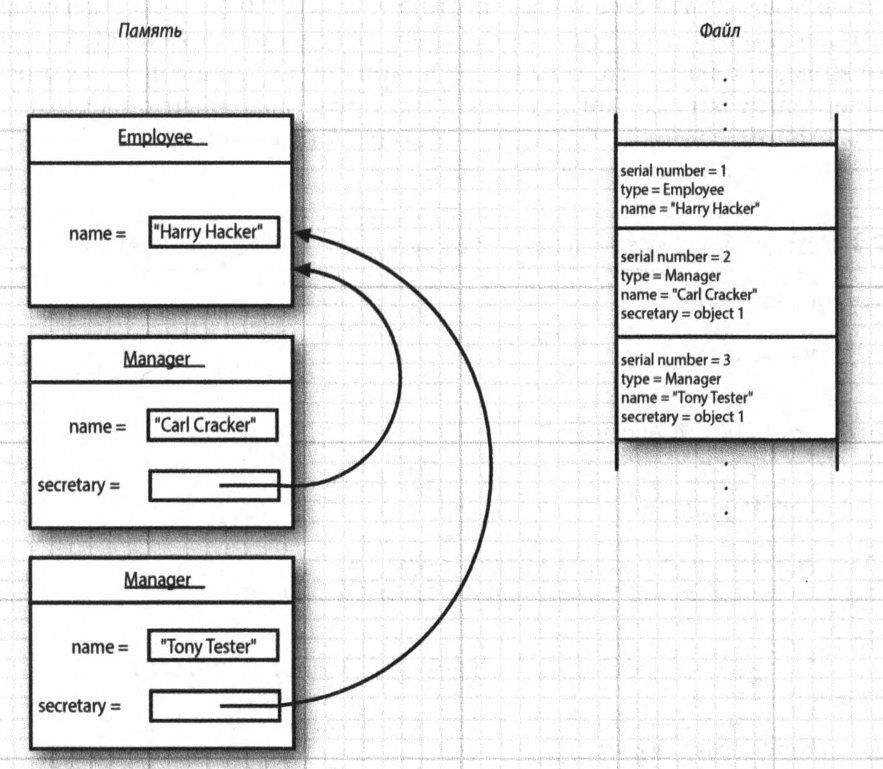
Сохранение такой разветвленной сети объектов оказывается непростой задачей. Разумеется, сохранять и восстанавливать адреса ячеек памяти для объектов секретарей нельзя. При повторной загрузке каждый такой объект, скорее всего, будет занимать уже совершенно другую ячейку памяти по сравнению с той, которую он занимал первоначально.

Поэтому каждый такой объект сохраняется под серийным номером, откуда, собственно говоря, и происходит название механизма сериализации объектов. Ниже описывается порядок действий при сериализации объектов.

1. Серийный (т.е. порядковый) номер связывается с каждой встречающейся ссылкой на объект, как показано на рисунке.

2. Если ссылка на объект встречается впервые, данные из этого объекта сохраняются в потоке ввода-вывода объектов.

3. Если же данные были ранее сохранены, просто добавляется метка "совпадает с объектом, сохраненным ранее под серийным номером х”.



При чтении объектов обратно из потока их ввода-вывода порядок действий меняется на обратный.

1. Если объект впервые указывается в потоке ввода-вывода объектов, он создается и инициализируется данными из потока, а связь серийного номера со ссылкой на объект запоминается.

2. Если же встречается метка "same as previously saved object with serial number x", то извлекается ссылка на объект по данному серийному номеру.

Пример(try в данном случае служит, не как отлов ошибок, а как блок в котором автоматически делается закрытие, т.е {}- это действие, а ()- это открытие потока , который потом автоматически закроется при выполнении действий):

try (ObjectOutputStream out = new ObjectOutputStream( new FileOutputStream("employee.dat"))) {

out.writeObject(staff); //запись объектов

}

try (ObjectlnputStream in = new ObjectInputStream( new FilelnputStream("employee.dat"))) {

Employee[] newStaff = (EmployeeU) in.readObject () ; //вывод объектов

for (Employee е : newStaff){ //вывод объектов массива

System.out.println(е); }}

**20) Класс Path :**

В интерфейсе Path имеется немало полезных методов для разделения путей к файлам на составляющие. В приведенном ниже фрагменте кода демонстрируется применение наиболее полезных методов из этого интерфейса.

* Path р = Paths.get("/home", "fred", "myprog.properties”);
* Path parent = p.getParent(); // путь /home/fred
* Path file = p.getFileName(); // путь myprog.properties
* Path root = p.getRootO; // путь /

Как пояснялось в первом томе настоящего издания, из объекта типа Path можно создать объект типа Scanner следующим образом:

* Scanner in = new Scanner(Paths.get("/home/fred/input.txt"));

**21) Чтение и запись данных в файл :**

Класс Files упрощает и ускоряет выполнение типичных операций над файлами. Например, содержимое всего файла нетрудно прочитать следующим образом:

* byte [ ] bytes = Files. readAUBytes (path) ;

Если же требуется прочитать содержимое файла в виде символьной строки, сначала необходимо вызвать метод readAUBytes (), как показано выше, а затем следующий конструктор:

* String content = new String(bytes, charset);

Но если требуется получить содержимое файла в виде последовательности строк, достаточно сделать следующий вызов:

* List<String> lines = Files. readAHLines (path, charset);

С другой стороны, если требуется записать в файл символьную строку, достаточно сделать приведенный ниже вызов.

* Files.write(path, content.getBytes(charset));

Кроме того, в файл можно записать целый ряд строк следующим образом:

* Files.write(path, lines);

Приведенные выше простые методы предназначены для манипулирования текстовыми файлами умеренной длины. Если же файл крупный или двоичный, то для манипулирования им можно воспользоваться упоминавшимися ранее потоками ввода-вывода или чтения и записи данных, как показано ниже. Имеющиеся у них удобные методы избавляют от необходимости обращаться непосредственно к классам Filelnputstream, FileOutputStream, BufferedReader или BufferedWriter.

* InputStream in = Files.newInputStream(path);
* OutputStream out = Files.newOutputStream(path);
* Reader in = Files.newBufferedReader(path, charset);
* Writer out = Files.newBufferedWriter(path, charset);

**22) Создание файлов и каталогов :**

Чтобы создать новый каталог, достаточно сделать следующий вызов:

* Files.createDirectory(path);

Чтобы создать пустой файл, следует сделать приведенный ниже вызов.

* Files.createFile(path);

**23) Копирование, перемещение, удаление файла:**

Чтобы скопировать файл из одного места в другое, достаточно сделать следующий вызов:

* Files.copy(fromPath, toPath);

А для того чтобы переместить файл, т.е. сделать его копию и удалить оригинал, следует сделать такой вызов:

* Files.move(fromPath, toPath);

И наконец, для удаления файла достаточно вызвать метод

* Files.delete(path);

Этот метод генерирует исключение, если файл не существует. Поэтому вместо него, возможно, придется вызвать приведенный ниже метод. Оба метода можно также использовать для удаления пустого каталога.

* boolean deleted = Files.deletelfExists(path);

**24) Получение сведений о файле :**Все файловые системы уведомляют об основных атрибутах, инкапсулированных в интерфейсе BasicFileAttributes. Они частично перекрывают прочие сведения о файлах. Ниже перечислены основные атрибуты файлов.

* Моменты времени, когда файл был создан, последний раз открывался и видоизменялся, в виде экземпляров класса java.nio.file.attribute. FileTime.
* Является ли файл обычным, каталогом, символической ссылкой или ничем из перечисленного.
* Размер файла.
* Файловый ключ — объект некоторого класса, характерный для применяемой файловой системы и способный (или не способный) однозначно определять файл.

Для получения перечисленных выше атрибутов файлов достаточно сделать следующий вызов:

* BasicFileAttributes attributes = files.readAttributes(path, BasicFileAttributes.class);