**Работа с JAR-архивами**

Иногда возникает потребность в том, чтобы java-программа могла просмотреть содержимое jar-архива, и извлечь его. В Интернете мне не удалось найти много информации по этому вопросу. Поэтому, решил разобраться во всём сам. Пошарив по документации, мне в голову пришли следующие мысли.

Если мы заранее знаем, что именно нам нужно извлечь из jar-файла, это не составит особого труда.

1. Для начала нам нужно создать объект класса **java.util.jar.JarFile** (далее **JarFile**), и указать для него имя просматриваемого jar-файла.
2. Затем, создаём объект класса **java.util.jar.JarEntry** (далее **JarEntry**) и указываем для него имя файла, который необходимо извлечь.
3. Для объекта **JarFile** создаём поток ввода с помощью метода **getInputStream()**. В качестве аргумента передадим ему объект **JarEntry**.
4. Ну а далее, работаем с потоками стандартным образом, используя методы **read()** и **write()**.

У нас должно получиться что-то вроде:

**JarFile** jarFile = **new** **JarFile**("some\_jar\_file.jar");

**JarEntry** jarEntry = **new** **JarEntry**("something.smth");

**InputStream** in = jarFile.getInputStream(jarEntry);

**FileOutputStream** out = **new** **FileOutputStream**(jarEntry.getName()); //будем создавать файл с тем же именем что и в jar-архиве

**int** t;

**while**((t = in.read()) != -1)

out.write(t);

Конечно же не забываем включить обработку исключения **IOException**.

Не правда ли, элементарно? Но что делать, если мы не знаем, что содержится в jar-архиве? Подумав немного над этим вопросом, я ознакомился с классом **ZipFile**. Ведь **JarFile** является его наследником. У класса **ZipFile** есть метод **entries()**, который возвращает объект интерфейса **Enumeration**, содержащий имена всех файлов, входящих в архив. Но так как пользоваться этим объектом, мягко говоря, неудобно, то имеет смысл перенести всё содержимое в объект класса **Vector**. Получаем что-то типа:

Enumeration<JarEntry> entries;

Vector<JarEntry> v;

…

**int** vc=0; /\* Vector capacity – количество элементов в векторе v.

Почему-то, метод v.capacity() выдаёт большее число,

чем на самом деле. Разбираться с этим не стал :) \*/

entries=jarFile.entries();

**while**(entries.hasMoreElements())

{

vect.add(entries.nextElement()); //заполняем коллекцию элементами (названиями каталогов и файлов), находящимися в архиве jar

vc++; //считаем количество таких элементов, для создания перебора этой коллекции в дальнейшем

}

Замечу сразу, что я писал программу для [Java 1.5](http://java.sun.com/j2se/1.5.0/download.jsp). Для [Java 1.4](http://java.sun.com/j2se/1.4.2/download.html) и более ранних версий работа с объектами **Enumeration** и **Vector** была бы немного другой, немного более трудной. Спасибо [Sun Microsystems](http://www.sun.com/" \t ") за облегчение и без того тяжкой участи программистов! :)

Ну а теперь, имея список содержимого **jar-архива**, мы спокойно можем распаковывать его, не забывая создавать подкаталоги, содержащиеся в архиве. Будем работать только с информацией находящейся в созданной нами коллекции “vect”. Для этого используем метод **mkdir()** класса **File**. Назовём наш метод **extract()**. Он может иметь следующий вид:

**public** **void** extract()

{

**File** tmpfile; /\* создаём временный объект, который будет создавать каталоги \*/

**JarEntry** tmpentry; /\* создаём временную ссылку на файл в архиве \*/

**FileOutputStream** out; /\* это и так понятно \*/

**InputStream** in; /\* и это тоже \*/

**int** t; /\* переменная для копирования файла \*/

**try**

{

**for**(**int** i=0;i<vc;i++) /\* создаём цикл для извлечения файлов

из архива. Вот нам и пригодилась переменная vc \*/

{

tmpentry=vect.get(i); /\* берём из вектора имя очередного файла или каталога \*/

tmpfile=new **File**(tmpentry.getName());

**if**(tmpentry.isDirectory()) /\* если tmpfile – каталог, \*/

{

**if**(!tmpfile.mkdir()) /\* то создаём его \*/

{

System.out.println("Can't create directory: "+tmpfile.getName()); /\* если он не создаётся, \*/

**return**; /\* выходим из функции \*/

}

}

**else** /\* ну а если tmpfile – не каталог, а файл, то спокойно извлекаем его \*/

{

in=jarFile.getInputStream(tmpentry);

out=new **FileOutputStream**(tmpfile);

**while**((t=in.read())!=-1)

out.write(t);

out.close(); /\* лучше потоки ввода и вывода закрывать, иначе наша программа \*/

in.close(); /\* может не сосем корректно работать (некоторые файлы могут теряться) \*/

}

}

}

**catch**(**IOException** e) /\* обрабатываем исключение \*/

{

System.out.println(e.getMessage());

e.printStackTrace(); /\* это, по-моему, не совсем обязательно \*/

System.exit(0);

}

}

Аналогичным образом можно работать и с zip-архивами. Необходимо только поменять JarFile и JarEntry на ZipFile и ZipEntry соответственно.

На основе этого можно сделать что-нибудь более сложное и подходящее под какие-то конкретные цели. Алгоритм может быть и более оптимально построен. В статье он не совсем оптимален для пущей наглядности.