ZIP files offer a packaging mechanism, allowing multiple files to be bundled together as one. Thus, when you need to download a group of files from the web, you can package them into one ZIP file for easier transport as a single file. The bundling can include additional information like directory hierarchy, thus preserving necessary paths for an application or series of resources once unbundled.

This tip will address three aspects of ZIP file usage:

- creating them
- adding files to them
- compressing those added files

Creating Zip Streams

The `ZipOutputStream` offers a stream for compressing the outgoing bytes. There is a single constructor for `ZipOutputStream`, one that accepts another `OutputStream`:

```
public ZipOutputStream(OutputStream out)
```

If the constructor argument is the type `FileOutputStream`, then the compressed bytes written by the `ZipOutputStream` will be saved to a file. However, you aren’t limited to using the `ZipOutputStream` with a file. You can also use the `OutputStream` that comes from a socket connection or some other non-file-oriented stream. Thus, for a file-oriented `ZipOutputStream`, the typical usage will look like this:

```
String path = "afile.zip";
FileOutputStream fos = new FileOutputStream(path);
ZipOutputStream zos = new ZipOutputStream(fos);
```

Adding Entries

Once created, you don’t just write bytes to a `ZipOutputStream`. Instead, you need to treat the output stream as a collection of components. Each component of a `ZipOutputStream` is paired with a `ZipEntry`. It is this `ZipEntry` that you create and then add to the `ZipOutputStream` before actually writing its contents to the stream.

```
String name = ...;
ZipEntry entry = new ZipEntry(name);
zos.putNextEntry(entry);
zos.write(<< all the bytes for entry >>);
```

Each entry serves as a marker in the overall stream, where you’ll find the bytes related to the entry in the library file. After the ZIP file has been created, when you need to get the entry contents back, just ask for the related input stream:

```
ZipFile zip = new ZipFile("afile.zip");
ZipEntry entry = zip.getEntry("anEntry.name");
InputStream is = zip.getInputStream(entry);
```

Now that you’ve seen how to create the zip file and add entries to that file, it is important to point out that the `java.util.zip` libraries offer some level of control for the added entries of the `ZipOutputStream`. First, the order you add entries to the `ZipOutputStream` is the order they are physically located in the .zip file. You can manipulate the enumeration of entries returned back by the `entries()` method of `ZipFile` to produce a list in alphabetical or size order, but the entries are still stored in the order they were written to the output stream.

Compressing Files

Files added to a ZIP/JAR file are compressed individually. Unlike Microsoft `CAB` files which compress the library package as a whole, files in a ZIP/JAR file are each compressed or not compressed separately. Before adding a `ZipEntry` to the `ZipOutputStream`, you determine whether its associated bytes are compressed. The `setMethod`

method of `ZipEntry` allows you to specify which of the two available compression formats to use. Use the `STORED` constant of `ZipEntry` to give you an uncompressed file and the `DEFLATED` setting for a compressed version. You cannot control the compression efficiency. That depends on the type of data in the associated entry. Straight text can be compressed to around 80% of its size quite easily, whereas MP3 or JPEG data will be compressed much less.

While you might think it obvious that everything should be compressed, it does take time to compress and uncompress a file. If the task of compressing is too costly a task to do at the point of creation, it may sometimes be better to just store the data of the whole file in a `STORED` format, which just stores the raw bytes. The same can be said of the time cost of uncompression. Of course, uncompressed files are larger, and you have to pay the cost with either higher disk space usage or bandwidth when transferring file. Keep in mind that you need to change the setting for each entry, not the `ZipFile` as a whole. However, it is more typical to compress or not compress a whole `ZipFile`, as opposed to different settings for each entry.

There is one key thing you need to know if you use the `STORED` constant for the compression method: you must explicitly set certain attributes of the `ZipEntry` which are automatically set when the entry is compressed. These are the size, compressed size, and the checksum of the entry's input stream. Assuming an input file, the size and compressed size can just be the file size. To compute the checksum, use the CRC32 class in the `java.util.zip` package. You cannot just pass in 0 or -1 to ignore the checksum value; the CRC value will be used to validate your input when creating the ZIP and when reading from it later.

```
ZipEntry entry = new ZipEntry(name);
entry.setMethod(ZipEntry.STORED);
entry.setCompressedSize(file.length());
entry.setSize(file.length());
CRC32 crc = new CRC32();
crc.update(<< all the bytes for entry >>);
entry.setCrc(crc.getValue());
zos.putNextEntry(entry);
```

To demonstrate, the following program will combine a series of files using the `STORED` compression method. The first argument to the program will be the ZIP file to create. Remaining arguments represent the files to add. If the ZIP file to create already exists, the program will exit without modifying the file. If you add a non-existing file to the ZIP file, the program will skip the non-existing file, adding any remaining command line arguments to the created ZIP.

```
import java.util.zip.*;
import java.io.*;

public class ZipIt {
    public static void main(String args[]) throws IOException {
        if (args.length < 2) {
            System.err.println("usage: java ZipIt Zip.zip file1 file2 file3"
            System.exit(-1);
        }
        File zipFile = new File(args[0]);
        if (zipFile.exists()) {
            System.err.println("Zip file already exists, please try another"
            System.exit(-2);
        }
        FileOutputStream fos = new FileOutputStream(zipFile);
        ZipOutputStream zos = new ZipOutputStream(fos);
        int bytesRead;
        byte[] buffer = new byte[1024];
        CRC32 crc = new CRC32();
        for (int i=1, n=args.length; i < n; i++) {
            String name = args[i];
            File file = new File(name);
            if (!file.exists()) {
                System.err.println("Skipping: " + name);
                continue;
```