Java

How to create a Java HashMap of user defined class type?

How to read a Matrix from user in Java?

Working with JAR and Manifest files In Java

Prerequisite – JAR file format

Whenever a developer wants to distribute a version of his software, then all he want is to distribute a single file and not a directory structure filled with class files. JAR files were designed for this purpose. A JAR file can contain both class files and other file types like sound and image files which may be included in the project. All the files in a JAR file are compressed using a format similar to zip.

Creating a JAR file – more Options

A jar file is created using jar tool. The general command looks somewhat like this:

```
jar options jar-file [manifest-file] file1 file2 file3 ...
```

- **jar – file :** name of jar file on which you want to use jar tool.
- **file1, file2, file3 :** files which you want to add inside a jar file. manifest-file is the name of file which contains manifest of that jar file, giving manifest-file as an argument is entirely optional.
- **c** : Creates a new or empty archive and adds files to it. If any of the specified file name are directories, then the jar program processes them recursively.
- **C** : Temporarily changes the directory.

- **e :** Creates an entry point in the manifest.
- **f** : Specifies the JAR file name as the second command-line argument. If this parameter is missing, jar will write the result to standard output (when creating a JAR file)or read it from standard input(when extracting or tabulating a JAR file).
- **i** : Creates an index file.
- **m** : Adds a manifest file to the JAR file. A manifest is a description of the archive contents and origin. Every archive has a default manifest, but you can supply your own if you want to authenticate the contents of the archive.
- **M** : Does not create a manifest file for the entries.
- **t :** Displays the table of contents.
- **u :** Updates an existing JAR file.
- **v :** Generates verbose output.
- **x :** Extract files. If you supply one or more file names, only those files are extracted Otherwise, all files are extracted.
- **0 :** Stores without zip compression.

The options of jar command are almost similar to that of UNIX tar command. In windows you can also get help about various options of jar command just by typing jar in cmd and then pressing enter, the output will be somewhat similar to this:

```
PS C:\UselessCoder> jar
Usage: jar {ctxui}[vfmn0PMe] [jar-file] [manifest-file] [entry-point] [-C dir] files ...
Options:
    -c  create new archive
    -t  list table of contents for archive
    -x  extract named (or all) files from archive
    -u  update existing archive
    -v  generate verbose output on standard output
    -f  specify archive file name
    -m  include manifest information from specified manifest file
    -n  perform Pack200 normalization after creating a new archive
    -e  specify application entry point for stand-alone application
        bundled into an executable jar file
    -0  store only; use no ZIP compression
    -P  preserve leading '/' (absolute path) and ".." (parent directory) components from file names
    -M  do not create a manifest file for the entries
    -i  generate index information for the specified jar files
    -C  change to the specified directory and include the following file
If any file is a directory then it is processed recursively.
The manifest file name, the archive file name and the entry point name are
specified in the same order as the 'm', 'f' and 'e' flags.

Example 1: to archive two class files into an archive called classes.jar:
       jar cvf classes.jar Foo.class Bar.class
Example 2: use an existing manifest file 'mymanifest' and archive all the
           files in the foo/ directory into 'classes.jar':
       jar cvfm classes.jar mymanifest -C foo/ .
```

Example :

For creating a JAR file which have two classes server.class and client.class and a Jpeg image logo.jpeg, one need to write following command :

```
jar cvf chat.jar server.class client.class logo.jpeg
```

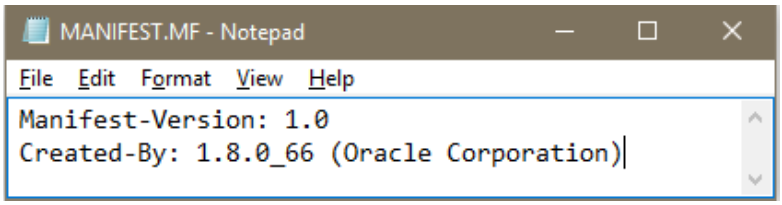
The output of above command will be somewhat like this:

```
PS C:\UselessCoder> jar cvf chat.jar client.class server.class logo.jpeg
added manifest
adding: client.class(in = 0) (out= 0)(stored 0%)
adding: server.class(in = 0) (out= 0)(stored 0%)
adding: logo.jpeg(in = 0) (out= 0)(stored 0%)
PS C:\UselessCoder>
```

It's a better practice to use **-v** option along with jar command as you will get to know how the things are going on.

Manifest File

Each JAR file contains a manifest file that describe the features of the archive. Each JAR file have a manifest file by default. Default manifest file is named as MANIFEST.MF and is present in the META-INF subdirectory of archive. Although the default manifest file contains just two entries, but complex manifest files can have way more. Here, is what a default manifest file looks like –



The entries of manifest files are grouped into sections. Each section have two entries section name and its value. We will see a bit later how these sections can really help us in controlling the properties of our archive. Manifest file can also be updated by using the **m** option of jar command. But there are certain things which need to kept in mind while updating manifest file otherwise you may get the following creepy message.

```
java.io.IOException: invalid manifest format
```

Things to keep in mind while handling Manifest files:

1. You should leave space between the name and value of any section in manifest file, like Version:1.1 is in valid section instead write Version: 1.1 that space between colon and 1.1 really matters a lot.
2. While specifying the main class you should not add .class extension at the end of class name. Simply specify the main class by typing:

   ```
   Main-Class: Classname
   ```

   (I'll be briefing about Main-Class section very shortly).
3. You must add newline at the end of file. You need not to write \n for specifying newline instead just leave the last line of your manifest file blank that will serve

the purpose.

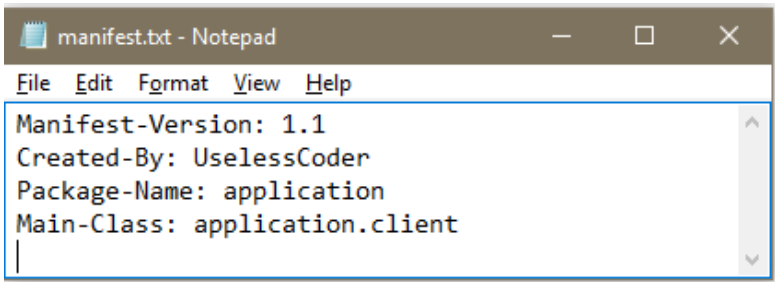
4. Text file for manifest must use UTF-8 encoding otherwise you may get into some trouble.

Example:

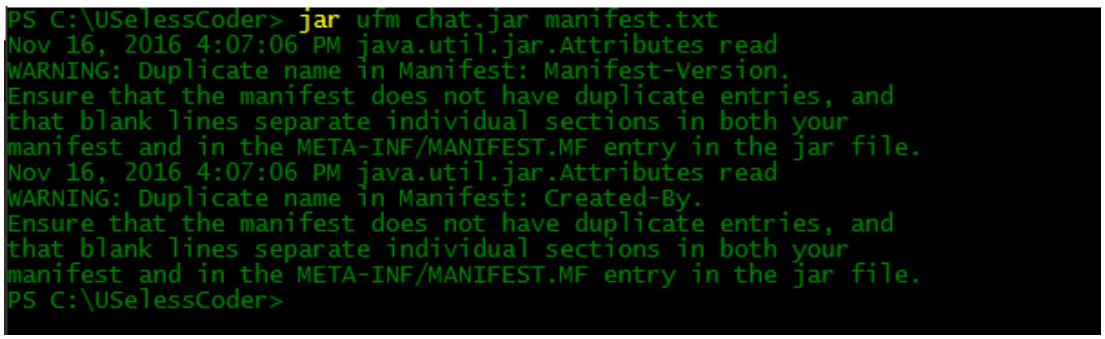
Now let's come back and update the contents of our chat.jar archive. To update the manifest file we simply need to write the following command:

```
jar uvfm chat.jar manifest.txt
```

Here manifest.txt is the new manifest file, which has following contents:



The output of above command will be somewhat like this:



Here we are getting two warnings because we are trying to overwrite to previously present entries.

Executable Jar Files

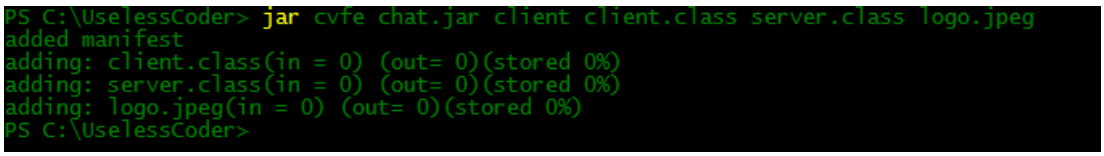
You can use the **e** option of jar command to specify the entry point of your program, ie. class which you normally want to invoke when launching your Java application.

Example:

To create chat.jar file having client class as main class you need to write following command –

```
jar cvfe chat.jar client client.class server.class logo.jpeg
```

The output of above command will be somewhat like this:



Remember not to add .class extension after the name of class which you want to set main class.

Alternatively you can add a Main-Class entry in the manifest file and then update it. For the above example you just need to add this entry:

```
Main-Class: client
```

With main class being set one can simply run a jar program by writing following command –

```
java -jar chat.jar
```

Depending on operating system configuration, users may even be able to launch application by double clicking the JAR file icon.

Package Sealing

Finally, we are going to discuss about package sealing in Java. We can seal a package in Java to ensure that no further classes can add themselves to it. You may want to seal a package if you use a package visible classes, methods and fields in your code. Without package sealing, other classes can add themselves to the same package and thereby gain access to package visible features.

- To achieve package sealing all one need to do is to put all classes of that package into a JAR file.
- By default the packages in a jar file are not sealed but one can change the global default by adding few lines in manifest file.
- Let's again consider the case of our chat.jar archive, now the package of classes client.class and server.class is application and we want to seal this package all we need to do is to add following line in the manifest file and update it.

```
 Name: application
Sealed: true
```

This is all from my side on how to work with JAR files. Stay Tuned!!

This article is contributed by **Abhey Rana(UselessCoder)**. If you like GeeksforGeeks and would like to contribute, you can also write an article using contribute.geeksforgeeks.org or mail your article to contribute@geeksforgeeks.org. See your article appearing on the GeeksforGeeks main page and help other Geeks.

Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above.

Recommended Posts:

Java.util.BitSet class methods in Java with Examples | Set 2
Shadowing of static functions in Java
How does default virtual behavior differ in C++ and Java ?
How are Java objects stored in memory?
How are parameters passed in Java?
Are static local variables allowed in Java?
final variables in Java
Default constructor in Java
Assigning values to static final variables in Java
Comparison of Exception Handling in C++ and Java
Does Java support goto?
Arrays in Java
Inheritance and constructors in Java
More restrictive access to a derived class method in Java
Comparison of static keyword in C++ and Java

Article Tags : Java

Practice Tags : Java

Be the First to upvote.