>javac -d . Hello.java

Will compile java file with package statement and create package structure in current directory.

>jad com.kc.dto.Hello

Will decompile above file on path and create Hello.jad file you can change extention to java

Make changes in code and again compile so will create .class file

To make jar

A normal jar file is the non-executable one, such as a library jar file or an applet jar file. The following command put all files under the kc\classes directory into a new jar file called example.jar:

>jar cfv example.jar -C kc\classes .

Note that there is a dot (.) at the end which denotes all files. The c option is to create, the f is to specify jar file name, the v is to generate verbose output, and the -C is to specify the directory containing the files to be added. Here’s the output (some entries are removed for brevity):

In case the current directory is build\classes, the -C option can be omitted (but the jar file needs to be moved upward two levels). For example:

>jar cfv ..\..\SwingEmailSender.jar .

If we want to add only files in a specific package:

>jar cfv SwingEmailSender.jar -C build\classes net\codejava\swing\mail

That takes only files in the package net.codejava.swing.mail (recursively).

The following command adds only one class (SwingEmailSender.class) to the jar file:

>jar cfv SwingEmailSender.jar -C build\classes net\codejava\swing\mail\SwingEmailSender.class

If we want to add multiple classes, separate them by spaces (with full path). Suppose that the current directory is build\classes, the following command adds two classes (SwingEmailSender.class and JFilePicker.class) to the jar file:

>jar cf ..\..\SwingEmailSender.jar net\codejava\swing\mail\SwingEmailSender.class net\codejava\swing\JFilePicker.class

2. Including/Excluding manifest file

By default, the jar tool automatically creates a manifest file when generating a new jar file. If we don’t want to have the manifest created, use the M option as in the following example:

>jar cfvM SwingEmailSender.jar -C build\classes .

In case we want to manually add an external manifest file, use the m option as in the following example:

1

jar cfm SwingEmailSender.jar manifest.txt -C build\classes .

Here, content of the manifest.txt is copied to the generated manifest file inside the jar file.

3. Creating executable jar file

An executable jar file (stand-alone desktop application) must have information about the application entry point (main class) defined in its manifest file. The entry point can be specified either directly on the command line (using the e option), or in an external manifest file.

The following command creates an executable jar file with the main class is net.codejava.swing.mail.SwingEmailSender:

jar cfe SwingEmailSender.jar net.codejava.swing.mail.SwingEmailSender -C build\classes .

If using an external manifest file, remember to specify the main class by adding the following entry to the manifest file. For example:

Main-Class: net.codejava.swing.mail.SwingEmailSender

And the following command creates an executable jar file with the main class is specified in the external manifest file (manifest.txt):

>jar cfm SwingEmailSender.jar manifest.txt -C build\classes .

4. Viewing content of jar file

The t option is used in conjunction with the f and v to list table of contents of a jar file. For example:

>jar tf SwingEmailSender.jar

Adding the v option will include file size and date time for each entry. For example:

>jar tfv SwingEmailSender.jar

It’s also possible to list only files under a specific package. The following command lists only files under the package net.codejava.swing.mail:

>jar tf SwingEmailSender.jar net/codejava/swing/mail

5. Updating content of jar file

The u option is used to update an existing jar file, e.g. adding new classes and removing/updating manifest file.

The following example adds/updates the classes under the package net.codejava.swing.download to the existing archive:

>jar uf SwingEmailSender.jar -C build\classes net/codejava/swing/download

The following command updates everything but removes the manifest file:

>jar ufM SwingEmailSender.jar -C build\classes .

And the following command updates everything including the manifest file:

jar ufm SwingEmailSender.jar manifest.txt -C build\classes .

Recommended Book: Core Java, Volume II--Advanced Features (9th Edition) (Core Series)

6. Extracting content of jar file

To extract a jar file, use the x option in conjunction with the f and v. The following command extracts all content of the SwingEmailSender.jar file to the current directory:

>jar xf SwingEmailSender.jar

Add the v option if we want verbose output:

>jar xfv SwingEmailSender.jar

If we want to extract only files under a specific package, pass the package path after the jar file name. For example:

>jar xfv SwingEmailSender.jar net/codejava/swing/mail

That extracts only files in the package net.codejava.swing.mail. Here’s the verbose output:

7. Disabling jar compression

By default, the generated jar file is compressed as ZIP/ZLIB compression format. However, it’s possible to disable the default compression by specifying the 0 option. For example:

1

jar cfv0 SwingEmailSender.jar -C build\classes .

8. Generating index information

The i option allows us to generate an INDEX.LIST file that contains information about the specified jar file, such as the dependent jar files and location of all packages. For example:

1

jar i SwingEmailSender.jar

The INDEX.LIST file is generated under the META-INF directory. Here’s a sample content of this file:

9. Using command line argument files

This is a useful feature that shortens and simplifies a lengthy jar command line. Just create a text file that contains jar options and arguments (separated by spaces or new lines), and then execute the jar command in the following form:

>jar @file1 @file2...

For example, create a classes.txt file containing the classes to be included as follows:

>jar cfv SwingEmailSender.jar -C build\classes @classes.txt

Create an options.txt file that contains all the options as follows:

cfve SwingEmailSender.jar

net.codejava.swing.mail.SwingEmailSender

-C build\classes

Then execute the following command:

1

jar @options.txt @classes.txt

10. Passing JRE options

The -J option allows us to pass options to the underlying Java Runtime Environment (JRE) when needed. For example:

1

jar cfv SwingEmailSender.jar -C build\classes . -J-Xmx128M

That passes the option -Xmx128M (maximum memory size is 128MB) to the runtime environment. Note that there must be no space between -J and the options.

References

<https://docs.oracle.com/javase/7/docs/technotes/tools/windows/jar.html>

<http://www.codejava.net/java-core/tools/using-jar-command-examples>