

[Advertiser Disclosure](#)

Custom Search

- [Java](#)
- [Microsoft & .NET](#)
- [Mobile](#)
- [Android](#)
- [Open Source](#)
- [Cloud](#)
- [Database](#)
- [Architecture](#)
- [Other](#)
 - [Project Management](#)
 - [PHP](#)
 - [Perl](#)
 - [Ruby](#)
 - [Services](#)
 - [Other Languages](#)
 - [White papers](#)
 - [Research Center](#)
- [NEW: Slideshows](#)
- [Sponsored](#)
-
-
-

May 13, 2020

Hot Topics:

[prev](#)

- [Android](#)
- [Java](#)
- [PHP](#)
- [Microsoft & .NET](#)
- [Cloud](#)
- [Open Source](#)
- [Database](#)

[next](#)[Developer.com](#)[Java](#)[Data & Java](#)[Read More in Data & Java »](#)

Effective Use of Java Command Line Tools

Download the authoritative guide: [Cloud Computing: Using the Cloud for Competitive Advantage](#)

-
- By [Manoj Debnath](#)
- [Send Email »](#)
- [More Articles »](#)

Tweet

Java command line tools are a sophisticated way to give a finer touch in the application development process. Java is bundled with several such tools. They can be found in the *bin* folder of the JDK installation directory. When working with our favourite IDE, we often do not notice the real power behind the tools because the IDE takes care of them in the background, giving the programmer a headache-free productive experience. This is quite appreciable, yet in cases blinds the programmer of the real essence of command line tools. Oracle provides extensive documentation on [Java tools and their utilities](#). In this article, we'll explore some of the tools that come in quite handy when programming in Java.

And, for the time being, forget about your IDE. We'll go command line from here and use a very basic text editor, something like Notepad or gedit for coding.

Compilation and Execution

Once the program code is complete, for compilation we need *javac* (a Java compiler that transforms source code into byte code or a .java file into a .class file) and to run the command, we need *java*.

The *java* command initiates the runtime environment (JRE), loads specific classes, and searches for the start-up and other application specific classes (in the location: bootstrap class path, installed extensions and user class path). The start-up class contains a program initiation method such as *main*, which is finally launched within the periphery of Java Virtual Machine.

Sample Program

- [Email Article](#)
- [Print Article](#)

```
class testapp{
    public static void main(String[] args){
        System.out.println("Testing command line tools");
    }
}
```

To compile the preceding code, we would write the following:

```
javac testapp.java
```

This will create the *testapp.class*, which is a byte code file in the current directory. To execute the program, we simply can write the following command:

```
java testapp
```

Now, suppose, Java source files are distributed across a complex hierarchy of packages in the current directory. How would we compile and then run the program?

Let's create the scenario.

Create three folders, one inside another, as */org/mano/example* in the current directory. Put the following Java source files, such as ***p1.java***, inside */org*, ***p2.java*** inside */org/mano*, ***p3.java***, and ***testapp1.java*** inside */org/mano/example*.

p1.java

```
package org;
/** This is my javadoc comment about the class*/
public class p1 {
    String str="from org package";
    /**
     * the method return void and prints String constant
     */
    public void printStr(){
        System.out.println(str);
    }
}
```

p2.java

```
package org.mano;
/** This is my javadoc comment about the class*/
public class p2 {
    String str="from org.mano package";
    /**
     * the method return void and prints String constant
     */
    public void printStr(){
        System.out.println(str);
    }
}
```

p3.java

```
package org.mano.example;
/** This is my javadoc comment about the class*/
public class p3 {
    String str="from org.mano.example package";
    /**
     * the method return void and prints String constant
     */
}
```

```

    */
    public void printStr(){
        System.out.println(str);
    }
}

```

testapp1.java

```

package org.mano.example;
import org.p1;
import org.mano.p2;
/** This is my avadoc comment about the class*/
public class testapp1 {

    /**
     * this is main method
     */
    public static void main(String[] args){
        p1 a=new p1();
        a.printStr();
        p2 b=new p2();
        b.printStr();
        p3 c=new p3();
        c.printStr();
    }
}

```

To compile, use *javac* as follows:

```

javac -cp . org/*.java org/mano/*.java org/
      mano/example/*.java

```

Note that the "-cp ." option denotes the class path to the current directory. The "*" denotes all Java source files in the particular directory. It, however, can be replaced by a specific file name, but "*" seems more convenient for our purpose at the moment. The command thus would compile and create *.class* files in the respective directory.

Now, to execute *testapp1* (the *testapp1* class contains the main function) use the following command:

```

java org.mano.example.testapp1

```



Figure 1: Output in a command line terminal

There are several other optional parameters we can use both with the *javac* and *java* commands to tweak compilation and program launch process, such as putting *.class* files in a specific directory, enabling/disabling assertions, loading a splash image before application launch, and so forth. Refer to documentation for more information about the optional parameters.

Packaging and Executing a JAR File

Archiving and packaging with the *jar* utility is another important command line tool to combine multiple files into a single package, called a JAR file. A JAR contains a manifest file, called the *content describer*, and, optionally, a signature file for security.

Let's create a JAR archive for the preceding example. First, create a *Manifest.txt* file in the current directory (in my case, it is *java_pro*, refer to Figure 1) with the content as follows:

```

Manifest-Version: 1.0
Created-By: 1.8.0_65 (Oracle Corporation)
Main-Class: org.mano.example.testapp1

```

Now, use the following command to create the archive:

```

jar cmf Manifest.txt testapp1.jar org

```

This will create a JAR file named *testapp1.jar* in the current directory. Now, to display the content of the jar file, use the following command:

```

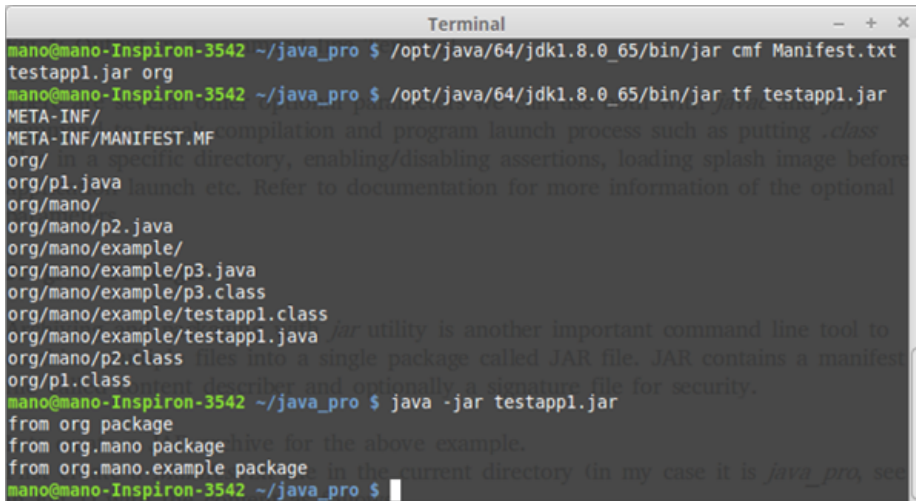
jar tf testapp1.jar

```

This command will display the contents as (shown in Figure 2). Now, to launch the JAR file, use the *java* command as follows:

```
java -jar testapp1.jar
```

That's it. The package is now ready to port and launch in a different machine (JVM).



```
Terminal
mano@mano-Inspiron-3542 ~/java_pro $ /opt/java/64/jdk1.8.0_65/bin/jar cmf Manifest.txt
testapp1.jar org
mano@mano-Inspiron-3542 ~/java_pro $ /opt/java/64/jdk1.8.0_65/bin/jar tf testapp1.jar
META-INF/
META-INF/MANIFEST.MF
org/
org/p1.java
org/mano/
org/mano/p2.java
org/mano/example/
org/mano/example/p3.java
org/mano/example/p3.class
org/mano/example/testapp1.class
org/mano/example/testapp1.java
org/mano/p2.class
org/p1.class
org/p1.class
mano@mano-Inspiron-3542 ~/java_pro $ java -jar testapp1.jar
from org package
from org.mano package
from org.mano.example package
mano@mano-Inspiron-3542 ~/java_pro $
```

Figure 2: Packaging with the *jar* utility and launching the application

Creating Documentation

The *javadoc* utility can be used to create documentation from the source code. The comments we have written in the source code in the preceding example will be used for method and class description. Now, to create documentation of the source code, we may create a *docs* folder in the current directory (otherwise, all HTML files will be created in the current directory) and use the following command:

```
javadoc -d ./docs/ org org.mano org.mano.example
p1.java p2.java p3.java testapp1.java
```

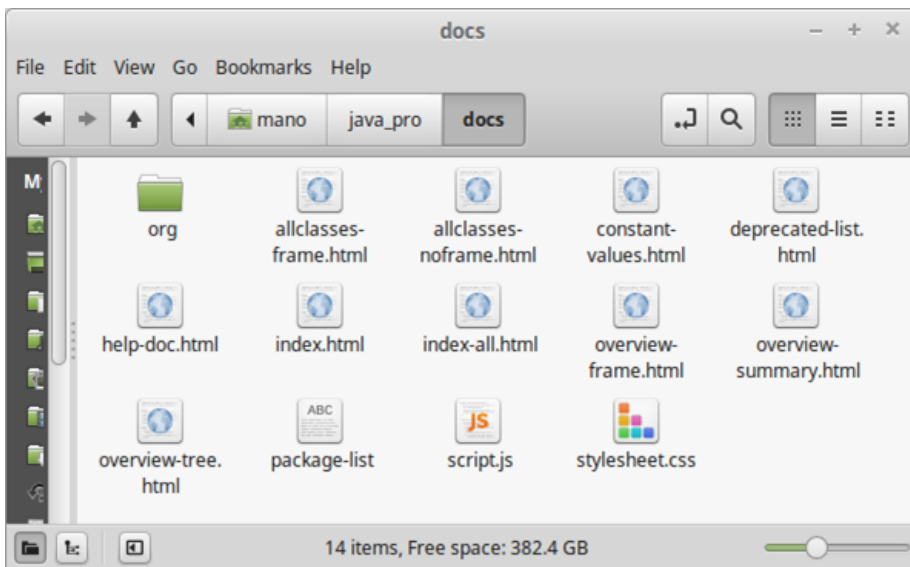


Figure 3: Java documentation files

Creating a usable and good documentation file is very important. There are standard procedures for creating one in Java. The purpose here is to show the usage of the *javadoc* tool, so it is better not to pay attention to the useless comments written in the source code but to the essence of *javadoc* command line utility. :)

Conclusion

There are numerous other command line utilities but these four are the basics. Here, I have tried to cover the fourfold utility of the command line tools such as compilation, execution, packaging, and documentation, with very specific implementation examples. If you are an IDE buff, I suggest that you stoop down to the command line sometime and feel the fun of using utility tools. You'll thank me later.

This article was originally published on January 8, 2016



Start the discussion!

LoudVoice Comments

Privacy Policy

Powered by OneAll

Enterprise Development Update

Don't miss an article. Subscribe to our newsletter below.

SIGN UP

Most Popular Developer Stories

- [Today](#)
- [This Week](#)
- [All-Time](#)
- [1 Using JDBC with MySQL, Getting Started](#)
- [2 An Introduction to Java Annotations](#)
- [3 MIDP Programming with J2ME](#)
- [4 An Introduction to JSP Standard Template Library \(JSTL\)](#)
- [5 Debugging a Java Program with Eclipse](#)
- [1 Using JDBC with MySQL, Getting Started](#)
- [2 An Introduction to Java Annotations](#)
- [3 An Introduction to JSP Standard Template Library \(JSTL\)](#)
- [4 MIDP Programming with J2ME](#)
- [5 Debugging a Java Program with Eclipse](#)
- [1 Using JDBC with MySQL, Getting Started](#)
- [2 An Introduction to Java Annotations](#)
- [3 An Introduction to JSP Standard Template Library \(JSTL\)](#)
- [4 MIDP Programming with J2ME](#)
- [5 Debugging a Java Program with Eclipse](#)

Most Commented On

- [This Week](#)
- [This Month](#)
- [All-Time](#)
- [1 10 Experimental PHP Projects Pushing the Envelope](#)
- [2 Day 1: Learning the Basics of PL/SQL](#)
- [3 C# Tip: Placing Your C# Application in the System Tray](#)
- [4 Logical Versus Physical Database Modeling](#)
- [5 Is Ubuntu Contributing as Much as It Should to Free Software Projects?](#)
- [1 Day 1: Learning the Basics of PL/SQL](#)
- [2 The 5 Developer Certifications You'll Wish You Had in 2015](#)
- [3 10 Experimental PHP Projects Pushing the Envelope](#)
- [4 An Introduction to Struts](#)
- [5 Inside Facebook's Open Source Infrastructure](#)
- [1 Creating Use Case Diagrams](#)
- [2 Day 1: Learning the Basics of PL/SQL](#)
- [3 C# Tip: Placing Your C# Application in the System Tray](#)
- [4 Using ASP.NET To Send Email](#)
- [5 Using JDBC with MySQL, Getting Started](#)



A Developer.com Property

[Terms of Service](#) | [About Us](#) | [Privacy Notice](#) | [Contact Us](#) | [Advertise](#) | [Sitemap](#) | [California - Do Not Sell My Info](#)

Copyright 2020 TechnologyAdvice. All Rights Reserved.

Advertiser Disclosure: Some of the products that appear on this site are from companies from which TechnologyAdvice receives compensation. This compensation may impact how and where products appear on this site including, for example, the order in which they appear. TechnologyAdvice does not include all companies or all types of products available in the marketplace.

Thanks for your registration, follow us on our social networks to keep up-to-date