**Java jar, war and ear files:**

A jar (Java Archive) is a package file format typically used to aggregate many Java .class files and associated metadata and resources (text file, image file, etc.) into one file to distribute application software or libraries on java platform.

(A ***JAR file*** includes ***all components*** to make a ***self-contained executable java application***).

Jar files are created using the **jar.exe** utility program from JDK. We can make **jar file runnable** by telling jar.exe **which class has** **main**. This should be defined in **manifest file**, it contains only Main-Class directive. **[ Main-Class: classFileNameOnly no extension of .class]**

**To create a jar file: [ c – to create, v – verbose output, m-to define manifest file, f- to specify archive file name].**

* jar -cvf <jarfilename.jar> <filename.class> --- with one class file.
* Jar -cvf <jarfilename.jar> A.class B.class C.class -- with three class files.
* Jar -cvf <jarfilename.jar> \*.class -- with all class files.
* Jar -cvf <jarfilename.jar > \*.\* -- with all files in current working directory.
* Jar -cvfm <jarfilename.jar > <filename.class> manifest.txt \*.class – to give manifest file.

**The easiest way:** navigate to **folder where java files are located**.

* **java \*.java** - - compile all java files
* **echo Main-class: *classNameOnly* > manifest.txt** - - create manifest file with ***className*** only as main class.
* **jar cvfm Myjar.jar manifest.txt \*.class [ \* - means it includes all sub folders]**

To run the executable jar

* **java -jar Myjar.jar**

**To run a jar file: [ java -jar to execute]**

* **java -jar <jarfilename.jar> -- to run a jar file.**

**To extract a jar file: [ x- to extract the file]**

* **jar -xvf <jarfilename.jar> -- extracts jar file.**

**To view table of contents: [v- to view in table format]**

* **jar -tvf <jarfilename.jar> -- to view table of contents.**

**To update a jar file: [u – to update the jar file]**

* **jar -uf <jarfilename.jar> inputfile(s) -- to update jar file with new files.**

Note: When we create a jar file, it automatically receives the default manifest file, there can be only one manifest file and it always has the classpath.(META-INF/MANIFEST.MF).

[Note: To use jar file we need to mention .jar filename]

**META-INF/MANIFEST.MF**: is directory which contains the manifest (list of files) of a JAR and created when we write JAR file.

**WEB-INF**: (contains all deployment information also contains web.xml file) is a directory contains a hierarchy in which we find the necessary configuration information for the web application and all the class files for servlets and classes that are called up by Java Server Pages. The WEB-INF directory is a vital component of web application, which will not without it.

**Note: Same commands are used to create .war file and .ear file with corresponding extension.**

For **.war** file we have special jar file with **web.xml file** in the **WEB-INF folder.**

For **.ear** file we have special jar file containing **application.xml file** in the **META-INF folder**

**WAR file:** contains web applications ( jsp, servlets, ejb, etc) each war file represents an web application. (***A WAR file contains files related to a web project, it contains servlet, JSP, XML, HTML, CSS and Javascript that can be deployed on any servlet/JSP container. The JDK’s jar tool helps to create a WAR file. These files are inside the WEB-INF folder of the project***).

[ note: to create .war go inside project folder outside (**WEB-INF**) and write **jar -cvf projname.war \***].

The **/WEB-INF** directory in **WAR file** contains a file named web.xml which defines the structure of the web application.

**EAR file**: contains anything from J2EE, represents ( jsp, servlets, ejb, etc).

**java:** console output will be executed, means sop will be displayed but other programs will run.

**javaw:** console output will not be executed, means no sop, uses esp, GUI based execution.

[ No associated console window, Java-window].

**javaws:** used to download and run the distributed web application, no console is associated.

[ java web start utility].

[Note: all are part of JRE and use same JVM].

**classpath:** describes the location of where required .class files are available.

**path:** represents location where binary executable are available.

[ Java compiler and JVM will use them].

**bin:** binary executables

**Java Binary:** Java source file is compiled into binary class file. These binary files are used in java virtual machine for execution.

**Web Server:** provides support only for web applications ( eg. Servlets, jsp, html etc.).

**Application Server:** any technology java J2EE (eg. Servlets, jsp, html, EJB, JMS etc.).

[ Note: in Application server there is built in Web Server].

**strictfp:** is a keyword in java, used for restricting floating point calculations and ensuring same result on every platform operation in floating point variable.

**volatile:** is a keyword in java, using volatile is making class **thread safe**.

**Thread safe:** means that a method or class instance can be used by multiple threads at the same time without any problem.

**Factory Method:** one java class method is able to return same class object or different class object.

@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

**Creating jar file and executing:**

Step1: compile .java file ---> in order to create a .class file.

Step 2: jar -cvf <MyJarFile.jar> <classfilename> ----> to create a jar file

Step 3: java -cp <MyJarFile.jar> <classfilename> -----> to execute class file.

Storing the class file in a jar file is that we can execute class from any location on the file system.

Or to execute from any where

* java -cp ./path to folder/<MyJarFile.jar> <classFileName>

**Creating war file:**

Step1: compile .java file ---> in order to create a .class file.

Step 2: jar -cvf <MyJarFile.war> <classfilename> ----> to create a jar file

Step 3: java -cp <MyJarFile.war> <classfilename> -----> to execute class file.

Storing the class file in a jar file is that we can execute class from any location on the file system.

Or to execute from any where

* java -cp ./path to folder/<MyJarFile.war> <classFileName>

**Creating ear file:**

Step1: compile .java file ---> in order to create a .class file.

Step 2: jar -cvf <MyJarFile.ear> <classfilename> ----> to create a jar file

Step 3: java -cp <MyJarFile.ear> <classfilename> -----> to execute class file.

Storing the class file in a jar file is that we can execute class from any location on the file system.

Or to execute from any where

* java -cp ./path to folder/<MyJarFile.war> <classFileName>

**For maven project:**

* mvn package ---> will create the {artifactid}-{version}.war file under ${basedir}/target directoy. If the packaging POM element is set to war in the pom.xml file.

Note: If no packing value has been specified, it will default to jar file.

**Classpath:** is a parameter in the Java Virtual Machine or Java compiler that specifies the location of user-defined classes and packages.

The parameter may be set either on command-line or through environment variable.

* When we execute java programs, the JVM finds and load classes.
* The classpath tells where to look in the file system of the classes.

JVM searched for and loads classes in order:

1. Bootstrap classes: the classes(bytecode) that are fundamental to the Java platform (public classes of Java library and private classes that are necessary for this library) .
2. Extension classes: packages that are in ext directory of JDK or JRE ( jre/lib/ext).
3. User-defined packages and libraries.

**Setting the path to execute Java programs:**

D:\myprogram\

|

---> org\

|

---> mypackage\

|

---> HelloWorld.class

---> SupportClass.class

---> UtilClass.class

Windows:

* Java -classpath D:\myprogram org.mypackage.HellowWord

Java -classpath <path to the package> <pkgName.classFileName>

Linux:

* Java -cp D:\myprogram org.mypackage.HellowWord

Java -cp <path to the package> pkgName.classFileName

**Setting the path through an environment variable:**

The environment variable named **CLASSPATH** may be alternatively used to set classpath.

* Set CLASSPATH=D:\myprogram

>java org.mypackage.Helloworld

set CLASSPATH=<path of the directory>

[ if none specified then the Current working directory is used as classpath (i.e. “.” ].

**Setting the path of a jar file:**

if a program uses a supporting library enclosed in a jar file physically located in a directory.

D:\myprogram\

|

---> lib\

|

---> supportLib.jar

|

---> org\

|

--> mypackage\

|

---> HelloWorld.class

---> SupportClass.class

---> UtilClass.class

* java -classpath D:\myprogram;D:\myprogram\lib\supportLib.jar org.mypackage.HelloWorld

java -classpath <path to the package<;<path including .jar file> <pkgName.classFileName>

or Alternatively.

* set CLASSPATH=<path to the package<;<path including .jar file> <pkgName.classFileName>

**Adding all jar file in a directory:**

* **java -classpath “.;<path to the directory>\\*” <directoryName>**

@@ important topic:

**Setting the path in a manifest file:** if the program has been enclosed in a jar file, located directly in D

D:\myprogram\

|

---> helloWorld.jar

|

---> lib\

|

---> supportLib.jar

The manifest file defined in helloWorld.jar has this definition.

Main-Class: org.mypackage.HelloWord

Class-Path: lib/supportLib.jar

The program is launched with following command.

* java -jar D:\myprogram\helloWorld.jar [app arguments]