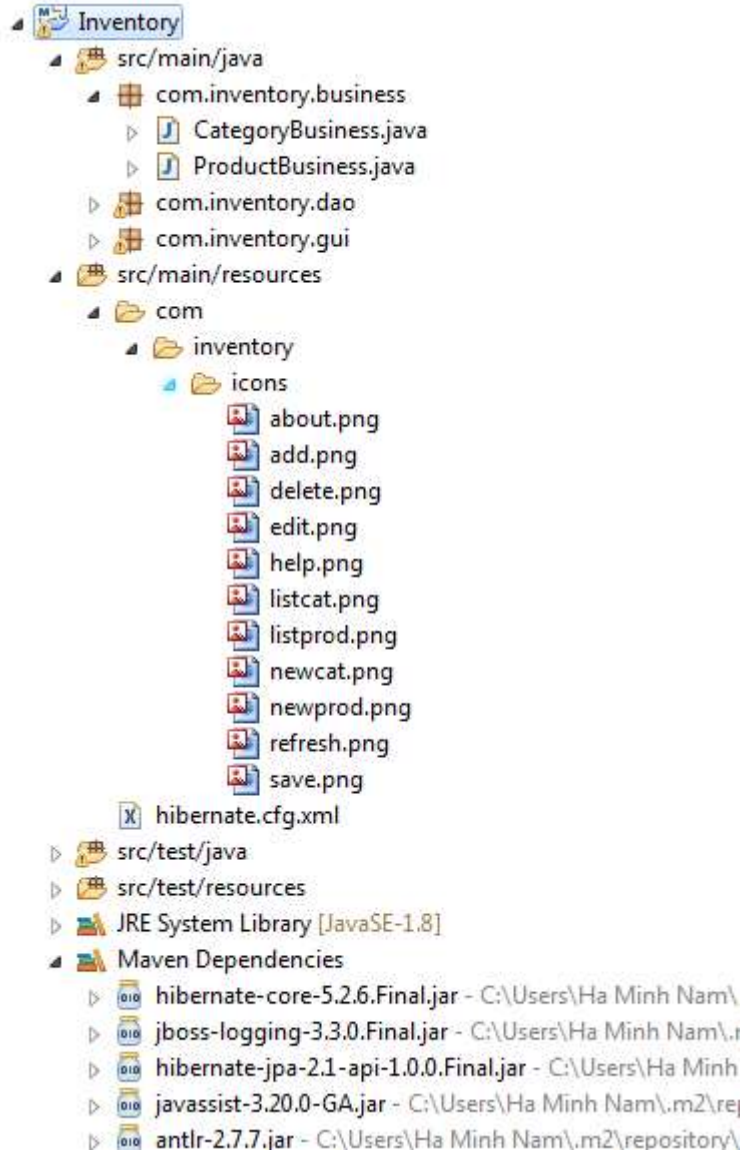
How to Create Executable JAR file with resources and dependencies using Maven in Eclipse

Last Updated on 21 February 2017 | Print Email

Java Spring Framework Masterclass: Beginner to Professional

In this article, we are going to guide you how to create an executable JAR file from a Java project which uses Maven build system. It's worth mentioning that the project contains resource files (XML configuration, images, etc) that are in a directory different from the Java source files directory. And the project also contains some dependencies as well. The following screenshot illustrates such a Java project:



As you can see, this Java project contains resource files like images and XML, and dependencies for Hibernate framework. So what we are going to show you is how to generate the executable JAR file of this project in a manner that the JAR file contains all the resources and dependencies (uber-JAR or fat JAR file).

And you know, creating such JAR file is not possible with Eclipse's Export function (File > Export > Java > Runnable JAR file). But the good news is that Maven provides a great plugin that is dedicated for creating executable JAR files with resources and dependencies. This plugin is called Maven Assembly Plugin.

To use the Maven Assembly Plugin, declare the following XML code in the **<plugins>** section which is under the **<build>**section of the **pom.xml** file:

```
1       <plugin>
2         <artifactId>maven-assembly-plugin</artifactId>
3         <configuration>
```

```
4             <archive>
5               <manifest>
6                 <mainClass>com.inventory.gui.InventoryApp</mainClass>
7               </manifest>
8             </archive>
9             <descriptorRefs>
10              <descriptorRef>jar-with-dependencies</descriptorRef>
11            </descriptorRefs>
12          </configuration>
13      </plugin>
```

That makes the pom.xml file looks like the following:

```
        <project ....>
1         <modelVersion>4.0.0</modelVersion>
2         <groupId>com.inventory</groupId>
3         <artifactId>Inventory</artifactId>
4         <packaging>jar</packaging>
5         <version>0.0.1-SNAPSHOT</version>
6         <name>Inventory Management</name>
7
8
9         <dependencies>
10          ....
11
12        </dependencies>
13
14        <build>
15          <plugins>
16            <plugin>
17              <artifactId>maven-assembly-plugin</artifactId>
18              <configuration>
19                <archive>
20                  <manifest>
21                    <mainClass>com.inventory.gui.InventoryApp</mainClass>
22                  </manifest>
23                </archive>
24                <descriptorRefs>
25                  <descriptorRef>jar-with-dependencies</descriptorRef>
26                </descriptorRefs>
27              </configuration>
28            </plugin>
29          </plugins>
30        </build>
31      </project>
32
```

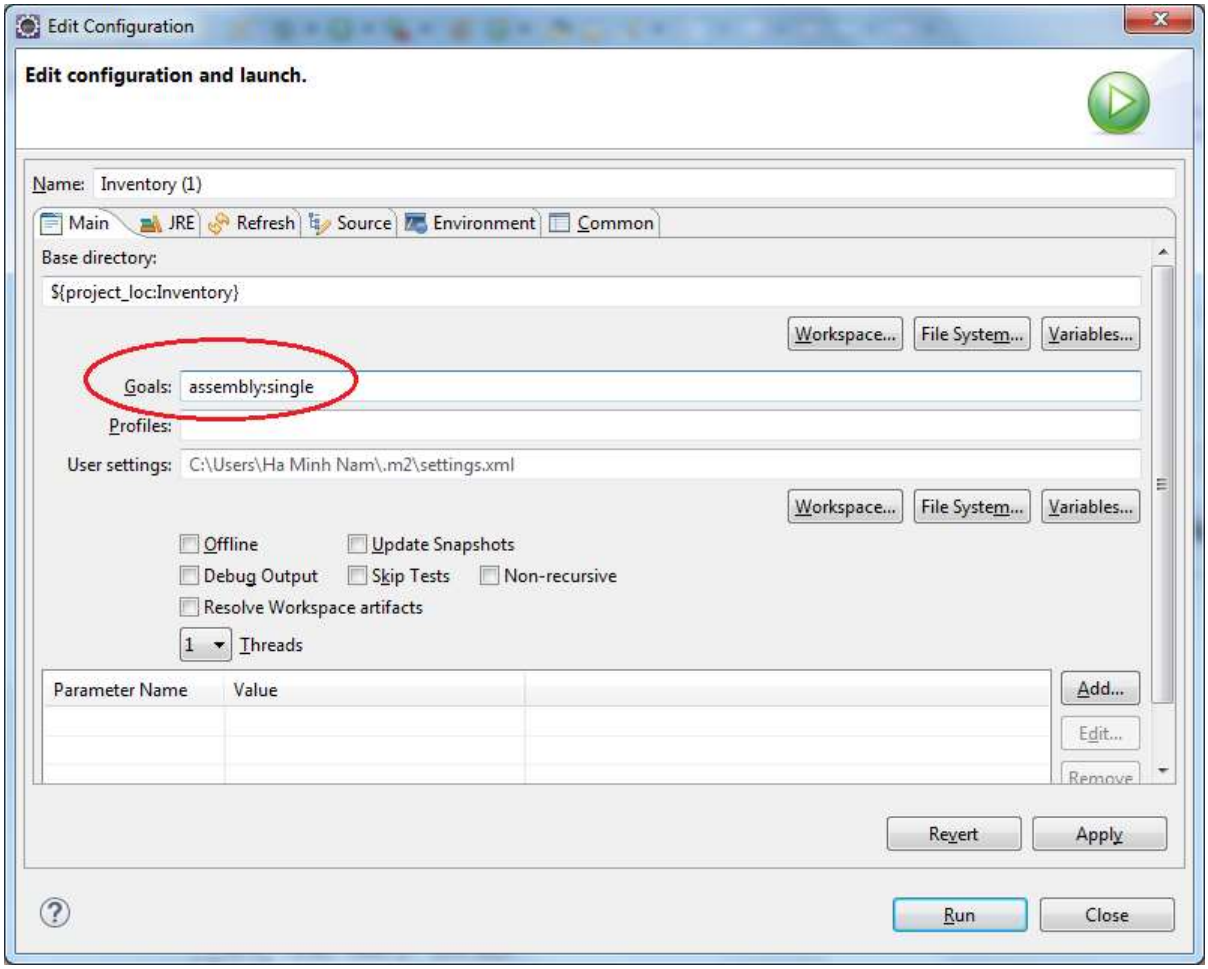
Here, there are two noteworthy points:

- The packaging type of the project must be jar: **<packaging>jar</packaging>**
- The **<mainClass>** element specifies the main class of your application in form of fully qualified name. For example, in the above XML we specify the main class is com.inventory.gui.InventoryApp

And to generate the JAR file from the project, run Maven with the goal **assembly:single**. For example, in the command line:

```
mvn clean install assembly:single
```

In Eclipse, right click on the project and select **Run As > Maven build**, and specify the goal **assembly:single** in the dialog, then click **Run**:



Wait a moment for the build to complete, and then check the JAR file generated under the project's **target** directory. Note that the JAR file contains all the resources and dependencies (fat JAR or uber-JAR).

References:

Maven Assembly Plugin