



OpenEye
Scientific Software

OpenEye Toolkit QuickStart – Java

Release 2012.Feb.1

OpenEye Scientific Software, Inc.

January 11, 2012

CONTENTS

1	Front Matter	1
2	Introduction	3
3	Prerequisites	5
3.1	Windows	5
3.2	Linux	5
3.3	OS X	5
3.4	Solaris - SPARC	5
4	Licensing	7
5	Download the OpenEye Java distribution	9
6	Installation	11
7	Using Eclipse	13
7.1	Creating a new project	13
7.2	Adding a class to our project	13
7.3	Running our example inside Eclipse	18
7.4	Creating an executable JAR	20
8	Creating an Example on the command line	23
9	Creating a custom oejava jar file	25
9.1	Examples of creating a special jar	25
	Index	27

FRONT MATTER

Copyright 1997-2012 OpenEye Scientific Software, Santa Fe, New Mexico. All rights reserved.

All rights reserved. This material contains proprietary information of OpenEye Scientific Software. Use of copyright notice is precautionary only and does not imply publication or disclosure.

The information supplied in this document is believed to be true but no liability is assumed for its use or the infringement of the rights of others resulting from its use. Information in this document is subject to change without notice and does not represent a commitment on the part of OpenEye Scientific Software.

This package is sold/licensed/distributed subject to the condition that it shall not, by way of trade or otherwise, be lent, re-sold, hired out or otherwise circulated without OpenEye Scientific Software's prior consent, in any form of packaging or cover other than that in which it was produced. No part of this manual or accompanying documentation, may be reproduced, stored in a retrieval system on optical or magnetic disk, tape, CD, DVD or other medium, or transmitted in any form or by any means, electronic, mechanical, photocopying recording or otherwise for any purpose other than for the purchaser's personal use without a legal agreement or other written permission granted by OpenEye.

This product should not be used in the planning, construction, maintenance, operation or use of any nuclear facility nor the flight, navigation or communication of aircraft or ground support equipment. OpenEye Scientific Software, shall not be liable, in whole or in part, for any claims arising from such use, including death, bankruptcy or outbreak of war.

Windows is a registered trademark of Microsoft Corporation. Apple, OS X, and Macintosh are registered trademarks of Apple Computer, Inc. AIX and IBM are registered trademarks of International Business Machines Corporation. UNIX is a registered trademark of the Open Group. RedHat is a registered trademark of RedHat, Inc. Linux is a registered trademark of Linus Torvalds. SPARC is a registered trademark of SPARC International Inc.

SYBYL is a registered trademark of TRIPOS, Inc. MDL is a registered trademark and ISIS is a trademark of Accelrys, Inc. SMILES, SMARTS, and SMIRKS may be trademarks of Daylight Chemical Information Systems. Macromodel is a trademark of Schrodinger, Inc.

Python is a trademark of the Python Software Foundation. Java is a trademark or registered trademark of Sun Microsystems, Inc. in the U.S. and other countries.

Other products and software packages referenced in this document are trademarks and registered trademarks of their respective vendors or manufacturers.

INTRODUCTION

The Java version of the OpenEye Toolkits is a JNI (Java Native Interface) wrapper created using SWIG. It is important to note that this is not a new version of the toolkits written in Java nor is it a Java interpretation of the toolkit API, but a rather faithful reproduction of the toolkit API in Java. As such there are some features that might not be as Java-like as a toolkit written entirely in Java, but most of these are minor issues and consistency with all the supported languages makes translation from one to another relatively easy and greatly enhances our ability to support all three versions.

A few of these idiomatic differences include:

- There are relatively few classes in OEChem and they have a rather shallow inheritance hierarchy.
- Most OEChem algorithms are in C++ free functions. These are mapped to static methods in the class *oechem*.
- Method names in most *OEChem* classes start with uppercase letters.
- Java doesn't support operator overloading, so C++ operators are mapped to methods. For example, operator `bool()` is mapped to a member function called `IsValid()`, operator `()` is mapped to `call()` and operator `const` is mapped to `constCall()`.
- Java doesn't support direct access to class attributes, so for some simple C++ classes and structs, direct access is mapped to a `get/set` pair of functions.

A Java version of the toolkits allows integration into Java desktop and server-side applications. However, the Java toolkits are not necessarily suited for applet programming since they require native code and platform-specific shared libraries. This can be achieved by building a multi-platform JAR as described in the final section, but this is considered for experts only.

Note that there are two sets of distributions for Linux. For the most part, if you are using a newer version of Linux (RHEL 5, RHEL 6, SLE 11 and Ubuntu 10.04 LTS), the regular distributions marked "Linux-x64" and "Linux-x86" are what you want. These are built with a newer version of GCC and provided a noticeable increase in performance over previous releases.

But, for RHEL 4 and SLED 10, these new releases are not compatible. For these two older Linux platforms, we've provided a pair of downloads labeled "Linux-x64-compatible" and "Linux-x86-compatible". Note that these "compatible" releases will work on all the supported Linux platforms, albeit with a noticeable decrease in performance.

PREREQUISITES

In order to use the OpenEye Java toolkits, a compatible version the Java SDK must be installed. For the most part, this means installing a version of the Oracle/Sun SDK for your platform. We also have support for the OpenJDK on some of our supported Linux platforms.

3.1 Windows

- Oracle/Sun JDK 1.6 - x86 and x64
- Oracle JDK 1.7 - x86 and x64

3.2 Linux

- Oracle/Sun JDK 1.6 - x86 and x64
- Oracle JDK 1.7 - x86 and x64
- OpenJDK 1.6 - x64, tested on these three platforms: - Ubuntu 10.04 LTS x64 - RHEL 5 x64 - RHEL 6 x64

3.3 OS X

- OS X 10.6 - Java 6 x86 and x64
- OS X 10.7 - Java 6 x64

3.4 Solaris - SPARC

- Sun JDK 1.6 - 32-bit and 64-bit

LICENSING

A license file from OpenEye Scientific Software is required to run any OpenEye application. A license file can be requested/obtained by contacting OpenEye at business@eyesopen.com.

At startup, the application looks for a valid license in the following default locations:

- In a file specified by the environment variable **OE_LICENSE**.
- In a file named `oe_license.txt` in the directory specified by the environment variable **OE_DIR**.
- In a file named `oe_license.txt` in the user's platform-specific local OpenEye application data directory. The location of this directory is detailed below:
 - Linux/UNIX:
 `~USERNAME/.OpenEye`
 - Mac OS X:
 `/Users/USERNAME/Library/OpenEye`
 - Microsoft Windows 2000/XP
 `C:\Documents and Settings\USERNAME\Application Data\OpenEye`
 - Microsoft Windows Vista/7
 `C:\Users\USERNAME\AppData\Local\OpenEye`
- In a file named `oe_license.txt` in the current working directory

DOWNLOAD THE OPENEYE JAVA DISTRIBUTION

Picking a distribution is relatively easy. For each platform (Windows, Linux, and OS X) we only have 2 downloads, a 32-bit version (x86) and a 64-bit version (x64). Just make sure to get the one for your specific architecture.

Download from <http://www.eyesopen.com/downloads>.

INSTALLATION

Untar the distribution into a directory of your choice. This will create a directory called “openeye”, with the following structure:

```
openeye/  
  oejava-*.jar  
  build.xml  
  examples/  
    openeye/  
      examples/  
      docexamples/
```

All the examples are found in the examples directory. A complete jar of all the examples can be made using Ant and the provided *build.xml* file.

USING ECLIPSE

Eclipse is a Java IDE available from eclipse.org.

7.1 Creating a new project

To create a new project, we choose from the File menu, File->New->Java Project. This will bring up a dialog like *New Eclipse Project 1*. Type in a name for this project, like “OESample1”, and then press the “Next” button.

Next, we need to add the OpenEye JAR to our project. Click the “Libraries” tab at the top of the dialog, then choose “Add External JARs...”. Browse to the location where you expanded the distribution you downloaded above and pick the appropriate “oejava-*-jar” for your platform. See *New Eclipse Project 2*.

You should then see something like: *New Eclipse Project 3*. Click the “Finish” button.

7.2 Adding a class to our project

Next we need to add a Java class to hold our sample code. Click File->New->Class to bring up the “New Class” dialog as in *Eclipse New Class*.

There are several things we want to do on this dialog.

- Choose a package name for this class. For example, “myoechemexamples”.
- Choose a name for the class: “OESample1”.
- Remove the default “Superclass” entry, since it is not required.
- Check the box to add a “main” to our class. And uncheck the box for “Inherits abstract methods”.
- Then click the “Finish” button.

This will take you back to Eclipse and open the new class in the editor, ready to add code.

Near the top of the file, add the import for oechem.

```
import openeye.oechem.*;
```

Then add our sample code inside the main method.

```
public static void main(String[] args) {  
    OEGraphMol mol = new OEGraphMol();  
    oechem.OEParseSmiles(mol, "ClCCCC1CCCB");  
}
```

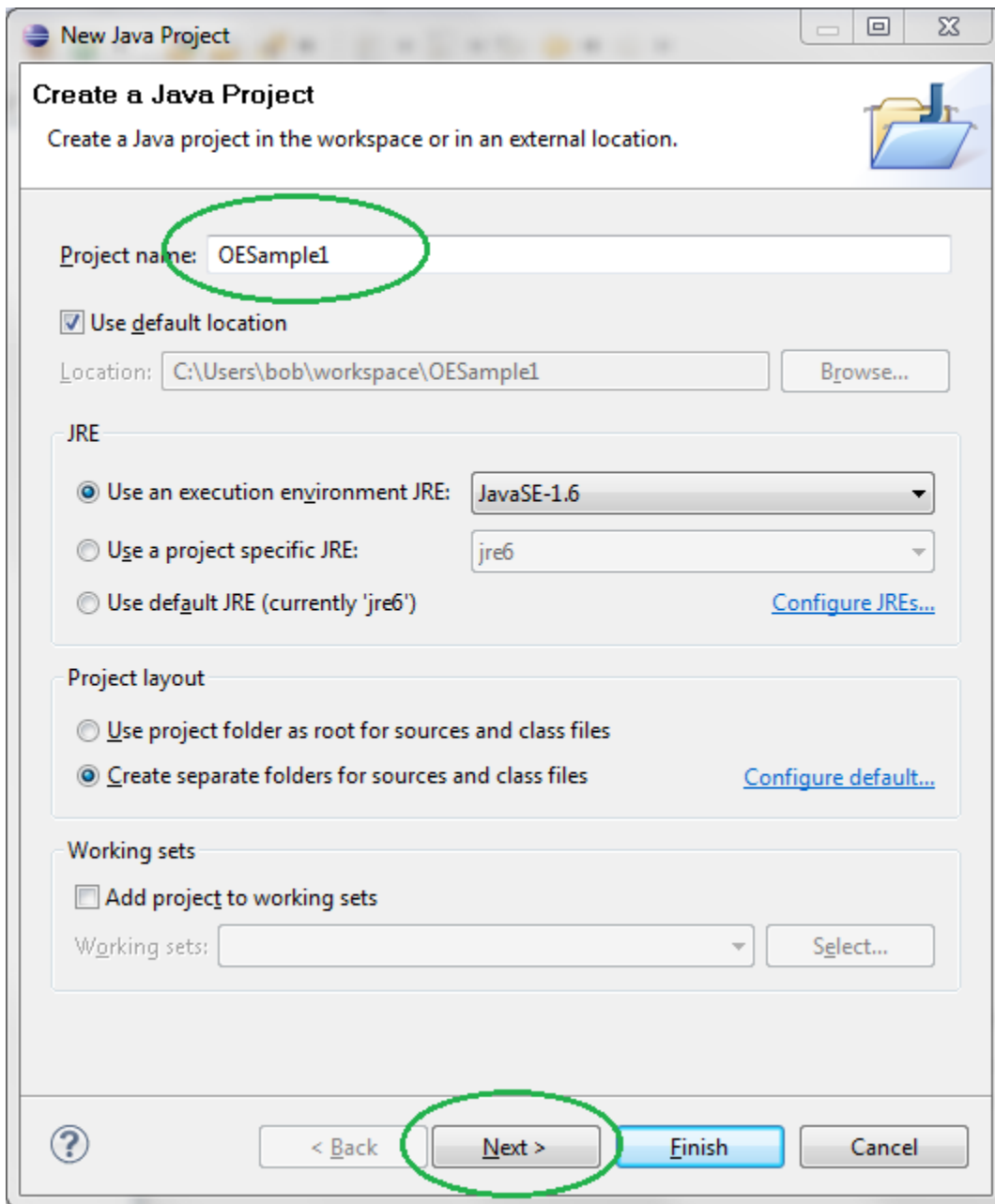


Figure 7.1: New Eclipse Project 1

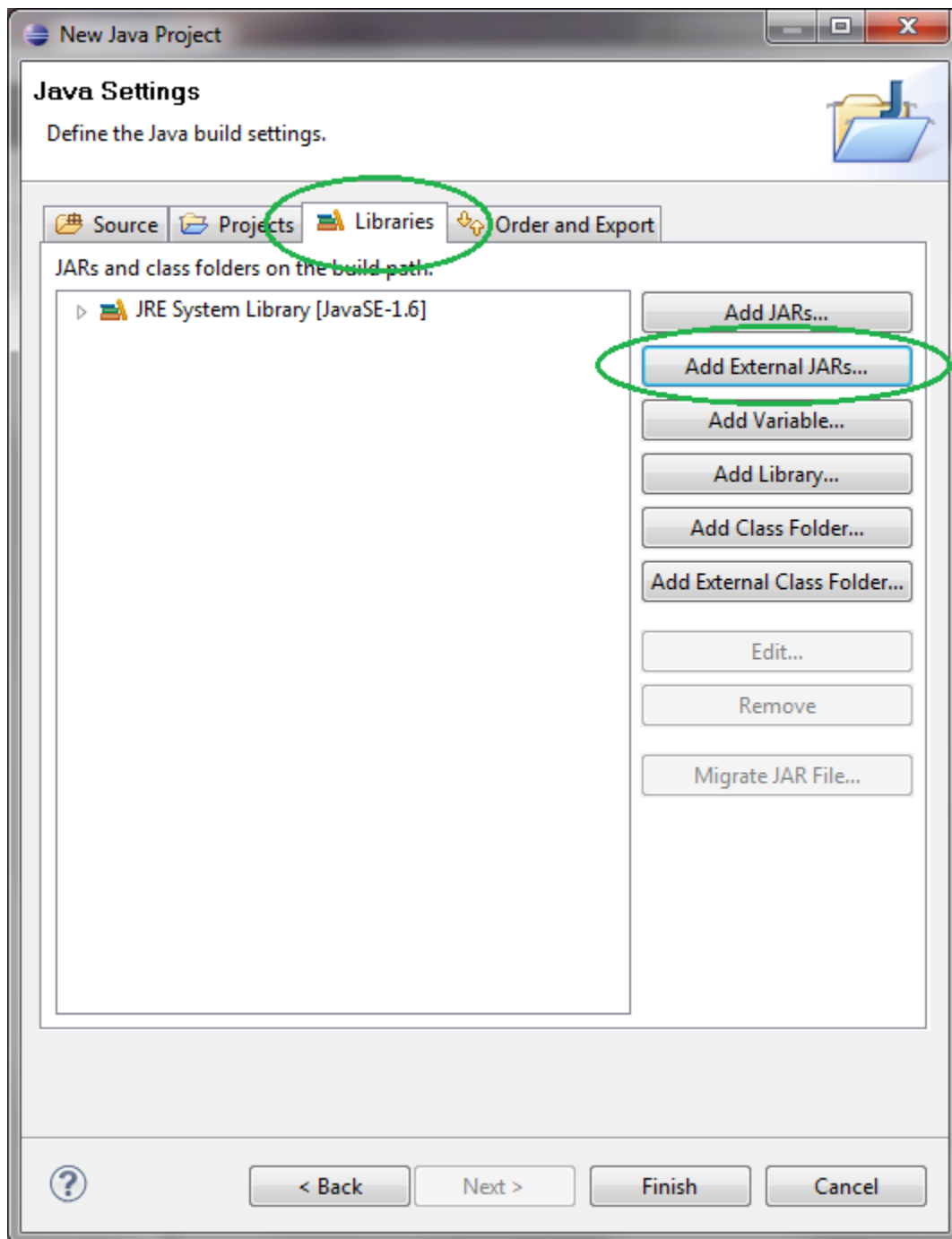


Figure 7.2: New Eclipse Project 2

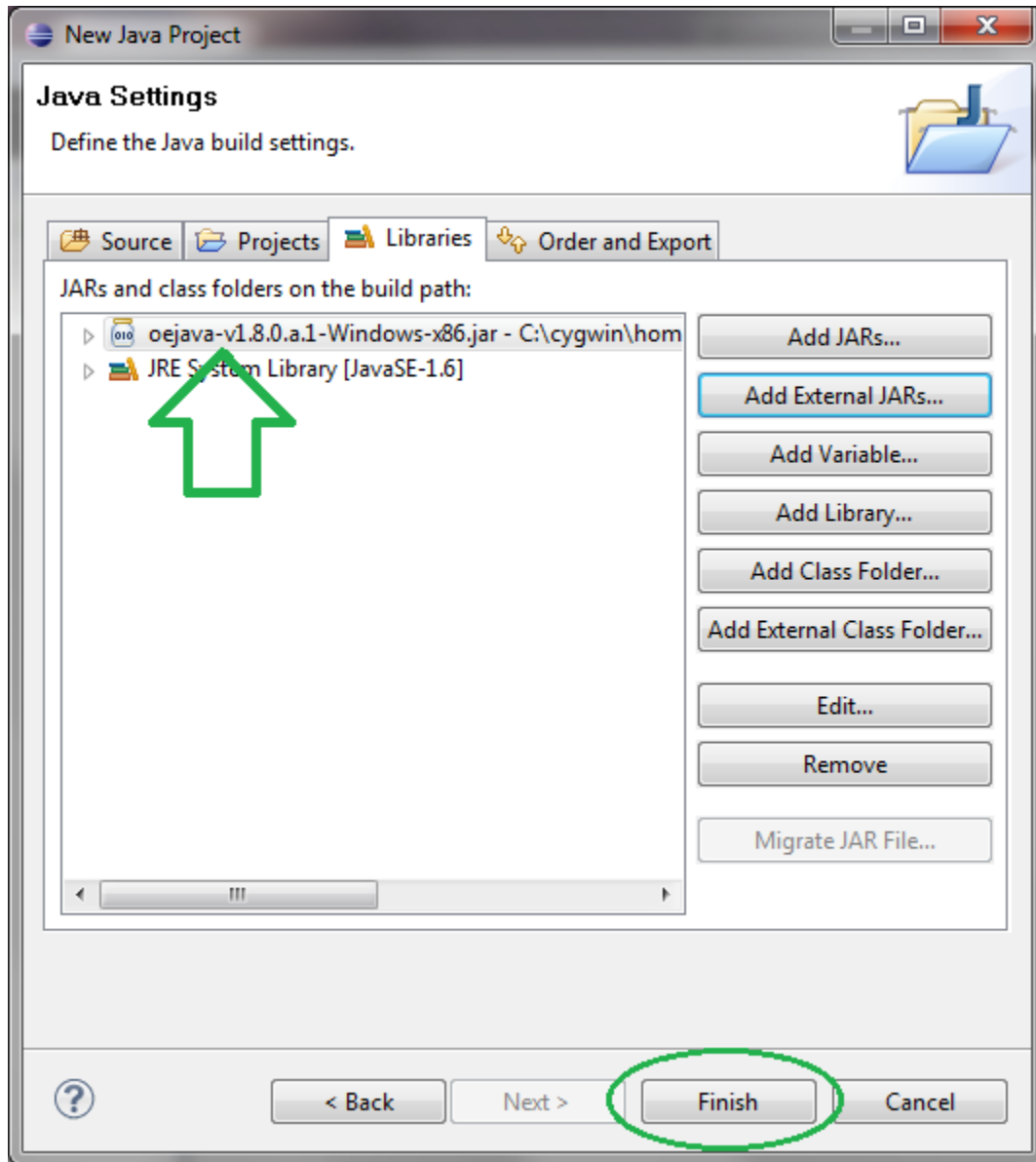


Figure 7.3: New Eclipse Project 3

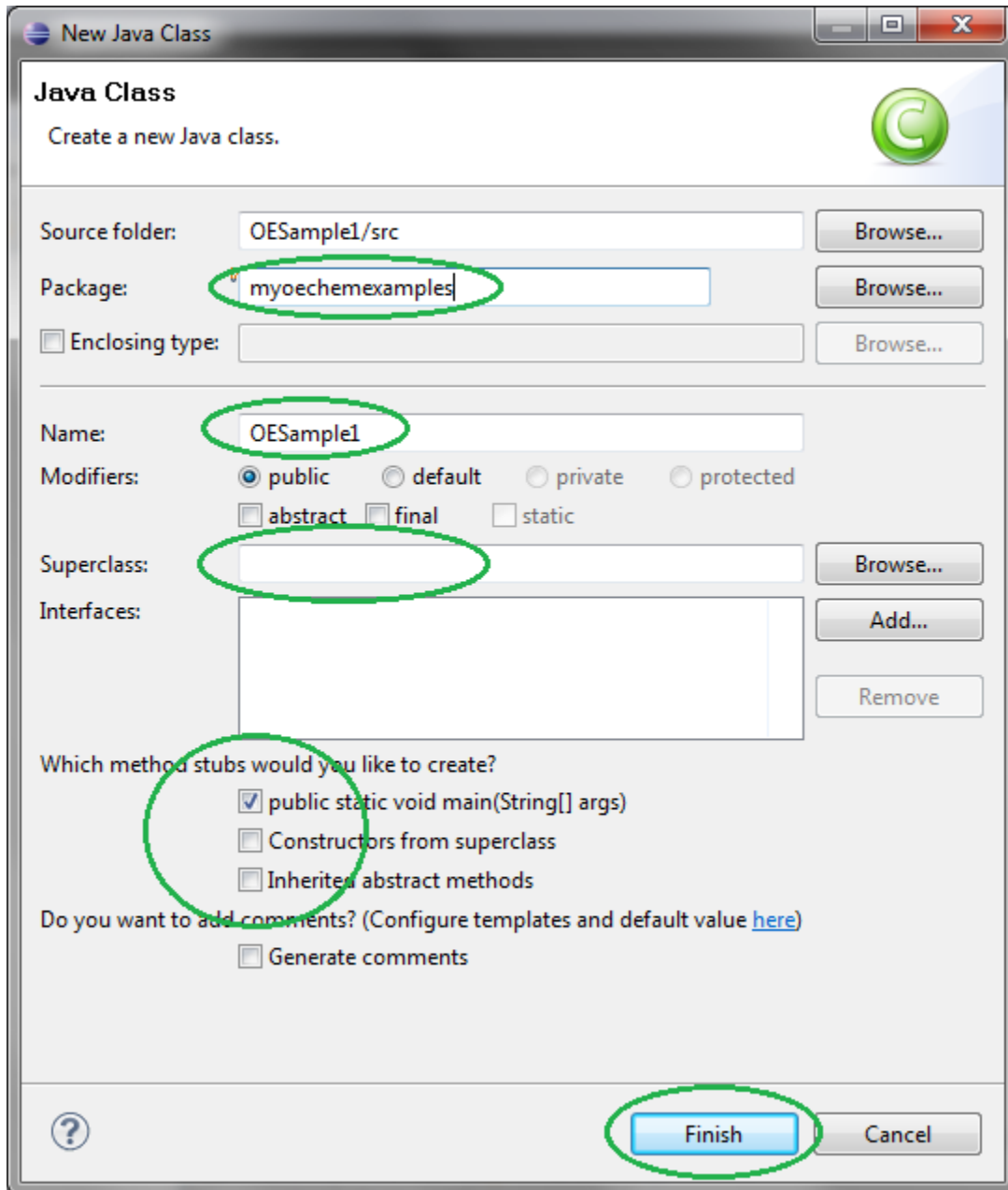


Figure 7.4: Eclipse New Class

```
System.out.format("mol has %d atoms\n", mol.NumAtoms());  
}
```

When you are finished, your sample should look something like *Editing in Eclipse*.

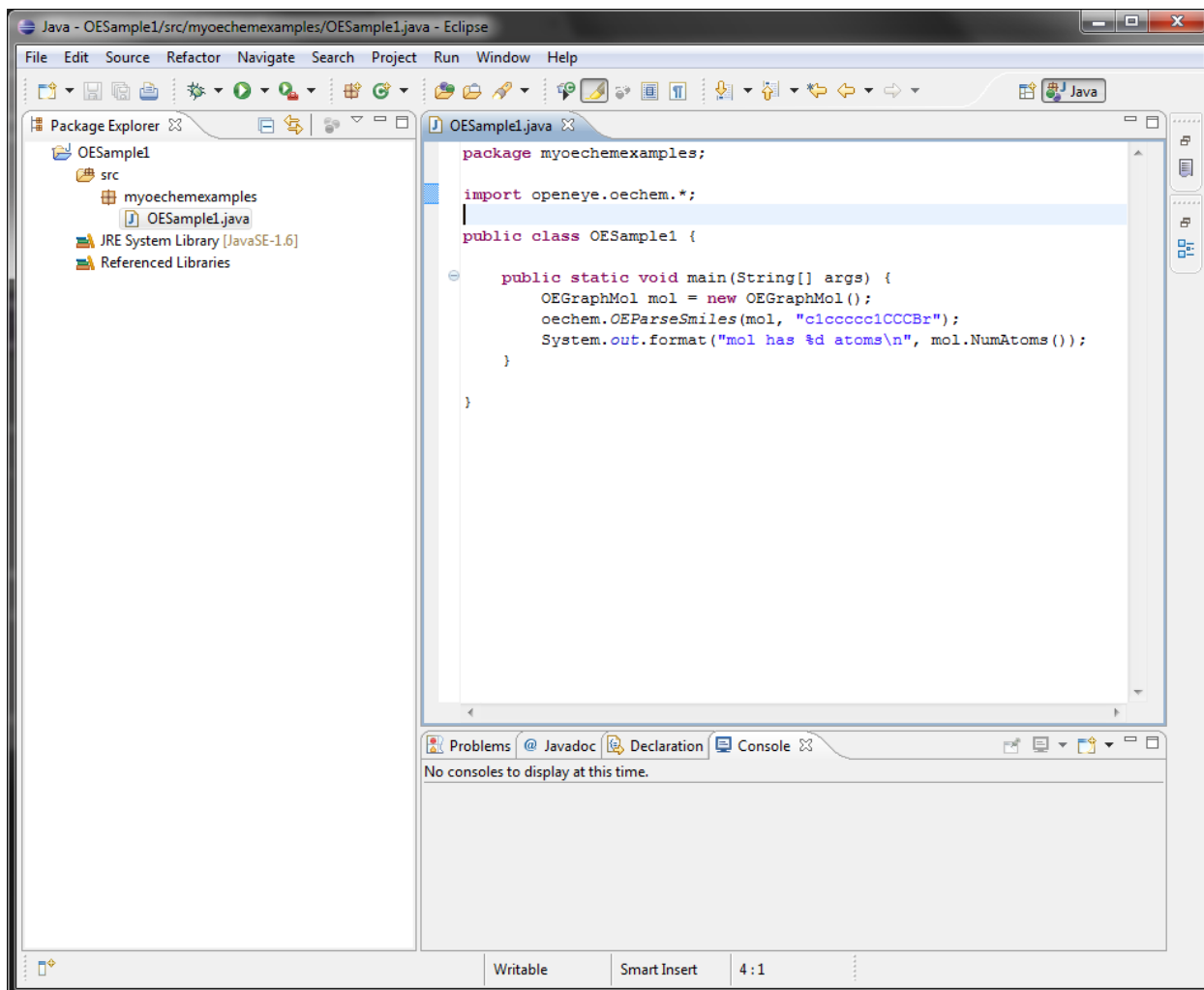


Figure 7.5: Editing in Eclipse

7.3 Running our example inside Eclipse

We can now test run our program in Eclipse. In the toolbar, click the “Run” button as shown in *Running in Eclipse*.

This should produce some output in the “Console” window like *Console output*.

And now you’ve created your first OEChem Java sample. You can now add code and continue to run inside Eclipse to try out more of the OEChem API.

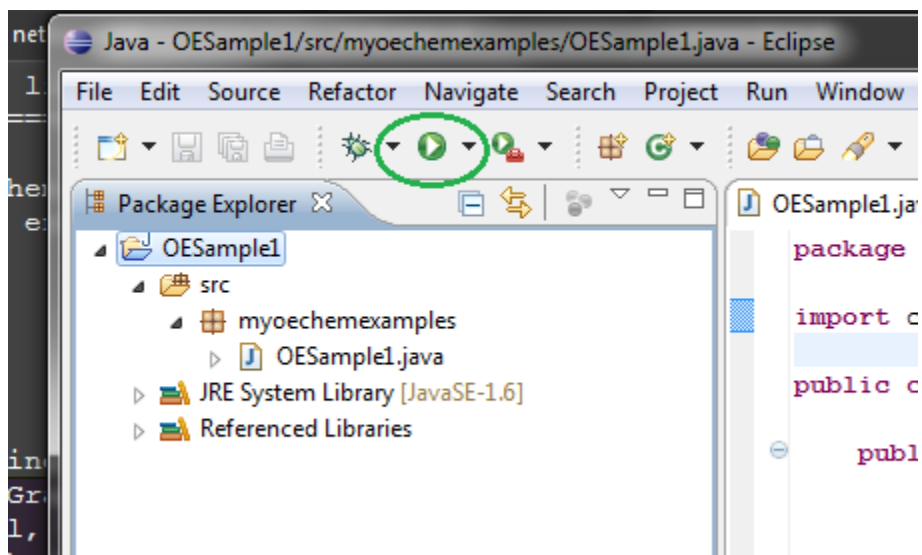


Figure 7.6: Running our example in Eclipse

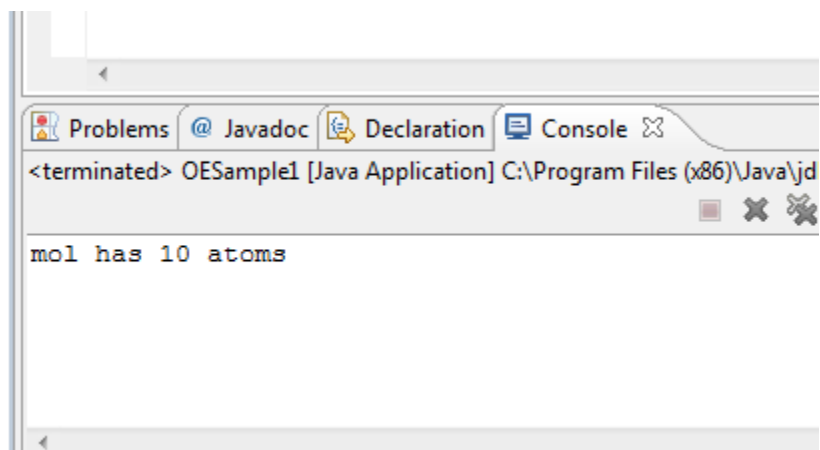


Figure 7.7: Console output from our example.

7.4 Creating an executable JAR

At some point, you may want to run your example outside of Eclipse, or you may want to deploy the application to another machine. One easy tool for this is to use the FAT JAR Eclipse plugin downloadable from <http://fjep.sourceforge.net/>.

Once installed, right click on the OESample1 project icon, in the Eclipse “Package Explorer” and choose “Build Fat Jar”

Then in the next dialog:

- Pick a name for you new JAR
- Make sure the name of the class we just created is set as the “Main-Class”.
- Click “Finish”.

You should now be able to open a terminal in the location where you saved the JAR file and run it from the command-line.

```
$> java -jar OESample1_fat.jar  
mol has 10 atoms
```

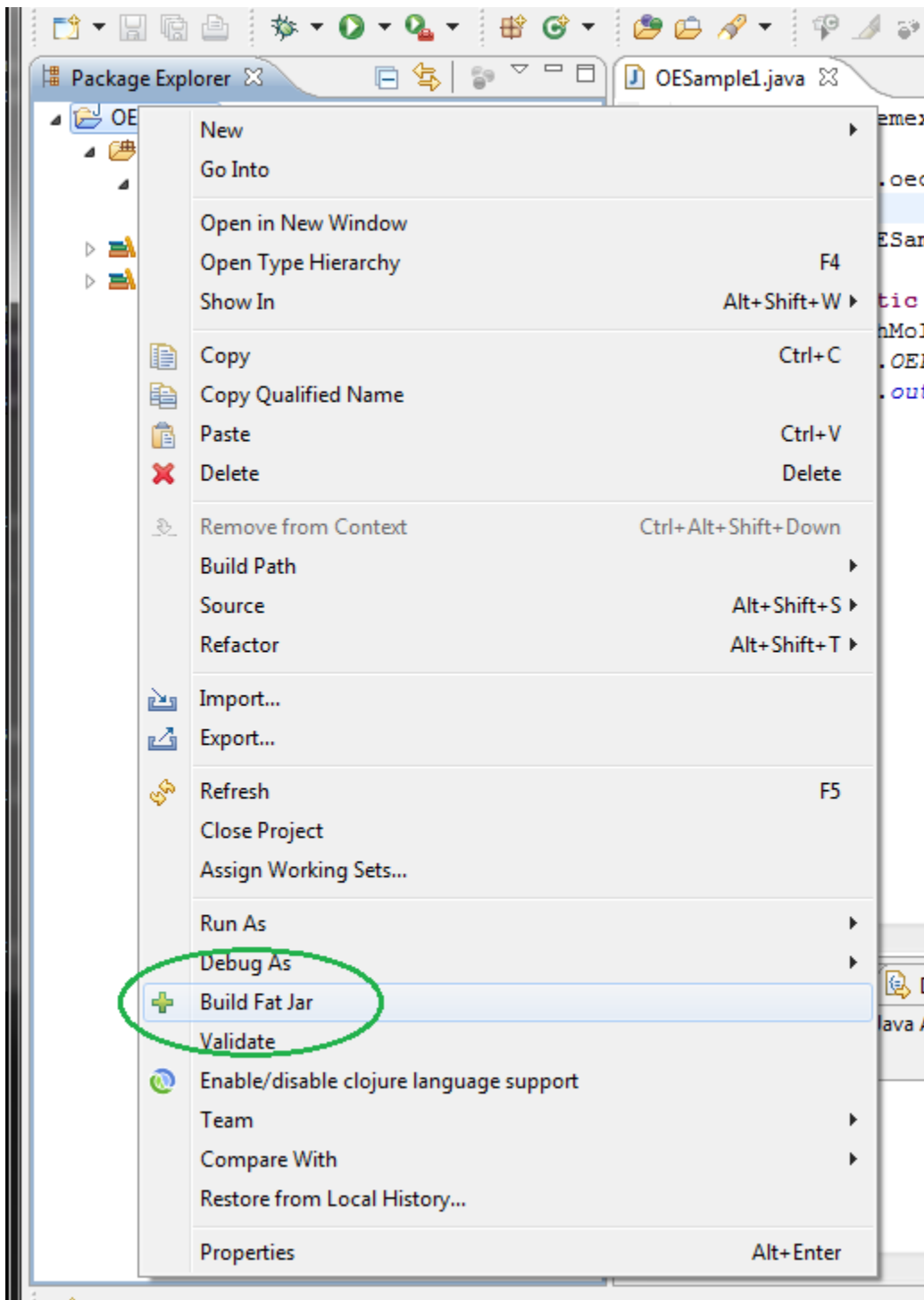


Figure 7.8: Choose Build Fat Jar.

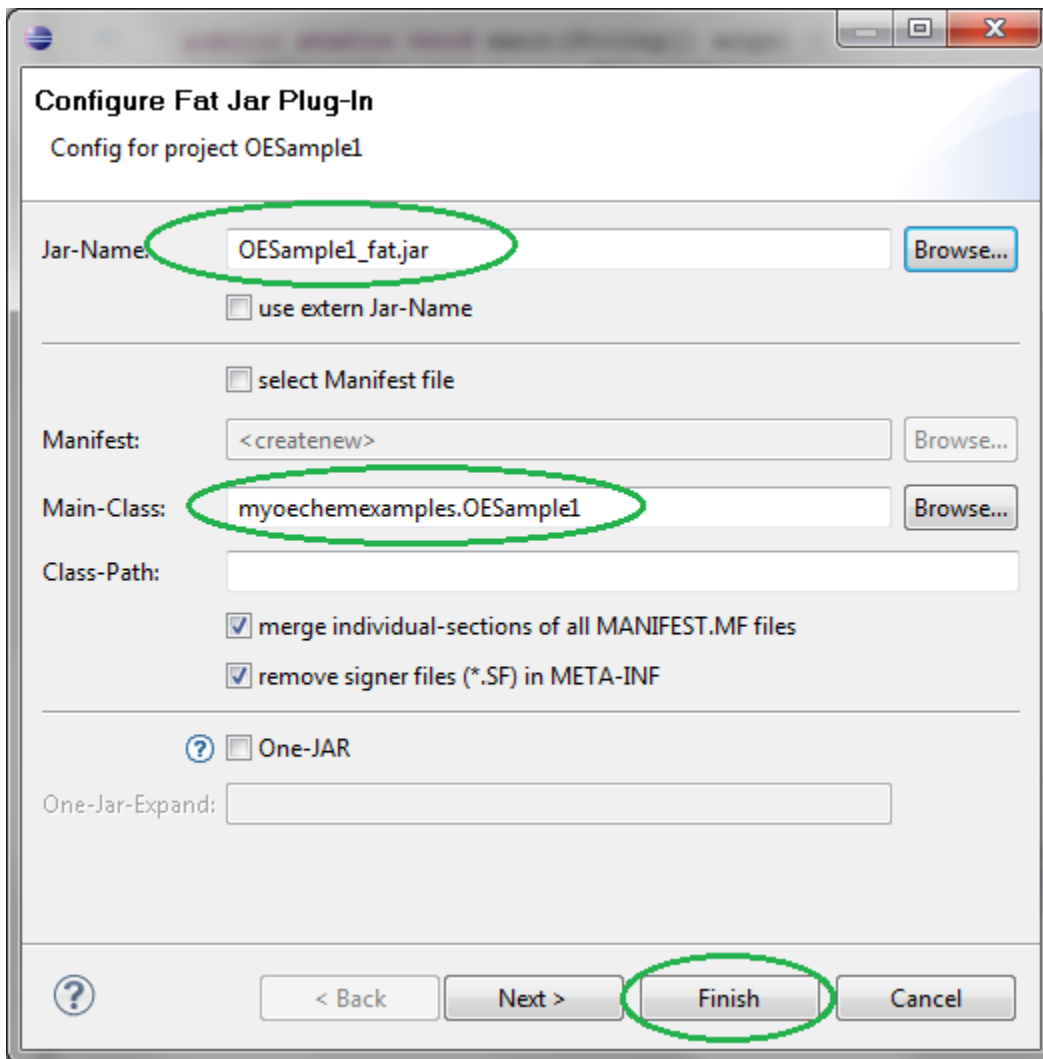


Figure 7.9: **Build Fat Jar.**

CREATING AN EXAMPLE ON THE COMMAND LINE

Let's start with a very simple OEChem program using command line tools. Open a new file in a text editor called *OESample1.java* and enter the following code.

```
import openeye.oechem.*;

class OESample1 {
    public static void main(String [] args) {
        OEGraphMol mol = new OEGraphMol();
        oechem.OEParseSmiles(mol, "ClCCCCClCCBr");
        System.out.format("mol has %d atoms\n", mol.NumAtoms());
    }
}
```

Save the file and let's compile it from the command line with *javac*. Note that you can pass the CLASSPATH as an argument to *javac* or you could declare the CLASSPATH environment variable. In either case, the only thing needed to add is the single jar file.

```
$> javac -cp oejava-1.8.0-Windows-x86.jar OESample1.java
```

This should produce *OESample1.class* if you don't have any typos. You can then run this sample on the command line using something like:

```
$> java -cp oejava-1.8.0-Windows-x86.jar:. OESample1
mol has 10 atoms
```


CREATING A CUSTOM OEJAVA JAR FILE

In the previous examples, we used the entire `oejava-*-jar` JAR file for the projects. In most cases, this is the easiest path since it contains all the toolkits for a particular platform.

But there are times where it may be desired to create a smaller, custom JAR containing only the specific toolkits for that project. Additionally, for a project that needs to run on several different platforms, we can also merge in two or more platforms, making a single JAR.

In the same directory that contains the `oejava-*-jar` file(s), you'll find another jar that contains a small program for generating these custom JARs. The next section shows some example uses of this program.

9.1 Examples of creating a special jar

If you run the program with no command line arguments, you'll see the basic syntax of the command as well as a list of available toolkits to include.

```
$ java -jar openeye-javautils.jar
```

```
usage: Main -products <prod list> -jars <jar list> -out <new jar name>
```

```
Available Products:
```

```
OEGrapheme  
OEDocking  
OEMolProp  
OEIUPAC  
OEZap  
OEShape  
OEQuacPac  
OEChem  
OESzybki  
OEBio  
OEGraphSim  
OEOmega  
OESpicoli  
OEDepict
```

The simplest example is to create a smaller JAR, containing a subset of the toolkits. Assuming we are working on 32-bit Windows, and want to create a new JAR with only `OEChem` and `OEGraphSim`, we would do something like:

```
$ java -jar openeye-javautils.jar -products OEChem OEGraphSim \  
-jars oejava-v1.8.0-Windows-x86.jar -out oechem-special-win32.jar
```

You can then use this new JAR for your new project. You'll note that it is significantly smaller, but still self-contained to support a project using OEChem and OEGraphSim only.

A potentially more useful feature is the ability to merge multiple platforms into one new JAR, allowing cross-platform project and cross-platform FAT JARs.

A standard scenario might be where you develop on Windows (using 32-bit Java) but want to deploy on 64-bit Linux. First gather the original JARs for the two platforms of interest.

```
$ ls oejava*  
oejava-v1.8.0-Linux-x64.jar  oejava-v1.8.0-Windows-x86.jar
```

Then run the program to create a new JAR with OEChem, OEDepict and OEGrapheme.

```
$ java -jar openeye-javautils.jar -products OEChem OEDepict OEGrapheme \  
-jars oejava-v1.8.0-Windows-x86.jar oejava-v1.8.0-Linux-x64.jar \  
-out oechem-oedepict-oegrapheme-Win32-Linux64.jar
```

Using this new JAR in your project in Eclipse will allow you to create a FAT JAR that will run on both Win32 and Linux64.

INDEX

E

environment variable
 OE_DIR, 7
 OE_LICENSE, 7

O

OE_DIR, 7
OE_LICENSE, 7