

Getting Started with Contract4J

by
Ivan A Krizsan

Version: May 29, 2008

Copyright 2008 Ivan A Krizsan. All Rights Reserved.

Table of Contents

Table of Contents.....	2
Purpose	3
Licensing	3
Disclaimers	3
Prerequisites.....	3
Downloading Contract4J.....	4
Building the Contract4J Library.....	6
Creating the Project.....	6
Adding Source Code and Libraries.....	8
Configuring the Build Path.....	9
Creating the Library JAR File.....	12
Using Contract4J.....	14
Creating the Project.....	14
Adding Libraries.....	14
Configuring the Build Path.....	14
Writing an Example Program.....	18
The MyBean Class.....	18
The Main Class.....	19
Running the Example Program.....	19

Purpose

This document aims to describe how to getting started with [Contract4J](#), a tool that supports design by contract programming in Java 5 and later.

Licensing

This document is licensed under a [Creative Commons Attribution-Noncommercial-Share Alike 3.0 License](#).

Disclaimers

Though I have done my best to avoid it, this document might contain errors. I cannot be held responsible for any effects caused, directly or indirectly, by the information in this document – you are using it on your own risk.

Submitting any suggestions, or similar, the information submitted becomes my property and you give me the right to use the information in whatever way I find suitable, without compensating you in any way.

All trademarks in this document are properties of their respective owner and do not imply endorsement of any kind.

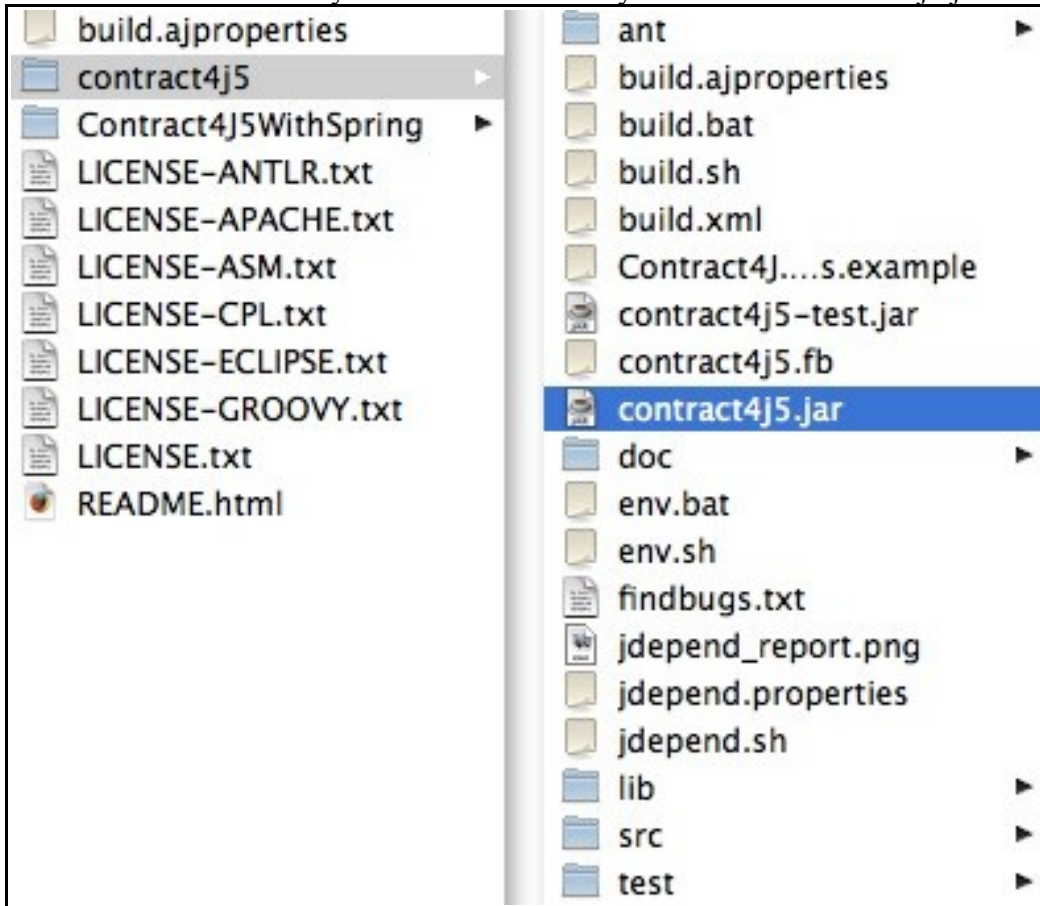
This document has been written in my spare time and has no connection whatsoever with my employer.

Prerequisites

All development in this document was done using [Eclipse](#) 3.3.2 with [AspectJ development tools](#) 1.5.2, which uses AspectJ 1.6.0.

Downloading Contract4J

Go to the [Contract4J web-page](#) and download the latest release. In my case it is Contract4J5 0.80. Unpack the archive and throw away the Contract4J library file named “contract4j5.jar”.

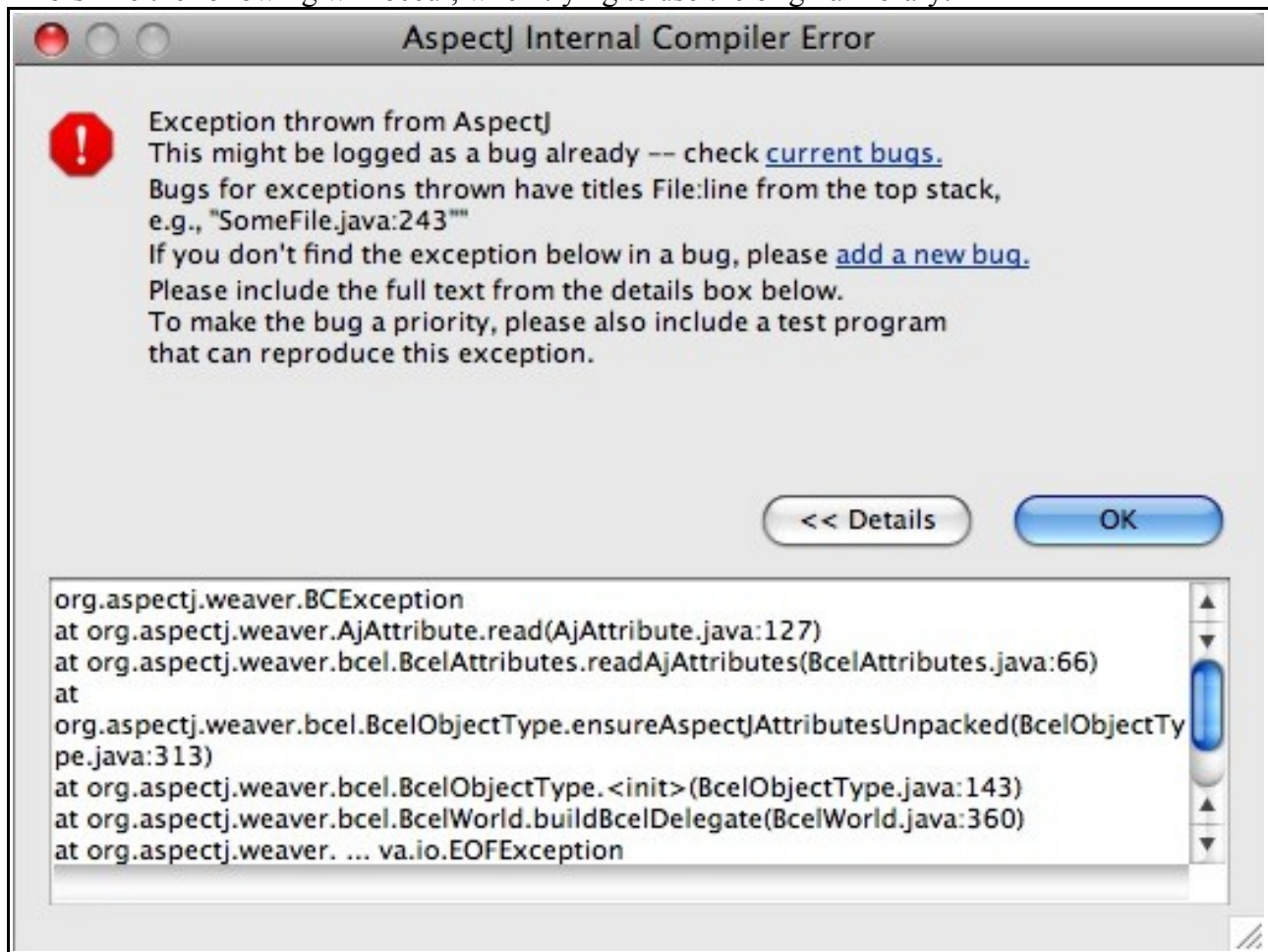


The location of the Contract4J library file in the unpacked Contract4J distribution.

Why throw away the Contract4J library?

The version of Contract4J I downloaded, Contract4J5 0.80, was apparently compiled with AspectJ 1.5, which regrettably is incompatible with AspectJ 1.6, which I am using, at the time of writing this document.

Errors like the following will occur, when trying to use the original library:



Error occurring when trying to use the original Contract4J5 0.80 library with AspectJ 1.6.

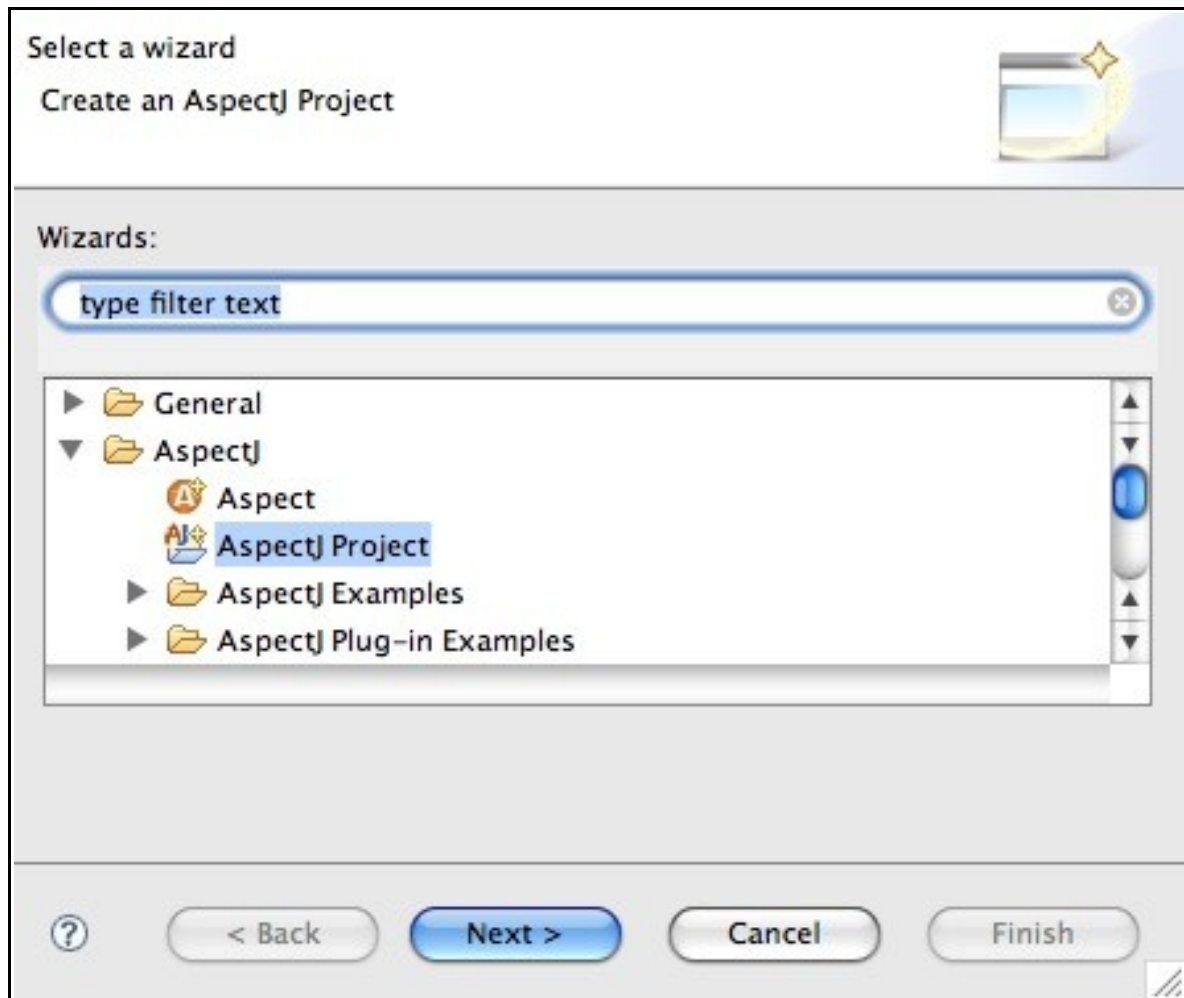
So, in order to use Contract4J, we have to build a new version of the library.

Building the Contract4J Library

Before we can actually use Contract4J, we need to build the Contract4J library. This will be done using Eclipse, to make the process easier.

Creating the Project

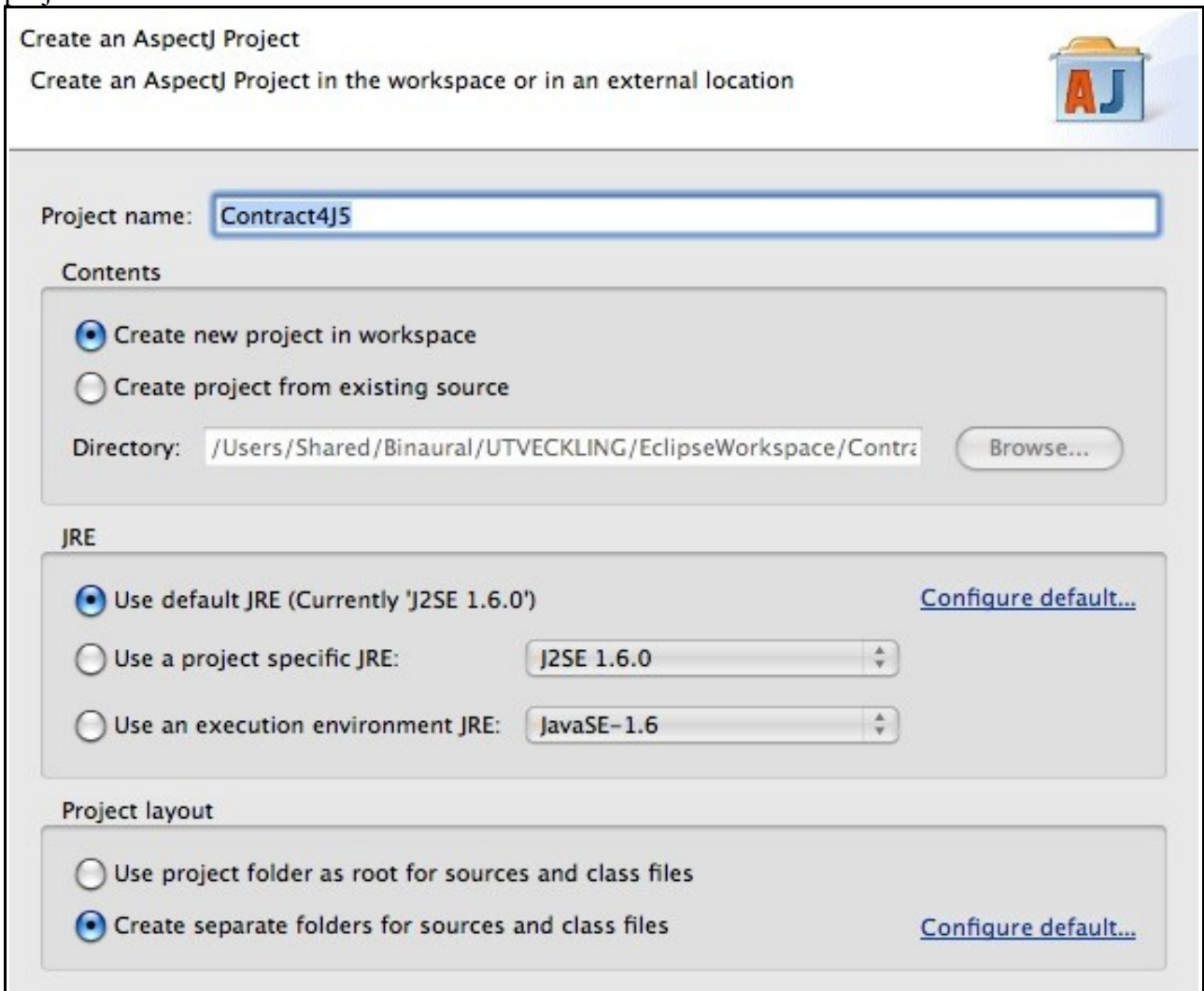
First we must create a project in Eclipse. Since Contract4J contains a number of aspects, we create an AspectJ project. Go to the File menu, select the New menu item and then select Project in the sub-menu that appears.



First step in creating the project for building the Contract4J library – selecting the project type.

Select the AspectJ Project type, as shown in the picture above and then click the Next button.

At this stage, we can name the project and configure the Java version it uses. I choose to name the project “Contract4J5” and use the default Java version.



Second step of creating the Eclipse project for building the Contract4J library.

Click the Finish button.

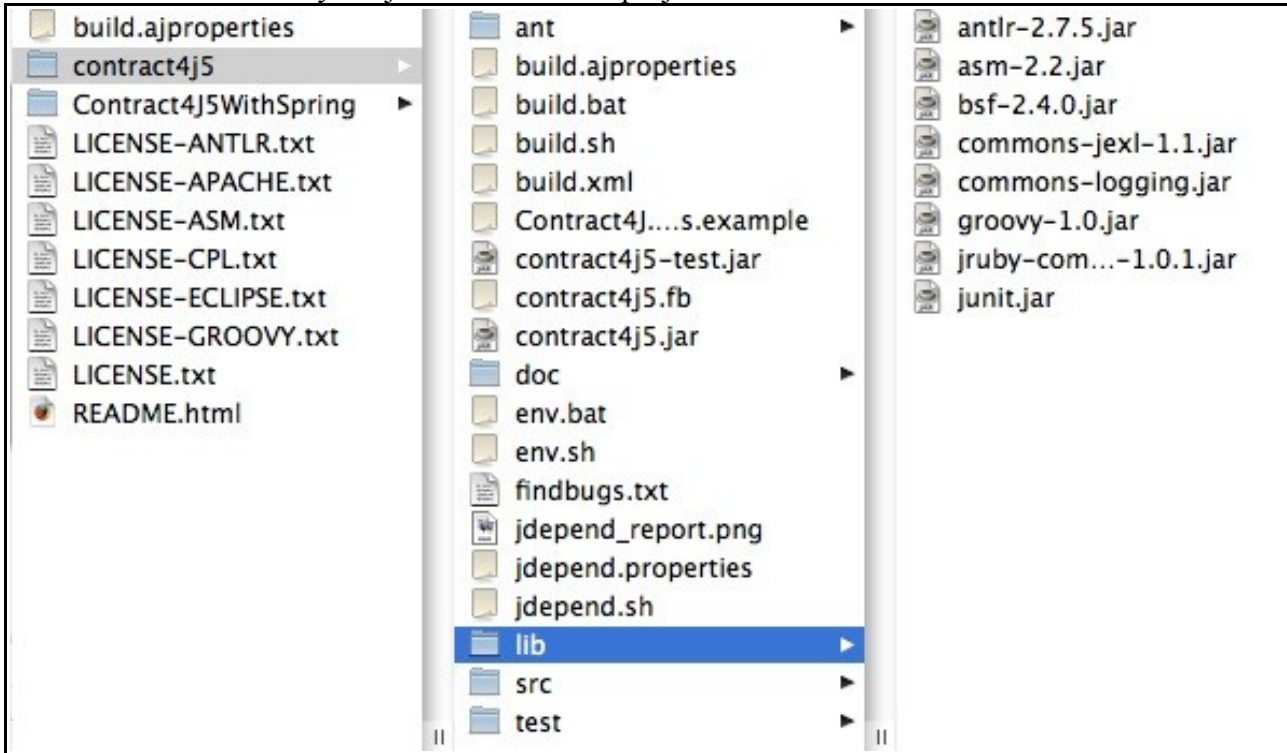
The project creation is now finished, the next step is to add source code and libraries to the project.

Adding Source Code and Libraries

We must now leave Eclipse for a while, in order to copy the source code and the libraries from the Contract4J distribution directory to the Eclipse project just created.

Start by locating the project director, usually located in the Eclipse workspace. In this directory, create a new directory named “lib”.

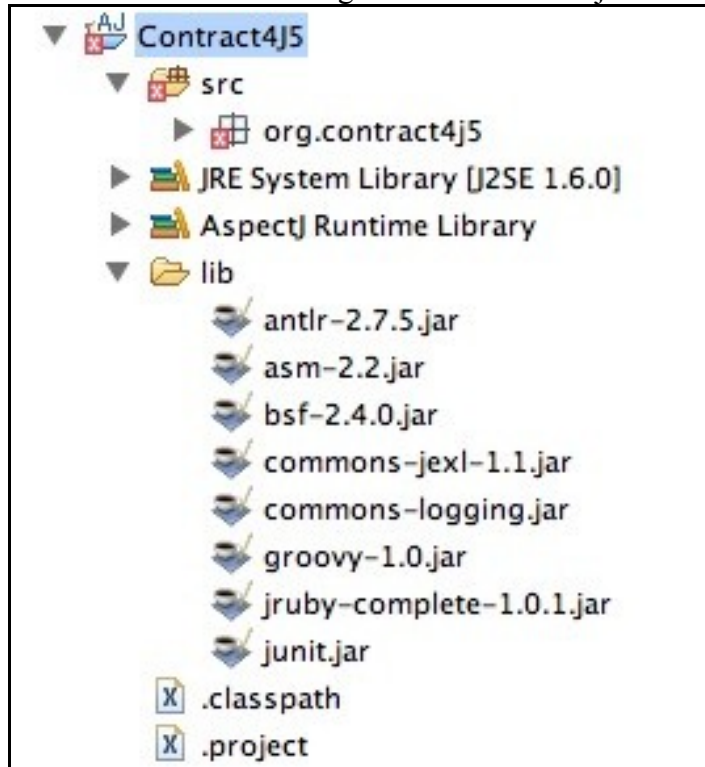
From the directory in the Contract4J distribution shown in the picture below, copy all of the JAR files to the “lib” directory we just created in the project.



Location of the libraries required to build the Contract4J library in the Contract4J distribution directory.

Next copy the contents of the “src” folder, located immediately below the “lib” folder in the above picture, to the “src” folder in the Eclipse project.

If we now go back to Eclipse, select the project and press the F5 key, in order to refresh the project contents, the project should now look something like this in the Project browser:

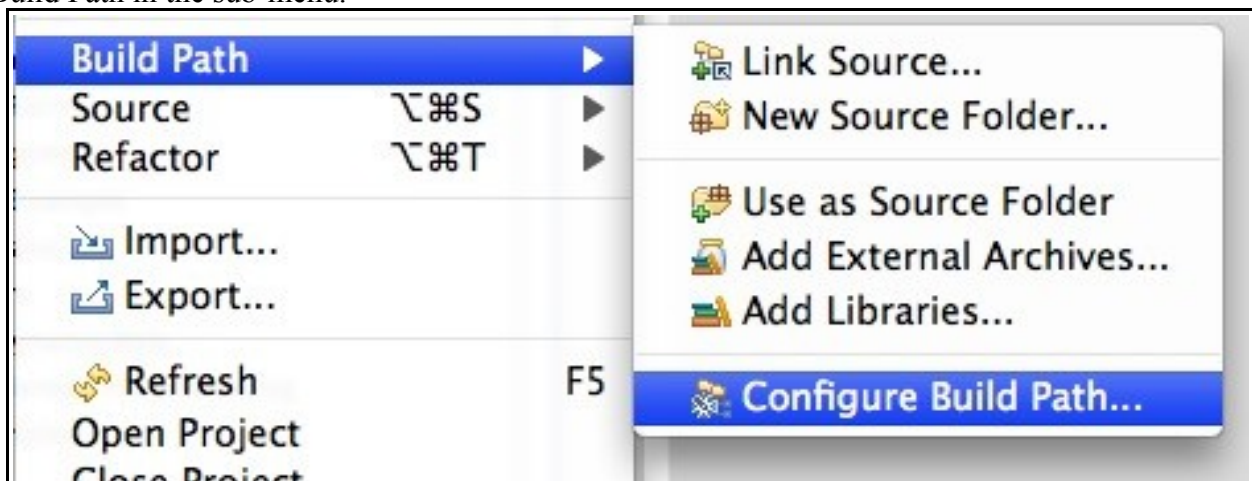


The Contract4J library project in Eclipse after having added the libraries and the source code.

As can be seen, there are compilation errors – this is because the libraries haven't been added to the Java build path yet.

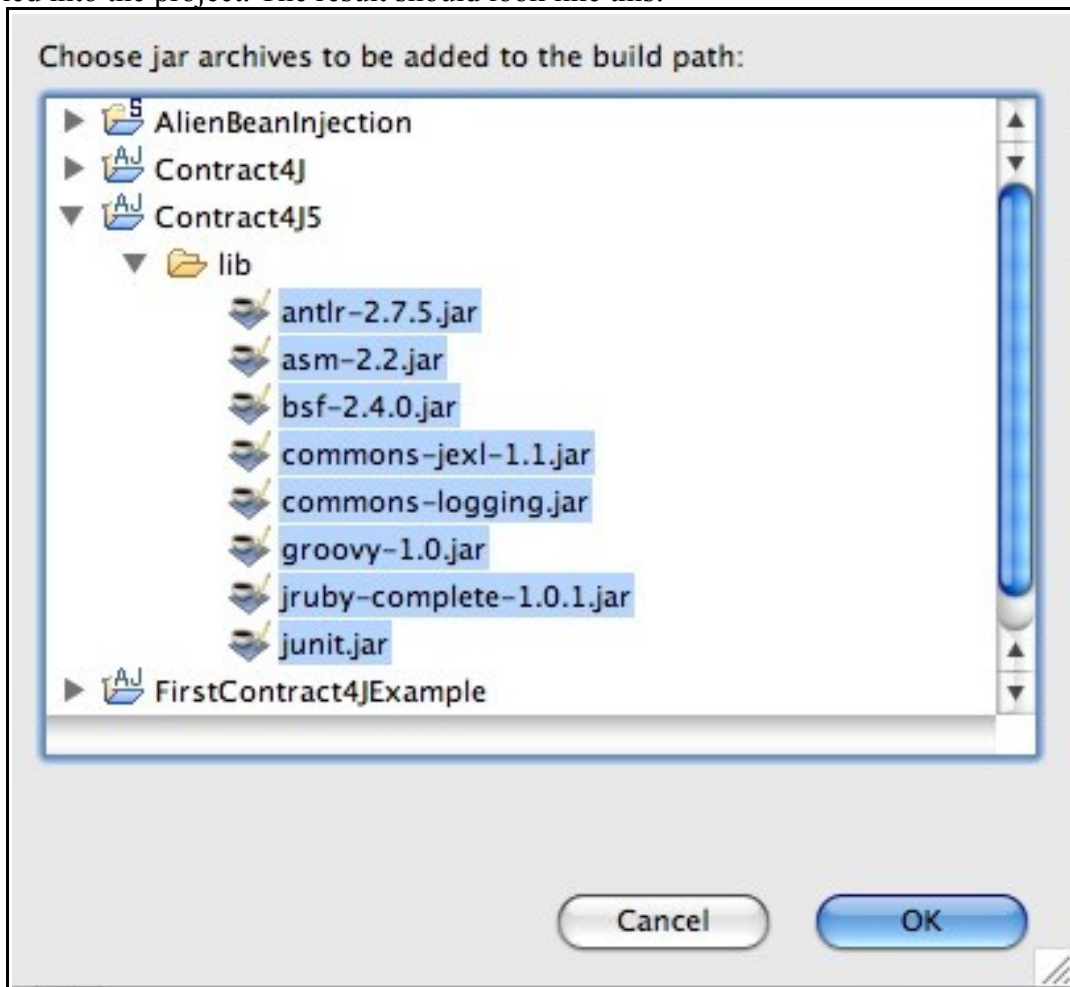
Configuring the Build Path

Right-click on the project name and select Build Path in the menu that appears and then Configure Build Path in the sub-menu.



Bringing up the Build Path configuration dialog for the Contract4J library project in Eclipse.

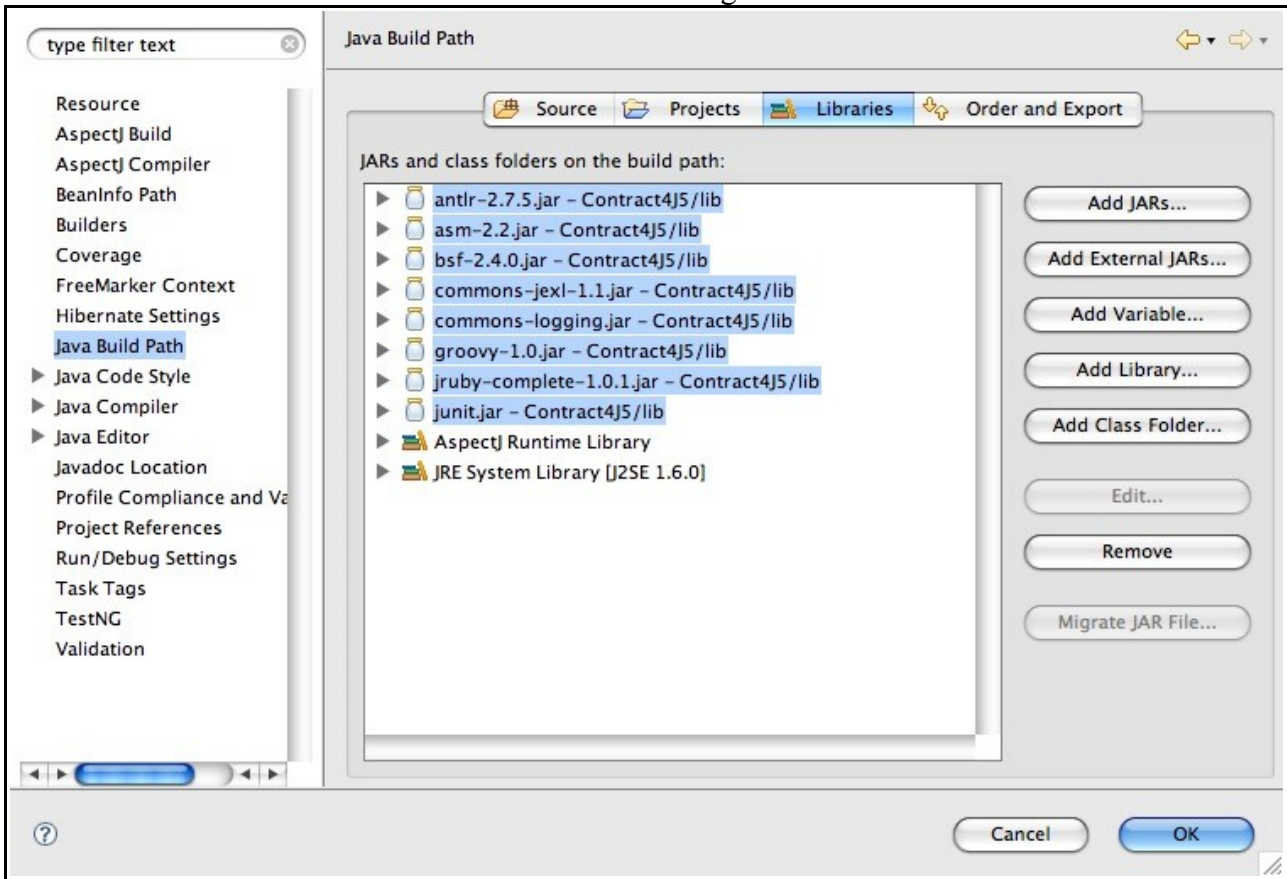
In the window that appears, click the Add JARs button on the right. Another window appears, in which we can select the JAR files to add to the build path. Open the Contract4J library project and the “lib” folder within the project. Then select all the library files you just copied into the project. The result should look like this:



Selecting the JAR files to add to the Contract4J library project's build path.

When finished, click the OK button.

The Java Build Path window should now look something like this:



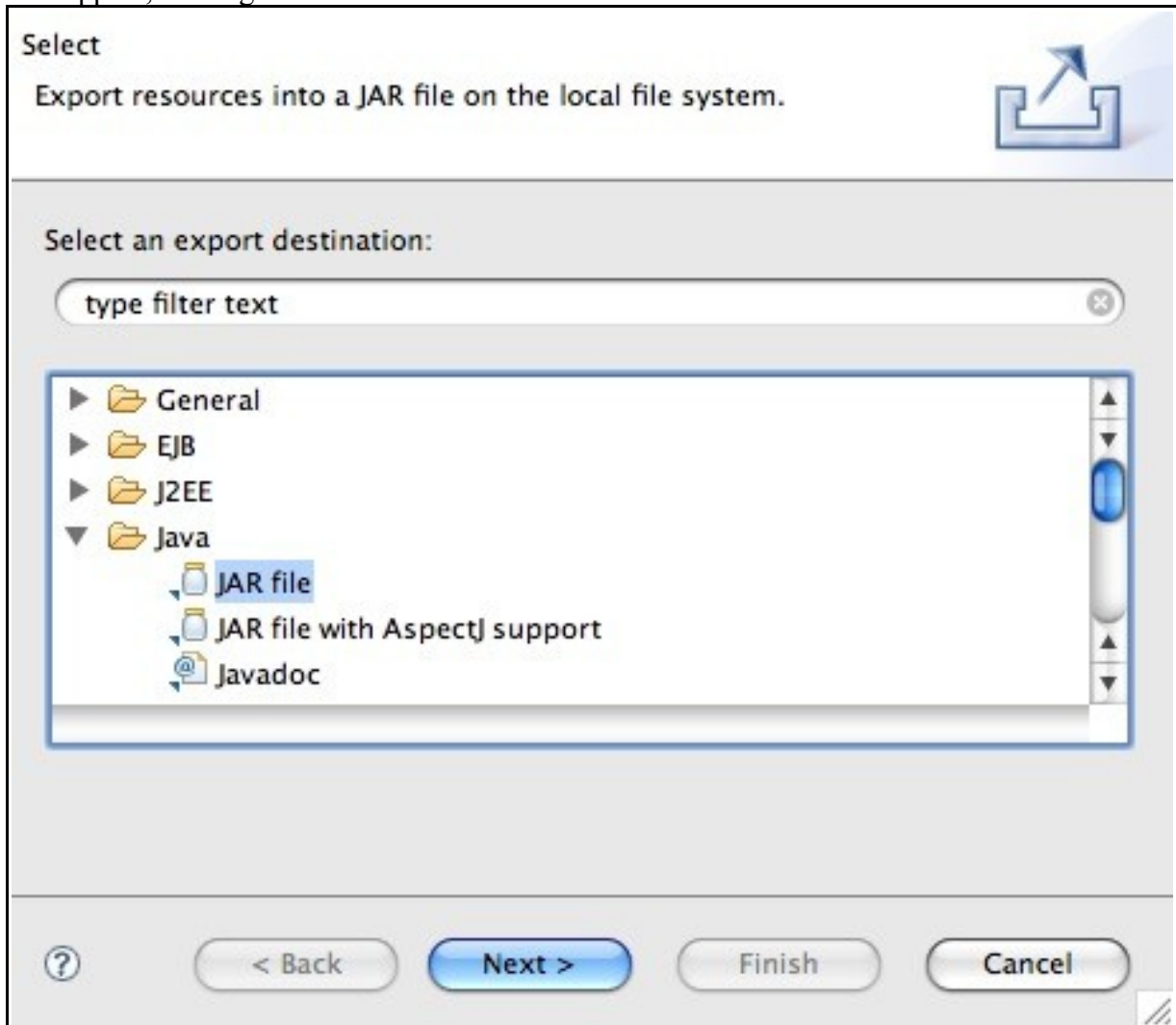
Java Build Path window after having added the required libraries to the Contract4J library project.

Click the OK button.

The project should now compile without any errors.

Creating the Library File

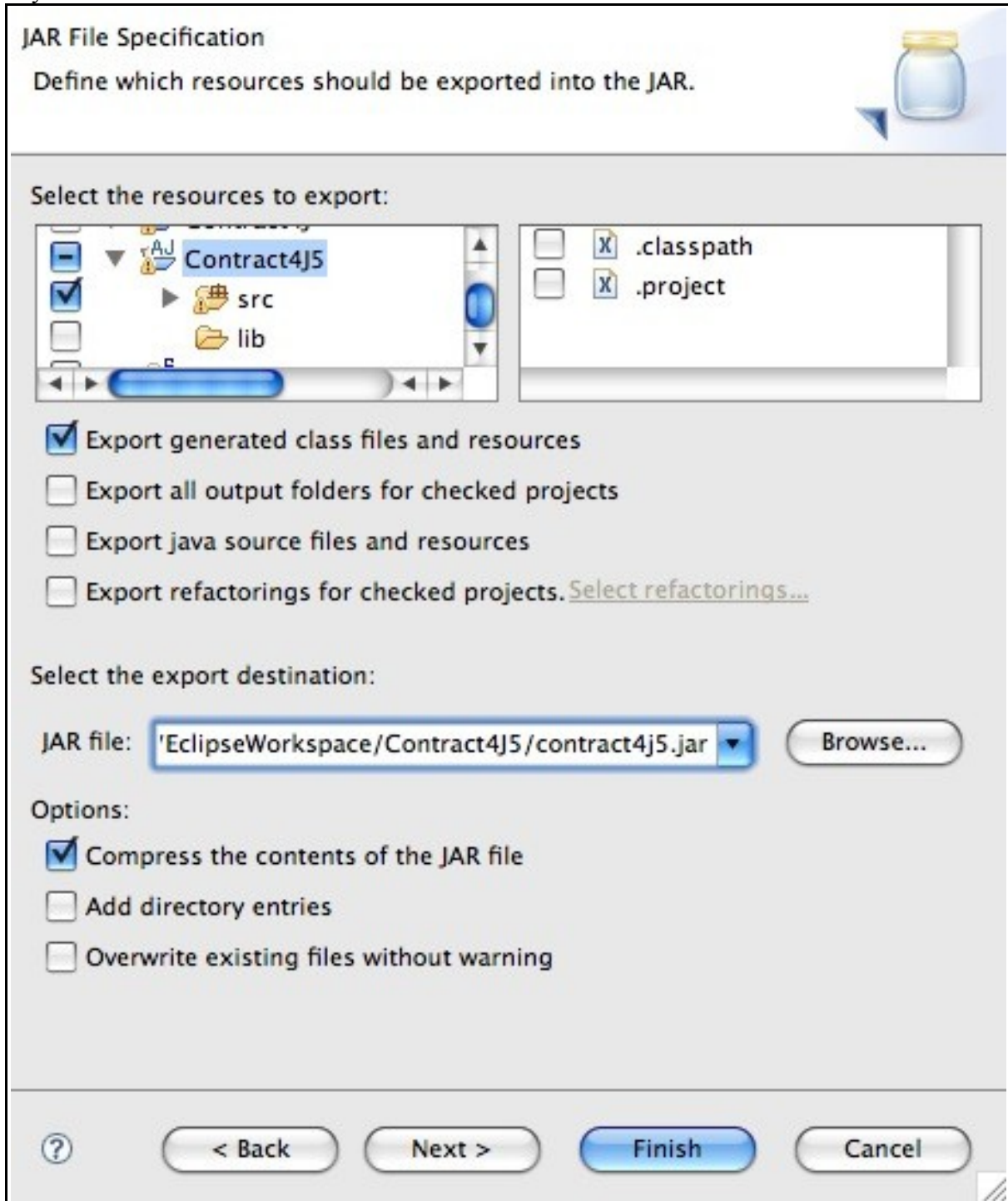
Select the Contract4J library project in the Eclipse project browser by clicking on it once. In the File menu, select the Export menu item – a window allowing you to select what to export should appear, looking like this:



First step in creating the Contract4J library – selecting export to JAR file.

Click the Next button.

In the following window we will select the resources that are to be exported to the Contract4J library.



Selecting the resources to be exported to the Contract4J library.

- Begin by de-selecting the .classpath and .project resources on the right.
- Click on the small triangle to the left of the Contract4J library project so you can see the “src” and “lib” folders.
- De-select the “lib” folder, as shown in the picture.
- Select the destination where the Contract4J library file will be saved and what name it will have by using the Browse button on the left.
- Click the Finish button to create the library.

Eclipse will complain about some error, but this will not affect the result of the build – just click the OK button in the error dialog to continue.

Now we are ready to use Contract4J!

Using Contract4J

Now we will write a small program that uses Contract4J, in order to get a glimpse of how it works. This example is by no means exhaustive and should not serve as more than a very brief introduction to some of the features of Contract4J.

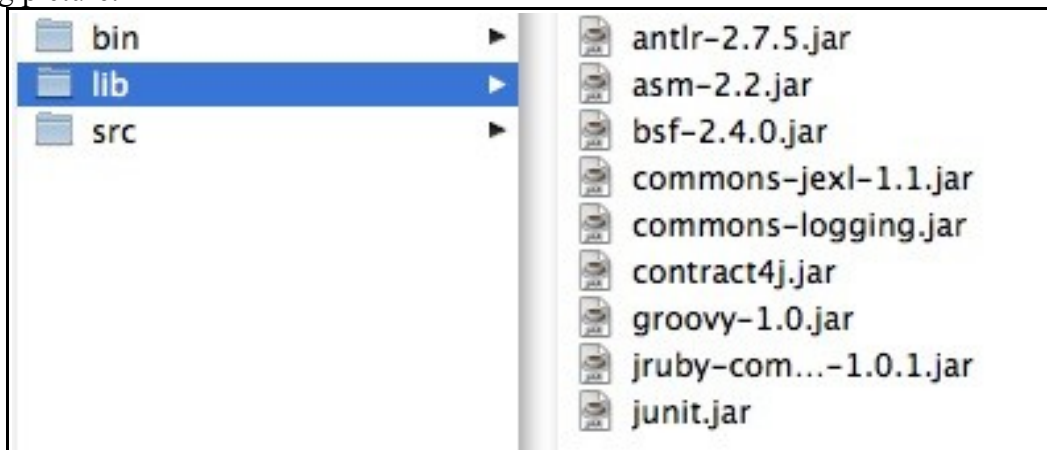
Creating the Project

In the same way as we did when creating the project for the Contract4J library described [above](#), create a new AspectJ project, but give it another name. I chose the name “FirstContract4JExample”.

Adding Libraries

Leave Eclipse for a moment to create a “lib” directory in the example program project directory and then copy the required libraries, as described in [above](#). The Contract4J library we created must also be copied to the project “lib” directory.

Having completed this step, the “lib” directory should now contain the items as shown in the following picture:



The example program “lib” directory with all the required libraries in place.

The Contract4J library I built is named “contract4j.jar” in the above picture.

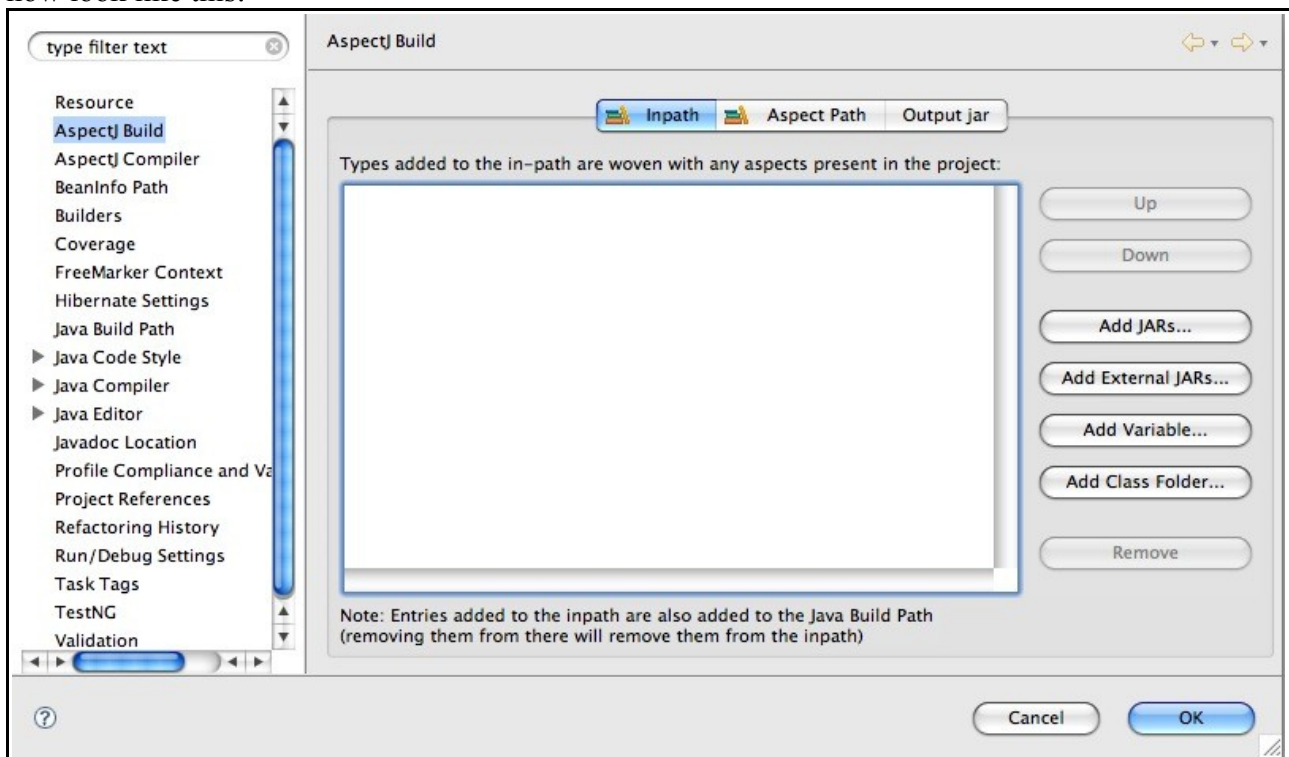
Configuring the Build Path

Next we return to Eclipse and add the libraries we just copied into the “lib” directory to the project. The procedure for adding libraries to a project was described in the section [Configuring the Build Path](#) above.

The result should be the same, with the addition of the Contract4J library JAR.

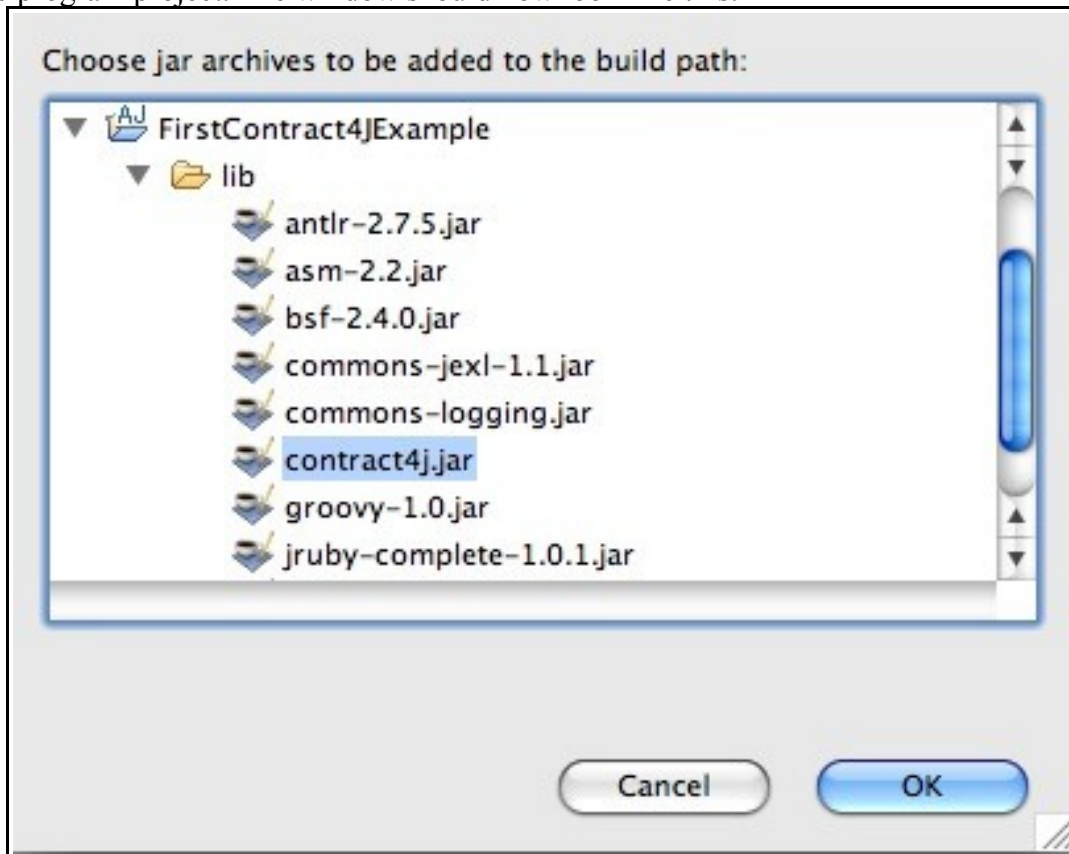
In addition to adding the libraries to the Java build path, we also have to add the Contract4J library to the AspectJ in-path.

This is done by right-clicking on the example program project in the Eclipse project browser and selecting Properties. Then select the AspectJ Build item in the list on the left. The window should now look like this:



Configuring the AspectJ in-path in Eclipse for the example program project.

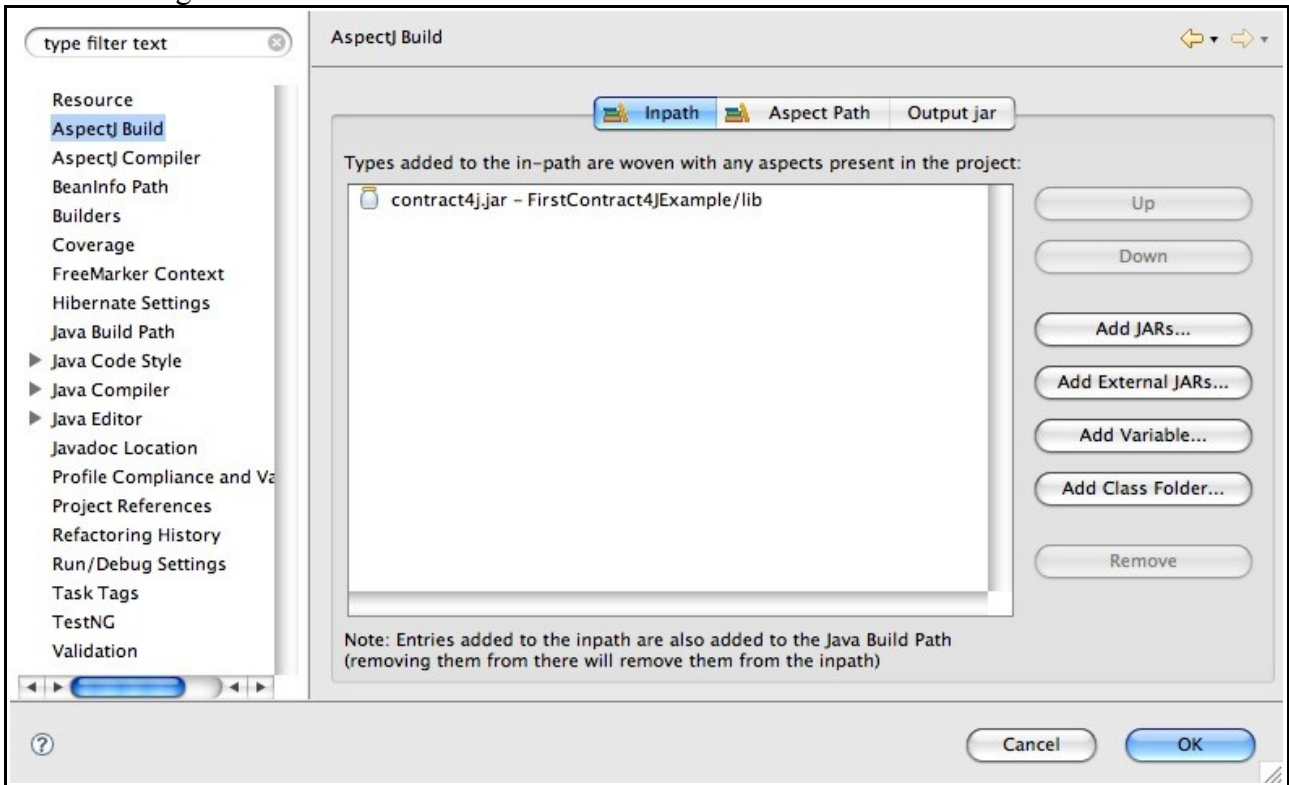
Click the Add JARs button on the right. In the window that appears, expand the example project node by clicking the small triangle to the left of the name and then expand the “lib” directory in the example program project. The window should now look like this:



Adding the Contract4J library to the AspectJ in-path of the example program project – selecting the library.

Select the Contract4J library and click the OK button.

The remaining window should now look like this:



Configuring the AspectJ in-path in Eclipse for the example program project.
The Contract4J library has been added to the AspectJ in-path.

Note that the Contract4J library should be added to the in-path, not the Aspect Path!
Click the OK button.

Writing an Example Program

Finally, we can start writing the example program, which just consists of two classes; a class with some properties that can be set and retrieved and a main class which creates and uses the previous class.

The MyBean Class

First, the class containing the properties:

```
package com.ivan;

import org.contract4j5.contract.Contract;
import org.contract4j5.contract.Invariant;
import org.contract4j5.contract.Pre;

@Contract
public class MyBean
{
    /** Some count that must be within a certain range. */
    @Invariant(value="mCount > 0 && mCount < 20",
        message="Count must be between 1 and 20")
    private int mCount = 1;
    /** A string that can't be set to null or the empty string. */
    private String mString = "default";

    public int getCount()
    {
        return mCount;
    }

    public void setCount(int inCount)
    {
        mCount = inCount;
    }

    public String getString()
    {
        return mString;
    }

    @Pre(value="inString != null && inString.size() > 0")
    public void setString(String inString)
    {
        mString = inString;
    }
}
```

The *MyBean* class of the example program that uses Contract4J.

In this class, there are a number of Contract4J annotations. A very brief introduction will be given for each, please refer to the [Contract4J web site](#) for more complete documentation.

- **@Contract**
All classes and interfaces that declares one or more contracts must be annotated with this annotation. If not, the contracts will be ignored.
- **@Invariant(value="mCount > 0 && mCount < 20" message="Count must be between 1 and 20")**
This annotation declares the contract for the `mCount` instance variable and specifies the message that will be given if this contract is broken.
- **@Pre(value="inString != null && inString.size() > 0")**
This annotation declares a pre-condition contract to the `setString` method, which says that the supplied string must not be null and must not be an empty string.

The Main Class

The *Main* class creates an instance of the *MyBean* class and tries to set some values that will violate the contracts that were specified.

```
package com.ivan;

public class Main
{
    public static void main(String[] inArgs)
    {
        MyBean theBean = new MyBean();

        System.out.println("BEFORE");
        System.out.println("Count = " + theBean.getCount());
        System.out.println("String = " + theBean.getString());

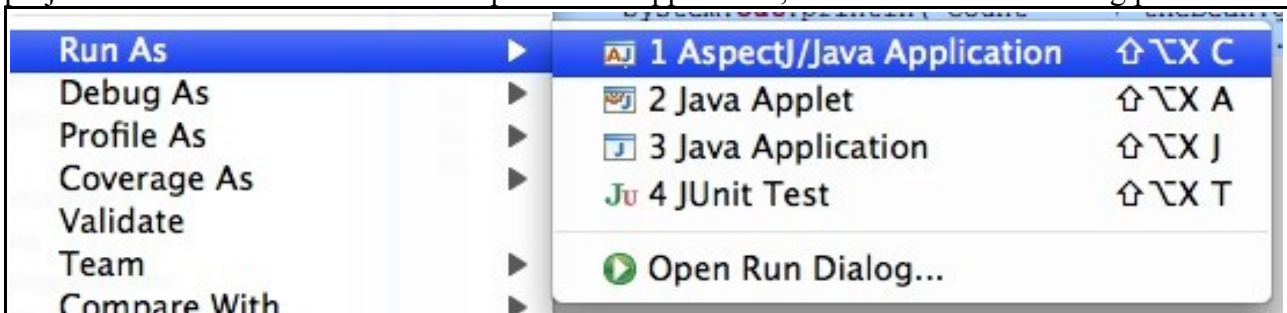
        theBean.setCount(25);
        theBean.setString("");

        System.out.println("\nAFTER");
        System.out.println("Count = " + theBean.getCount());
        System.out.println("String = " + theBean.getString());
    }
}
```

The *Main* class of the example program that uses Contract4J.

Running the Example Program

No special preparations are needed in order to run the example program, just right-click on the project and select Run As and then AspectJ/Java Application, as shown in the following picture.



Running the example program.

In this first version, the program will generate the following console output:

```
BEFORE
Count = 1
String = default
[FATAL] DefaultContractEnforcer: *** Contract Failure (MyBean.java:41): Invar test
"mCount > 0 && mCount < 20" for "com.ivan.MyBean" failed. Count must be between 1 and
20 [failure cause = null]
Exception in thread "main" org.contract4j5.errors.ContractError: *** Contract Failure
(MyBean.java:41): Invar test "mCount > 0 && mCount < 20" for "com.ivan.MyBean" failed.
Count must be between 1 and 20 [failure cause = null]
    at
    org.contract4j5.enforcer.ContractEnforcerHelper.makeContractError(ContractEnforcerHelper
.java:183)
    at
    org.contract4j5.enforcer.defaultimpl.DefaultContractEnforcer.finishFailureHandling(Default
ContractEnforcer.java:29)
    at
    org.contract4j5.enforcer.ContractEnforcerHelper.handleFailure(ContractEnforcerHelper.jav
a:110)
    at
    org.contract4j5.enforcer.ContractEnforcerHelper.invokeTest(ContractEnforcerHelper.java:9
2)
        at com.ivan.MyBean.mCount_aroundBody5$advice(MyBean.java:190)
        at com.ivan.MyBean.setCount(MyBean.java:41)
        at com.ivan.Main.main(Main.java:34)
```

Console output from the first attempt at running the example program using Contract4J.

We can see that the default values of the attributes in the instance of the *MyBean* class are printed onto the console, but then an exception occurs. When Contract4J detects a breach of a contract, the program will immediately be stopped and a message specifying which contract was broken output to the console.

Note the highlighted text, which is the message that we entered in the contract.

Also note that the line numbers in the above console output do not match the line numbers in the previous source file. This is since most comments have been removed from the source code, to conserve space in this document.

To rectify the contract breach, the parameter of the *setCount* method is changed to 19. The *main* method in the *Main* class will now look like this:

```
...
public static void main(String[] inArgs)
{
    MyBean theBean = new MyBean();

    System.out.println("BEFORE");
    System.out.println("Count = " + theBean.getCount());
    System.out.println("String = " + theBean.getString());

    theBean.setCount(19);
    theBean.setString("");

    System.out.println("\nAFTER");
    System.out.println("Count = " + theBean.getCount());
    System.out.println("String = " + theBean.getString());
}
...
```

The modified *main* method of the *Main* class.

If we try to run the example program once again, it will now generate the following console output:

```
BEFORE
Count = 1
String = default
[FATAL] DefaultContractEnforcer: *** Contract Failure (MyBean.java:50): Pre test
"inString != null && inString.size() > 0" for "setString" failed. [failure cause =
null]
Exception in thread "main" org.contract4j5.errors.ContractError: *** Contract Failure
(MyBean.java:50): Pre test "inString != null && inString.size() > 0" for "setString"
failed. [failure cause = null]
    at
    org.contract4j5.enforcer.ContractEnforcerHelper.makeContractError(ContractEnforcerHelper
.java:183)
    at
    org.contract4j5.enforcer.defaultimpl.DefaultContractEnforcer.finishFailureHandling(DefaultContractEnforcer.java:29)
    at
    org.contract4j5.enforcer.ContractEnforcerHelper.handleFailure(ContractEnforcerHelper.java:110)
    at
    org.contract4j5.enforcer.ContractEnforcerHelper.invokeTest(ContractEnforcerHelper.java:92)
    at
    org.contract4j5.aspects.MethodBoundaryConditions.ajc$before$org_contract4j5_aspects_MethodBoundaryConditions$1$d79ef325(MethodBoundaryConditions.aj:121)
    at com.ivan.MyBean.setString(MyBean.java:52)
    at com.ivan.Main.main(Main.java:35)
```

Console output from the second attempt at running the example program using Contract4J.

This time, no special message is displayed, since no message was defined for the contract that has been broken this time. We can, however, see that the contract broken is for the *setString* method in the *MyBean* class.

Once again we modify the *main* method of the *Main* class, so that the parameter to the *setString* method does not break the contract for the method. The *main* method now looks like this:

```
...
public static void main(String[] inArgs)
{
    MyBean theBean = new MyBean();

    System.out.println("BEFORE");
    System.out.println("Count = " + theBean.getCount());
    System.out.println("String = " + theBean.getString());

    theBean.setCount(19);
    theBean.setString("test");

    System.out.println("\nAFTER");
    System.out.println("Count = " + theBean.getCount());
    System.out.println("String = " + theBean.getString());
}
...
```

The modified *main* method of the *Main* class.

A final attempt at running the example program now renders the following console output:

```
BEFORE
Count = 1
String = default

AFTER
Count = 19
String = test
```

Console output from the third attempt at running the example program using Contract4J.

No contracts are broken and the example program can execute normally.