

# **Standardized Geoprocessing with 52°North Open Source Software**

**- FOSS4G 2008 Workshop Tutorial -**

2008-09-29

**Bastian Schaeffer**

**`schaeffer@52north.org`**

This tutorial will guide you through the process of setting up the 52°North WPS, creating a new Process, executing and exporting the process.

## 1. Build the WPS

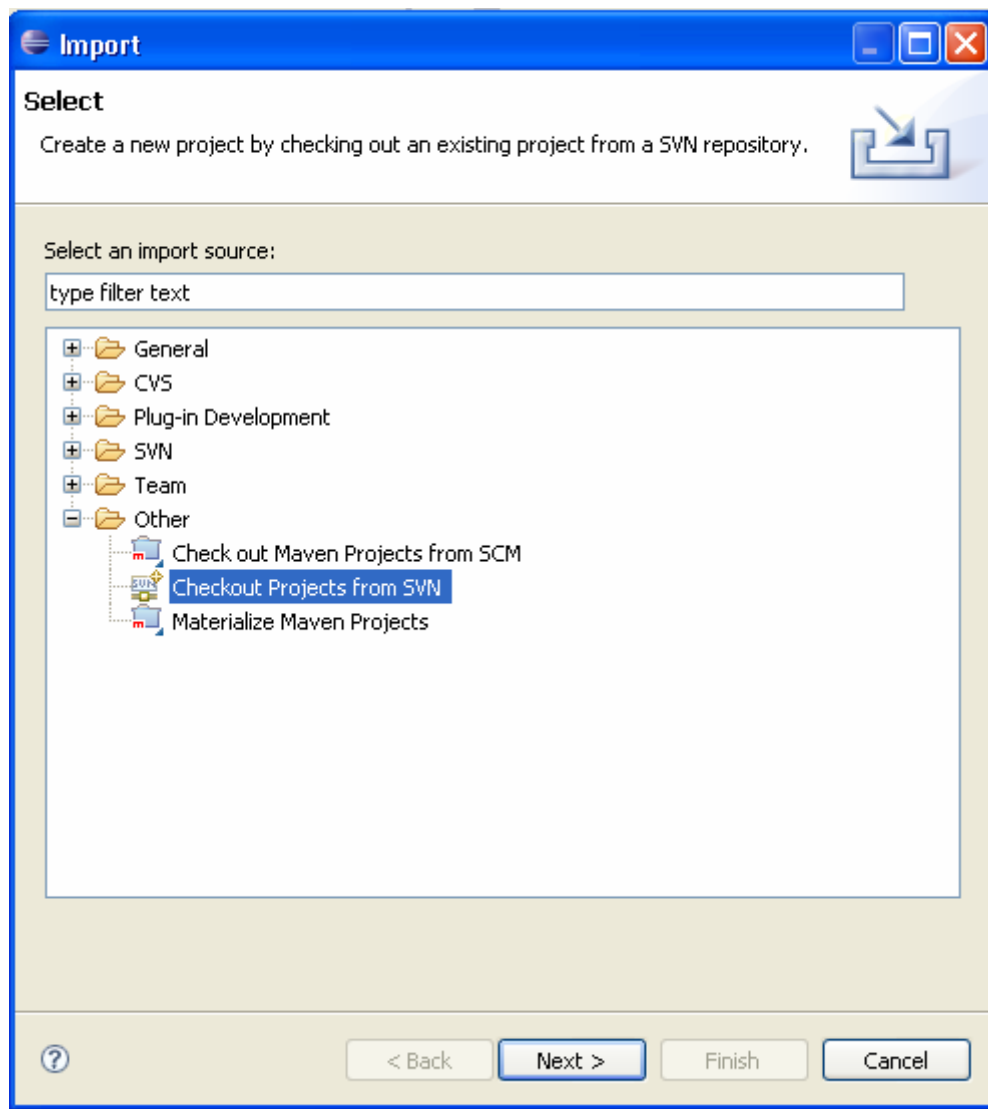
Start eclipse located on your desktop



### 1.1 Get the source code

Go and get the source from 52North via SVN

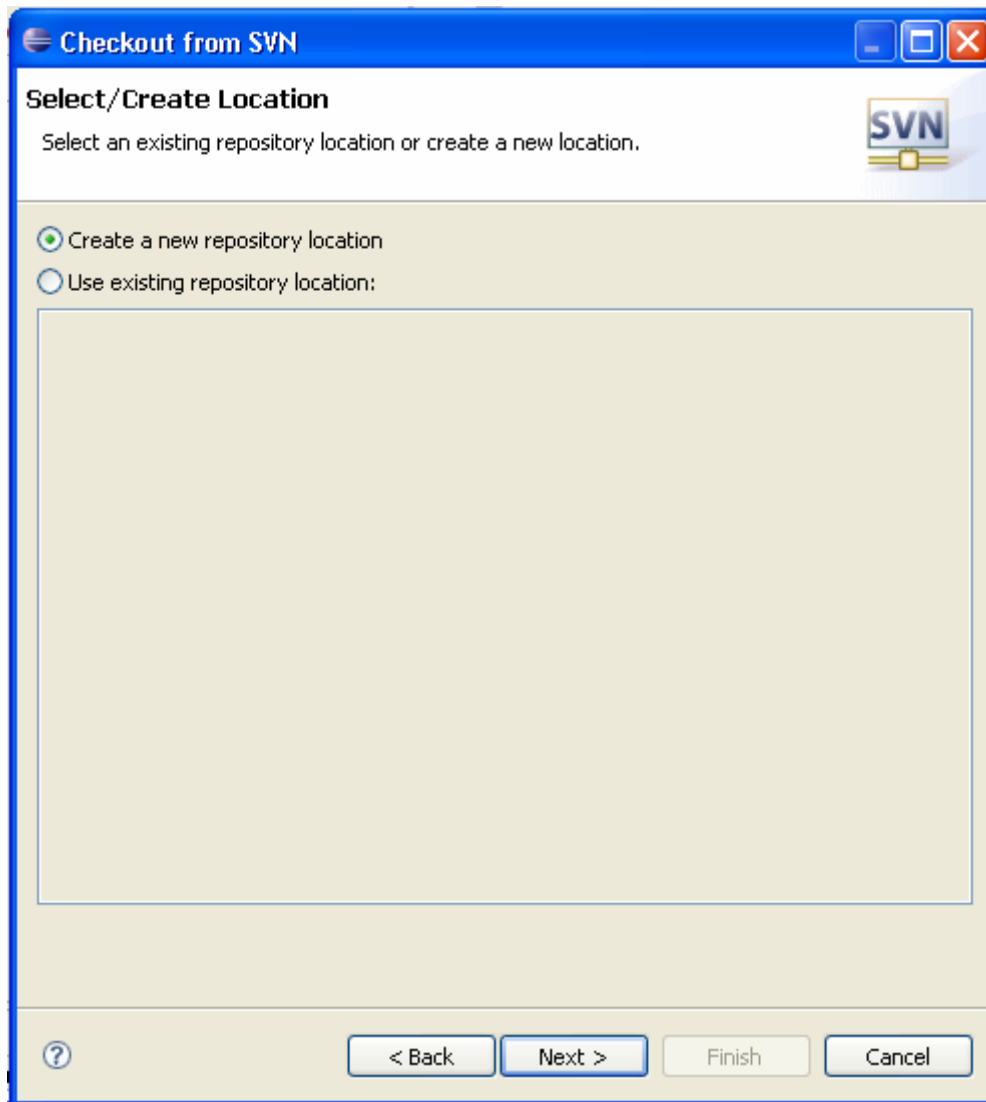
Go to *File->Import→Other→Checkout Projects from SVN*



Click *Next*

Click on:

*create a new repository location.*

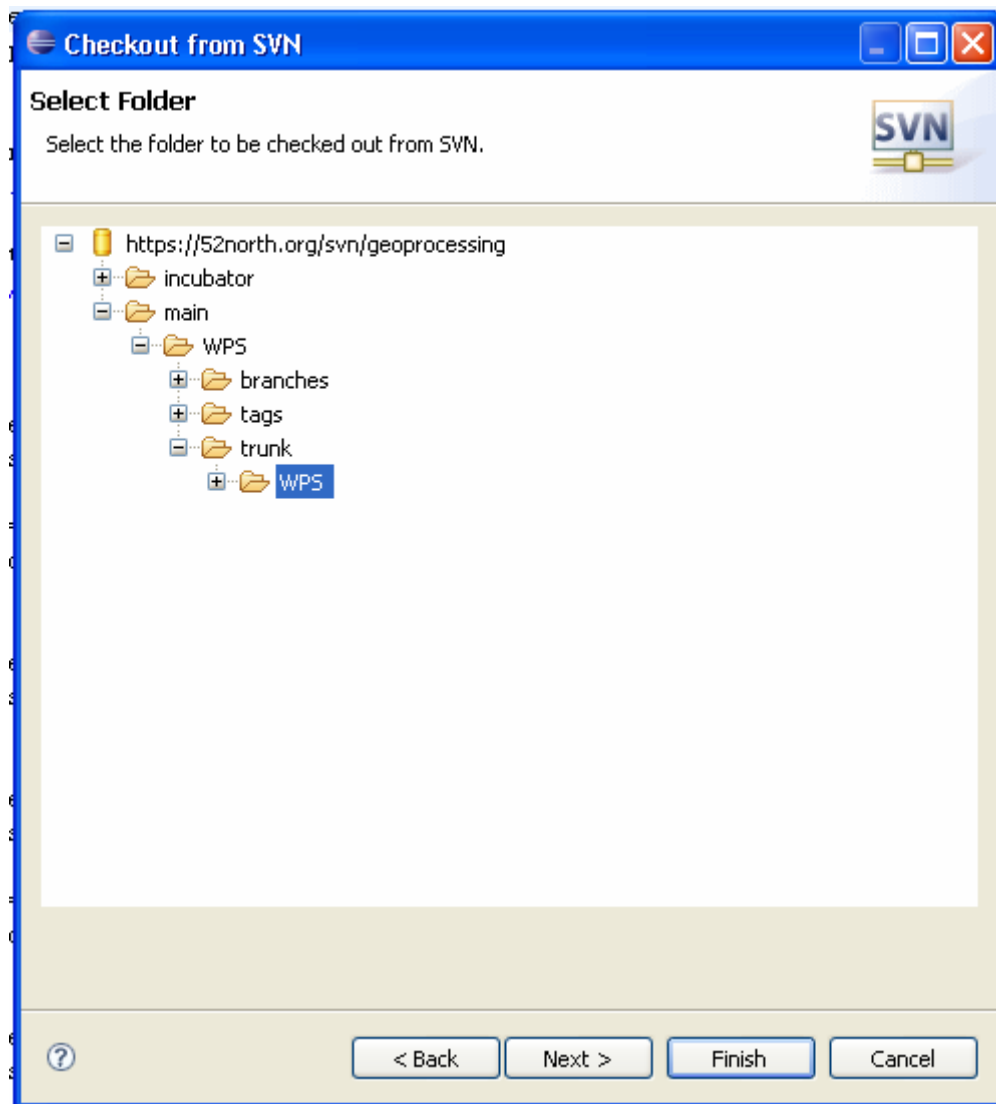


Use the following URL:

<https://52north.org/svn/geoprocessing>

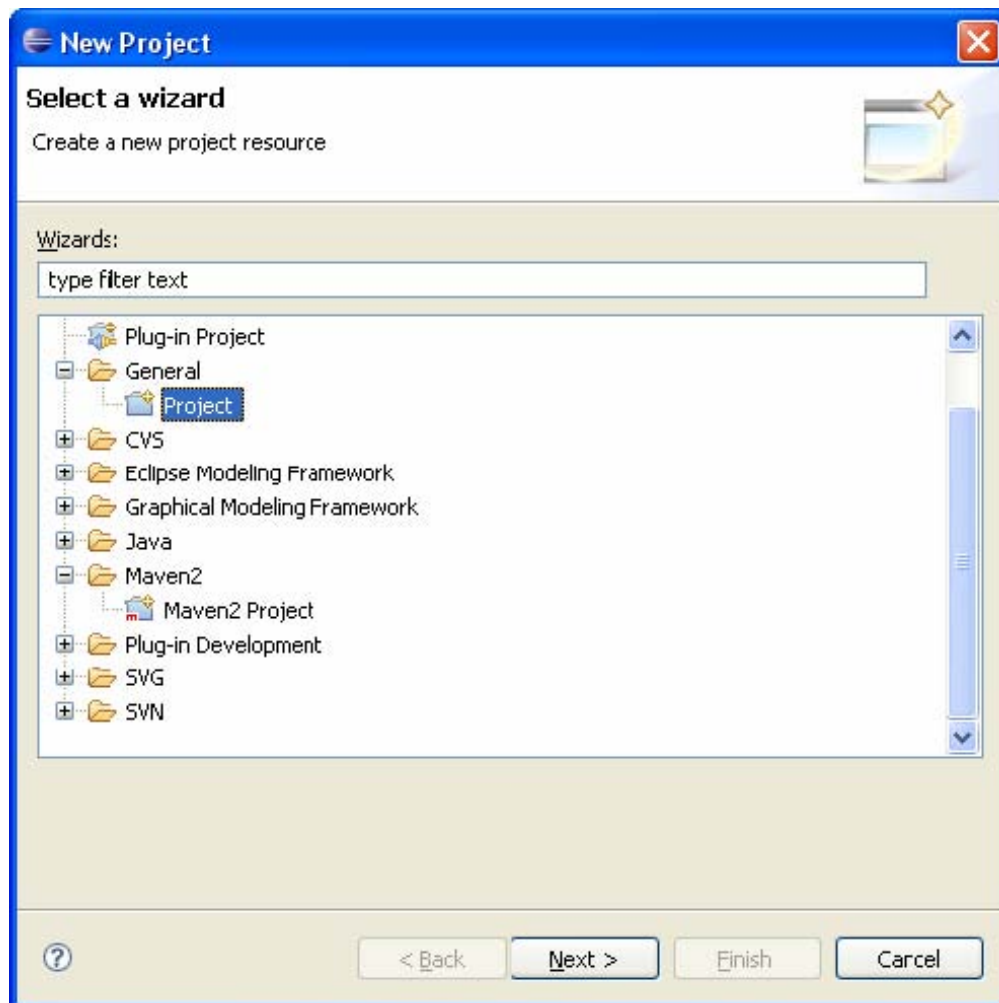
Click *Next* and accept all Certificates

Navigate to *main* → *WPS* → *trunk* → *WPS*  
And select *WPS*

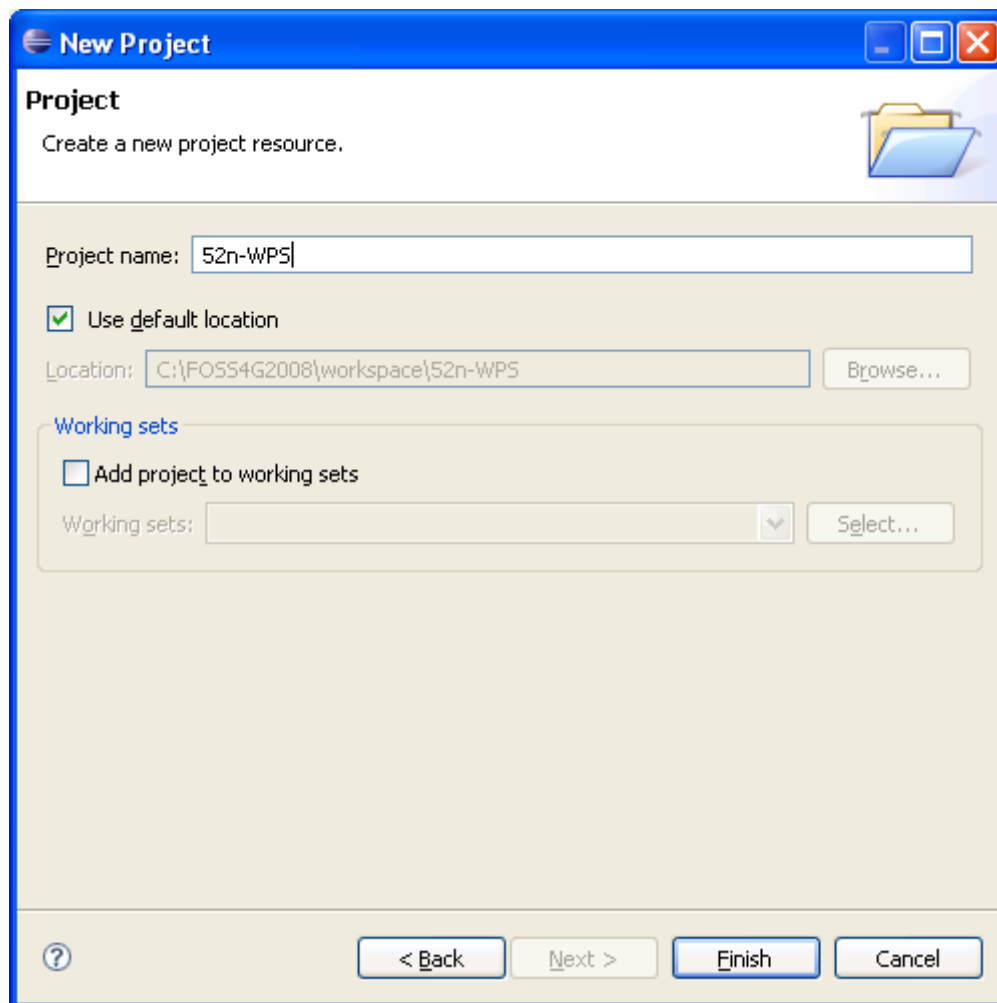


Click *Finish*.

And create a new Project as shown in the next screenshot under *General*→*Project*



Click *Next*

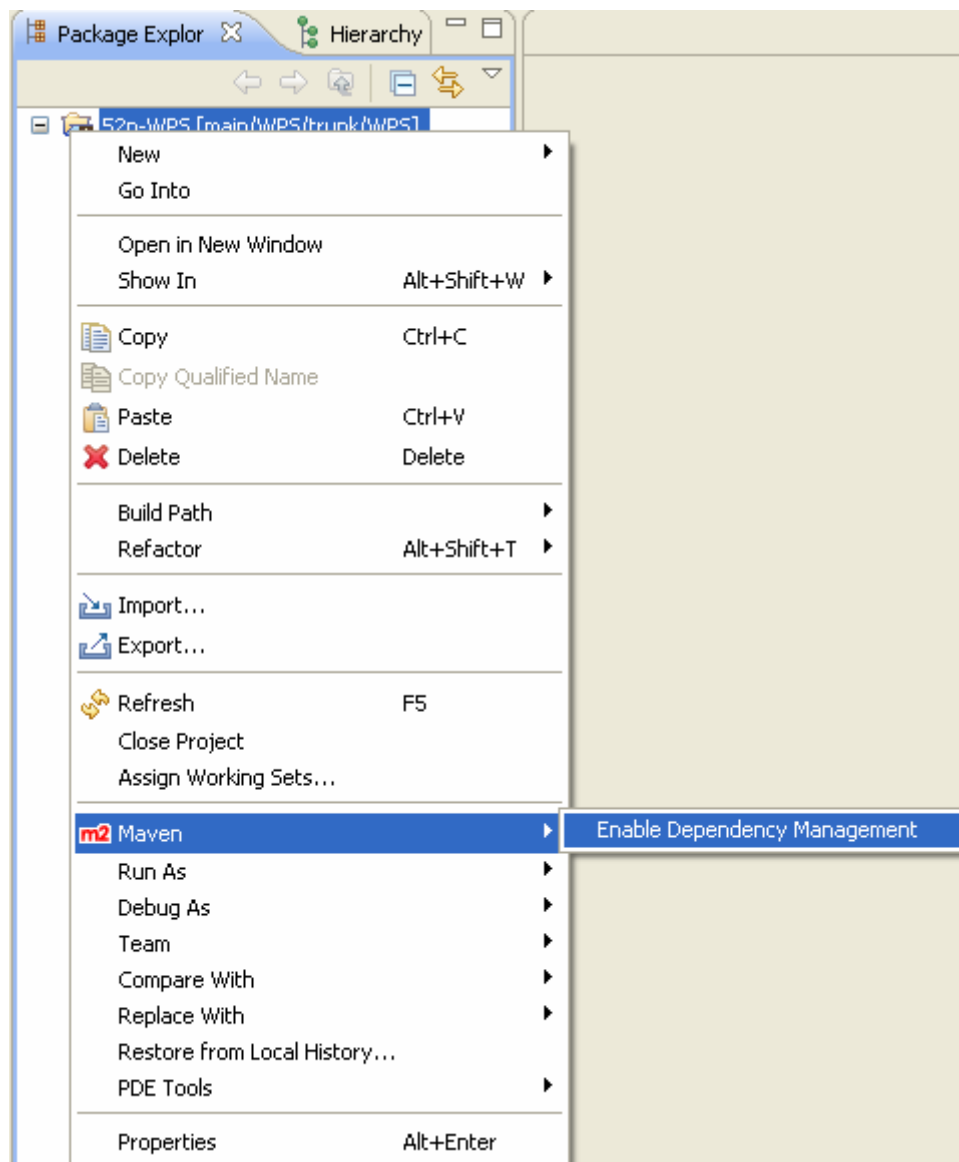


And type in 52n-WPS as the project Name.

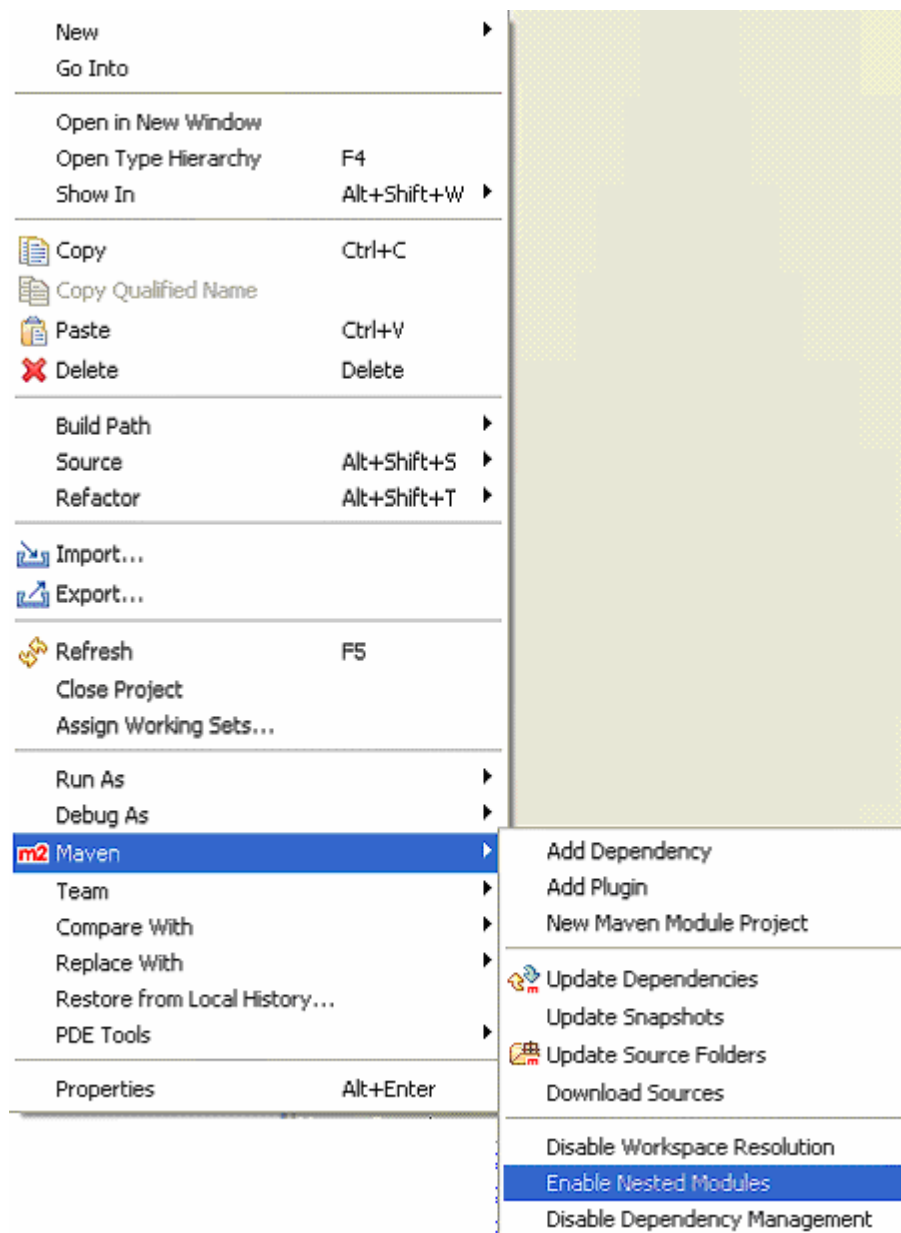
Click *Finish*

## 1.2 Setting the project as an Maven2 Project

Right click on your project in the package explorer. Select *maven* → *Enable Dependency Management* as seen below:

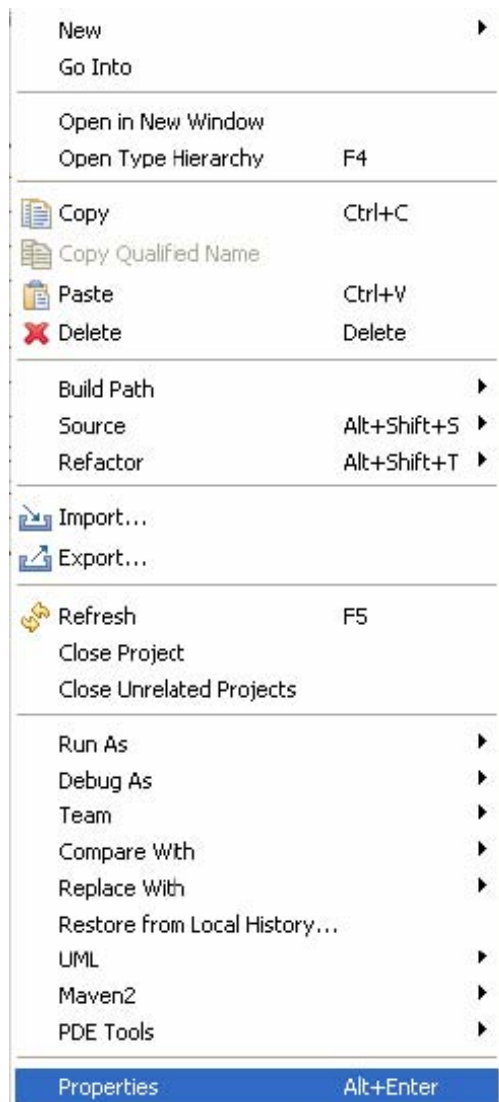


Again, right click on your project, *maven* → *Enable Nested Modules*.

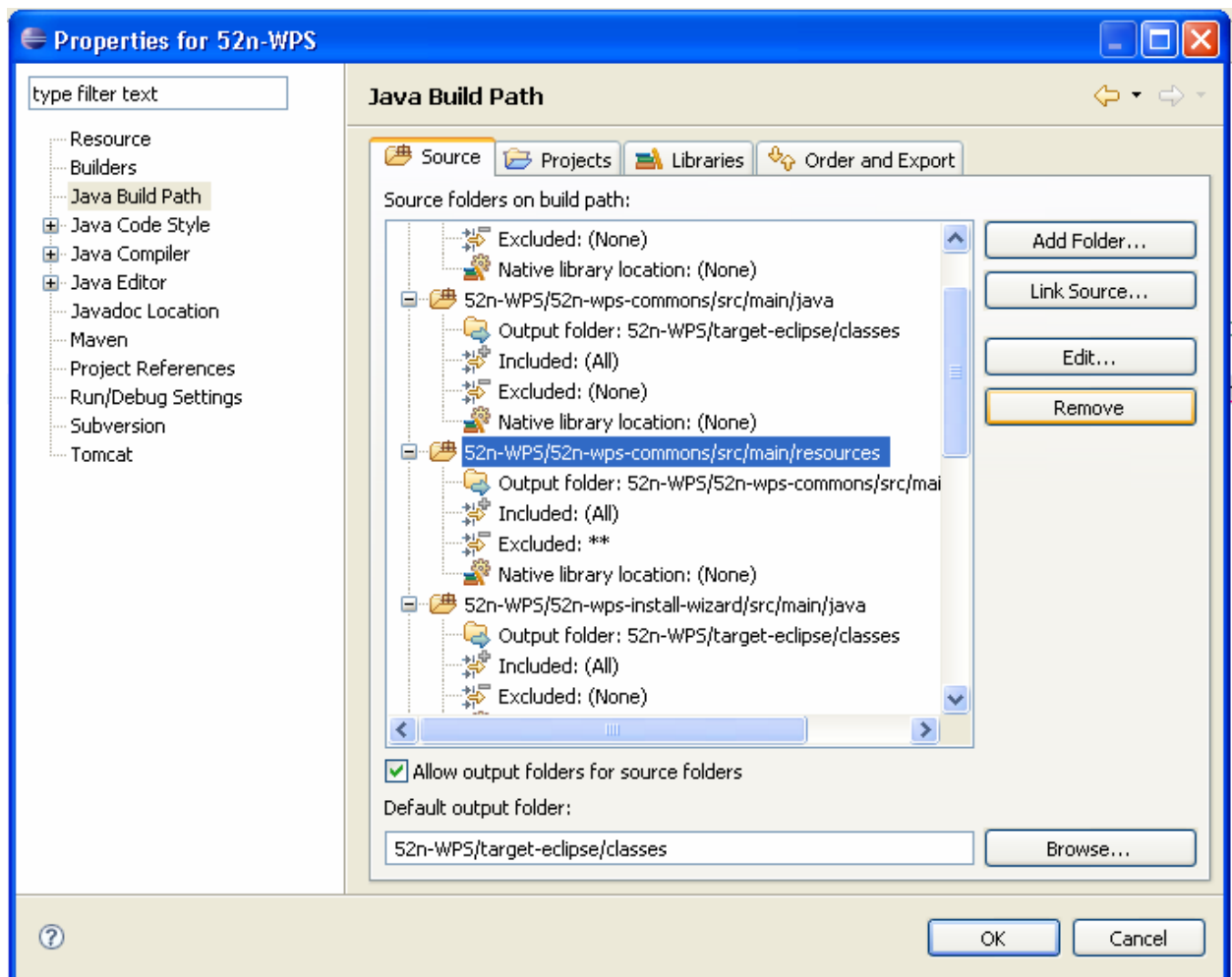


After that, again right click on your project. Go to *properties* → *java build path* → *source*

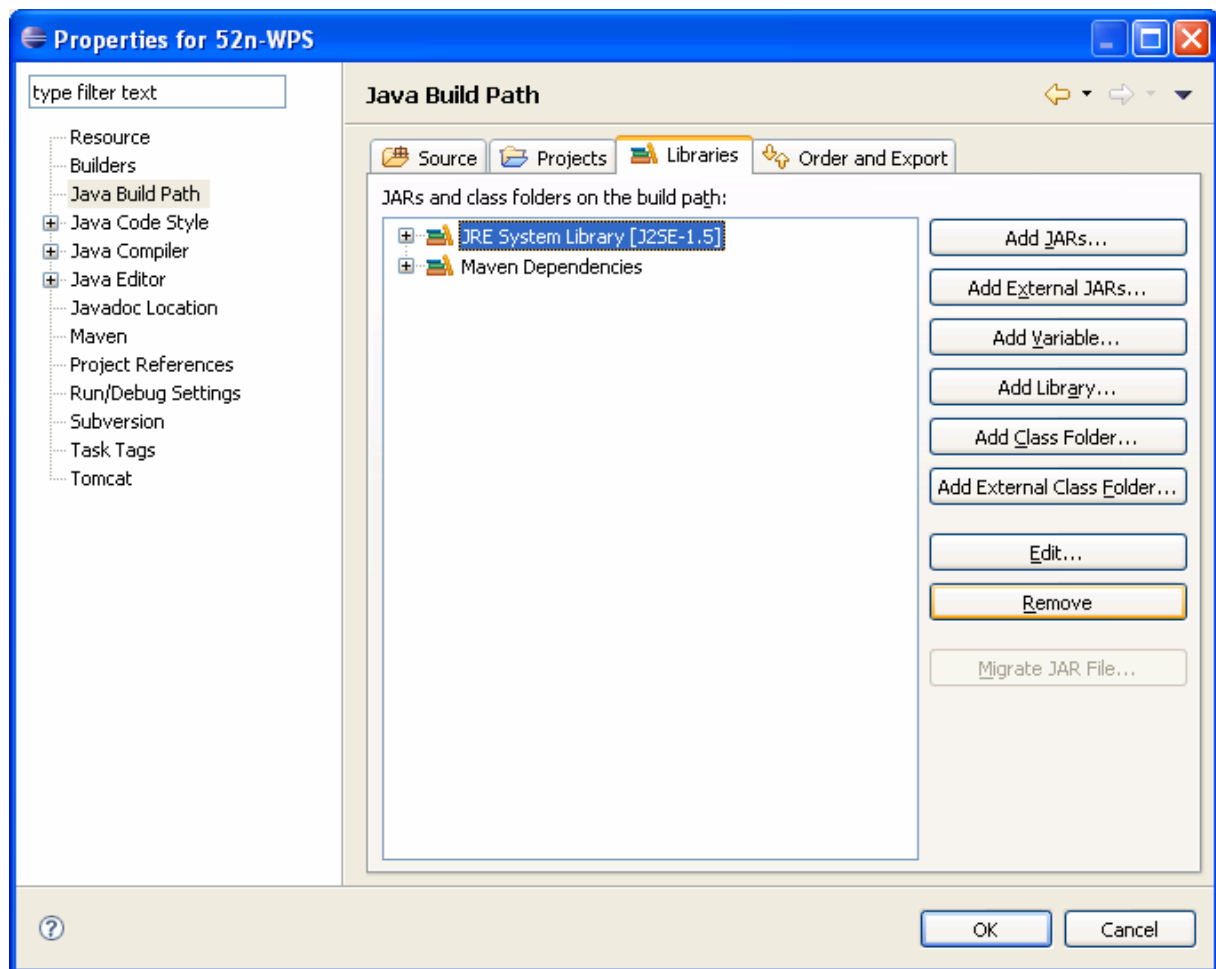




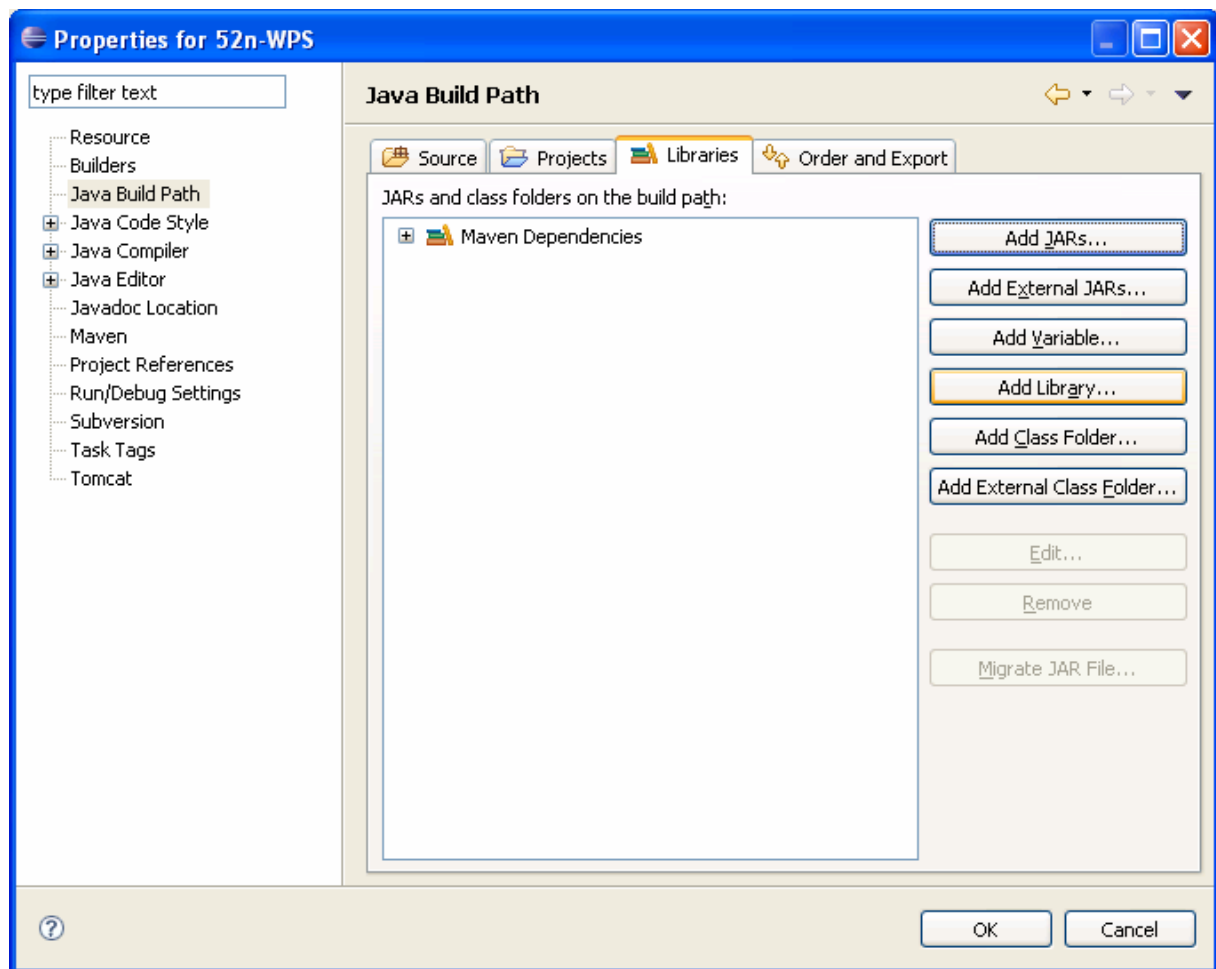
And remove the resources folders (there are three) from the source folder list.



Switch to the *Libraries* tab

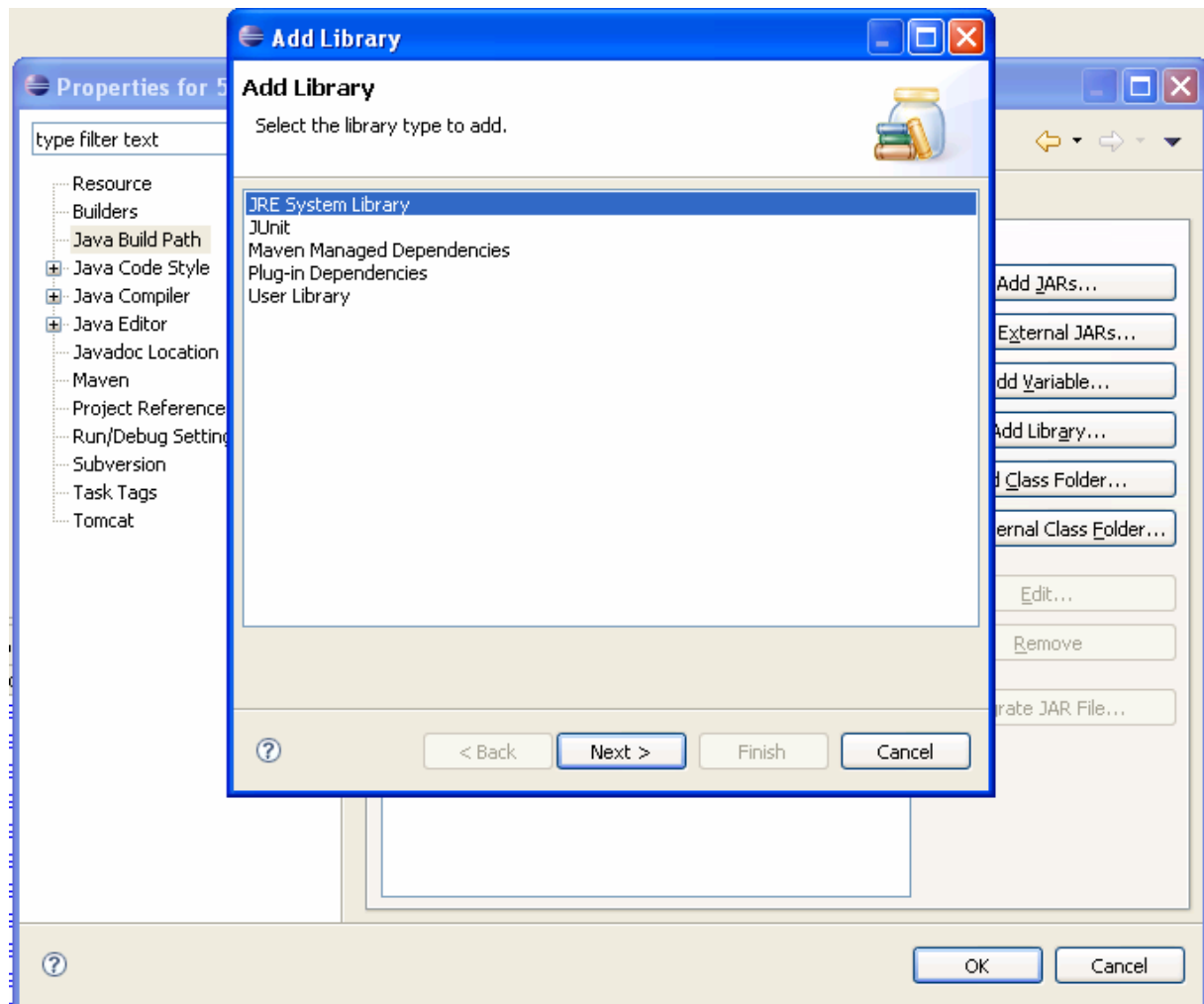


Select *JRE System Library* and click on *Remove*



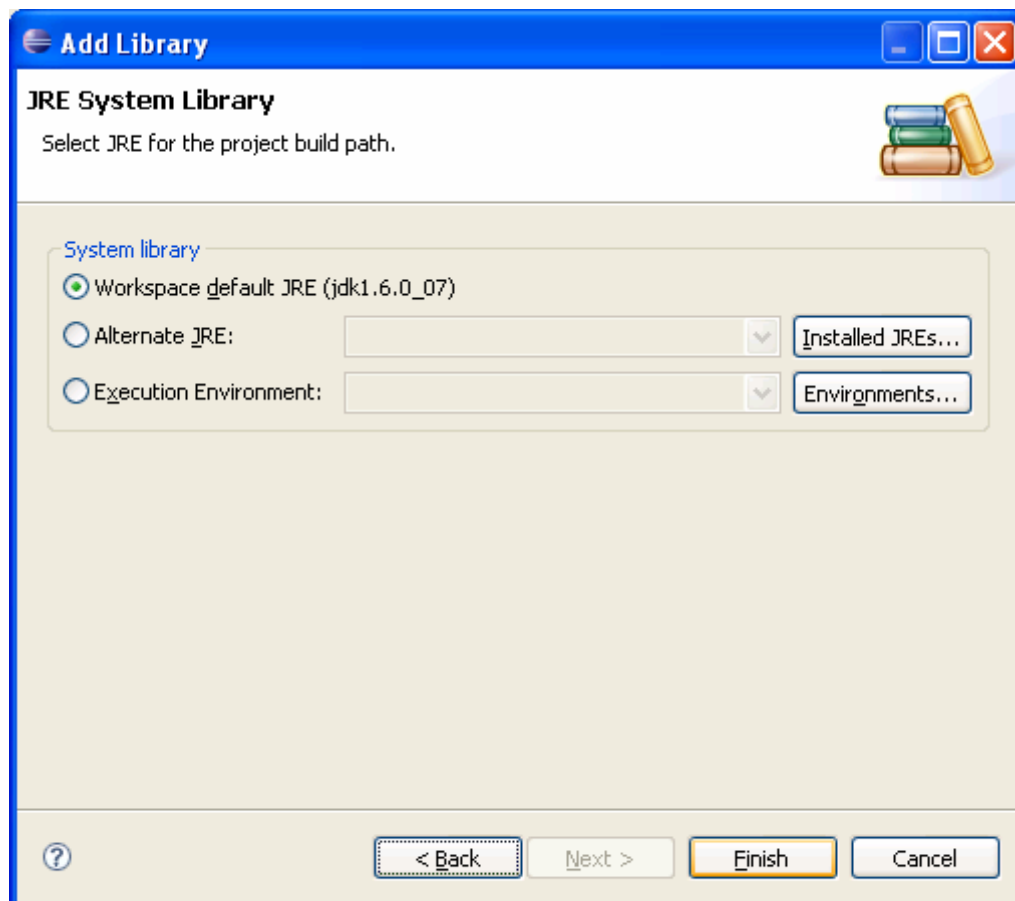
Click on *Add Library*

Select *JRE System Library*



Click *Next*

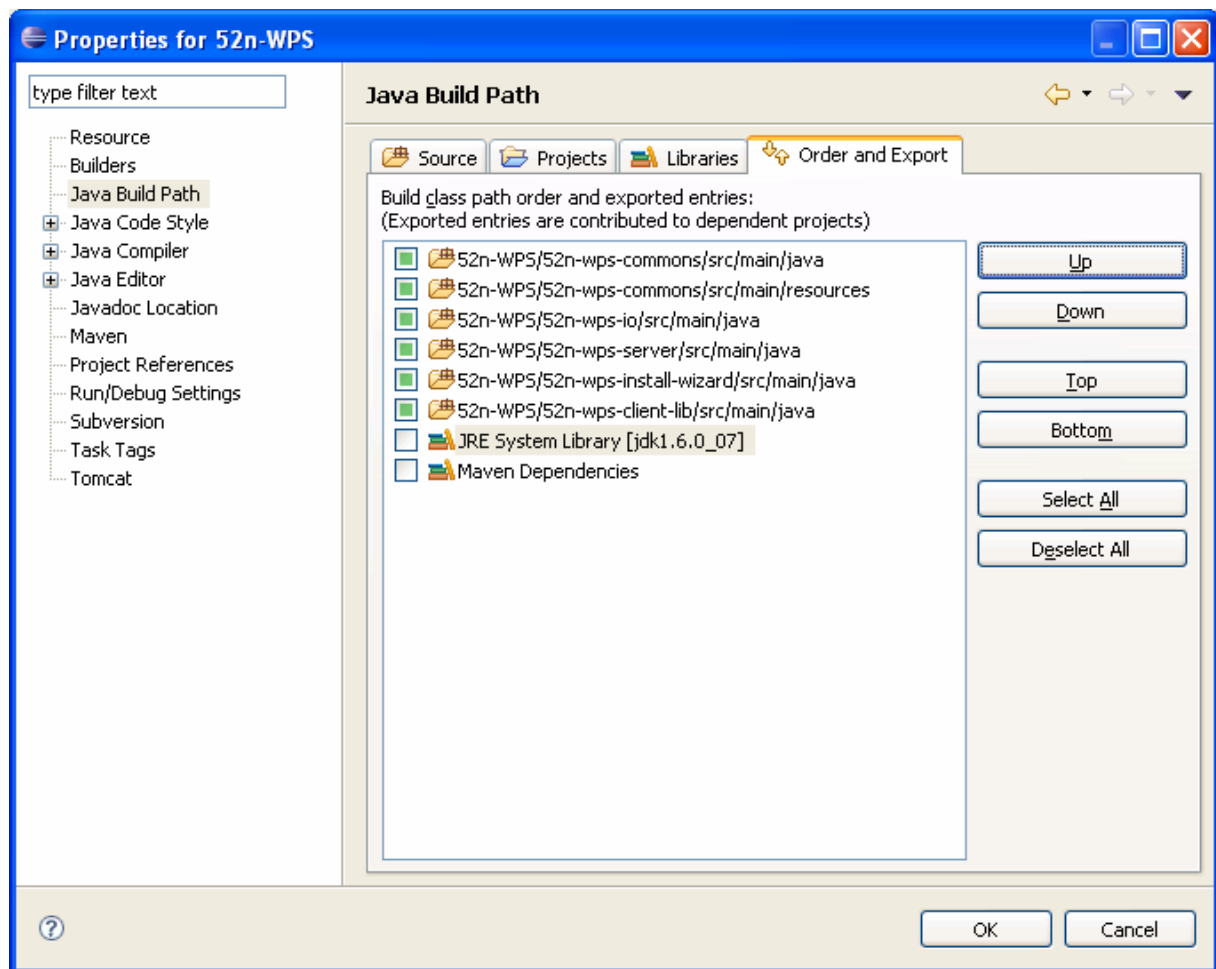
Click on *Finish*



Switch to the *Order and Export* Tab.

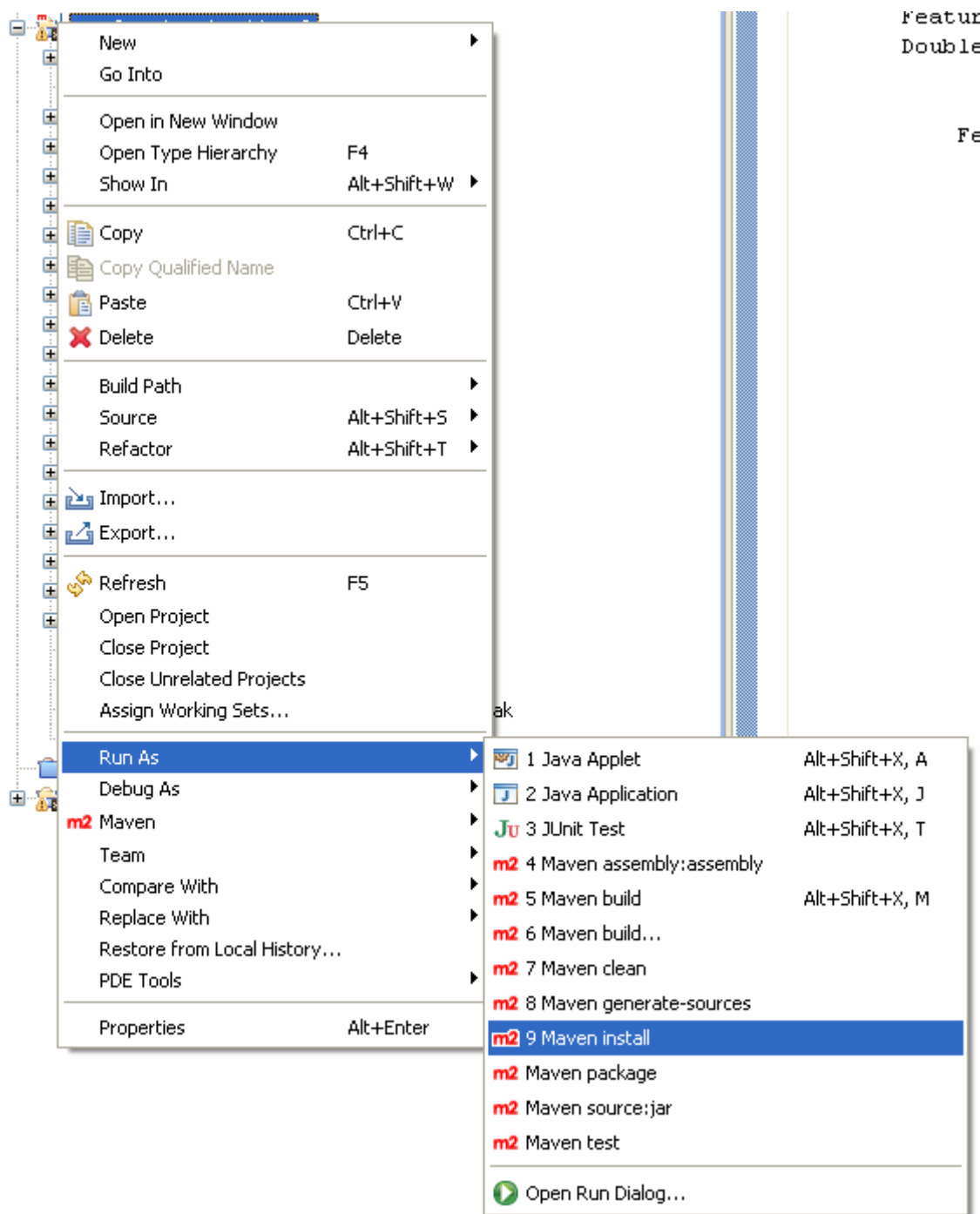
Select *JRE System Library* and hit *Up*

Click *OK*



### 1.3 Compile

Right mouse click on the WPS Project. Go to *Run as* → *Maven install*



As a result, you should see in the console:



```
Problems @ Javadoc Declaration Console
<terminated> Executing install in C:/Dokumente und Einstellungen/bs1980x/Desktop/Müll/WPS Test Install/52n-WPS [Maven Build] C:\Programme\Ja
[INFO] Reactor Summary:
[INFO] -----
[INFO] 52north processing ..... SUCCESS [2.640s]
[INFO] 52north 52n-wps-commons ..... SUCCESS [3.203s]
[INFO] 52north 52n-wps-io ..... SUCCESS [17.746s]
[INFO] 52north 52n-wps-server ..... SUCCESS [19.527s]
[INFO] 52north 52n-wps-nstall-wizard library ..... SUCCESS [2.109s]
[INFO] 52n WPS Web Application ..... SUCCESS [21.574s]
[INFO] 52north 52n-wps-client library ..... SUCCESS [6.983s]
[INFO] -----
[INFO] BUILD SUCCESSFUL
[INFO] -----
[INFO] Total time: 1 minute 14 seconds
[INFO] Finished at: Wed Jun 18 10:45:45 CEST 2008
[INFO] Final Memory: 7M/44M
[INFO] -----
```

## 1.4 Deploy the WPS as a Web application

You can choose many ways to deploy a Web Application in tomcat. In the following only one way is described for Tomcat 5.5:

Go to your *Tomcat home directory* (C:\FOSS4G2008\apache-tomcat-5.5.26) → *conf*  
→ *Catalina* → *localhost*

Create a new wps.xml file with the following content:


```
<Context path="/wps" privileged="true" docBase="<path to your
WPS>\52n-wps-webapp\target\ 52n-wps-webapp-1.0-rc3-SNAPSHOT"
debug="1"/>
```

where <path to your WPS> points to the folder where your wps project resides.

**Hint:** path to your WPS = C:\FOSS4G2008\workspace\52n-WPS

Save the file.

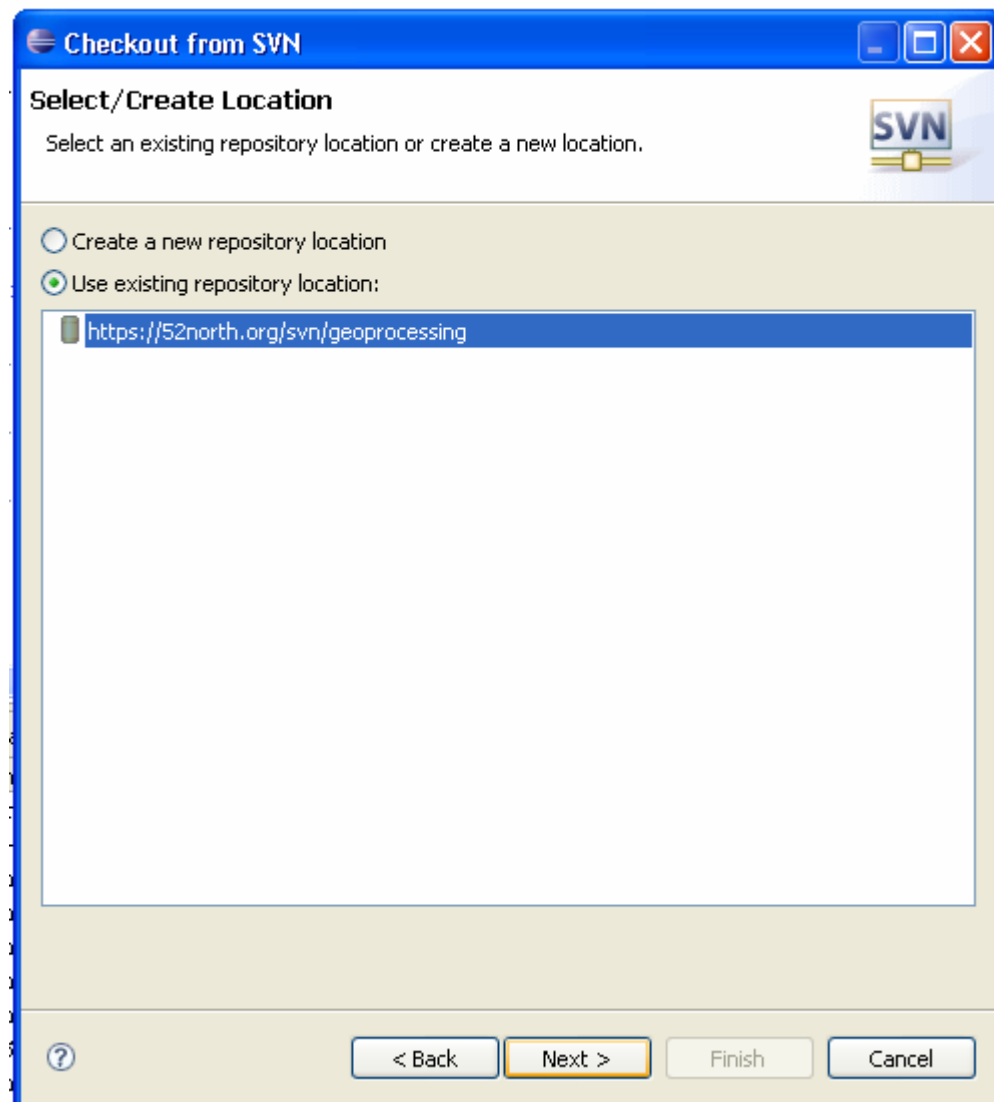
## 1.5 Start Tomcat

You can choose many ways to deploy a Web Application in tomcat. In the following only Click on the following icon in your Eclipse toolbar 

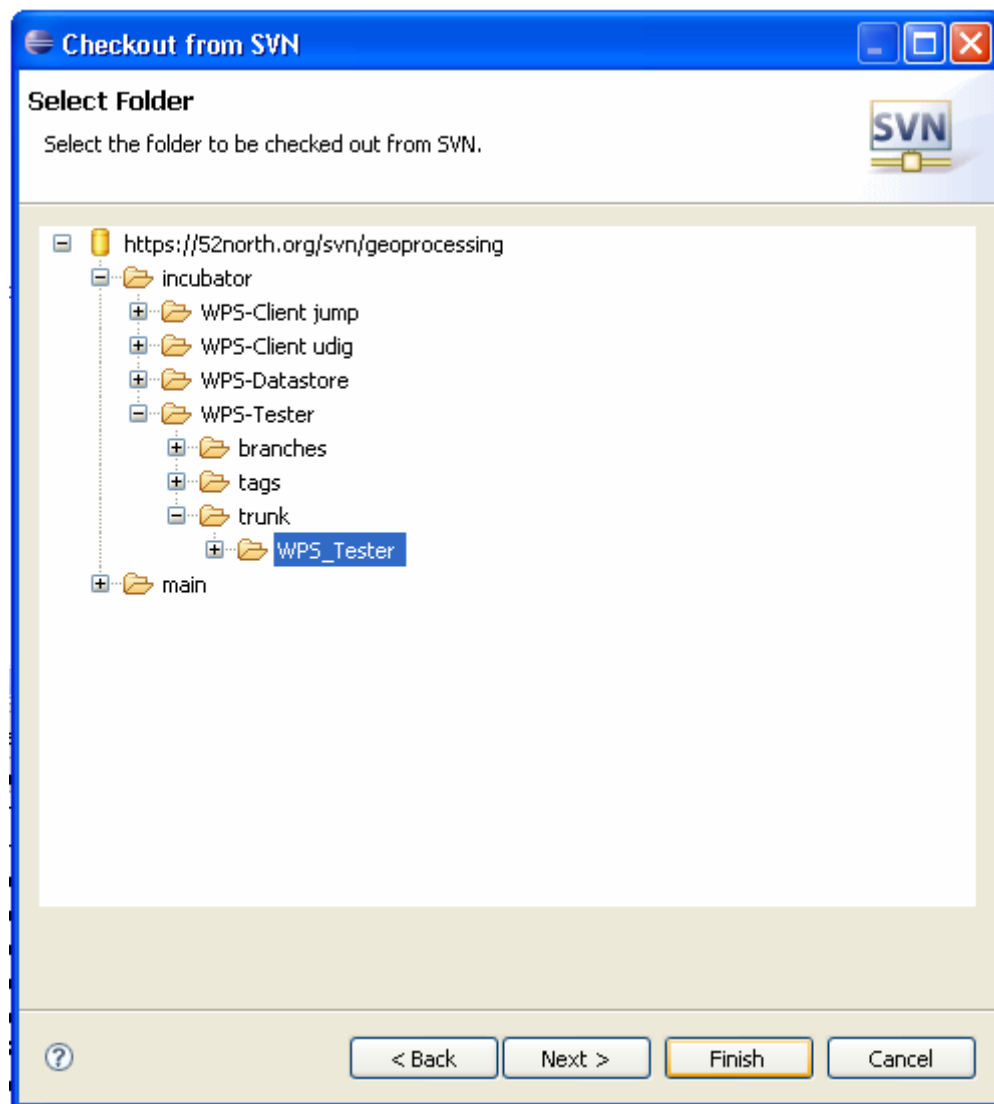
## 1.6 Test your installation

You can test your application with an additional project:

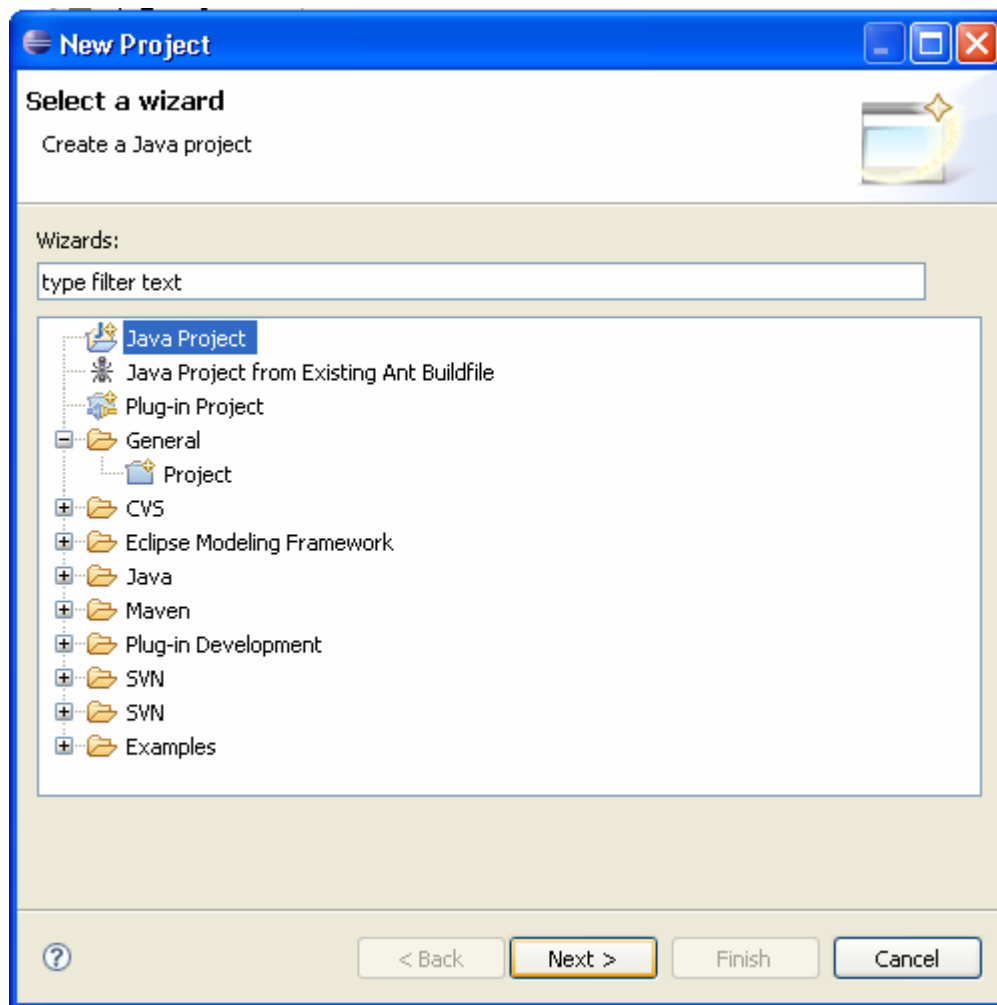
Import a new Project to your Workspace. Use again the SVN client with the existing URL:



This time, go to the *incubator* → *WPS\_Tester* → *trunk* → *WPS\_Tester*

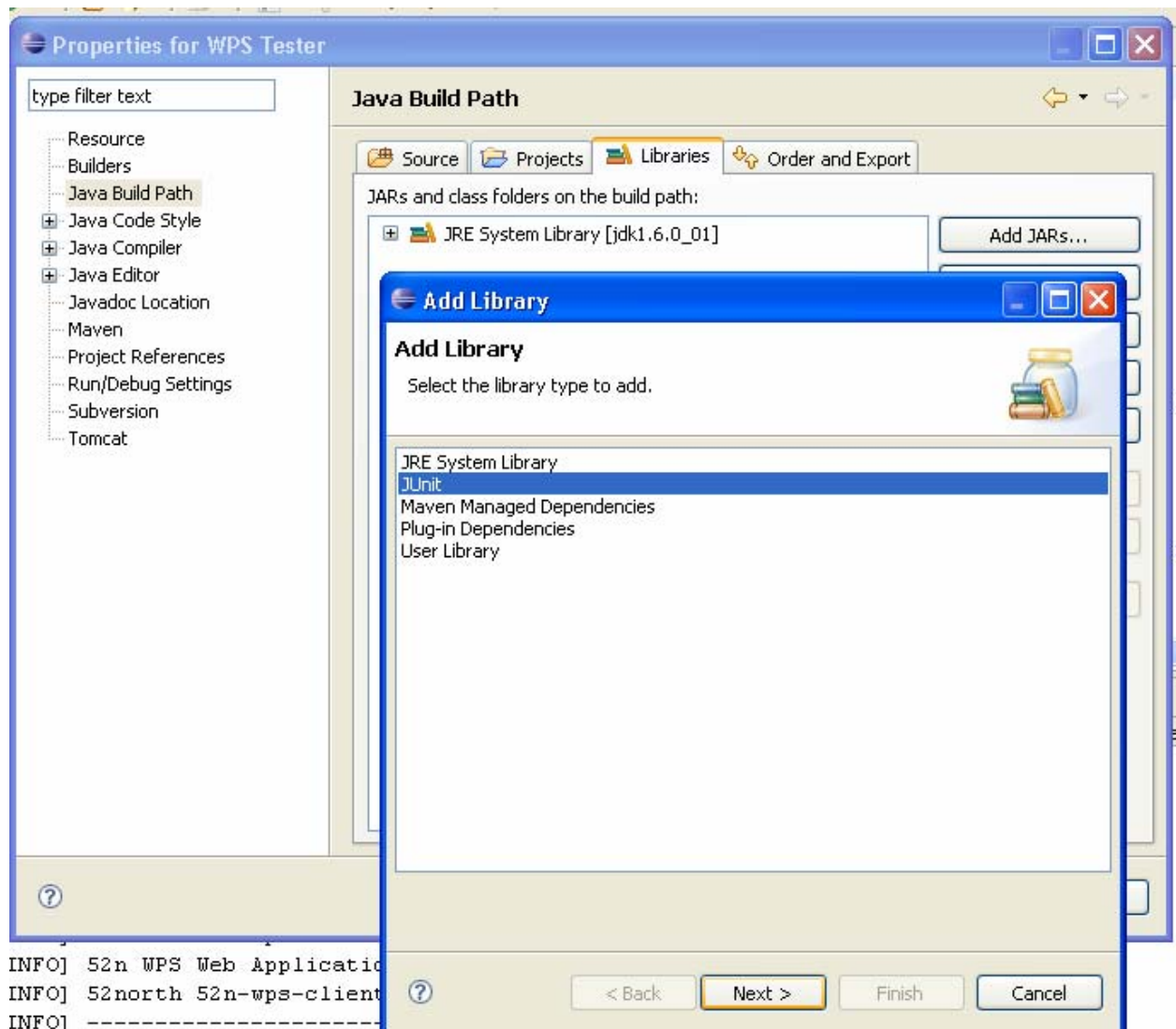


Click *Finish* and create a new Java Project:

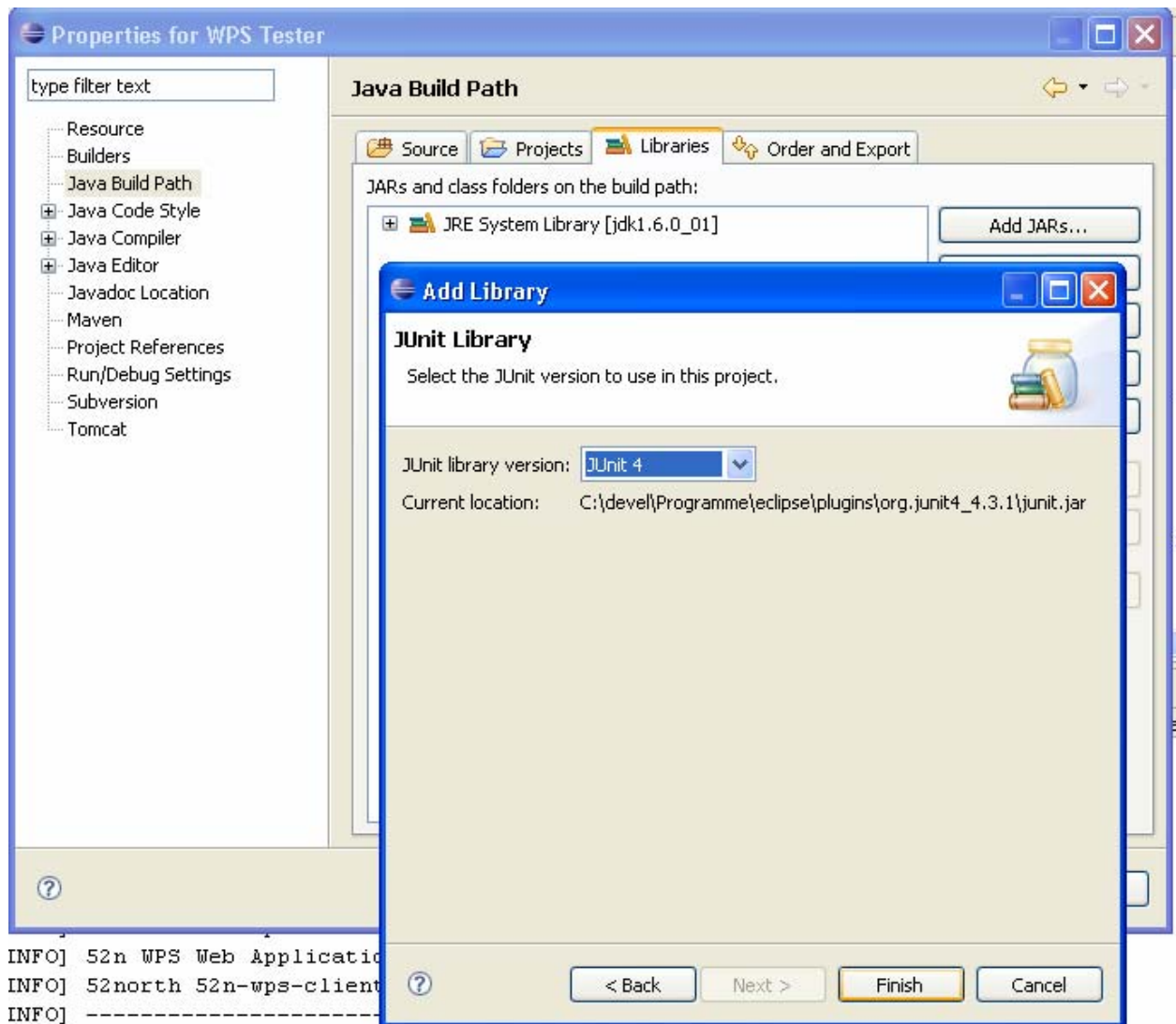


Now you have to add JUnit to the classpath via:

Right mouse click on your *project* → *properties* → *java build path* → *libraries* → *add Library*. Select JUnit.

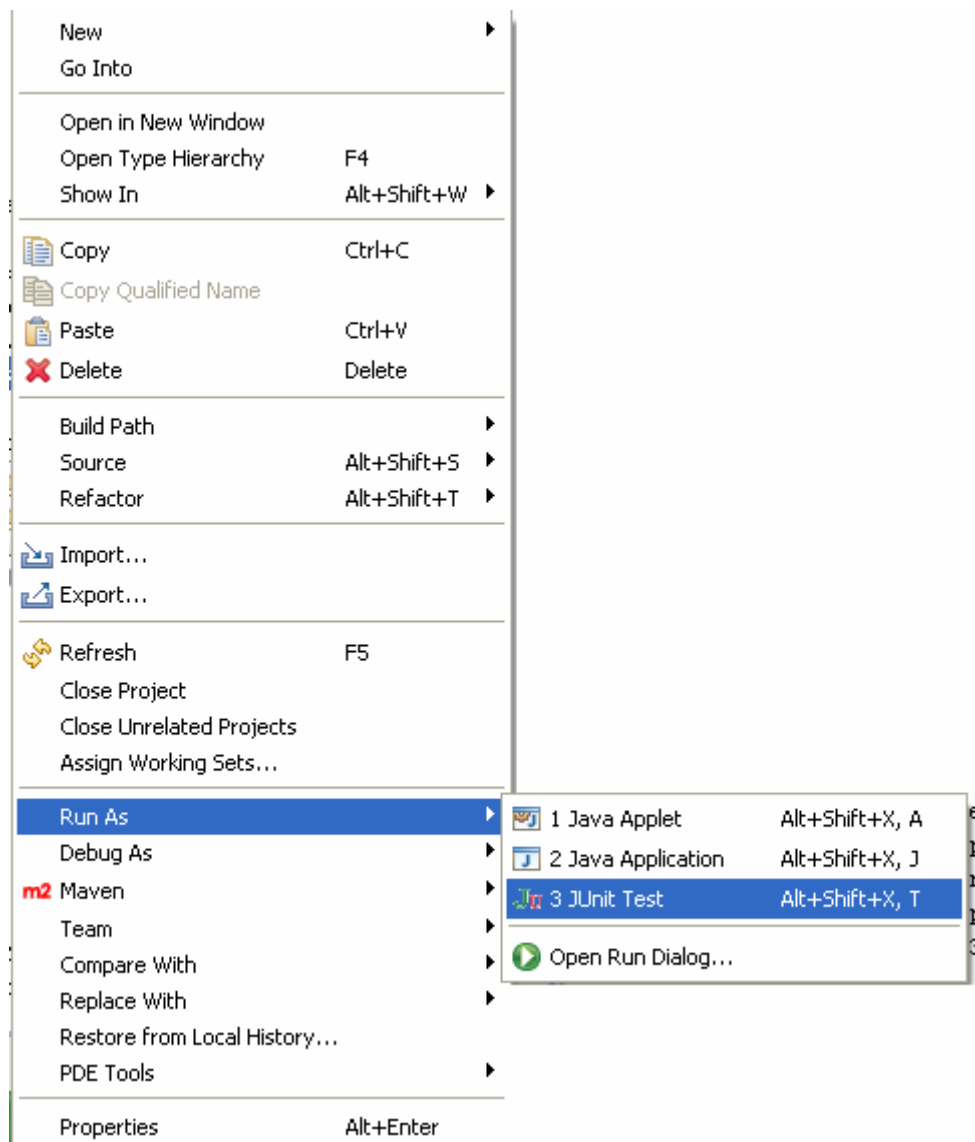


You need to add *JUnit4* and click *Finish*.

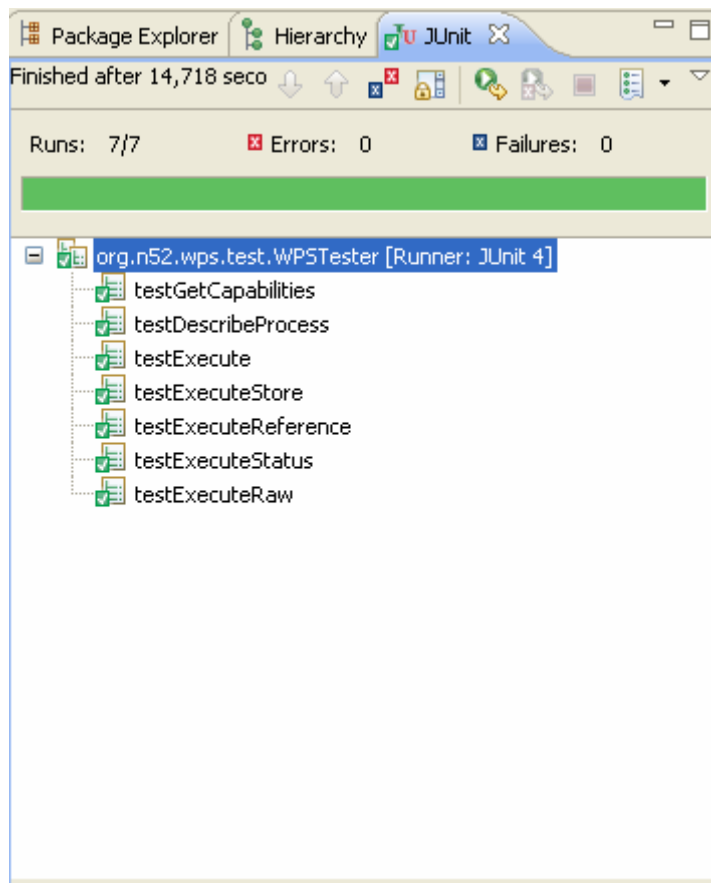


Click **OK**

Right click on your *WPS\_Tester project* → *run as* → *3 Junit Test*



The result should look like:



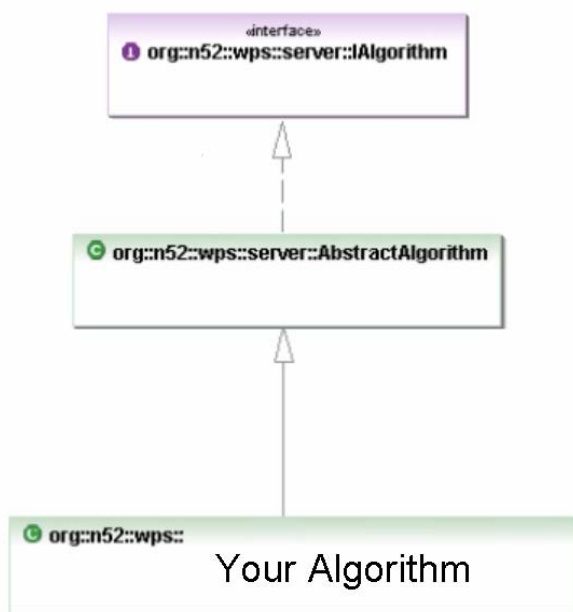
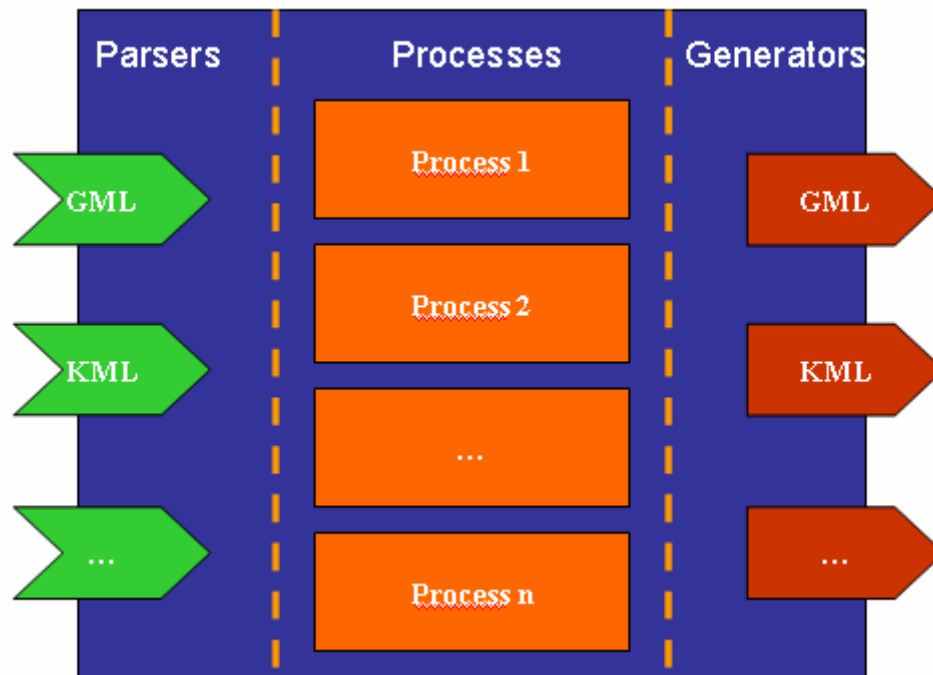
Your WPS is up and running.

## 2. Create your own process

### 2.1 Architecture

In order to understand the big picture, the next two figures remind you of the 52°North WPS Architecture.



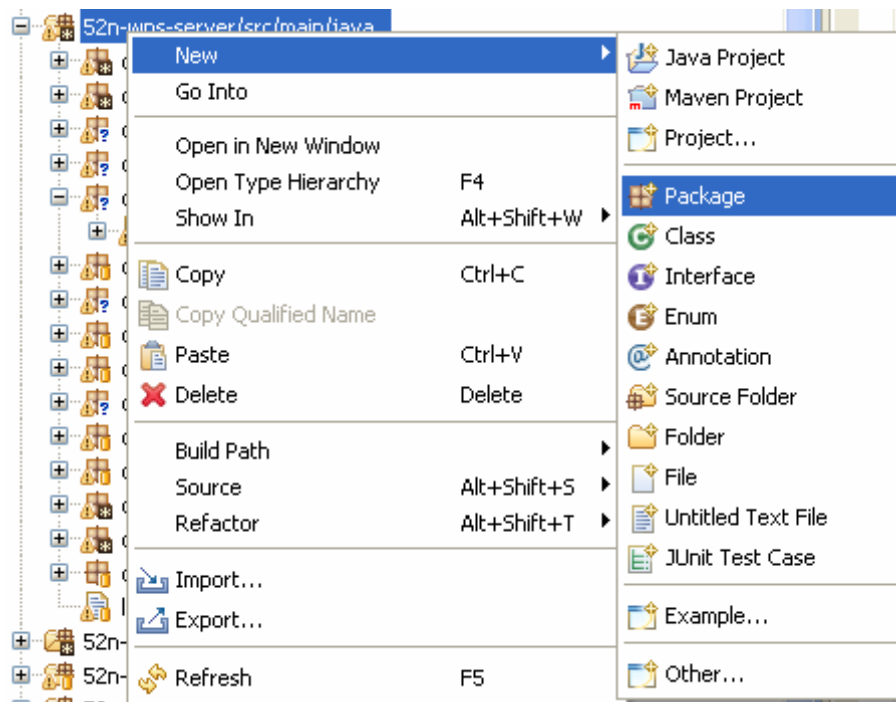


## 2.2 Implementation

In this section you will implement your own WPS algorithm class. This class will be used to calculate on the fly line simplifications.

### 2.2.1 Create a new Package

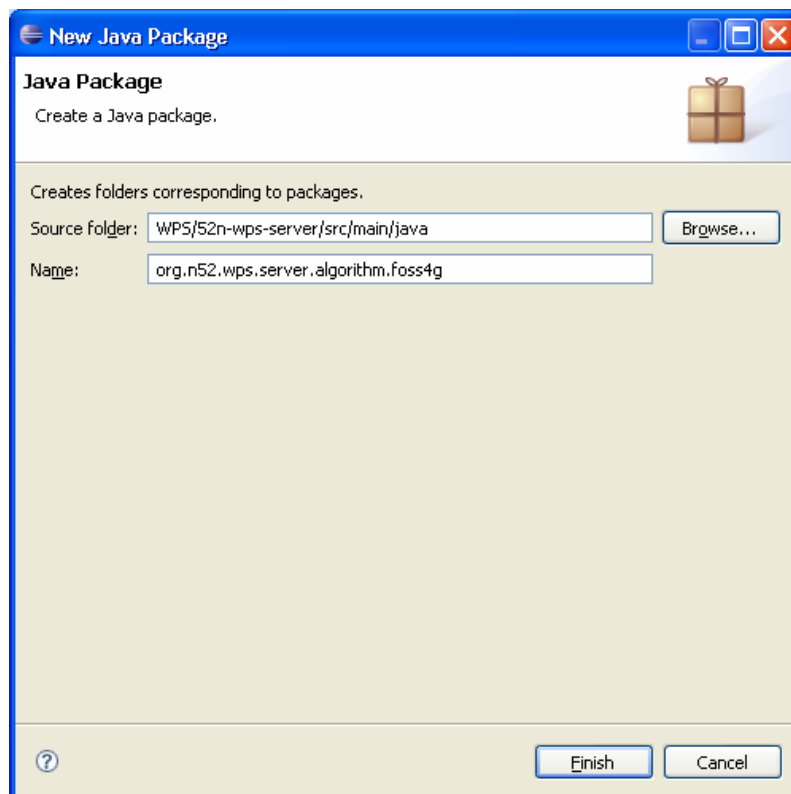
In the package explorer on the left go to `52n-WPS` → `52n-wps-server/src/main/java` → `New` → `Package`



and enter:

```
org.n52.wps.server.algorithm.foss4g
```

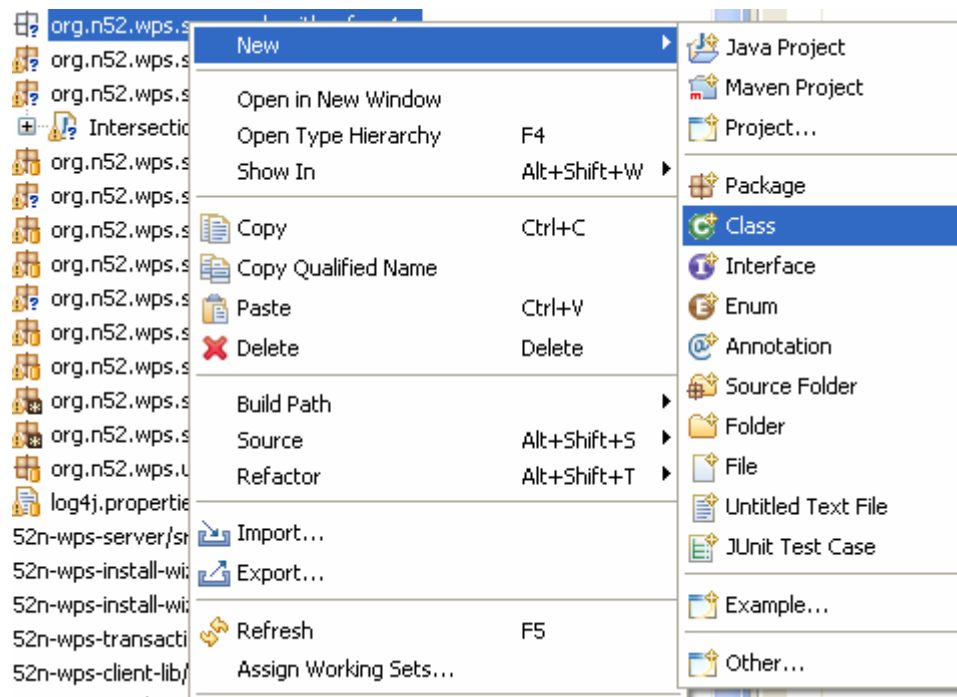
as the package name.



Click *Finish*

### 2.2.2 Create a new Class

Right click on the newly created `org.n52.wps.server.algorithm.foss4g` package and go to *New*→*Class*

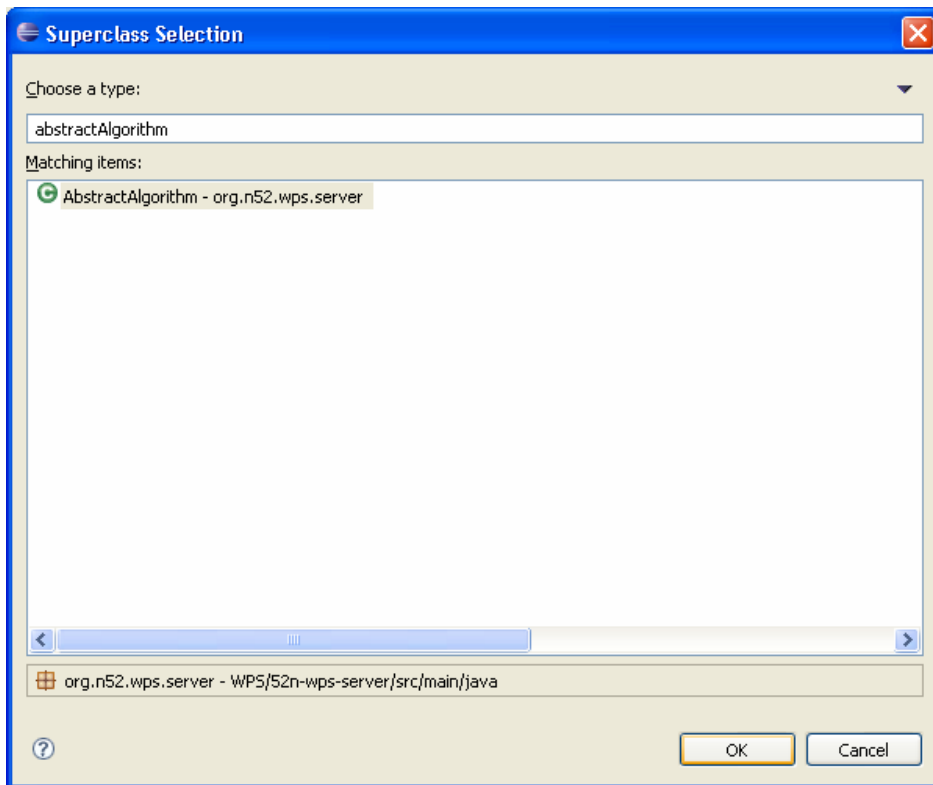


In the following dialog, enter

```
DouglasPeukerSimplificationAlgorithm
```

as the class name.

Click on the *Browse* button next to *Superclass* to select a superclass. Type `AbstractAlgorithm` and select the class as shown in following figure.



Click *OK*

Click *Finish*.

### 2.2.3 Code

Note that by inheriting from `AbstractAlgorithm`, you only have to deal with implementing the business logic in the `run` method and not with other things like loading the process description etc. .

As a preparation, copy the following import statements below the toplevel package declaration:

```
import java.util.HashMap;
import java.util.Map;

import org.geotools.feature.Feature;
import org.geotools.feature.FeatureCollection;
import org.geotools.feature.FeatureIterator;
import org.geotools.feature.IllegalAttributeException;
import org.n52.wps.server.AbstractAlgorithm;

import com.vividsolutions.jts.geom.Geometry;
import com.vividsolutions.jts.geom.LineString;
import com.vividsolutions.jts.geom.MultiLineString;
import com.vividsolutions.jts.geom.MultiPolygon;
import com.vividsolutions.jts.geom.Polygon;
import com.vividsolutions.jts.simplify.DouglasPeuckerSimplifier;
```

#### 2.2.3.1 Get the input data

The

```
public Map run(Map layers, Map parameters)
```

method has two parameters. The `layers` parameter provides parsed complex data. The `parameters` parameter supplies literal data. Since a `Map` datastructure is used, the actual data is accessible as key-value-pair tuples. The key is the input parameter name used in the `ProcessDescription` for a certain input. Therefore, you can obtain for instance input data, which is required by the `ProcessDescription` Document under the name "data" by

```
layers.get("Data")
```

Since parsed data is returned back, the generic result type (`java.lang.Object`) of this operation has to be casted to the datastructure used by the parser. For GML data, the parsers return an `org.geotools.feature.FeatureCollection`.

For our case, delete the generated contents of the `run` method and enter as the first two lines:

```
FeatureCollection collection =  
(FeatureCollection)layers.get("FEATURES");  
Double tolerance = (Double)parameters.get("TOLERANCE");
```

in the `run` method to get both expected input parameters.

### 2.2.3.2 Business Logic

The actual business logic implements the *DouglasPeucker* algorithm.

According to Wikipedia ([http://en.wikipedia.org/wiki/Ramer-Douglas-Peucker\\_algorithm](http://en.wikipedia.org/wiki/Ramer-Douglas-Peucker_algorithm)), the algorithm's idea is "that through a series of points or lines given, curve by omitting individual points easier. In doing so, the shape of the curve as such but maintained. It is not the number of targets are controlled, but there is a gap measure for the maximum distance between the starting points of the curve and the target curve.

The algorithm now produces an approximation curve consisting of a subset of the starting points of the curve. For the production of the curve, the output curve gradually divided into sections, then the algorithm as a separate task through. The target curve arises from the different approximated sections so that no starting point of the curve is further from the target curve away, as the predetermined distance dimension."

This idea can be applied to each feature. Therefore, we have to iterate through the previously obtained feature collection by typing:

```
FeatureIterator iter = collection.features();  
  
while(iter.hasNext()) {  
    Feature feature = iter.next();
```

Now, we can simplify the feature (after a simple check):

```
if(feature.getDefaultGeometry() == null) {  
    throw new NullPointerException("defaultGeometry is null in feature  
id: " + feature.getID());
```

```
}
```

and store the user data for further use:

```
Object userData = feature.getDefaultGeometry().getUserData();
```

Next, we have to extract the geometry because we want to perform the operation on the geometry.

```
try{
    Geometry in = feature.getDefaultGeometry();
```

Since we want to reuse existing code (and we are lazy developers ;-), we can use the `simplify` method on the `com.vividsolutions.jts.simplify.DouglasPeuckerSimplifier` class. Type in the next line:

```
Geometry out = DouglasPeuckerSimplifier.simplify(in, tolerance);
```

After this method call, we can update the feature with the new geometry (with special checks):

```
if(in.getGeometryType().equals("MultiPolygon") &&
out.getGeometryType().equals("Polygon"))
{
    MultiPolygon mp = (MultiPolygon)in;
    Polygon[] p = {(Polygon)out};
    mp = new MultiPolygon(p,mp.getFactory());
    feature.setDefaultGeometry(mp);
}
else if(in.getGeometryType().equals("MultiLineString") &&
out.getGeometryType().equals("LineString")) {
    MultiLineString ml = (MultiLineString)in;
    LineString[] l = {(LineString)out};
    ml = new MultiLineString(l,ml.getFactory());
    feature.setDefaultGeometry(ml);
}
else {
    feature.setDefaultGeometry(out);
}
feature.getDefaultGeometry().setUserData(userData);
}
catch(IllegalArgumentException e) {
    throw new RuntimeException("geometrytype of result is not matching",
e);
}
```

And close the loop

```
}
```

### 2.2.3.3 Return data back

In the last step, the updated features have to be returned back to allow a Generator to create the requested encoding.

Thus, a new `HashMap` has to be created:

```
HashMap<String, FeatureCollection> result = new HashMap<String,
FeatureCollection>();
```

And the updated `FeatureCollection` collection has to be dropped in there. The key `"SIMPLIFIED_FEATURES"` is determined by the `ProcessDescription` output parameter id (See next section).

```
result.put("SIMPLIFIED_FEATURES", collection);
return result;
```

Press Ctrl+S to save the file.

**Hint:** If something did not work out, you can find the complete code here.

```
package org.n52.wps.server.algorithm.foss4g;

import java.util.HashMap;
import java.util.Map;

import org.geotools.feature.Feature;
import org.geotools.feature.FeatureCollection;
import org.geotools.feature.FeatureIterator;
import org.geotools.feature.IllegalAttributeException;
import org.n52.wps.server.AbstractAlgorithm;

import com.vividsolutions.jts.geom.Geometry;
import com.vividsolutions.jts.geom.LineString;
import com.vividsolutions.jts.geom.MultiLineString;
import com.vividsolutions.jts.geom.MultiPolygon;
import com.vividsolutions.jts.geom.Polygon;
import com.vividsolutions.jts.simplify.DouglasPeuckerSimplifier;

public class DouglPeukerSimplificationAlgorithm extends AbstractAlgorithm
{

    public String getErrors() {
        // TODO Auto-generated method stub
        return null;
    }

    public Map run(Map layers, Map parameters) {
        FeatureCollection collection =
        (FeatureCollection)layers.get("FEATURES");
        Double tolerance= (Double)parameters.get("TOLERANCE");

        FeatureIterator iter = collection.features();
        while(iter.hasNext()) {
            Feature feature = iter.next();
            if(feature.getDefaultGeometry() == null)
            {
                throw new
                NullPointerException("defaultGeometry is null in feature id: " +
                feature.getID());
            }
            Object userData =
            feature.getDefaultGeometry().getUserData();
            try{
                Geometry in =
                feature.getDefaultGeometry();
```

```

        Geometry out =
DouglasPeuckerSimplifier.simplify(in, tolerance);

        if(in.getGeometryType().equals("MultiPolygon") &&
out.getGeometryType().equals("Polygon"))
        {
            MultiPolygon mp = (MultiPolygon)in;
            Polygon[] p = {(Polygon)out};
            mp = new
MultiPolygon(p,mp.getFactory());
            feature.setDefaultGeometry(mp);
        }
        else
        if(in.getGeometryType().equals("MultiLineString") &&
out.getGeometryType().equals("LineString")) {
            MultiLineString ml =
(MultiLineString)in;
            LineString[] l = {(LineString)out};
            ml = new
MultiLineString(l,ml.getFactory());
            feature.setDefaultGeometry(ml);
        }
        else {
            feature.setDefaultGeometry(out);
        }

feature.getDefaultGeometry().setUserData(userData);
    }
    catch(IllegalArgumentException e) {
        throw new
RuntimeException("geometrytype of result is not matching", e);
    }
}
HashMap<String, FeatureCollection> result = new
HashMap<String, FeatureCollection>();

result.put("SIMPLIFIED_FEATURES", collection);
return result;

    }

}

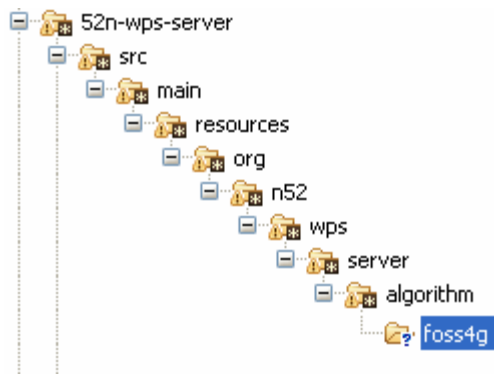
```

#### 2.2.4 ProcessDescription

Every Process needs a ProcessDescription which will be delivered via the *getProcessDescription* operation.

The ProcessDescription has to be created manually. The 52n WPS follows the convention that an XML ProcessDescription file should be found under 52n-wps-server/main/resources/<path to class>





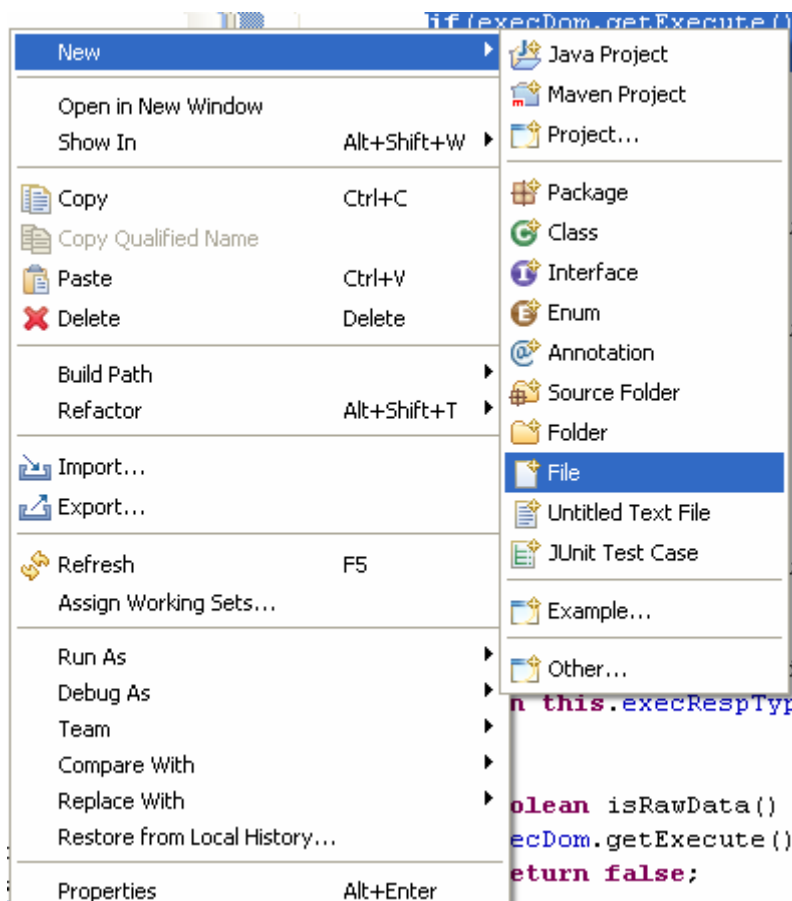
in our case we have to create the folder

*foss4g*

under

*52n-wps-server/src/main/resources/org/n52/wps/server/algorithm*

Inside this folder, we create a new file



by using the right-mouse click inside the folder. Go to *New* → *File*.

Label the file with the same name as our implemented algorithm:

*DouglasPeukerSimplificationAlgorithm.xml*

Double click on the created file to open it.

Click on the *source* tab in the lower left corner of that window.

Copy the following XML into that file and save it with CTRL+S.

```
<?xml version="1.0" encoding="UTF-8"?>
<!--This example describes a buffer command that accepts polygon
coordinates in GML, and used a buffer distance in meters to produce a
buffered polygon feature, which is output in GML, in either UTF-8 or base64
encoding. The polygon can be returned directly as output, or stored by the
service as a web-accessible resource. Ongoing processing status reports
are not available. -->
<wps:ProcessDescriptions xmlns:wps="http://www.opengis.net/wps/1.0.0"
xmlns:ows="http://www.opengis.net/ows/1.1"
xmlns:xlink="http://www.w3.org/1999/xlink"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.opengis.net/wps/1.0.0
http://geoserver.itc.nl:8080/wps/schemas/wps/1.0.0/wpsDescribeProcess_respo
nse.xsd" service="WPS" version="1.0.0" xml:lang="en-US">
  <ProcessDescription wps:processVersion="2" storeSupported="true"
statusSupported="false">

    <ows:Identifier>org.n52.wps.server.algorithm.foss4g.DouglasPeuckerSim
plifyAlgorithm</ows:Identifier>
    <ows:Title>DouglasPeuckerSimplifyAlgorithm</ows:Title>
    <ows:Abstract>Uses JTS implementation. Does not support
topological awareness</ows:Abstract>
    <ows:Metadata xlink:title="spatial" />
    <ows:Metadata xlink:title="geometry" />
    <ows:Metadata xlink:title="douglas peucker" />
    <ows:Metadata xlink:title="GML" />
    <DataInputs>
      <Input minOccurs="1" maxOccurs="1">
        <ows:Identifier>FEATURES</ows:Identifier>
        <ows:Title>input features</ows:Title>
        <ows:Abstract>Just features</ows:Abstract>
        <ComplexData>
          <Default>
            <Format>
              <MimeType>text/XML</MimeType>

            <Schema>http://schemas.opengis.net/gml/2.1.2/feature.xsd</Schema>
            </Format>
          </Default>
          <Supported>
            <Format>
              <MimeType>text/XML</MimeType>

            <Schema>http://geoserver.itc.nl:8080/wps/schemas/gml/2.1.2/gmlpacket.
xsd</Schema>
            </Format>
          </Supported>
        </ComplexData>
      </Input>
      <Input minOccurs="1" maxOccurs="1">
        <ows:Identifier>TOLERANCE</ows:Identifier>
        <ows:Title>Tolerance Value for DP Alg</ows:Title>
        <ows:Abstract></ows:Abstract>
        <LiteralData>
```

```

        <ows:DataType
ows:reference="xs:double"></ows:DataType>
        <ows:AllowedValues>
            <ows:Value></ows:Value>
        </ows:AllowedValues>
        </LiteralData>
    </Input>
</DataInputs>
<ProcessOutputs>
    <Output>

        <ows:Identifier>SIMPLIFIED_FEATURES</ows:Identifier>
        <ows:Title>smooth geometries</ows:Title>
        <ows:Abstract>GML stream describing the smooth
feature.</ows:Abstract>
        <ComplexOutput>
        <Default>
        <Format>
            <MimeType>text/XML</MimeType>

        <Schema>http://schemas.opengis.net/gml/2.1.2/feature.xsd</Schema>
        </Format>
        </Default>
        <Supported>
            <Format>
                <MimeType>text/XML</MimeType>

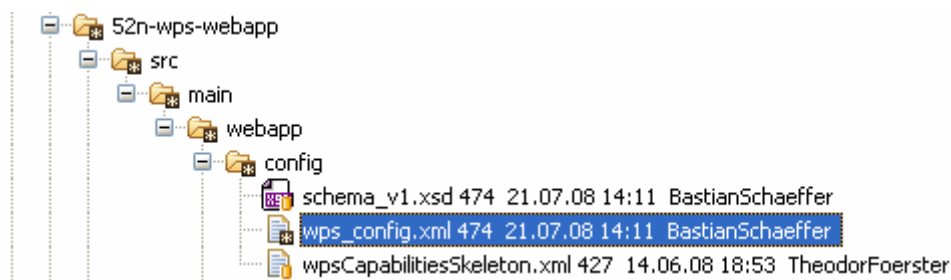
        <Schema>http://geoserver.itc.nl:8080/wps/schemas/gml/2.1.2/gmlpacket.
xsd</Schema>
            </Format>
        <Format>
            <MimeType>application/vnd.google-
earth.kml+xml</MimeType>

        <Schema>http://www.opengis.net/kml/2.2</Schema>
        </Format>
        </Supported>
        </ComplexOutput>
    </Output>
</ProcessOutputs>
</ProcessDescription>
</wps:ProcessDescriptions>

```

### 2.2.5 Configuration

The WPS has to be informed about the newly available process. This can be easily done by editing the `config_wps.xml` file, which is located here:



Double click on the file.

Add the element

```
<Property  
name="Algorithm">org.n52.wps.server.algorithm.foss4g.DouglasPeu  
kerSimplificationAlgorithm</Property>
```

As a child element to

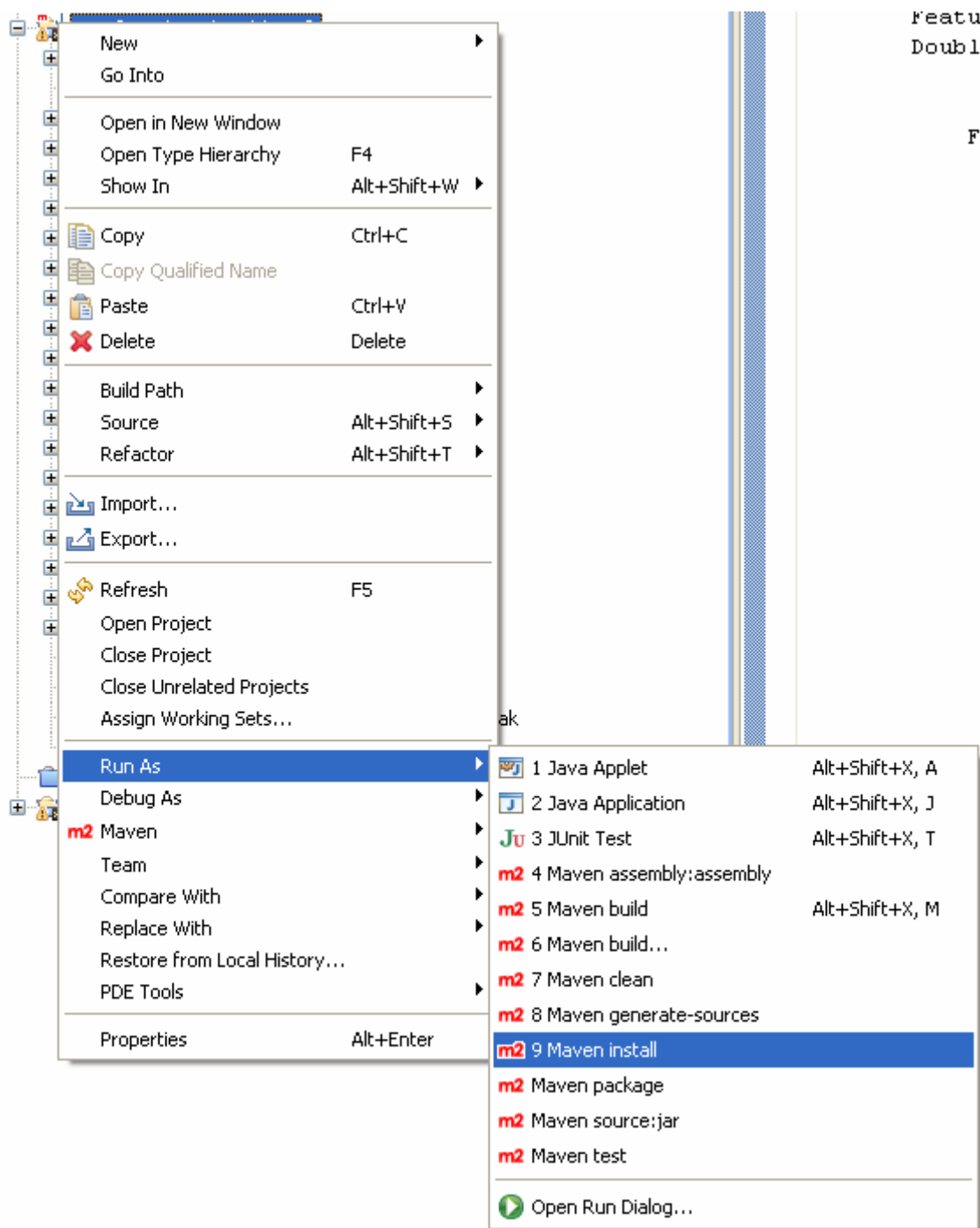
```
<AlgorithmRepositoryList>  
  <Repository name="LocalAlgorithmRepository"  
    className="org.n52.wps.server.LocalAlgorithmRepository">
```

Note that the identifier is the fully qualified class name.


Save the file by pressing CTRL+S

### 2.2.6 Compile again

Right mouse click on the WPS Project. Go to *Run as* → *Maven install*



## 2.2.7 Restart Tomcat

Just click on this  Symbol in the toolbar. (If not running start it again).

Check if the new process exists by performing a simple get Capabilities request:

<http://localhost:8080/wps/WebProcessingService?Request=GetCapabilities&Service=WPS>

You should find something similar to:

```
<wps:Process wps:processVersion="2">
<ows:Identifier>
```

```
        org.n52.wps.server.algorithm.foss4g.DouglasPeuckerSimplificatio
        nAlgorithm
    </ows:Identifier>
    <ows:Title>DouglasPeuckerSimplifyAlgorithm</ows:Title>
</wps:Process>
```

### 3 Execute the implemented process

After successfully implementing a new algorithm, we are now able to execute this algorithm.

This section guides you through the process of executing the implemented algorithm with the help of the user friendly desktop GIS (uDig).

#### 3.1 Setup uDig WPS Client

Download the 52°North uDig WPS client ([org.n52.wps.client.udig 1.2.0.jar](http://org.n52.wps.client.udig.1.2.0.jar)) and drop it into the `C:\FOSS4G2008\udig\1.1-RC14\eclipse\plugins` directory

Start uDig  `udig.exe` from the desktop.

#### 3.2 Add Data

First, we need some data to process.

Go to *Layers* → *Add...* → *Web Feature Server*

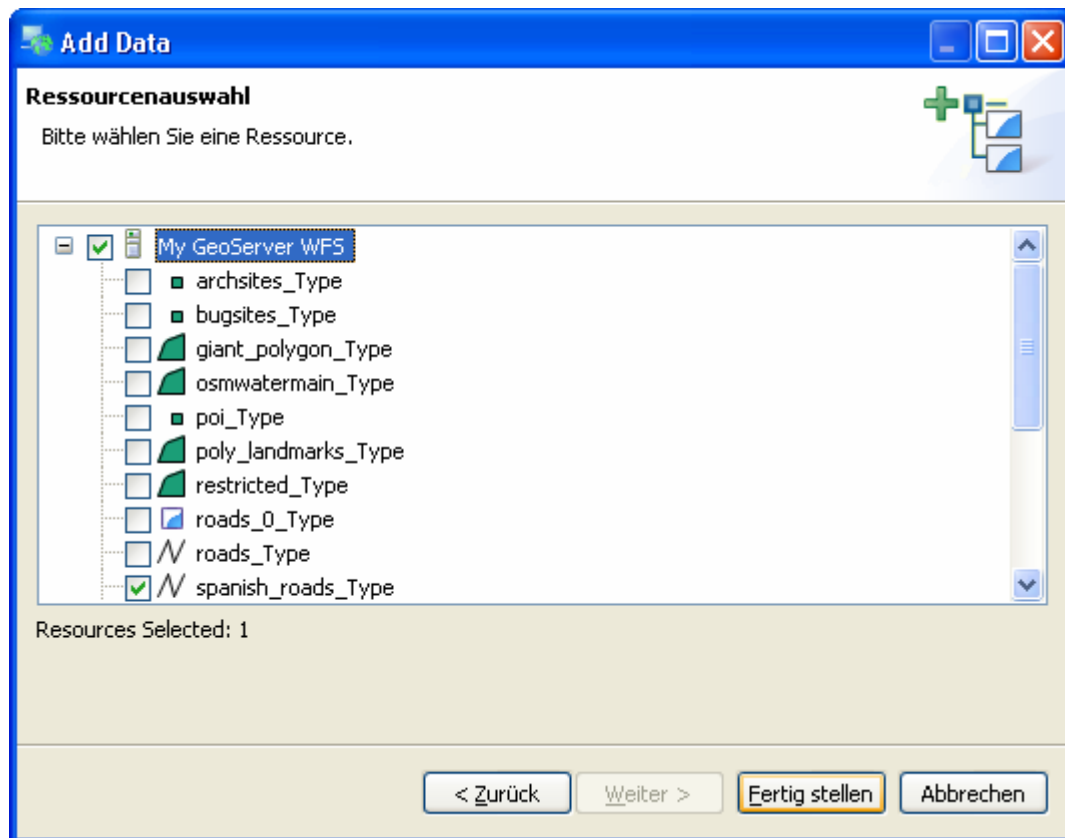
Click *Next*

Enter <http://geoserver.itc.nl:8080/geoserver/wfs> as the URL

Click *Next*

Select the *spanish\_roads\_type* layer

Click *Finish*

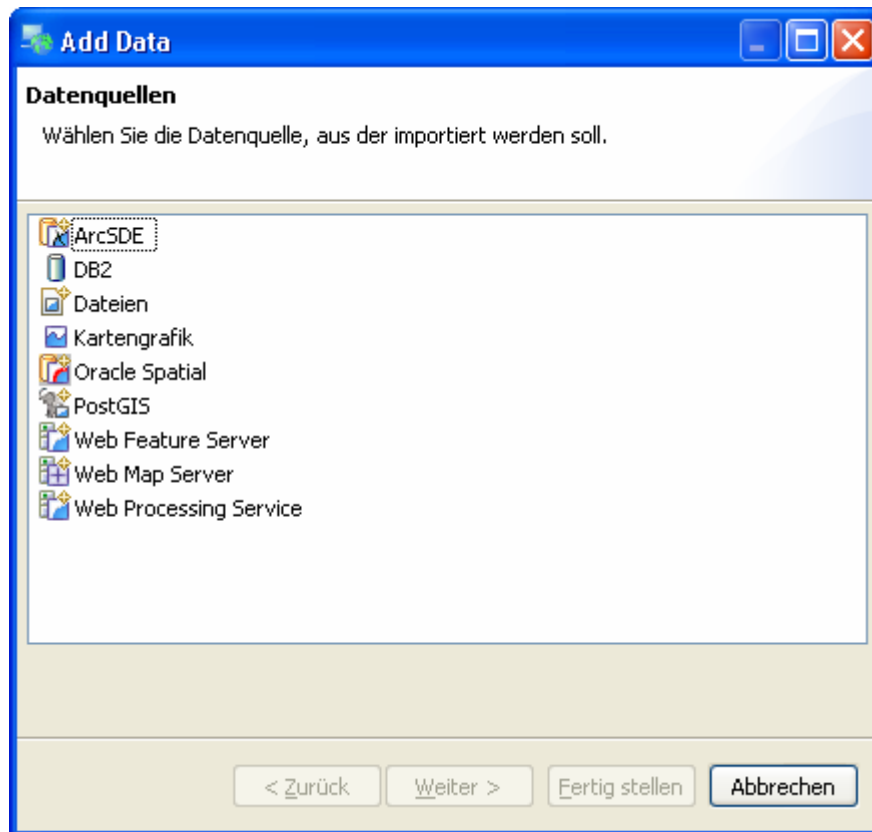


### 3.3 Execute WPS Process

Now we are ready to execute our implemented process with the recently added data.

**Note:** Make sure that tomcat is running

*Go to Layer→Add...→ Web Processing Service*



Click *Next*

Enter the URL of your WPS (<http://localhost:8080/wps/WebProcessingService>)  
(Theoretically you could use the WPS of your neighbour-but it is configured as "localhost" therefore only accessible via localhost)

Click *Next*

Select your `org.n52.wps.algorithm.foss4g.DouglasPeuckerSimplificationAlgorithm` process

Note the input and output description on the right side.

Click *Next*

Select the previously added *spanish\_roads\_type* layer as *input features*.



**Add Data**

**Web Processing Service**  
Set the Input values for the previously selected process

input features:

Tolerance Value for DP Alg:

< Zurück Weiter > Fertig stellen Abbrechen

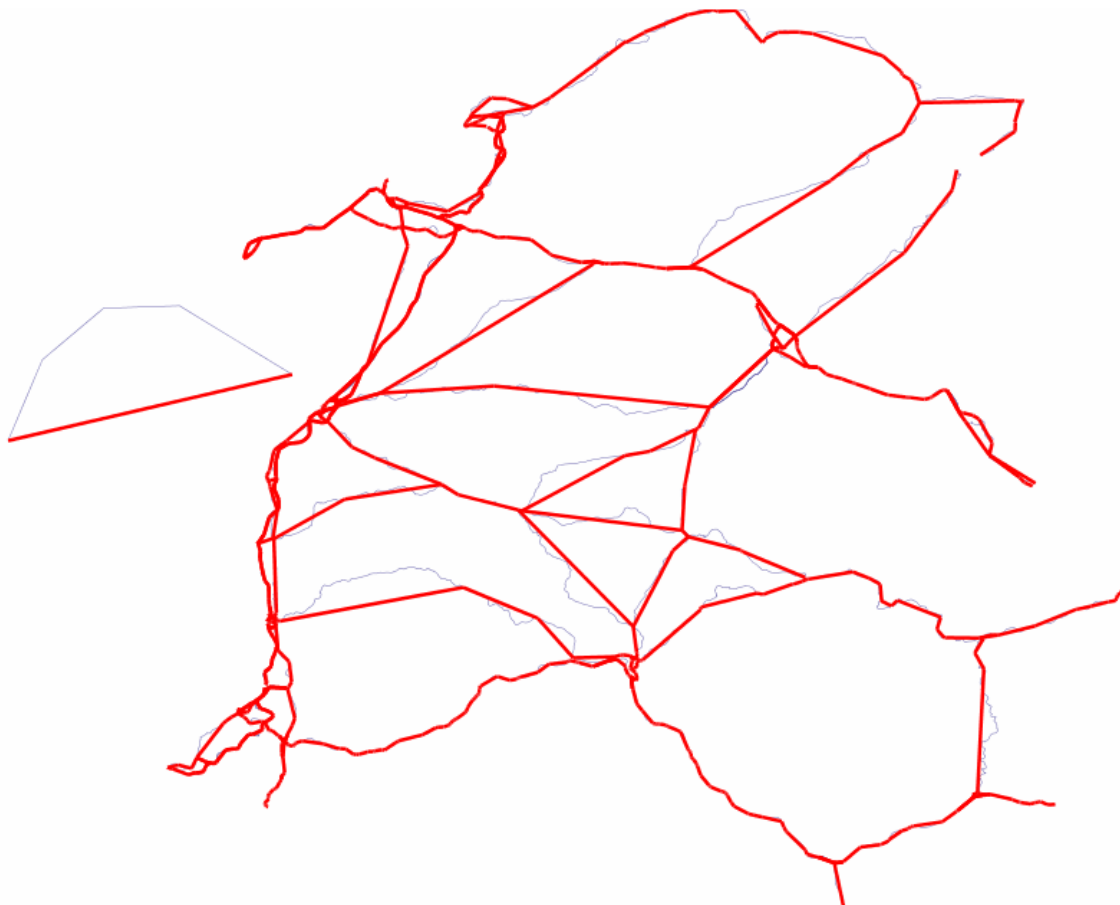
Enter 1 as the *Tolerance Value for DP Alg*

Click *Next*

Click *Finish*

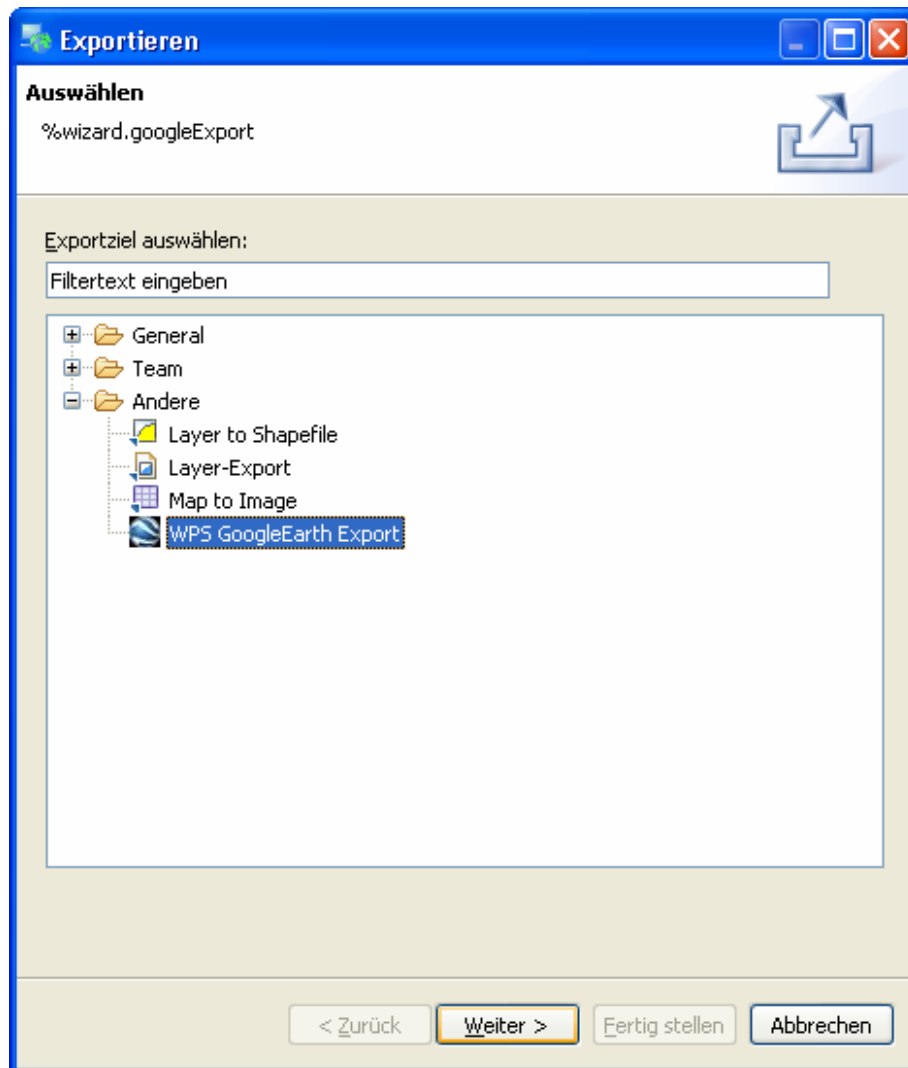
After the process finishes, you should notice a new layer *smooth geometries* in your layerlist.

The result should look similar to this:



## 4 Export Process to Google Earth

Right mouse click on the newly added WPS Process Result layer (*smooth geometries*) → *export* → *WPS Google Earth Export*

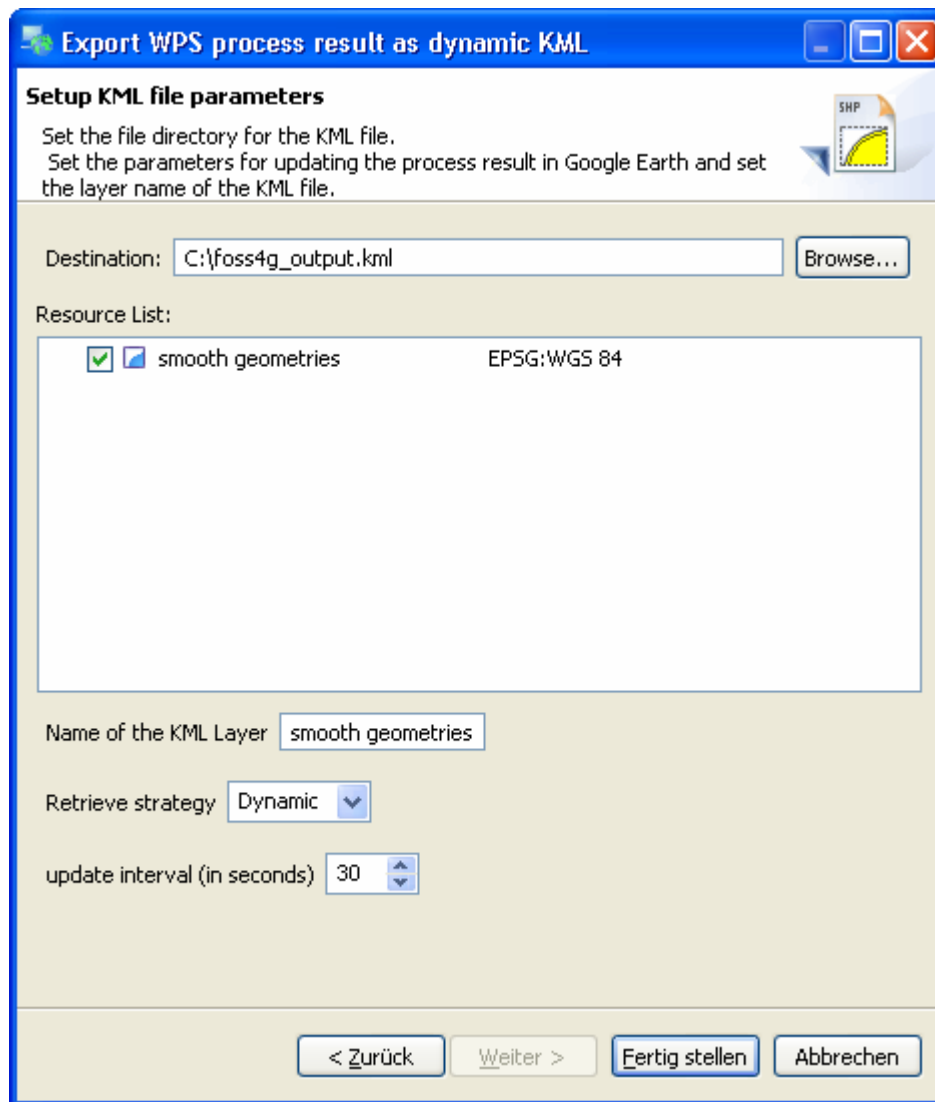


Click *Next*

Enter *C:\foss4g\_output.kml* in the Destination field.

Select *Dynamic* as the retrieve Strategy

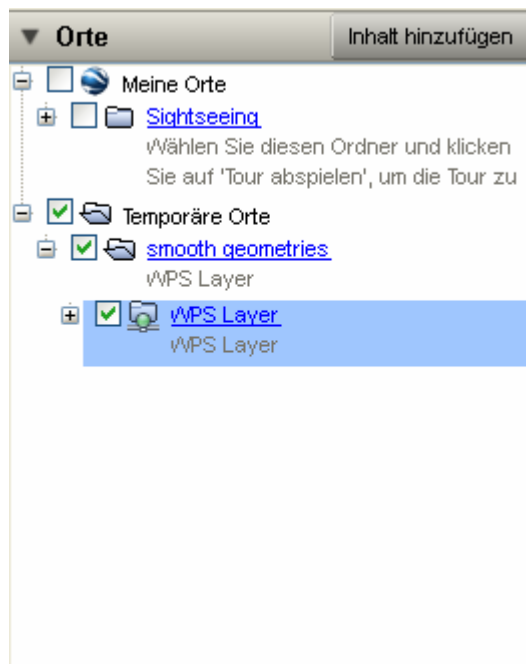
Type in *120* as the update interval



Click *Finish*

Go to your C: Drive and double click the created *foss4g\_output.kml* file.

Select in the *Temporary Place Box* your new layer



Note that Google Earth will request your WPS every 30 second for the latest results.



More at:

<http://www.52north.org/wps>

or contact Bastian Schaeffer

`schaeffer@52north.org`