HOW TO COPE WITH GEOSPATIAL

The Pragmaic Intro for Java Developers

28 September 2007

GeoTools
The open source Java GIS toolkit

Refractions
RESEARCH
THE GEOSPATIAL EXPERTS

TABLE OF CONTENTS

1 GOALS

After completing this workbook, you will have:

- Installed a Java Development Kit onto your computer
- Installed GeoServer from the Windows installer
- Started up and Customize GeoServer
- Started the Open Layers WFS-T Demonstrate

2 WELCOME

Are you new to GeoSpatial? Are you not cool enough to be a Neo-Geographer AJAX empowered meta tagging Ruby wonder kid? Does scientific mumbo-jumbo make your head hurt? Are you (gasp!) just out to get the job done?

This workbook offers a survey of the Java GIS landscape; if you are new to the GeoSpatial scene it offers an introduction to current concepts and projects, and how to avoid common pitfalls.

We will start with something nice, fun and visual - practicing fetching content from Web Map Servers on the Internet using the GeoTools toolkit. We can talk about what is going on, but the focus is on you and the code you need to get the job done.

Moving on, we will explore what maps are made of, sugar and spice and all things nice? Would you believe they are made of Features (literally things you can draw on a Map), Geometry (what to actually draw) and details like coordinate reference systems, units and projections.

The good news is all this stuff is captured at the Java level as nice normal objects by the GeoTools and Java Topology Suite projects. There are utility classes around so we can avoid going down into crazy scientific detail.

We will work with a couple of common GeoSpatial data formats and show how to make queries and modify information.

On the visualization side of things we will make use of one of the available rendering systems and do so with Style. Well, we can show you how to use a Style Layer Descriptor document and then hack apart the result to see what makes it tick.

3 INTRODUCTION AND SETUP

Welcome FOSS5G Lab participants – your computer has already been set up with Eclipse and the required source code.

There are many really good Java toolkits and libraries out there. We are going to focus on the GeoAPI, JTS and GeoTools libraries.

- GeoAPI contains interfaces (and javadocs) for many of the concepts in the geospatial world. These interfaces are informed by OGC and/or ISO specifications if applicable.
- JTS Topology Suite is an implementation of Geometry (ie Point, Lines and Polygons) that matches up with the Simple Feature for SQL specification. You will find JTS code has been ported to C++ and .NET.
- GeoTools is an implementation of everything else from referencing (so we know where the shapes are located) to data access and rendering.

3.1 PROJECT FOR REQUIRED JARS

In this section we are going to create a new project to hold all the library jars, other projects in our workspace are going to depend on this project.

You will find that many of the wiki instructions make use of a tool called Maven. Maven is an apache build tool like ANT or MAKE that makes that downloads jars as needed.

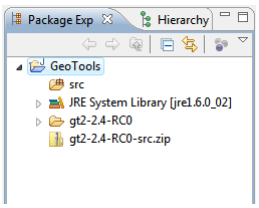
1. The first task is to get the libraries ready to go in our IDE. We are using Eclipse (http://www.eclipse.org/downloads/) for our example but the same principles apply for NetBeans (or even ANT).
2. Download the latest stable release of GeoTools from http://docs.codehaus.org/display/GEOTOOLS/Downloads
3. For this Workbook I used the following two files
   - **gt2-2.4-RC0-bin.zip**
     All the jars needed for GeoTools
   - **gt2-2.4-RC0-src.zip**
     The source code so you can step through in the debugger
4. Start up Eclipse
5. Create a new Java Project using **File > New > Java Project** in the menu bar.
6. Project name: GeoTools

7. Press **Finish**

8. We can now place our GeoTools downloads into this project:
   Unzip the **gt2-2.4-RC0-bin.zip** file to produce a new folder:
   gt2-2.4-RC0

7-Zip is a good choice when unzipping files on windows.

9. Just drag the **gt2-2.4-RC0-src.zip** file into your eclipse project.
   Here is what it looks like when you are done



10. If you open up your bin folder you can count 115 jars – that is a lot of code! We are going to remove jars that we are not going to use in the workbook.

We only need one "epsg" plugin and epsg-hsql is really good.

11. We only need one epsg plugin – please REMOVE:
    gt2-epsg-access-2.4-RC0.jar
    gt2-epsg-postgresql-2.4-RC0.jar
    gt2-epsg-wkt-2.4-RC0.jar

12. Some of the database require you to supply a driver; for now please REMOVE:
    gt2-arcsde-2.4-RC0.jar
    gt2-db2-2.4-RC0.jar
    gt2-oracle-spatial-2.4-RCO.jar

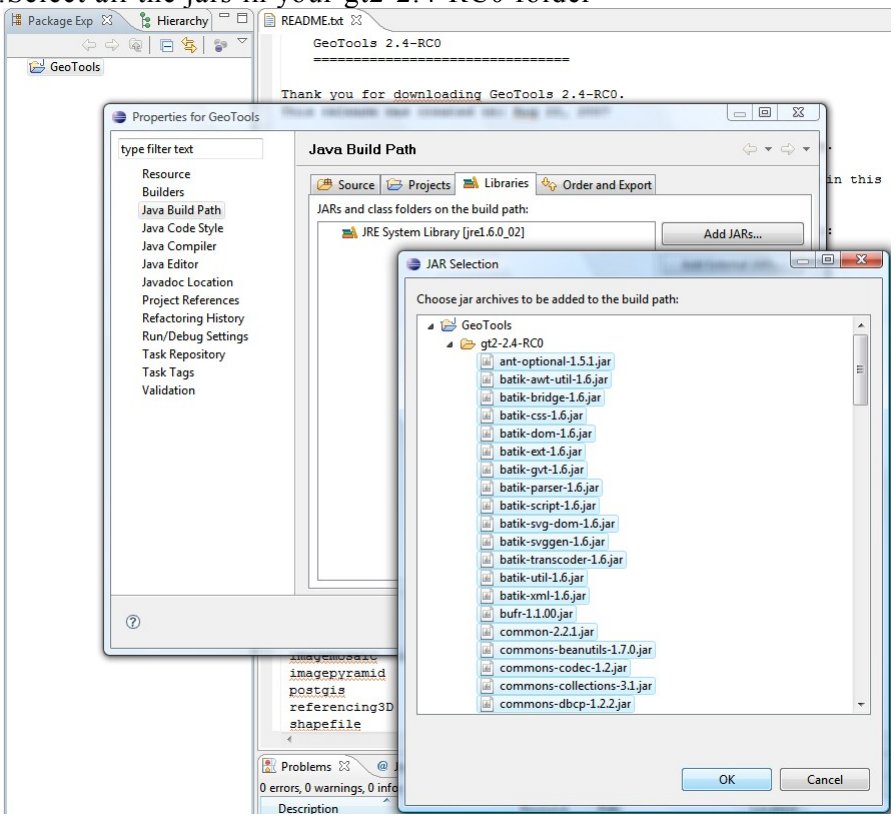
13. Now that we have a good list of jars we can add them to our Java Build path.
    Right click on the project and select properties

14. Choose **Java Build Path** in the list property pages.

15. Select the **Libraries** tab

16. Press the **Add JARs** button

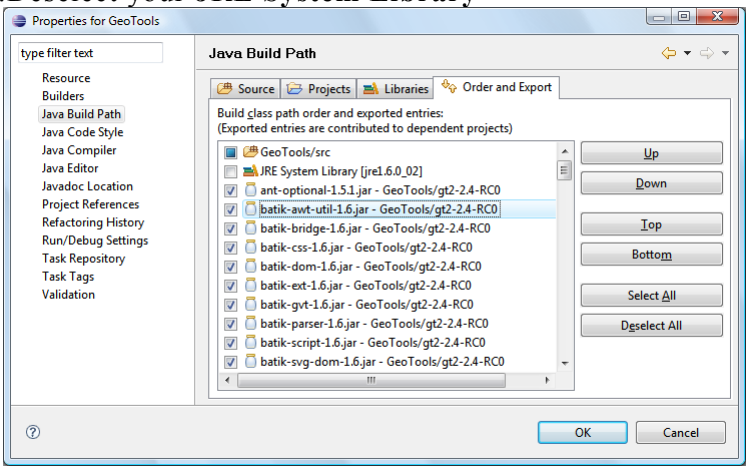
17. Select all the jars in your gt2-2.4-RC0 folder



18. Switch to the **Order and Export** tab

19. Press **Select All**

20. Deselect your **JRE System Library**

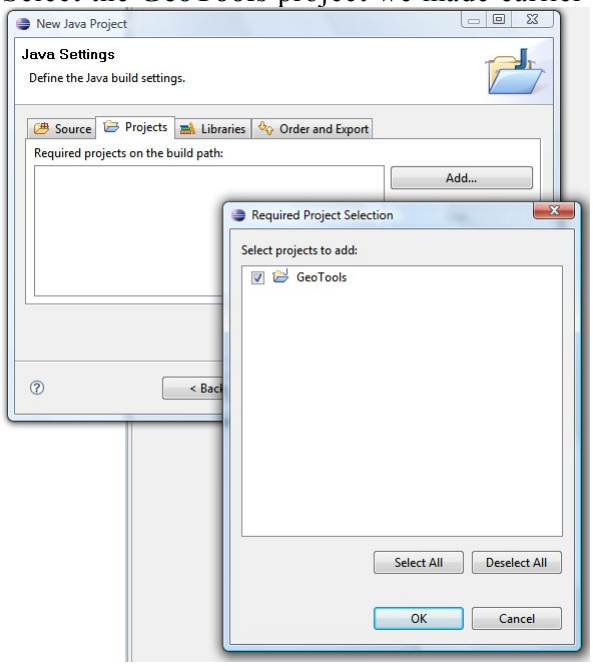


21. And press **OK**

3.2 PROJECT FOR THIS WORKBOOK

We can now create additional projects that depend on our GeoTools project.

1. From the file menu create a **New Java Project**
2. Project name: **Example**
3. Press **Next**
4. Switch to the **Projects** tab and press **Add...**
5. Select the **GeoTools** project we made earlier



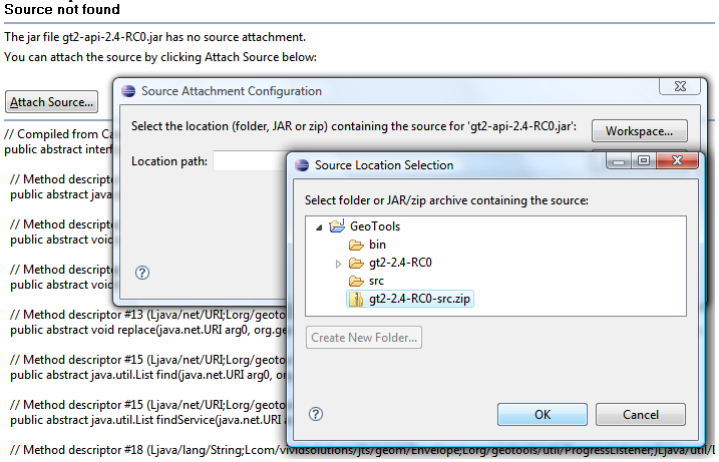
6. And press **OK**
7. From the file menu **File > New > Class**
8. Package: **org.geotools.demo**
9. Name: **WMSLab**
10. All the files are available on this page: http://svn.geotools.org/geotools/branches/2.4.x/demo/example/src/main/java/org/geotools/demo/
11. Select the WMSLab.java link to show the code we are going to start with
12. Select All and Copy
13. Return to eclipse and press Paste
14. You can repeat this process for each section of the workbook

3.3 JAR TO SOURCE ASSOCIATIONS

The first time you go to look at any of the GeoTools classes you will need to tell Eclipse where the source code for that jar is.

1. Open up a GeoTools class by typing Control-Shift-T this will bring up the Type finder
2. Enter the class name GeoTools (A class that records the current version number)

1. From the "Source not found" editor press the Attach Source... Button.
2. Press the Workspace button
3. Select the gt2-2.4-RC0-src.zip
4. And press OK



3.3.1 Alternative for Power Eclipse Users

If you are interested in the details, the above steps made a change to your .classpath file:

```
<classpathentry exported="true" kind="lib"
                path="gt2-2.4-RC0/gt2-api-2.4-RC0.jar"
                sourcepath="gt2-2.4-RC0-src.zip"/>
```

You can quickly cut and paste sourcepath="gt2-2.4-RC0-src.zip for the rest of the gt2 jars.

4 WMS LAB

We are going to start in nice pragmatic fashion by cheating; rather than draw a map of our own we are going to talk to a server and ask it to do the hard work.

We are starting with the outline of a class available here WMSLab.java.

4.1 CONNECT TO A WMS

1. To start please open the **WMSLab** file and add the following to your main method.

```
public static void main(String[] args) throws Exception {
    URL server = getServerURL(args);
    WebMapServer wms;

    System.out.println("Connecting to " + server);
    wms = new WebMapServer(server);

    System.out.println("Welcome");
    WMSLab wmsLab = new WMSLab(wms);
    wmsLab.setVisible(true);
}
```

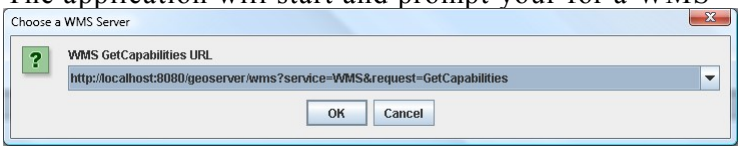
2. That is right connecting to a WMS is as simple as passing the URL to the service. Have a look at the getServerURL method to see some example servers.
3. The first thing we are going to do is have a look a at the "capabilities" of the WMS. The capabilities bean describe the operations the server can do and the service itself.
4. To obtain the title from the capabilities fill in the following getWMSTitle method

```
public String getWMSTitle(WebMapServer wms) {
    WMSCapabilities capabilities = wms.getCapabilities();
    Service service = capabilities.getService();
    String title = service.getTitle();
    return title;
}
```

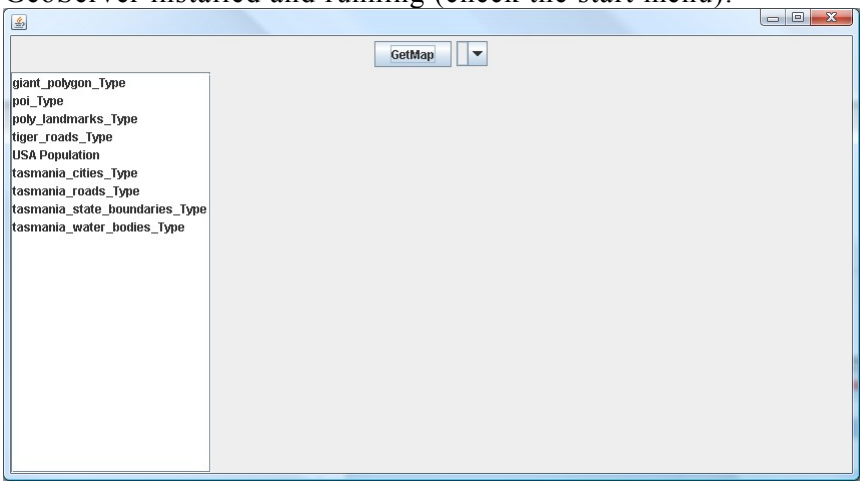
This workbook assumes you are running the default GeoServer installation

5. We are going to run the application just so you can see us actually connecting to the a WMS. If you get a chance to install GeoServer on your machine you will find this a nice quick way to test functionality. We have listed several other recommended servers in the source code.

6. Select WMSLab in the Package Explorer
7. Press the Run Button
8. The application will start and prompt your for a WMS



9. Please select the default local geoserver shown above if you have GeoServer installed and running (check the start menu).



10. Congratulations, now let's start getting graphical.

4.2 OPERATIONS ON A LAYER

1. Please cut and past the following into getLegendGraphics

```
public Icon getLegendGraphics(Layer layer) {
    Icon icon = null;
    WMSCapabilities capabilities = wms.getCapabilities();
    WMSRequest request = capabilities.getRequest();
    OperationType description =
                          request.getGetLegendGraphic();
    if (description == null) return null;

    GetLegendGraphicRequest legendGraphicsRequest = wms
            .createGetLegendGraphicRequest();
    legendGraphicsRequest.setLayer(layer.getName());
    legendGraphicsRequest.setStyle(getNamedStyle(layer));
    legendGraphicsRequest.setFormat((String)
description.getFormats()
            .iterator().next());
    URL url = legendGraphicsRequest.getFinalURL();
    return new ImageIcon(url);
}
```

2. This time we are using the WMSRequest bean to obtain information about the getLegendGraphic OperationType. Not all WMS servers support this operation so we are careful to check if the value is null.

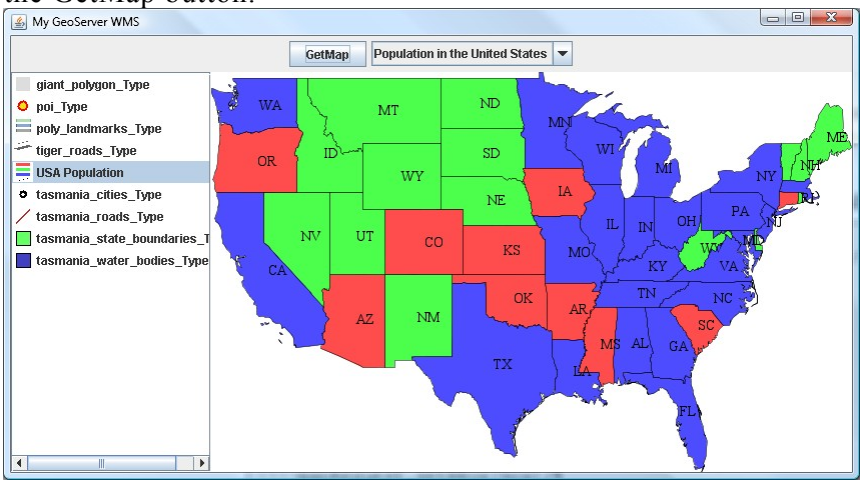
3. We use the WebMapServce class to create a GetLegendGraphicRequest. We will our new request with the details about the layer, style and format and ask it for the final url we can use to obtain the image.

4. We are going to hook the getMap method up in a similar fashion.

```
private void getMap(Layer layer) throws Exception {
    GetMapRequest mapRequest = wms.createGetMapRequest();
    String style = getNamedStyle( layer );
    mapRequest.addLayer(layer, style);

    mapRequest.setFormat(getImageFormat(wms));
    CRSEnvelope box = getCRSEnvelope(layer, null);
    if (box == null) {
        box = layer.getLatLonBoundingBox();
        box.setEPSGCode("EPSG:4326");
    }
    mapRequest.setSRS(box.getEPSGCode());
    mapRequest.setBBox(box);
    mapRequest.setDimensions(
         panel.getWidth(), panel.getHeight());
    URL url = mapRequest.getFinalURL();
    ImageIcon load = new ImageIcon(url);
    image = load.getImage();
    panel.repaint();
}
```

5. Now when you run the program you can select a layer and press the GetMap button.



6. We have an example of fetching several layers at once, change the getMap() callback method to the following and have a look.

```
public void getMap() {
    try {
        Object selection[] = layers.getSelectedValues();
        List layerList = Arrays.asList(selection);
        getMap(layerList);
    } catch (Exception e1) {
        image = null;
    }
}
```

5 CSV TO SHAPE

On the other extreme we have the defacto file format of spatial data the ESRI shapefile. In this example we are going to create a shapefile from a really simple comma separated value file just to get use to the idea.

Don't tell the Open Street Map project but this is just a guess of where London is.

1. To start with let's create a really simple CSV file in Notepad. Please create a new file **landmarks.csv** file with the following contents.

```
"Latitude","Longitude","Name"
-123.31,48.4,Victoria
0,52,London
```

2. To start we will grab our **csv** file from the user and create the schema for our new shapefile (like the file header). Add the following to **main**

```
File file = getCSVFile(args);
FeatureType type = DataUtilities.createType("Landmarks",
                "location:Point,name:String");
```

3. Now we can read our csv file in line by line and add features one by one to a FeatureCollection we have in memory.

```
FeatureCollection collection =
FeatureCollections.newCollection();
BufferedReader reader = new BufferedReader(new
FileReader(file));
GeometryFactory geometryFactory = new GeometryFactory();
try {
    String line = reader.readLine();
    System.out.println("Header: " + line);
    for (line = reader.readLine(); line != null; line =
reader
            .readLine()) {
        String split[] = line.split("\\,");
        double longitude = Double.parseDouble(split[0]);
        double latitude = Double.parseDouble(split[1]);
        String name = split[2];
        Coordinate coordinate = new Coordinate(longitude,
latitude);
        Point point =
geometryFactory.createPoint(coordinate);
        Feature feature = type.create(new Object[] { point,
name });
        collection.add(feature);
    }
} finally {
    reader.close();
}
```

4. The important part of the above example is the use of GeometryFactory to create a Point. This is the famed JTS Geometry classes which you can use to do all kinds of hands on spatial operations.

5. Now that we have the some Features we can create a new shapefile.

```
File newFile = getNewShapeFile(file);

DataStoreFactorySpi factory = new
ShapefileDataStoreFactory();

Map create = new HashMap();
create.put("url", newFile.toURI().toURL());
create.put("create spatial index", Boolean.TRUE);

ShapefileDataStore newDataStore = (ShapefileDataStore)
    factory.createNewDataStore(create);
newDataStore.createSchema(type);
newDataStore.forceSchemaCRS(DefaultGeographicCRS.WGS84);
```

6. Finally we can use a transaction to write the contents to disk. The FeatureStore interface contains methods to add, modify and remove features.

```
Transaction transaction = new DefaultTransaction("create");
String typeName = newDataStore.getTypeNames()[0];
FeatureStore featureStore = (FeatureStore) newDataStore
        .getFeatureSource(typeName);
featureStore.setTransaction(transaction);
try {
    featureStore.addFeatures(collection);
    transaction.commit();
} catch (Exception problem) {
    problem.printStackTrace();
    transaction.rollback();
} finally {
    transaction.close();
}
```

7. Even for simple file formats like shape the GeoTools library has gone to the trouble introduce the concept of Transactions (complete with rollback).

8. If you are finished try loading your shape file with the following tools:

    - uDig
    - QGis

These applications are installed on your lab computer. If you are working at home they are a quick download away.

6 POSTGIS LAB

Moving on from shapefiles – in this lab we bring out the big guns a real spatial database. If you you are working in an enterprise that has Oracle, DB2 or even (gasp!) ArcSDE you can use the lessons in this lab to connect to your existing infrastructure.

1. In the last example you saw us create a shape file in a pretty odd fashion (using a Map? With a URL?) and you probably thought we were crazy.
   Hopefully this make the practice a little more clear – update your **main** method to contain the following:

```
Map connectionProperties = getConnectionProperties(args);
DataStore dataStore =
      DataStoreFinder.getDataStore( connectionProperties );
```

2. We are going to ask the user for connection information; and we are going to look up on the classpath for the "best" implementation. If you write your application this way once – your user can work with ArcSDE, DB2, Oracle, PostGIS and Shapefile.

3. We are just going to do a quick sanity check to make sure we connected.

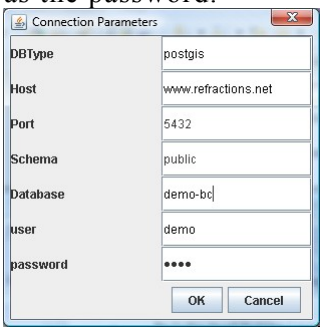
```
String[] typeNames = dataStore.getTypeNames();
if (typeNames == null) {
    JoptionPane.showConfirmDialog(null,
         "Could not conntect");
    System.exit(0);
}
```

4. And then start up a nice window to work with your DataStore.

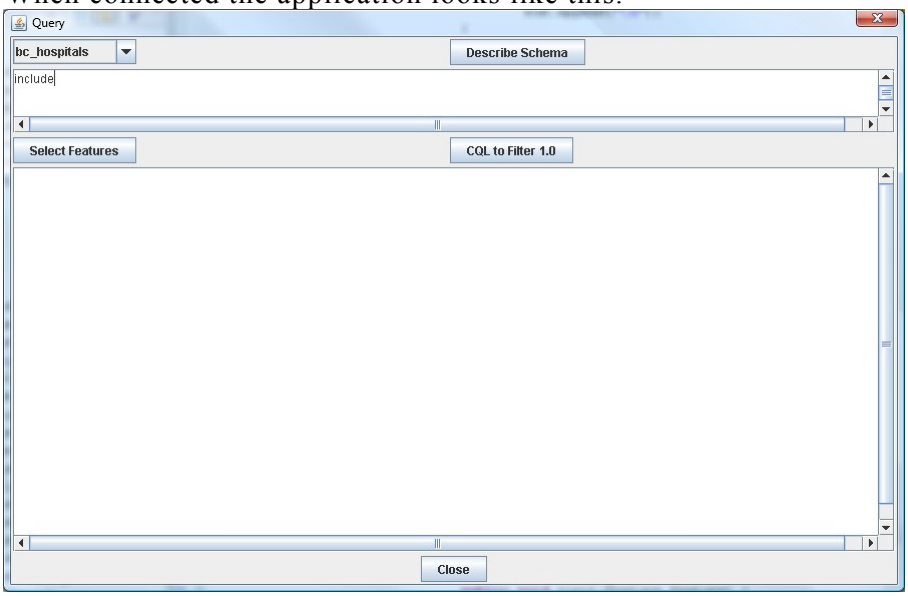
```
JQuery dialog = new JQuery(dataStore);
dialog.setVisible(true);
dialog.dispose();
System.exit(0);
```

The connection settings for FOSS4G database: example user: postgres password: postgres

5. Here is what that looks like when run and about to connect to the same www.refractions.net database. To connect please use "demo" as the password.



6. When connected the application looks like this.



7. We are going to spend the rest of this section filling in the details and talking about what information is available.

8. The most important thing is to request a FeatureCollection from the database. This is going to occur in a couple of steps – creating a Filter from "Common Query Language".

```
public FeatureCollection filter(String text) throws
Exception {
    Filter filter;
    filter = CQL.toFilter(text);
```

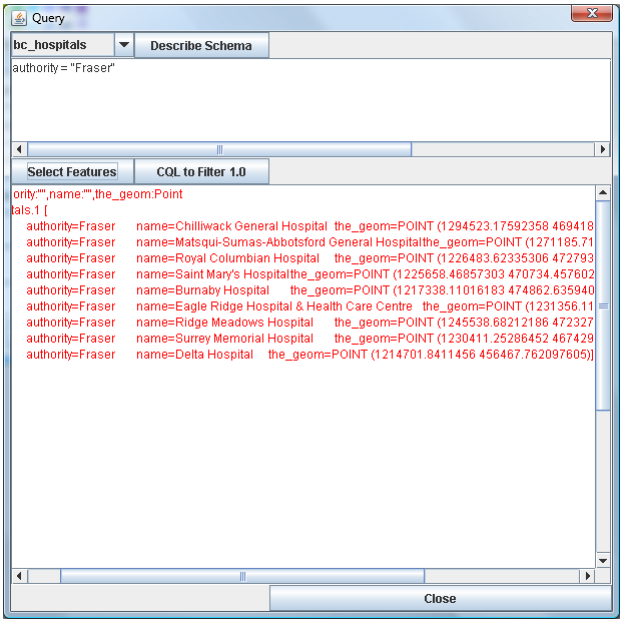
9. Figuring out the typeName and setting up a query,

```
    String typeName = (String)
                typeNameSelect.getSelectedItem();
    DefaultQuery query = new DefaultQuery();
    query.setTypeName(typeName);
    query.setFilter(filter);
    query.setMaxFeatures(1000);
```

10. And finally requesting a FeatureCollection:

```
    FeatureSource table =
dataStore.getFeatureSource(typeName);
    return table.getFeatures(query);
}
```

11. Here is what that looks like in action:



12. Here are some more filters to try out:

    - include
    - authority="Fraser"
    - BBOX(the_geom, 1200000, 450000, 1400000, 460000)

13. The CQL filter syntax we are using here can be coverted to the more common (and capable) Filter 1.0 XML format. Fill in the **display(Filter)** method shown:

```
protected void display(Filter filter) throws Exception {
    FilterTransformer transform = new FilterTransformer();
    transform.setIndentation(2);
    String xml = transform.transform(filter);

    show.setText(xml);
}
```

7 IMAGE LAB

The sample dataset includes several image files clouds.jpg and earthlights.jpg.

A raster file or image format is called a GridCoverage (a coverage is something that completely covers an area; and a raster just happens to be arranged in a grid – who thinks up this stuff?).

The raster support available is really cool and based on Java Advanced Imaging and Image IO libraries.

1. We are going to focus on a specific file format this time out - "World Plus Image" that consists of a normal PNG, GIF or JPEG file combined with a text file that describes where in the world it is located.
2. In this lab we are going to be filling in the **main** method to open up WorldImageReader.

```
File file = getImageFile( args );
WorldImageReader reader = new WorldImageReader( file );
```

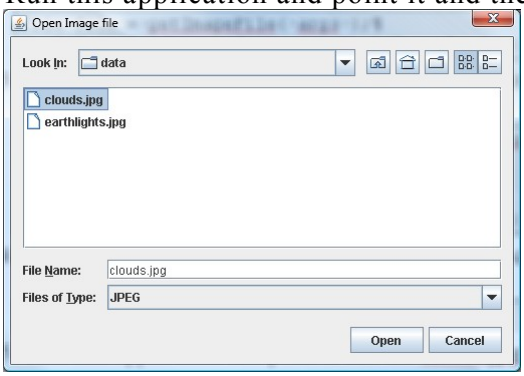
3. We are then going to set up a MapContext describing what we want to see on the screen.

```
MapContext map = new DefaultMapContext(reader.getCrs());
Style style = createStyle();
map.addLayer(reader.read(null), style);
map.setAreaOfInterest(new
ReferencedEnvelope(reader.getOriginalEnvelope()));
```

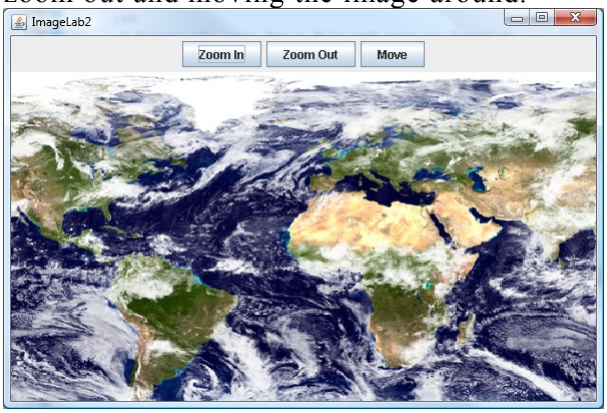
4. Finally we can show our Map using a utility class called JmapPane.

```
showMap(map);
```

5. Run this application and point it and the clouds.jpg file



6. You can use the buttons to change the mode between zoom in, zoom out and moving the image around.



7. The style used to display the map was created around a single RasterSymbolizer. Try modifying the following code to change the opacity.

```
private static Style createStyle() {
    StyleFactory styleFactory =
                CommonFactoryFinder.getStyleFactory(null);
    RasterSymbolizer symbolizer =
                styleFactory.createRasterSymbolizer();
    Rule rule = styleFactory.createRule();
    rule.setSymbolizers(new Symbolizer[] {symbolizer});
    FeatureTypeStyle fts =
                styleFactory.createFeatureTypeStyle();
    fts.setRules(new Rule[] {rule});
    Style style = styleFactory.createStyle();
    style.addFeatureTypeStyle(fts);
    return style;
}
```

8 SHAPE LAB

Finally we are going to open up a shape file and draw in on the screen. This time out you can have considerable flexibility to modify the style.

- You can create rules using the same **Filter** concepts covered in the PostGIS lab. Rules select which features are going to be draw.
- You can define "Symbolizers" that define how lines, points, polygons and text are displayed. The symbolizers can make use of expression – allowing both the use of feature data and on the fly calculations during rendering.
- Finally you can define several rules each with more than one symbolizer making for some very sophisticated effects.

1. This lab looks very similar to the earlier examples – please fill in the following main method.

```
public static void main(String[] args) throws Exception {
    File file = getShapeFile(args);

    ShapefileDataStore shapefile = new
ShapefileDataStore(file.toURL());
    String typeName = shapefile.getTypeNames()[0];
    FeatureSource featureSource =
shapefile.getFeatureSource();
    FeatureType schema = featureSource.getSchema();
    CoordinateReferenceSystem crs =
schema.getDefaultGeometry()
            .getCoordinateSystem();

    MapContext map = new DefaultMapContext(crs);
    Style style = createStyle(file,schema);
    map.addLayer(featureSource, style);

    showMap(map);
}
```

2. This time we are going to look into creating a style for each kind of content. The create style looks at the instanceof the "default geometry" indicated by your schema in order to figure out what kind of content is being displayed.

3. For all of these examples we will make use of a factory (StyleFactory) to create a Style, we will also make use of a FilterFactory to create the Expressions used to look up feature properties using Xpath, preform calculations and simply hold literal values.

4. Provide an implementation for createPolygonStyle and open up polygon shapefile (like countries.shp or worldborders.shp)

```
private static Style createPolygonStyle(){
    Style style;
    PolygonSymbolizer symbolizer =
styleFactory.createPolygonSymbolizer();
    Fill fill = styleFactory.createFill(
            filterFactory.literal("#FFAA00"),
            filterFactory.literal(0.5)
    );
    symbolizer.setFill(fill);
    Rule rule = styleFactory.createRule();
    rule.setSymbolizers(new Symbolizer[] { symbolizer });
    FeatureTypeStyle fts =
styleFactory.createFeatureTypeStyle();
    fts.setRules(new Rule[] { rule });
    style = styleFactory.createStyle();
    style.addFeatureTypeStyle(fts);
    return style;
}
```

5. In the above example we specify the fill color using a text string; and make the fill slightly transparent.
   You can also use a normal java Color as a literal expression; the code is very smart about adapting to what is needed.

```
    private static Style createLineStyle() {
        Style style;

        LineSymbolizer symbolizer =
styleFactory.createLineSymbolizer();
        SLD.setLineColour(symbolizer, Color.BLUE);
        symbolizer.getStroke().setWidth(filterFactory.liter
al(1));
        symbolizer.getStroke().setColor(filterFactory.liter
al(Color.BLUE));

        Rule rule = styleFactory.createRule();
        rule.setSymbolizers(new Symbolizer[]
{ symbolizer });
        FeatureTypeStyle fts =
styleFactory.createFeatureTypeStyle();
        fts.setRules(new Rule[] { rule });
        style = styleFactory.createStyle();
        style.addFeatureTypeStyle(fts);
        return style;
    }
```

6. The exciting here is mixing and matching all of these ideas; try looking at **TextSymbolizer** on your own and see if you can label one of your shape files.

7. To read an SLD document for disk, we can make use of an SLDParser. You can write out any style you create and use them in a wide range of applications.

```
private static Style createFromSLD(File sld) {
    SLDParser stylereader;
    try {
        stylereader = new SLDParser(styleFactory,
sld.toURL());
        Style[] style = stylereader.readXML();
        return style[0];
    } catch (Exception e) {
        JOptionPane.showMessageDialog(null,
e.getMessage());
        System.exit(0);
    }
    return null;
}
```

8. You can now load the **timezones.shp** file and it will pick up and parse the matching **timezones.sld** file.

9. You can combine many shapefiles, databases and raster images onto the same MapContext.

10. Some WMS servers even let you provide an SLD as part of your GetMap request.