GeoTools

Filter Workbook

FOSS4G 2009 Geospatial for Java Tutorials

27 September 2009

Jody Garnett
Michael Bedward

Table of Contents

1 Welcome

Welcome to Geospatial for Java -this workbook is aimed at Java developers who are new to geospatial and would like to get started.

Please set up your development environment prior to starting this tutorial (both a GeoTools NetBeans Quickstart and GeoTools Eclipse Quickstart are available). We will list the maven dependencies required at the start of the workbook.

If you found this tutorial online go ahead and skip to the code examples – the text is provided if you have any questions.

This is an exciting workbook where we actually sit down and start working with spatial data. The focus of this workbook is the Filter API used to query datastores, such as shapefiles and databases, and Web Feature Servers and ask them for their contents.

We are trying out a code first idea with these workbooks – offering you a chance to start with source code and explore the ideas that went into it later if you have any questions.

This workbook is part of the FOSS4G 2009 conference proceedings.

Jody Garnett

Jody Garnett is the lead architect for the uDig project; and on the steering committee for GeoTools; GeoServer and uDig. Taking the role of geospatial consultant a bit too literally, Jody has presented workshops and training courses in every continent (except Antarctica.) Jody Garnett is an employee of LISAsoft.

Michael Bedward

Michael Bedward is a researcher with the NSW Department of Environment and Climate Change and an active contributor to the GeoTools users' list. He has a particularly wide grasp of all the possible mistakes one can make using GeoTools.

2 Filter Lab

This example will go through using a Filter to select a FeatureCollection from a shapefile or other DataStore.

We are going to be using connection parameters to connect to our DataStore this time; and you will have a chance to try out using PostGIS or a Web Feature Server at the end of this example.

Let us start the lab:

1. The dependencies we are going to use this time are:

```
<dependencies>
    <dependency>
        <groupId>org.geotools</groupId>
        <artifactId>gt-main</artifactId>
        <version>${geotools.version}</version>
    </dependency>
    <dependency>
        <groupId>org.geotools</groupId>
        <artifactId>gt-shapefile</artifactId>
        <version>${geotools.version}</version>
    </dependency>
    <dependency>
        <groupId>org.geotools</groupId>
        <artifactId>gt-swing</artifactId>
        <version>${geotools.version}</version>
        <exclusions>
            <exclusion> <!-- we are not using svg icons right now -->
                <groupId>org.apache.xmlgraphics</groupId>
                <artifactId>batik-transcoder</artifactId>
            </exclusion>
        </exclusions>
    </dependency>
</dependencies>
```

2. Create the file org.geotools.demo.QueryLab using your IDE.
3. We are going to start by filling in the imports used through out the rest of the application..

```
package org.geotools.demo;

import java.awt.BorderLayout;
import java.awt.Dimension;
import java.awt.event.ActionEvent;
import java.io.IOException;
import java.util.Map;

import javax.swing.JComboBox;
import javax.swing.JFrame;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JOptionPane;
import javax.swing.JScrollPane;
import javax.swing.JTable;
import javax.swing.JTextField;
import javax.swing.table.DefaultTableModel;
```

4. We then have some GeoTools, GeoAPI and JTS imports to bring in.

```
import org.geotools.data.DataStore;
import org.geotools.data.DataStoreFinder;
import org.geotools.data.DefaultQuery;
import org.geotools.data.FeatureSource;
import org.geotools.data.shapefile.ShapefileDataStoreFactory;
import org.geotools.feature.FeatureCollection;
import org.geotools.feature.FeatureIterator;
import org.geotools.filter.text.cql2.CQL;
import org.geotools.swing.action.SafeAction;
import org.geotools.swing.data.JDataStoreWizard;
import org.geotools.swing.table.FeatureCollectionTableModel;
import org.geotools.swing.wizard.JWizard;
import org.opengis.feature.simple.SimpleFeature;
import org.opengis.feature.simple.SimpleFeatureType;
import org.opengis.feature.type.FeatureType;
import org.opengis.filter.Filter;

import com.vividsolutions.jts.geom.Coordinate;
import com.vividsolutions.jts.geom.Geometry;
import com.vividsolutions.jts.geom.Point;

public class QueryLab extends JFrame {
    DataStore datastore;
    JComboBox types;
    JTable table;
    JTextField text;
    public QueryLab(DataStore data) {
        this.datastore = data;
        // USER INTERFACE
    }
}
```

5. As you can see we are building a user interface this time around; starting from a JFrame and adding a text field to enter a filter condition, and a table to display the results.

```
    public QueryLab(DataStore data) {
        this.datastore = data;
        // USER INTERFACE
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        getContentPane().setLayout(new BorderLayout());

        try {
            types = new JComboBox(datastore.getTypeNames());
        } catch (IOException e1) {
            JOptionPane.showMessageDialog(null, "Unable to find any published content");
            System.exit(0);
        }

        text = new JTextField(80);
        text.setText("include"); // include selects everything!
        getContentPane().add(text, BorderLayout.NORTH);

        table = new JTable();
        table.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);
        table.setModel(new DefaultTableModel(5, 5));
        table.setPreferredScrollableViewportSize(new Dimension(500, 200));

        JScrollPane scrollPane = new JScrollPane(table);
        getContentPane().add(scrollPane, BorderLayout.CENTER);

        JMenuBar menubar = new JMenuBar();
        setJMenuBar(menubar);

        menubar.add(types);
        JMenu menu = new JMenu("Data");
        menubar.add(menu);
        pack();
        // ACTIONS
}
```

6. We can now add an Action to the menu to list features

```
        // ACTIONS
        menu.add(new SafeAction("Get features") {
            public void action(ActionEvent e) throws Throwable {
                filterFeatures();
            }
        });
```

7. And now we can create our Filter and ask for the FeatureCollection to display.

```
    public void filterFeatures() throws Exception {
        String typeName = (String) types.getSelectedItem();
        FeatureSource source = datastore.getFeatureSource(typeName);

        Filter filter = CQL.toFilter(text.getText());
        FeatureCollection features = source.getFeatures(filter);
        FeatureCollectionTableModel model = new FeatureCollectionTableModel(features);
        table.setModel(model);
    }
```

This is what is happening:

- First we get the feature type name selected by the user and retrieve the corresponding FeatureSource from the DataStore.
- Next we get the query condition that was entered in the text field and use the CQL class to create a Filter object.
- We pass the filter to the getFeatures method which returns the features matching the query as a FeatureCollection (you met FeatureCollection before in CSV 2 SHP Lab).
- Finally, we create a FeatureCollectionTableModel for our dialog's JTable. This GeoTools class takes a FeatureCollection and retrieves the feature attribute names and the data for each feature.

8. With the user interface in place we can now create a main method

```
public static void main(String[] args) throws Exception {
        JDataStoreWizard wizard = new JDataStoreWizard(new ShapefileDataStoreFactory());

        int result = wizard.showModalDialog();
        if (result != JWizard.FINISH) {
            System.exit(0);
        }

        Map<String, Object> connectionParameters = wizard.getConnectionParameters();
        DataStore dataStore = DataStoreFinder.getDataStore(connectionParameters);
        if (dataStore == null) {
            JOptionPane.showMessageDialog(null, "Could not connect - check parameters");
            System.exit(0);
        }

        JFrame frame = new QueryLab(dataStore);
        frame.setVisible(true);
    }
```

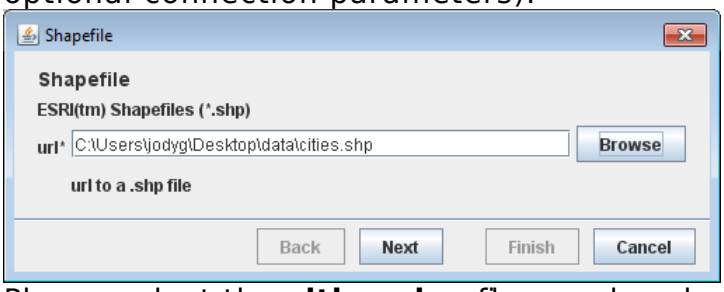
As you can see most of the code in the main method is to prompt the user for a shapefile and connect to it.

- In Quickstart and other examples we used JFileDataStoreChooser to prompt the user for a shapefile. In this example we are using JDataStoreWizard which requires a few more lines of code. The advantage of the wizard here is that we can easily modify the code above to work with a PostGIS database instead of a shapefile.

2.1 Running the Application

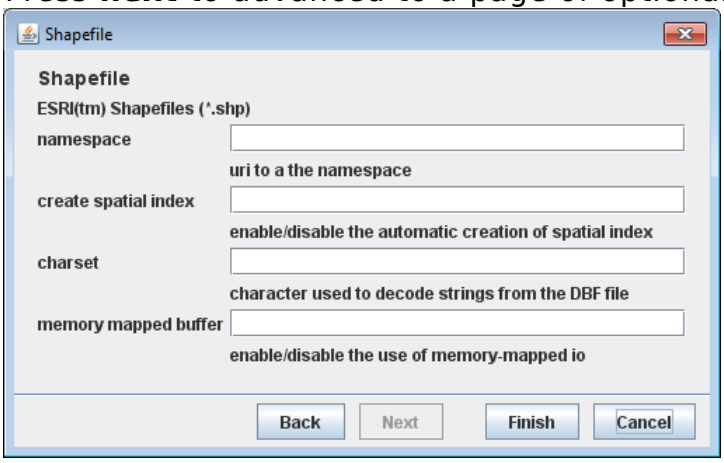
We can now run the application and try some queries.

1. Run the application and it will prompt you for a shape file (and several optional connection parameters).

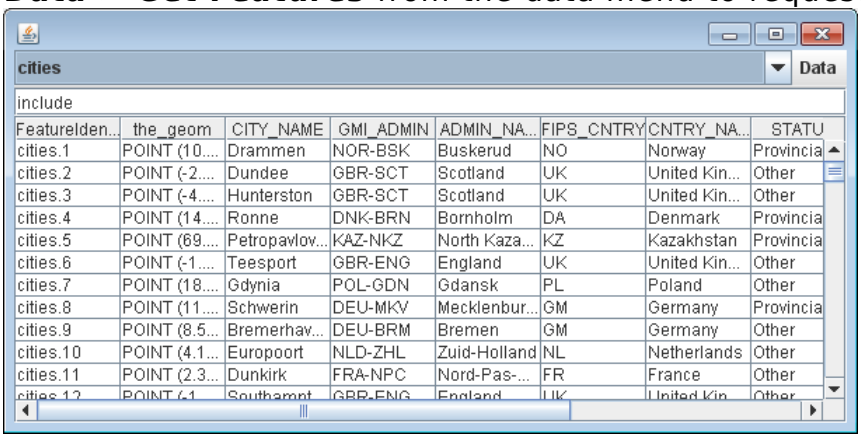


   Please select the **cities.shp** file you downloaded for the quickstart.

2. Press **next** to advanced to a page of optional parameters.



3. The default query is “include” which will include all the data; select **Data > Get Features** from the data menu to request all the cities.

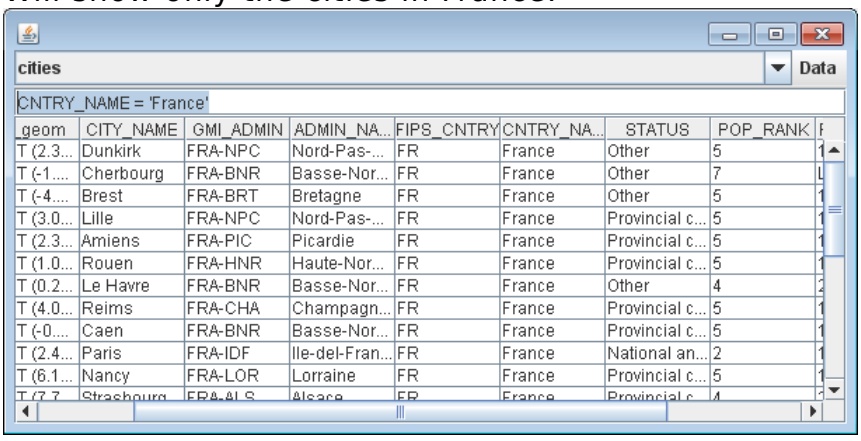


Featurelden...	the_geom	CITY_NAME	GMI_ADMIN	ADMIN_NA...	FIPS_CNTRY	CNTRY_NA...	STATU
cities.1	POINT (10....	Drammen	NOR-BSK	Buskerud	NO	Norway	Provincia
cities.2	POINT (-2....	Dundee	GBR-SCT	Scotland	UK	United Kin...	Other
cities.3	POINT (-4....	Hunterston	GBR-SCT	Scotland	UK	United Kin...	Other
cities.4	POINT (14....	Ronne	DNK-BRN	Bornholm	DA	Denmark	Provincia
cities.5	POINT (69....	Petropavlov...	KAZ-NKZ	North Kaza...	KZ	Kazakhstan	Provincia
cities.6	POINT (-1....	Teesport	GBR-ENG	England	UK	United Kin...	Other
cities.7	POINT (18....	Gdynia	POL-GDN	Gdansk	PL	Poland	Other
cities.8	POINT (11....	Schwerin	DEU-MKV	Mecklenbur...	GM	Germany	Provincia
cities.9	POINT (8.5...	Bremerhav...	DEU-BRM	Bremen	GM	Germany	Other
cities.10	POINT (4.1...	Europoort	NLD-ZHL	Zuid-Holland	NL	Netherlands	Other
cities.11	POINT (2.3...	Dunkirk	FRA-NPC	Nord-Pas-...	FR	France	Other
cities.12	POINT (-1....	Southampt...	GBR-ENG	England	UK	United Kin...	Other

4. Here are some sample queries to try:

```
CNTRY_NAME = 'France'
```

Will show only the cities in France:



geom	CITY_NAME	GMI_ADMIN	ADMIN_NA...	FIPS_CNTRY	CNTRY_NA...	STATUS	POP_RANK
T (2.3...	Dunkirk	FRA-NPC	Nord-Pas-...	FR	France	Other	5
T (-1....	Cherbourg	FRA-BNR	Basse-Nor...	FR	France	Other	7
T (-4....	Brest	FRA-BRT	Bretagne	FR	France	Other	5
T (3.0...	Lille	FRA-NPC	Nord-Pas-...	FR	France	Provincial c...	5
T (2.3...	Amiens	FRA-PIC	Picardie	FR	France	Provincial c...	5
T (1.0...	Rouen	FRA-HNR	Haute-Nor...	FR	France	Provincial c...	5
T (0.2...	Le Havre	FRA-BNR	Basse-Nor...	FR	France	Other	4
T (4.0...	Reims	FRA-CHA	Champagn...	FR	France	Provincial c...	5
T (-0....	Caen	FRA-BNR	Basse-Nor...	FR	France	Provincial c...	5
T (2.4...	Paris	FRA-IDF	Ile-del-Fran...	FR	France	National an...	2
T (6.1...	Nancy	FRA-LOR	Lorraine	FR	France	Provincial c...	5

5. Select all features with value >= 5 for the POP_RANK attribute

```
POP_RANK >= 5
```

6. Select features which satisfy two conditions

```
CNTRY_NAME = 'Australia' AND POP_RANK > 5
```

7. This is a bounding box query that will select all features within the area bounded by 110 - 155 ° W, 10 - 45 ° S (a loose box around Australia). Notice that we give the name of the geometry attribute which, for the cities shapefile, is Point type.

```
BBOX(the_geom, 110, -45, 155, -10)
```

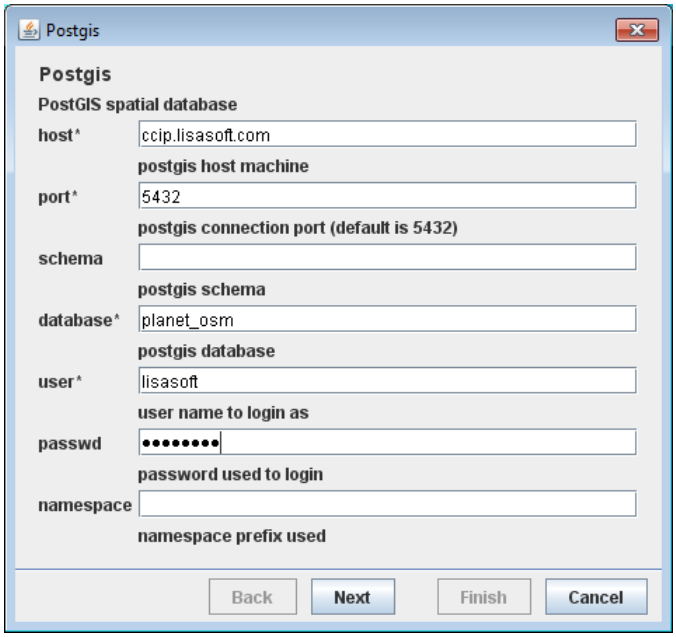
3 Things to Try

Additional ideas to try:

- Try using PostGIS – you will need to change your main method to:

```
JDataStoreWizard wizard = new JDataStoreWizard(new PostgisDataStoreFactory());
```

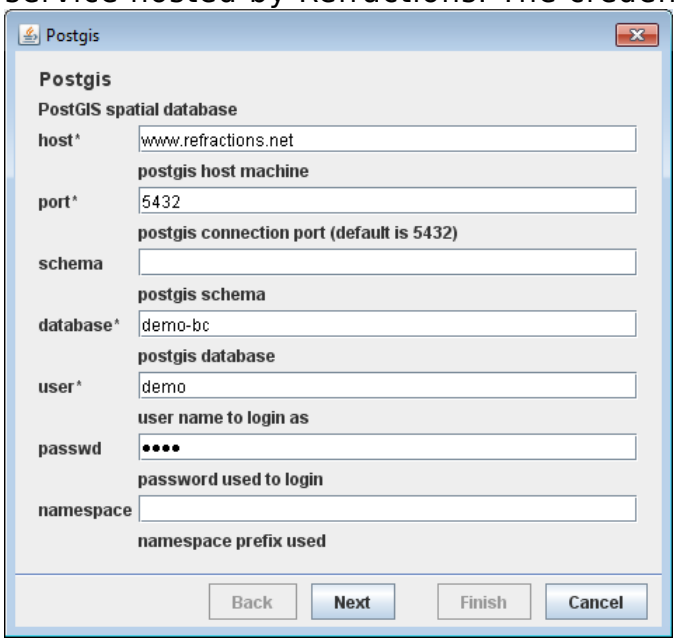
- And then connect with the following parameters. The login credentials are readonly/readonly



The database is provided by the Climate Change Integration Plugfest for the FOSS4G conference.

Remember you will need to add **gt-postgis** as a dependency in your pom.xml file.

- If you are using this workbook after the FOSS4G conference try this service hosted by Refractions. The credentials are demo/demo.



  Refractions is responsible for the PostGIS spatial extension to PostgreSQL – although now it is an independent OSGeo project.

- FeatureCollection has a few more things you can do with it – for example count the number of features returned.

  Add the following button:

```
        menu.add(new SafeAction("Count") {
            public void action(ActionEvent e) throws Throwable {
                countFeatures();
            }
        });
```

  And fill in the code for countFeatures.

```
    public void countFeatures() throws Exception {
        String typeName = (String) types.getSelectedItem();
        FeatureSource source = datastore.getFeatureSource(typeName);

        Filter filter = CQL.toFilter(text.getText());
        FeatureCollection features = source.getFeatures(filter);

        int count = features.size();
        JOptionPane.showMessageDialog(text, "Number of selected features:" + count);
    }
```

- We have seen how to represent a Filter using CQL; there is also the original XML representation to work with.

```
Configuration configuration = new org.geotools.filter.v1_0.OGCConfiguration();
Parser parser = new Parser( configuration );
...
Filter filter = (Filter) parser.parse( inputstream );
```

  If you need a xml file to start from you can write one out using.

```
Configuration = new org.geotools.filter.v1_0.OGCConfiguration();
Encoder encoder = new org.geotools.xml.Encoder( configuration );

encoder.encode( filter, org.geotools.filter.v1_0.OGC.FILTER, outputstream );
```

- GeoTools provides a configuration for both Filter 1.0 and Filter 1.1.

- In the Examples above we only used a Filter to select content – there is also DefaultQuery that can be used if you would like more control over what you get back.

  You can use DefaultQuery to specify sort order; limit the number of attributes returned and perform a couple of out of the box transformations.

```
public void queryFeatures() throws Exception {
    String typeName = (String) types.getSelectedItem();
    FeatureSource source = datastore.getFeatureSource(typeName);

    FeatureType schema = source.getSchema();
    String name = schema.getGeometryDescriptor().getLocalName();

    Filter filter = CQL.toFilter(text.getText());

    DefaultQuery query = new DefaultQuery(schema.getName().getLocalPart(), filter,
            new String[] { name });

    FeatureCollection<SimpleFeatureType, SimpleFeature> features = source.getFeatures(query);

    FeatureCollectionTableModel model = new FeatureCollectionTableModel(features);
    table.setModel(model);
}
```

- Finally, one of the interesting things to do is actually process the features for a result. Here is some code that will go through the features and find the center.

```
private void centerFeatures() throws Exception {
    String typeName = (String) types.getSelectedItem();
    FeatureSource source = datastore.getFeatureSource(typeName);

    Filter filter = CQL.toFilter(text.getText());
    FeatureCollection<SimpleFeatureType, SimpleFeature> features = source.getFeatures(filter);

    double totalX = 0.0;
    double totalY = 0.0;
    long count = 0;
    FeatureIterator<SimpleFeature> iterator = features.features();
    try {
        while (iterator.hasNext()) {
            SimpleFeature feature = iterator.next();
            Geometry geom = (Geometry) feature.getDefaultGeometry();
            Point centroid = geom.getCentroid();
            totalX += centroid.getX();
            totalY += centroid.getY();
            count++;
        }
    } finally {
        iterator.close(); // IMPORTANT
    }
    double averageX = totalX / (double) count;
    double averageY = totalY / (double) count;
    Coordinate center = new Coordinate(averageX, averageY);
    JOptionPane.showMessageDialog(text, "Center of selected features:" + center)
}
```

4 Filter

To request information from a FeatureSource we are going to need to describe (or select) what information we want back. The data structure we use for this is called a Filter.

We have a nice parser in GeoTools that can be used to create a Filter in a human readable form.

```
Filter filter = CQL.toFilter("POPULATION > 30000");
```

CQL is defined as part of the OGC Catalog specification.

The format used here is called “Common Query Language” and is similar to the where clause of an SQL select statement.

We can also make spatial filters using CQL – geometry is expressed using the same Well Known Text format employed earlier for JTS Geometry.

```
Filter pointInPolygon = CQL.toFilter("CONTAINS(THE_GEOM, POINT(1 2))");
Filter clickedOn = CQL.toFilter("BBOX(ATTR1, 151.12, 151.14, -33.5, -33.51)";
```

You may also skip CQL and make direct use of a FilterFactory.

```
FilterFactory ff = CommonFactoryFinder.getFilterFactory( null );

Filter filter = ff.propertyGreaterThan( ff.property( "POPULATION"), ff.literal( 12 ) );
```

Your IDE should provide command completion allowing you to quickly see what is available from FilterFactory.

Note, filter is a real live java object that you can use do to work.

```
if( filter.evaluate( feature ) ){
    System.out.println( "Selected "+ feature.getId();
}
```

The implementation in GeoTools is very flexible and able to work on Features, HashMaps and JavaBeans.

4.1 Expression

You may have missed it in the last section; but we also described how to access data using an expression. Here are some examples:

```
ff.property( "POPULATION" ); // expression used to access the attribute POPULATION from a feature
ff.literal( 12 );            // the number 12
```

You can also make function calls using the expression library; here are some examples in CQL:

```
CQL.toExpression("buffer( THE_GEOM)");
CQL.toExpression("strConcat( CITY_NAME, POPULATION)");
CQL.toExpression("distance( THE_GEOM, POINT(151.14,-33.51) )");
```

4.2 Query

The Query data structure is used to offer finer grain control on the results returned. The following query will request THE_GEOM and POPULATION from a FeatureSource “cities”.

```
DefaultQuery query = new DefaultQuery( "cities", filter, new String[]{ "THE_GEOM", "POPULATION" } );
```

4.3 FeatureCollection

Previously we added features to a FeatureCollection during the CSV2SHP example. This was easy as the FeatureCollection was in memory at the time. When working with spatial data we try to not have a FeatureCollection in memory because spatial data gets big in a hurry.

Special care is needed when stepping through the contents of a FeatureCollection with a FeatureIterator. A FeatureIterator will actually be streaming the data off disk and we need to remember to close the stream when we are done.

Even though a FeatureCollection is a “Collection” it is very lazy and does not load anything until you start iterating through the contents.

The closest Java concepts I have to FeatureCollection and FeatureIterator come from JDBC as shown below.

FeatureSource	View
FeatureStore	Table
FeatureCollection	PreparedStatement
FeatureIterator	ResultSet

If that is too much just remember – please **close** your feature iterator when you are done. If not you will leak resources and get into trouble.