Support Multi-Valued Attributes in Filter Comparison Operators

Motivation:	X-path expressions can return multiple values. Comparison operators on values (=, <, >, etc) and geometries (overlaps, intersects, etc...) in the filters currently do not handle multiple values as input.
Contact:	Niels Charlier
Tracker:	http://jira.codehaus.org/browse/GEOT-3576
Tagline:	filter x-path

This page represents the **current** plan; for discussion please check the tracker link above.

Children:

Description

The proposal is to modify the following filters:

- comparison filters: equals, not equals, gt, gte, lt, lte, between, like
- all geometry filters

To allow collections of values as input, besides single values. if an expression returns a collection of values rather than just a single value, the filters will be able to handle it. The OGC Filter standard specifies that the way to deal with multiple values can be modified using the "matchAction" flag:

7.7.3.3
matchAction parameter
The matchAction attribute can be used to specify how the comparison predicate shall be evaluated for a collection of values (e.g. in XML, properties having maxOccurs > 1) and not including some additional context to identify a specific value from the collection to be tested. Possible values for the attribute are: All, Any or One. A value of All means that all values in the collection shall satisfy the predicate. A value of Any means that any of the value in the collection can satisfy the predicate. Finally, a value of One means that only one of the values in the collection shall satisfy the predicate.
If the value of the matchAction attribute is One, additional context (e.g. XPath index) can be included to indicate which value in the collection should satisfy the predicate.

EXAMPLE

The following example illustrates the use of the matchAction attribute. Consider the following XML fragment, which is an instance of a GML (see ISO 19136) feature:

```
<ex:Building gml:id="b123">
  <gml:name>175 Fifth Ave.</gml:name>
  <gml:name>Flatiron</gml:name>
  <gml:name>Acme Building</gml:name>
  <!- ... ->
</ex:Building>
```

And consider the following filter expression:

```
<fes:Filter>
  <fes:PropertyIsEqualTo matchAction="...">
    <fes:ValueReference>gml:name</fes:ValueReference>
    <fes:Literal>Flatiron</fes:Literal>
  </fes:PropertyIsEqualTo>
</fes:Filter>
```

If the value of the matchAction attribute is set to Any, this predicate will evaluate to true since there is at least one gml:name value that satisfied the predicate. If the value of the matchAction attribute is All, this predicate will evaluate to false since not all gml:name values are Flatiron. Finally, if the matchAction attribute is set to One then the expression will evaluate to true since only one gml:name value is Flatiron.
If the value of the matchAction attribute is Any or All, the ValueReference XPath expression shall not include an index predicate. If the matchAction attribute is One an XPath index predicate may be specified and the predicate shall only evaluate to true if not only one value matches the predicate but the specific value indicates by the index matches the value.

The default value of "matchAction" is "Any" i.e. the result will be the OR-ed aggregation of all possible combinations. For example, if we compare the following values

(x, y, z) = (a, b)

the result will be true if x=a OR x=b OR y=a OR y=b OR z=a OR z=b.

Plan

MatchAction

The plan is to introduce an Enum MatchAction for ANY, ALL, ONE.

Update Filter with getMatchAction() methods

The update is limited to:

- BinaryComparisonOperator
- PropertyIsBetween

- PropertyisLike
- SpatialOperator

Update FilterFactory

Update FilterFactory and FilterFactory2 interfaces to allow MatchAction to be supplied.

For reference here is a code example showing how things stand:

```
Filter filter1 = ff.equals( expr1, expr2); // assume default case sensitive match
Filter filter2 = ff.equals( expr1, expr2, false ); // case insensitive match
```

The idea is to explicitly allow for MatchCase enumeration:

```
Filter filter3 = ff.equals( expr1, expr2, MatchCase.ANY );
Filter filter4 = ff.equals( expr1, expr2, true, MatchCase.ONE );
```

Update Filter implementations

Inside the implementation of equals we will need to update the code to expect a List of values (rather than a single value).

Here is an example of how any filter implementation can be made compatible with multiple values:

```
public boolean evaluate(Object feature) {
  final Object object1 = eval(expression1, feature);
  final Object object2 = eval(expression2, feature);

  if(!(object1 instanceof Collection) && !(object2 instanceof Collection)) {
    return evaluateInternal(value1, value2);
  }

  Collection<Object> leftValues = object1 instanceof Collection ? (Collection<Object>)
object1
                : Collections.singletonList(object1);
  Collection<Object> rightValues = object2 instanceof Collection ?
(Collection<Object>) object2
                : Collections.singletonList(object2);

  int count = 0;

  for (Object value1 : leftValues) {
     for (Object value2 : rightValues) {
        boolean temp = evaluateInternal(value1, value2)) {
        if (temp) {
          count++;
        }

        switch (matchAction){
         case ONE: if (count > 1) return false; break;
         case ALL: if (!temp) return false; break;
         case ANY: if (temp) return true; break;
        }
    }
  }

   switch (matchAction){
         case ONE: return (count == 1);
         case ALL: return true;
         case ANY: return false;
   }
}

public abstract boolean evaluateInternal(Object value1, Object value2);
```

Status

Proposal accepted, work is proceeding on the 8.x branch:

- Andrea Aime +1
- Ben Caradoc-Davies: +1
- Christian Mueller +0
- Ian Turton +1
- Justin Deoliveira +1
- Jody Garnett +1
- Simone Giannecchini +0

Tasks

	no progress		done		impeded		lack mandate/funds/time		volunteer needed

1. NC: Modify filter implementation to work with default "ANY" implementation + add unit tests for new behaviour
2. NC: Check for any effects on other modules like gt-render
3. NC: Update Filter interface to retrieve "matchAction" property (default: MatchAction.ANY)
4. NC: Update Filter implementation to deal with MatchAction.ALL and MatchAction.ANY properties for MatchAction.
5. NC: Update Filter factories to create filters with customised "matchAction" properties
6. JD: Update XML parser to parse the 'matchAction' property.
7. : Review user documentation for changes
   - http://docs.geotools.org/latest/userguide/library/main/filter.html
   - http://docs.geotools.org/latest/userguide/library/opengis/filter.html
   - Update pictures in images/ folder using ObjectAID eclipse plugin
   - bonus: add example for matchCase since that was not done

API Changes

FilterFactory

BEFORE:

```
interface FilterFactory {
...
PropertyIsEqualTo equals(Expression expr1, Expression expr2);
PropertyIsEqualTo equal(Expression expr1, Expression expr2, boolean matchCase);
...
}
```

AFTER:

```
interface FilterFactory {
...
PropertyIsEqualTo equals(Expression expr1, Expression expr2);
PropertyIsEqualTo equal(Expression expr1, Expression expr2, boolean matchCase);
PropertyIsEqualTo equal(Expression expr1, Expression expr2, boolean matchCase,
MatchAction matchAction);
 ...
}
```

(This is done for any filter that supports matchAction).

Filters

BEFORE:

```
public interface BinaryComparisonOperator extends Filter {
...
}

public interface PropertyIsBetween extends Filter {
...
}

public interface PropertyIsLike extends Filter {
...
}
public interface SpatialOperator extends Filter {
...
}
```

AFTER:

```
public interface MultiValuedFilter extends Filter {
    enum MatchAction {ANY, ALL, ONE};

    MatchAction getMatchAction();
}


public interface BinaryComparisonOperator extends MultiValuedFilter {
...
}

public interface PropertyIsBetween extends MultiValuedFilter {
...
}

public interface PropertyIsLike extends MultiValuedFilter {
...
}
public interface SpatialOperator extends MultiValuedFilter {
...
}
```

(The same is done for PropertyIsBetween, PropertyIsLike and SpatialOperator)