RegexpKeyedMap

A map that uses a regular expression as a key.

Author: Manik Surtani (manik at surtani dot org)

Purpose: To create a Java Map that allowed me to pass in any arbitrary String as a key to it's get() method, and for the Map implementation to attempt to match this key against various regexp-keys it would store, and return any matching value (or null if nothing matches)

Dependencies: This class relies on v1.3 of Jakarta's Regexp package

Examples: Ok, that explanation above was rubbish. Nothing works quite as well as an example. Here we go:

```
RegexpKeyedMap map = new RegexpKeyedMap();
map.put( "^Green.*$", "Your sentence starts with Green" );
map.put( "^Blue.*$", "Your sentence starts with Blue" );
map.put( "Orange", "Your sentence contains the word Orange" );


// contains some stuff typed in my the user
String sentence = "Green apples happen to be my favourite";

// responseText will be "Your sentence starts with Green"
String responseText = map.get( sentence );

sentence = "I like green apples a lot";

// responseText will be null since nothing matched
responseText = map.get( sentence );


sentence = "Green apples and Oranges are both quite good";

// responseText will be indeterminate - it could be either of "Your sentence starts with Green"
// or "Your sentence contains the word Orange" - this is because the RegexpKeyedMap uses a HashMap
// to store its contents and ordering of elements is not guaranteed.  The first match is returned.
responseText = map.get( sentence );
```

Source code:

```
package org.apache.regexp.collections;

import java.util.HashMap;
import java.util.Iterator;

import org.apache.regexp.RE;
import org.apache.regexp.RESyntaxException;



/**
 * This map implementation uses a hashmap as the underlying storage.
 * Note that the keySet() method will return a set of regular expressions rather than actual keys.
 * The put() method uses a regexp as a key.
 * The get() method gets any value that matches one of the regexps.  If there is more than one matching regexp,
the first one
 * encountered is returned - and hence could be indeterminate!
 *
 * @author Manik Surtani
 *
 */
public class RegexpKeyedMap extends HashMap
{
    public Object put(Object key, Object value)
    {
        if (key instanceof String)
```

```
            return super.put(key, value);
        else
            throw new RuntimeException("RegexpKeyedMap - only accepts Strings as keys.");
    }

    /**
     * The key passed in should always be a String.  The map will return the first element whose key, treated
as a regular expression, matches the key passed in
     * NOTE: It is possible for this map to have more than one return value, for example, if a key is passed
into get() which matches more than one regexp.
     *
     * E.g., consider the following keys in the map - '[A-Za-z]*' and 'Hello'.  Passing in 'Hello' as a key to
the get() method would match either of the regexps,
     * and whichever apears first in the map (which is indeterminate) will be returned.
     *
     */
    public Object get(Object key)
    {
        Iterator regexps = keySet().iterator();
        String keyString;
        Object result = null;

        String stringToMatch = cleanKey( key );

        while (regexps.hasNext())
        {
            keyString = regexps.next().toString();
            try
            {
                RE regexp = new RE(keyString);
                if (regexp.match(stringToMatch))
                {
                    result = super.get(keyString);
                    break;
                }
            }
            catch (RESyntaxException e)
            {
                // invalid regexp.  ignore?
            }
        }
        return result;
    }

    /**
     * Strip any 'dirty' chars from the key we are searching for,
     * otherwise we end up with funny results from the RE
     *
     * @param obj
     * @return
     */
    private String cleanKey( Object obj )
    {
        String retVal = obj.toString();

        // remove any '^' from start of key - prevents the RE from matching !?!?
        return ( retVal.charAt(0) == '^' )  ? retVal.substring(1) : retVal;
    }

}
```

Feedback: is very much appreciated! Let me know how this works for you, how it can be improved, etc.