ForgeRock Developer Programming Exercise

Background:

An application manages access to “resource” objects (e.g a user) and we need to provide a Java API for querying the set of available resources. One of the requirements for the query API is to have filtering functionality allowing clients to select resources which match certain criteria. For the sake of simplicity, a “resource” is represented using a Map<String,String> which maps “property” names (keys) to their values. Property names are case sensitive, but property values are not.

Task:

Design and implement a “Filter” API which can be used to determine whether or not a resource matches a given set of criteria. More specifically, the filter API should provide the following functionality:

1. The ability to determine whether or not a filter matches a given resource (where a resource is represented using a Map<String,String>).

2. Support for the following types of filter predicate (it is not necessary to implement all of these but they should try to implement at least one of each category):
    a. boolean literals (constants): “true” and “false”
    b. logical operators which can be used to combine the results of other filters: AND, OR, and NOT
    c. comparison operators (care should be taken to deal with missing properties):
        i. property is present
        ii. property is equal to some value
        iii. property is less than some value
        iv. property is greater than some value
        v. property matches a regular expression.

3. The ability to programmatically construct arbitrarily complex filters.

4. A string representation, including the ability to generate and parse filters from the string representation. IMPORTANT NOTE: you are not required to implement the string parsing logic since this could take too long and may prevent you from completing the rest of this exercise in the provided time.

5. Extensibility:
    a. Consider how the API could be extended to include support for new types of filter.
    b. Consider how the API could support 3rd party applications which need to perform some logic based on the structure and content of a filter in a type-safe manner.

Example usage:

```
// Create user resource having various properties:
Map<String, String> user = new LinkedHashMap<String, String>();
user.put(“firstname”,     “Joe”);
user.put(“surname”,       “Bloggs”);
user.put(“role”,          “administrator”);
user.put(“age”,           “35”);

// Create a filter which matches all administrators older than 30:
Filter filter = ???;              // Create a filter using your API.
assert filter.matches(user);      // Filter should match.

user.put(“age”,           “25”);
assert !filter.matches(user);     // Filter should not match.
```