User Manual for the `rebac` Java program

The code for the system is **open source** and available at github.com/andrewcortese/rebac. Use of the system requires Java 8 or newer. The package includes a pre-compiled executable called `rebac`.

In order to execute the program, a network file and resource file are needed. Several such sample files are included in the system package. The program can be executed with a command of the form:

```
java -jar rebac networkfile resourcefile
```

Where *networkfile* is a path to the network file and *resourcefile* is the path to the resource file. For best results, these files should be in the same directory as the `rebac` executable, and their relative pathnames should be used.

Once initialization is complete, a prompt will be displayed. The user can enter any number of access requests and see the result for each. The specified user and resource must exist within the dataset.

The source code is obviously also included in the package. This consists of several `.java` text files. In order to compile the code from source, these files must be in the present working directory. The following two commands can then be issued:

```
javac *.java
```

```
jar cvfe rebac Main *.class
```

The first command compiles the `.java` source files into `.class` object files. The second command links the object files and generates an executable called `rebac`.

The format of a policy string has some differences from the syntax of the abstract policy specification language. The only difference is that spaces between constructs are not allowed, and there are special symbols for propositional operators. @ is used instead of $\wedge$, and # is used instead of $\vee$. This is due to the character set for Java (there is no $\vee$), and the fact that some characters are reserved for special use in regular expressions.

For example, the policy:

$\langle parent\rangle\, a \vee \langle friend\rangle\, a \wedge \langle sibling\rangle\, a$

Would look like:

```
<parent>a#<friend>a@<sibling>a
```

An *affirmative* policy is one that contains no negation operators ($\neg$), no relation inverse operators ($-$), and no truth constants ($\top$ and $\bot$) Thus, it obeys the syntax: $\phi,\ \psi\ ::= \mathbf{a} \mid \phi \vee \psi \mid \psi \wedge \phi \mid \langle i \rangle\, \phi$. Our system only works with affirmative policies. Thus, the resource file cannot contain policies that do not obey this syntax (as well as the formatting rules discussed above.

To test the system, we created a simple synthetic social network and resource set. The information is outlined below:

$\mathcal{I} = \{friend, family, coworker\}$

$\mathcal{U} = \{u_i \mid 0 \leq i \leq 11\}$

$\mathcal{R} = \{r_0, r_1, r_2, r_3\}$

$owner(r_0) = u_0$
$policy(r_0) = \langle friend \rangle\, a \ \vee \ \langle family \rangle\, a$

$owner(r_1) = u_1$
$policy(r_1) = \langle friend \rangle\, a \ \wedge \ \langle coworker \rangle\, a$

$owner(r_2) = u_7$
$policy(r_2) = \langle friend \rangle\, a \ \vee \ \langle family \rangle \langle coworker \rangle\, a$

$owner(r_3) = u_7$
$policy(r_3) = \langle friend \rangle\, a \ \vee \ \langle coworker \rangle\, a \ \wedge \ \langle family \rangle\, a$

The sample network and resource files contain data to initialize the system to this state. Access requests can then be made, and the system will determine whether access is allowed. Below are some access requests that were made, and the authorization decision for each. (true indicates that access was granted, while false indicates that it was denied).

- access(u_1, r_0) = true
- access(u_7, r_0) = false
- access(u_6, r_1) = true
- access(u_2, r_1) = false
- access(u_{10}, r_2) = true
- access(u_0, r_2) = false
- access(u_{11}, r_3) = true
- access(u_4, r_3) = false