CS3152 Data Structures and Discrete Mathematics II
Project 1: Access Control
Due: 11:55pm, Sunday, Sep 22, 2012

Access Control

The Web makes information easily accessible to the public. However, in many cases the information should be accessed in a selective way. For example, credit card information should be accessible by a company employee that handles commercial transactions of the company. The same information should not be accessible by customers of the company. In this project, you should implement a simple *Policy Decision Point (PDP)*. The PDP is the part of an access control system that decides whether a request for data access is granted.

The Access Control Model

Every user of the access control system has a unique user ID. Access rights define which objects can be accessed by the users. Access rights are specified through so-called *rules*. A single rule is a triple of the form (*usr, dat, act*), which can be interpreted as "the user *usr* has the access right *act* for the data object *dat*." For example, a rule might be the triple (`usr001`, `drug`, `read`). This means that the user with id `usr001` can read the drug information of patients. We consider here only the access rights `read`, `write`, and `deny`. There is no order imposed on the rules. Thus the rules form a set of triples.

Policy Decision Point

An *access request* by a user is forwarded to the Policy Decision Point (PDP) in the form of the triple (*usr, dat, act*). At this point, the request has already been checked for syntactic and semantic errors. That means the user *usr* is a valid user. The data object *dat* is an existing data object. The value of *act* is either `read` or `write`. Note that it may happen that no rule applies to the data object *dat* or to the user *usr*. The PDP has to decide whether the request should be granted. Observe that the PDP does not actually access any data for the user. For example, in the case of the request (`usr001, drug, write`), the PDP determines whether a user `usr001` has `write` access to the object `drug`. By default, if there is no rule that grants access, the request is denied. Thus, the PDP checks the set of rules in order to make a decision. If, for example, the rule (`usr001`, `drug`, `write`) exists, but not the rule (`usr001`, `drug`, `deny`), then the access is granted. Depending on the access policy, the request may even be granted if both rules, (`usr001`, `drug`, `write`) and (`usr001`, `drug`, `deny`), exist. See the next section about access policies for details.

If the user and the data object of a rule do not match the user and the object of a request, the rule is irrelevant for the respective request.

Access Policies

Several contradictory rules may apply to a request. For example, there might be a rule that grants write access, e.g. (`usr001`, `drug`, `write`), but also a rule that denies access, e.g. (`usr001`, `drug`, `deny`). In this case, the outcome of an access request depends on the specified policy. In case of the policy DENY_OVERRIDES, a read or write action for a data object

dat by a user *usr* is denied if the rule (*usr, dat*, `deny`) exists—no matter whether one or more additional rules grant access. In case of the policy PERMIT_OVERRIDES, a read (respectively write) access request for an object *dat* by a user *usr* is granted if the rule (*usr, dat*, `read`) (respectively (*usr, dat*, `write`)) exists—no matter whether an additional rule denies access.

In some systems, a rule that grants write access should also imply read access. For example, if the rule (`usr001`, `drug`, `write`), but not the rule (`usr001`, `drug`, `read`) exists, a read request of the form (`usr001`, `drug`, `read`) should be granted. In this case, the policy WRITE_IMPLIES_READ must have been specified. Otherwise, the policy WRITE_WITHOUT_READ applies. In case of the policy WRITE_WITHOUT_READ, a read request is only granted if a rule explicitly grants read access.

The Project

In this project you will implement a policy decision point and its necessary classes. The learning objectives are as follows:

- Implement a given interface.
- Use the terminology from set theory.
- Meet specifications that use propositional logic.
- Test and debug a program in an object-oriented programming language.

You have to work in a team of two students. You are responsible for finding a partner for this project.

Implementation

Download the starter project AccessControl.zip which is given as an eclipse project. The project contains the interfaces PDP, `RuleSet`, and `Rule`, the enum types `Action`, `OverridePolicy`, and `ReadWritePolicy`, and the class `AccessRequest`. The *enum type* is used to define constants. See the tutorial by Oracle at http://docs.oracle.com/javase/tutorial/java/javaOO/enum.html for details if you are not familiar with the enum type. There are many different access control models that are based on different kinds of rules. Thus the interface `RuleSet` is a generic type to allow for the specification of the appropriate type of rules. Note that the type parameter must be implementing the interface `Rule`. In turn, a PDP is based on a specific type of rule set. Hence, the interface PDP is a generic type as well. Its type parameter must implement the `RuleSet`. For details about generics see the Oracle tutorial at the website http://docs.oracle.com/javase/tutorial/java/generics/index.html.

In this project you should to implement the three interfaces PDP, `RuleSet`, and `Rule`. Below are further instructions.

1. Implement a class called `UserRule` that implements the interface Rule. Each object of class `UserRule` represents a triple of the form (user, data, action). Two rules are equal if and only if the corresponding triples are equal. Add methods to the class as necessary.
2. Implement a class called `UserRuleSet` that implements the interface `RuleSet`. Add methods to the class as necessary. You can use the following class header:

```
public class UserRuleSet implements RuleSet<Rule>
```

Carefully decide on the most suitable data structure to implement the collection of rules. Keep in mind that the most frequent operation is the look-up operation (i.e. method contains). The add and remove operation occur only rarely.

3. Implement the interface `PDP`. The resulting class, say `MyPDP`, can have the following class header:

```
public class MyPDP implements PDP<UserRuleSet>
```

4. Carefully test your solution. Make sure that all methods of your implementation meet the given specifications of the corresponding interface. You do not have to include your tests in the final submission.

Submission

Submit a zip-file with your *s*ource code. In particular, submit the files Rule.java, RuleSet.java and PDP.java and any additional files that are required to execute your implementation. Only one student in your team has to submit the project. Include the names of both team members in the name of your zip-file. If both team members submit a solution, then the most recently submitted solution will be graded. The due date is Sunday, Sep 22, 11:55pm.

Grading

Points will be assigned as follows:

- Implementation of interface `Rule: 20 points`
- Implementation of interface `RuleSet:` 45 points
- Implementation of interface PDP: 35 points
  (Method requestIsGranted: 25 points)

A program that cannot be compiled or executed will receive 0 points.