CSE 425S: Programming Systems & Languages **Fall 2008**

Project #2: Denotational Semantics and Runtime-System Design

Assigned: October 27, 2008 *Due Date: November 17, 2008*

Overview

In this assignment, you will be given an informal description of a security-oriented programming language. You will be asked to provide a BNF description of the language's syntax, you will be asked to formally define the language using denotational semantics, and you will be asked to implement an interpreter and runtime-system for the language using Java. You will be required to submit a printed report in class on the listed due date, and you will also be required to electronically submit — via email — a copy of your compilable and runnable code on the due date before the start of class.

Informal Language Description

You are part of a team involved in the development of a highly secure data access system. A proposal has been made to use the class concept as a vehicle for encapsulating and managing security information. The idea is to use instance variables to keep track of password requirements, to treat methods as capabilities (i.e., access rights), and to employ class membership as the mechanism by which security obligations (passwords) are imposed upon and capabilities are granted to objects. Dynamic redefinition of classes allows for changes in the security requirements. Since a class can be defined in terms of other classes, inheritance may be used to combine obligations and capabilities. The general rule about inheritance is that obligations (passwords) accumulate and that capabilities (methods) can be granted by exactly one authority (i.e., a class) at any given time. In order to investigate these ideas further, you are to define and implement a simple language in which only the security issues are present.

Language Syntax

The language has three types of statements: class definition, class instantiation, and type-checking. A class definition defines a new security class in terms of any number of required passwords and provided capabilities. Security classes may inherit required passwords and available capabilities from any number of other security classes. However, security classes cannot inherit from themselves, neither directly nor indirectly; in other words, circular class definitions are not permitted. New classes may be defined at any time, and existing classes may be redefined dynamically. The following code creates a new — or redefines the existing — security class named "A", causing it to require the passwords "x" and "y" — in addition to the passwords required by "B" and "C" — and to inherit the capabilities of "B" and "C" (but not if the same capability can be granted by both base classes) as well as to provide — and possibly override — the capabilities "p" and "q":

```
define A
inherits B, C
```

```
passwords x, y
capabilities p, q;
```

A class instantiation statement creates a new object of a given security class or reassigns an existing object to a class. For example, the following instantiation statement reassigns, to the object “u”, membership in class “A”:

```
grant A to u;
```

A type-checking statement may be used to verify or discover the current security status of a given object. The check statement displays all the passwords associated with a given object, and it displays all capabilities available to the object, along with the security class from which the given capabilities are inherited. For example, the following check statement displays the security status of the object “u”:

```
check u;
```

The statement above — assuming that the security classes “B” and “C” have no passwords and no capabilities (or are undefined) — results in the following output:

```
password x
password y
capability p from A
capability q from A
```

Syntactic Language Specification

Create an abstract BNF description of the language syntax.

Semantic Language Specification

Formally define the language behavior using the denotational semantic model. Follow the method presented in class for developing denotational semantic specifications. In this situation the environment is much more complex than merely maintaining a mapping from identifiers to values. It is advisable to employ several functions or relations, which together characterize the abstract state of the system configuration, one might refer to it as the “security configuration”.

Runtime-System Design

Think about what algorithms and data structures you will need in order to implement your denotational semantic specification. Since you will actually implement this design, you should keep efficiency in mind while designing your runtime-system. List all the algorithms and data structures you will need, and explain how you will use them in implementing your language. If you use any non-standard algorithms or non-standard data structures, explain what they do and how they operate. Draw a diagram or other visual representation of how the components of your runtime-system interact. Also, if you use any non-standard data structures, draw a diagram of those structures.

Download the Java skeleton code for the interpreter from the course website. Look over the code, so that you understand what it does and how you can plug your own code into it. Fill out any parts of the skeleton which require filling-out, and alter the provided parser — if necessary — so that it agrees with your syntactic definition of the language. Implement your runtime-system design in actual Java code, and plug it into your modified skeleton. Be sure to test your code thoroughly, both with valid and invalid input, to make sure that it is both correct and robust (e.g. What happens when you provide a circular definition? What happens when you mistype a keyword?).

Sample Test

Print a transcript from executing your interpreter with the following input:

```
define A
inherits B, C
passwords x, y
capabilities p, q;

grant A to u;
check u;

define B
inherits none
passwords x,z
capabilities p, r;

check u;

define C
inherits none
passwords w,z
capabilities s, r;

check u;
```

What To Turn-In

At the start of class on November 17, 2008, you must submit a printed report containing everything but your Java code. The report should be in the following format:

1. Cover Page

    - Course title
    - Project number
    - Project name
    - Due date
    - Your name

2. Language Syntax
    - Brief Informal Overview
    - Abstract Syntax in BNF
3. Language Semantics
    - Denotational Semantic Model
4. Runtime System Design
    - Textual Description
    - Graphical Illustration
5. Transcript of Sample Test

In addition to the above, you must submit an archive of your code by email before the start of class on November 17, 2008. Please use either the *.zip, the *.tar.gz, or the *.tar.bz2 archive formats. The archive should contain all of your code as well as: a README explaining how to build, run, and test your code; an AUTHORS file containing your name, your email address, and your student ID#; an electronic copy of your printed report with the name "paper-version" and in DOCX, DOC, ODT, or PDF format. The email should be sent to *cse425s@gmail.com*, and the subject line should be "[CSE 425S] Project #2 Submission: *your_name*; Student ID# *your_id*".