Law of Demeter Checker

Design Document

Homework 1
COM 3205

September 29, 2002

Law of Demeter Checker Design Document

Table of Contents

Law of Demeter Checker Design Document

1.0 Introduction:

1.1 Law of Demeter:

The Law of Demeter (LoD) is a programming style rule that attempts to minimize the coupling between modules of a given program. It is based on the principle that the less any one module knows about the other modules in a given program, the easier that program will be to maintain. For example, if modules B, C, D, and E are written in a way in which they all rely on the structure of module A, then if the structure of module A changes, it is possible that code in modules B, C, D, and E may have to be modified in order for the program to continue to work properly. The LoD can be applied to *classes* as well as class instances (or *objects*) and it can be applied in both a *strong* and a *weak* form. Each variation of the LoD has its benefits and drawbacks.

1.2 System Objectives:

The goal of the Law of Demeter Checker application is to provide a means to check all four variations of the LoD in any Java program. The user will be allowed to choose the variation of the law that he/she wishes to apply to his program. Depending on the variation chosen, "violation notifications" will be reported at compile and/or run time and suggested coding corrections will be produced. The LoD Checker will be provided as a plugin for the Eclipse Integrated Development Environment (IDE).

2.0 Functional Requirements

2.1 Class Form Checker for the Law of Demeter

The LoD checker application will verify that the Java program in question follows the Class Form of the LoD (see definition below) at compile-time. If the Java application in question contains statements that do not follow the class form of the LoD, error statements will be produced notifying the user of the violations. For every violation a suggested code change will be produced in addition to the violation statement. This feature should be optional, i.e. the user must choose to use the class form checker instead of the object form checker. This feature should be mutually exclusive with the checker for the Object Form of the Law of Demeter. This feature should not be mutually exclusive with either the Weak Form or Strong Form Law of Demeter Checker.

Class Form of LoD: For all classes C, and for all methods M attached to C, all objects to which M sends a message must be instances of classes associated with the following classes:

1. The argument classes of M (including C).
2. The instance variable classes of C.
3. Objects created by M, or return types of local methods.
4. Objects in global variables.

2.2 Object Form Checker for the Law of Demeter

The LoD checker application will verify that the Java program in question follows the Object Form of the LoD (see definition below) at run-time. If the Java application in question contains statements that do not follow the Object Form of the LoD, error statements that describe the violations will be written to a log file. For every violation a suggested code change will be written to the log file in addition to the violation statement. This feature should be mutually exclusive with the checker for the Class Form of the Law of Demeter. This feature should be a default option when running the application, i.e. the user must choose to use the class form checker instead of the object form checker. This feature should not be mutually exclusive with either the Weak Form or Strong Form Law of Demeter Checker.

Object Form of LoD: For all classes C, and for all methods M attached to C, all objects to which M sends a message must be one of the following:

1. M's argument objects, including the this/self object
2. The instance variable objects of C.
3. Objects created by M, or objects returned by local method calls.
4. Objects in global variables.

2.3 Weak Form Checker for the Law of Demeter

The LoD checker application will verify that the Java program in question follows the Weak Form (see definition below) of either the Class or Object form of the LoD, depending on which form of the law the user has selected to check. If the Java application in question contains statements that do not follow the Weak Form of the LoD:

- if the check is being performed at compile time (i.e. user has selected the class form of the LoD) error statements will be produced
- if the check is being performed at run-time (i.e. the user has selected the object form of the LoD) error statements that describe the violations will be written to a log file.

For every violation a suggested code change will be produced in addition to the violation statement. This feature should be mutually exclusive with the checker for the Strong Form of the Law of Demeter. This feature should be a default option when running the application, i.e. the user must choose to use the strong form checker instead of the weak form checker. This feature should not be mutually exclusive with either the Class Form or Object Form Law of Demeter Checker.

Weak Form of LoD: The Weak Law of Demeter defines instance variables as being BOTH the instance variables that make up a given class AND any instance variables inherited from other classes.

2.4 Strong Form Checker for the Law of Demeter

The LoD checker application will verify that the Java program in question follows the Strong Form (see definition below) of either the Class or Object form of the LoD, depending on which form of the law the user has selected to check. If the Java application in question contains statements that do not follow the Strong Form of the LoD:

- if the check is being performed at compile time (i.e. user has selected the class form of the LoD) error statements will be produced
- if the check is being performed at run-time (i.e. the user has selected the object form of the LoD) error statements that describe the violations will be written to a log file.

For every violation a suggested code change will be produced in addition to the violation statement. This feature should be mutually exclusive with the checker for the Weak Form of the Law of Demeter. This feature should be optional when running the application, i.e. the user must choose to use the strong form checker instead of the weak form checker. This feature should not be mutually exclusive with either the Class Form or Object Form Law of Demeter Checker.

Strong Form of LoD: The Strong Law of Demeter defines instance variables as being ONLY the instance variables that make up a given class. Inherited instance variable types may not be passed messages.

2.5 Package Attachment:

The LoD Checker application must be attached as an additional package to the application being checked. The LoD Checker must be contained completely within a separate package from the application being checked.

2.6 "Checked Application" Modification:

The Java code of the application being checked will not be modified in any way by the LoD checker. AspectJ Aspects will instead be used to wrap checks around application "join points" when the application is compiled using the AspectJ compiler.

3.0 Non-Functional Requirements

3.1 AspectJ:

The application classes will be written primarily as AspectJ Aspects. The AspectJ Aspects will wrap code around specific "join points" in the application being checked when the application in question is recompiled using the AspectJ compiler with the LoD Checker added as an addition package. This will allow for non-intrusive runtime LoD checking of an application.

3.2 Eclipse:

The application will eventually be modified to function as a plugin for the Eclipse IDE.

3.3 Log4J:

The application will use Log4J to produce the log file of Object Form violations at run-time.

4.0 Test Cases

A : B | C *common* D
B = E
C = F
D = F
E = D
F = G
G = .

D :: func(){}
F :: func(){}
G :: func(){}

4.1 Weak Class Form

Pass

```
B :: foo1( ){
  e.d.func();
}

C :: foo2(){
  f.func();
}
```

Fail

```
C :: bar1(){
  f.g.func();
}

B :: bar2(){
  e.d.f.func();
}
```

4.2 Strong Class Form

Pass

```
C :: foo2(){
  f.func();
}
```

Fail

```
B :: foo1( ){
  e.d.func();
}

C :: bar1(){
  f.g.func();
}

B :: bar2(){
  e.d.f.func();
}
```

4.3 Weak Object Form

Pass

```
B :: foo1( ){
  d.func();
}

C :: foo2(){
  f.func();
}
```

Fail

```
C :: bar1(){
  f.g.func();
}

B :: bar2(){
  e.d.f.func();
}
```

4.4 Strong Object Form

Pass

```
C :: foo2(){
  f.func();
}
```

Fail

```
B :: foo1( ){
  d.func();
}

C :: bar1(){
  f.g.func();
}

B :: bar2(){
  e.d.f.func();
}
```

Law of Demeter Checker Design Document

Glossary:

advice: code that executes when a "join point" is reached in an AspectJ program (see aspectj.org)

Aspect: a collection of "join points" and "advice" as defined by the AspectJ syntax (see aspectj.org)

AspectJ: an extension to the Java Programming Language that allows for "Aspect Oriented Programming" (see aspectj.org)

Class Form of LoD: For all classes C, and for all methods M attached to C, all objects to which M sends a message must be instances of classes associated with the following classes:

5. The argument classes of M (including C).
6. The instance variable classes of C.
7. Objects created by M, or return types of local methods.
8. Objects in global variables.

Eclipse: an open source IDE (see eclipse.org)

IDE: (see) Integrated Development Environment

Integrated Development Environment (IDE): an application that provides multiple tools to facilitate code development.

join point: a well-defined point in the execution of a program (see aspectj.org)

Law of Demeter (LoD): a programming style rule that attempts to minimize the coupling between modules of a given program; Each unit should have only limited knowledge about other units: only units "closely" related to the current unit. Or: Each unit should only talk to its friends; Don't talk to strangers. (see also: *Object Form of LoD*; *Class Form of LoD*; *Weak Form of LoD*; *Strong Form of LoD*)

LoD: (see) Law of Demeter

Object Form of LoD: For all classes C, and for all methods M attached to C, all objects to which M sends a message must be one of the following:

5. M's argument objects, including the this/self object
6. The instance variable objects of C.
7. Objects created by M, or objects returned by local method calls.
8. Objects in global variables.

pointcut: a program element that picks out join points, as well as data from the execution context of the join points (see aspectj.org)

Strong Form of LoD: The Strong Law of Demeter defines instance variables as being ONLY the instance variables that make up a given class. Inherited instance variable types may not be passed messages.

Weak Form of LoD: The Weak Law of Demeter defines instance variables as being BOTH the instance variables that make up a given class AND any instance variables inherited from other classes.