Introduction to Object Oriented Programming 2E Timothy A. Budd

Chapter 17

Visibility and Dependency

© Timothy A. Budd 1996 (Modifications © H. C. Cunningham 1997)

Connections – The Bane of Large Scale Programming

- Difficulties in developing large-scale programs not so much algorithmic as *communication* complexity
- Need to control how much information one must have about another programmer's code

© Timothy A. Budd 1996 (Modifications © H. C. Cunningham 1997)

Visibility

Visibility is an attribute of *names*.

- Names of variables, functions, fields, etc.
- If you can't name something, you can't manipulate it
- Older languages have mechanisms for the control of name visibility (e.g., block structure, modules)
- OOP languages introduce a few new twists

© Timothy A. Budd 1996 (Modifications © H. C. Cunningham 1997)

Dependency

Dependency: degree one component relies on another to perform its responsibilities

- high degree of dependency limits code reuse
- makes moving component to new projects difficult

© Timothy A. Budd 1996 (Modifications © H. C. Cunningham 1997)

Coupling and Cohesion

Ideas from the software engineering community, pre-dating OOP

Coupling: extent one component uses another to perform actions

– goal is usually to *decrease* coupling

Cohesion: extent actions of a component tied together in purpose

– goal is usually to *increase* cohesion

© Timothy A. Budd 1996 (Modifications © H. C. Cunningham 1997)

Varieties of Coupling

Beginning with the least desirable

Internal data coupling: to internal data in other classes

Global data coupling: to shared global data

Control (or sequence) coupling: perform actions in fixed order

Parameter coupling: on method signatures

Subclass coupling: to parent class

© Timothy A. Budd 1996 (Modifications © H. C. Cunningham 1997)

Varieties of Cohesion

Beginning with the least desirable

Coincidental cohesion: arbitrary grouping

Logical cohesion: logically related, but no shared data or control (e.g., math library)

Temporal cohesion: must be performed at approximately same time (e.g., initialization)

Communication cohesion: manager for shared data or device

Sequential cohesion: activated in a particular order

Functional cohesion: relate to performance of single function

Data cohesion: implements a single data abstraction

© Timothy A. Budd 1996 (Modifications © H. C. Cunningham 1997)

Limiting Coupling – the Law of Demeter

The Law of Demeter attempts to limit the way components interact.

Law of Demeter: In any Method M attached to a class C, only methods defined by the following classes may be used:

1. The instance-variable classes of C.
2. The argument classes of method M (including C) – global objects or objects created inside the method M are considered arguments to M.

© Timothy A. Budd 1996 (Modifications © H. C. Cunningham 1997)

Rewritten in Terms of Messages

Law of Demeter (weak form): Inside a method, it is only permitted to access or send messages to the following objects:

1. Arguments of the method being executed (including the self object)
2. Instance variables for the receiver of the message
3. Global variables
4. Temporary variables created inside the method

© Timothy A. Budd 1996 (Modifications © H. C. Cunningham 1997)

What is Ruled Out?

- One object cannot directly manipulate internal data values of another object
- Instead, all access to another component should be through procedures – thereby reducing data coupling to the weaker parameter coupling

© Timothy A. Budd 1996 (Modifications © H. C. Cunningham 1997)

Class-Level versus Object-Level Visibility

Question:

Are siblings allowed to examine each others private data fields?

Answers:

– YES means class-level visibility as in C++ and Java

– NO means object-level visibility as in Smalltalk

© Timothy A. Budd 1996 (Modifications © H. C. Cunningham 1997)

Active Values

Problem: have an existing program and want to observe a data value – see when it changes value

Solution: create a new subclass that just changes those methods that set or read the data value

This solution requires no direct manipulation of the data value.

© Timothy A. Budd 1996 (Modifications © H. C. Cunningham 1997)

Public, Subclass and Private Faces

Previously noted that objects have public and private faces

– inheritance introduces the subclass face

Public features: aspects users of component must have access to

Private features: aspects implementor of component must have access to

Protected features: aspects implementors of child classes have access to

© Timothy A. Budd 1996 (Modifications © H. C. Cunningham 1997)

Visibility in Java – Packages

- package facility for grouping related class definitions
- import statement for making other packages' public features available (without package qualifiers)
- CLASSPATH file search path for packages based on package name

© Timothy A. Budd 1996 (Modifications © H. C. Cunningham 1997)

Visibility in Java – Access

- default (friendly) access features accessible everywhere in package
- private features accessible by instance and its class siblings
- public features accessible everywhere package is visible
- protected features accessible by the class and its subclass methods and everywhere inside package
- final modifier to prevent inheritance from classes, overriding of methods, or changes to variables

© Timothy A. Budd 1996 (Modifications © H. C. Cunningham 1997)

Intentional Dependency

- Sometimes want code to depend upon another class
- Sometimes want this even if one class not "knowledgeable" of other
- Introduce separate dependency manager
  - When an object changes, it tells the manager "notify my dependents"
  - E.g., Java 1.1 AWT event model

© Timothy A. Budd 1996 (Modifications © H. C. Cunningham 1997)