CIS W338

Introduction to Object Oriented Design

Encapsulation and Connascense
Encapsulation

• Level 0
  – Machine Instructions
  – Source Lines of Code
• Level 1
  – Functions and Procedures
• Level 2
  – Classes
Levels of Encapsulation

The Class
Procedural Reusability

• Very Limited Success with procedures.
• Language Support Libraries only exception.
• Larger data structures now part of C++ standard library.
• Java has a very extensive library.
• Classes have a greater potential for reuse.
• Components (an emerging market) even greater.
# Interactions

<table>
<thead>
<tr>
<th></th>
<th>Level-0</th>
<th>Level-1</th>
<th>Level-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Level-0</td>
<td>Structured Programming</td>
<td>Message fan-out</td>
<td></td>
</tr>
<tr>
<td>Level-1</td>
<td>Cohesion</td>
<td>Coupling</td>
<td></td>
</tr>
<tr>
<td>Level-2</td>
<td></td>
<td>Class cohesion</td>
<td>Class coupling</td>
</tr>
</tbody>
</table>
Definitions

• Fan-out is a measure of the number of references to other procedures by lines of code within a given procedure.
• Cohesion is a measure of the “single-mindedness” of the lines of code within a given procedure.
• Coupling is a measure of the number and strength of connections between procedures.
Connasence

- Derived from Latin, means “having been born together.” Also “having intertwined destinies in life.”
- Two software elements $A$ and $B$ have connasence if:
  - A change in $A$ may require a change in $B$ to preserve correctness
  - A change requires $A$ and $B$ to be changed together
Connascence of type

• int i; // line A
• ...
• i = 7; // line B
• If line A is changed to char i, then B will probably have to be changed.
Connascence of Name

- int i;     // Line A
- ...
- i = 7;     // Line B
- If line A is changed to int j, then line B also needs to be changed.
Connascence of Position

- X: JUMP Y + 38
- ...
- Y: CLEAR R1
- … // 38 bytes of code go here
- CLEAR R2 // This is the target of the jump.
Connascence of Convention

• Use positive account numbers for people and negative ones for companies.
• Code will be filled with statements of the form

        if (accountNo > 0)
              ...
else
       ...

Convention

• Direction:
  – 0 = north, 1 = east, 2 = south, 3 = west
  – N = north, E = east, S = south, W = west
  – 0 = north, 90 = east, 180 = south, 270 = west
Connascence of Algorithm

• Assume that the sort function has a “bug” such that the last two values are in the incorrect order.

• Assume that everybody who uses this knows about it and has a work-around in their code.

• Assume some day that the “bug” gets fixed!
Connascence of Execution

• Many routine examples
  – Initialization before use
  – Changing and reading global variables in the correct order
  – Setting and testing semaphore variables

• Some may be subtle or not obvious
  – Mask interrupt before disabling interrupts
Contrancence

• Sometimes things need to be different.
• This is a form of connascence.
• Example: Two classes used as base classes in multiple inheritance should have different public and protected attributes and operations.
Contranscence Example

```
Void ProgramRentalItem::someMethod
{
    ...
    if (PhysicalInventoryItem::length ...)
        ...
    ...
    if (RecordingMedium::length ...)
        ...
    ...
}
```
Encapsulation is a Check on Connascence

• Imagine a 100,000 LOC main program.
  – Need to add a new variable!
  – Is the same variable name used in different parts intentionally or is it mere coincidence?
Example -- Hash Table

• Pre Object Oriented Design
  – Global array held the table
  – Global function to insert
  – Global function to remove not necessarily close to the insert function

• Object Oriented
  – Table and functions encapsulated into a class
  – Only public interface it the remove and insert functions.
Connascence and Maintainability

• Minimize overall connascence.
• Minimize remaining connascence that crosses encapsulation boundaries.
• Maximize the connascence within encapsulation boundaries.
Abuses

- The *friend* of C++
- Inheritance
- Accidents of Implementation
The **friend** of C++

- By its nature, it violates encapsulation.
- When used judiciously it permits including an item within the encapsulation boundary that is otherwise not permitted by the syntax.
  - Overloaded stream insertion and extraction.
- Can be misused by declaring a function or class to be a friend of multiple classes.
Inheritance and Protected Items

• If all members of a base (super) class are protected, they are visible to all derived (sub) classes.

• Any change to the super-class may require a change to the sub-classes.
Accidents of Implementation

• A set class always retrieved items in the same order in which they were inserted.
  – The underlying data representation was a linked list.
• Programmer(s) relied upon this.
• Set class later replaced with one the retrieved items in a different order.
  – The C++ Standard Library uses a binary search tree.
Domains

- Application
- Business
- Architecture
- Foundation
The Foundation Domain

• Fundamental classes or build-in types
  – int, float, char, bool
• Structural classes
  – list, stack, string
• Semantic classes
  – Date, Time, Angle, Money, Point, Line, …

Universal
The Architecture Domain

• Communication classes
  – Port, Socket, RemoteMachine

• Database manipulation
  – Transaction, Backup, …

• Human interface
  – Window, CommandButton

  Computer or Operating System Specific
Business Domain

- **Attribute Classes**
  - Balance
  - BodyTemperature

- **Role Classes**
  - Customer
  - Patient

- **Relationship Classes**
  - AccountOwnership
  - PatientSupervision

Specific to the Business, but useful in multiple applications
Application Domain

- Event recognizer classes
  - PatientTemperatureMonitor
    - patientDevelopsFeaver
    - patientBecomesHypothermic

- Event manager classes
  - WarmHypothermicPatient
  - SchedulePatientForSurgery

Specific to the application, little reusability
Sources

• Foundation Classes
  – Part of the language support environment
  – Third party market

• Architecture Classes
  – May be part of support environment
  – Larger third party market

• Business Classes
  – Part of your product line
  – Some third party market
Encumbrance

- Encumbrance measures the total ancillary machinery of a class.

- Direct class-reference set
  - C inherits from D
  - C has an attribute of D
  - C has an operation with an input argument of D
  - C has a method that returns a value of D
  - C has a method with a local variable of D
  - C is a friend of D
Encumbrance (cont.)

• Indirect class-reference set
  – Union of the direct class-reference set of the direct class-reference classes.
Rectangle

Point

Length

Boolean

Real
The Use of Encumbrance

• Classes high in the Domain hierarchy should have high encumbrance.
  – Those that don’t probably are not taking advantage of reuse opportunities.

• Classes low in the Domain hierarchy should have low encumbrance.
  – Those that don’t probably do not have high cohesion.
The Law of Demeter

For an object $\text{obj}$ of class $\text{C}$ and for any operation defined for $\text{obj}$, each target of a message within the implementation of $\text{op}$ must be one of the following:

1. The object itself
2. An object refereed to by an argument of $\text{op}$.
3. An object referred to by a variable $\text{o obj}$.
4. An object created by $\text{op}$.
5. An object referred to by a global variable.
Class Cohesion

• Class cohesion is the measure of the interrelatedness of the features of a class.
• Low cohesion: features that do not belong together.
• High cohesion: features all contribute to the type abstraction.
• Should be apparent from the external interface.
Mixed-instance

• Some features are undefined for some objects of the class.

<table>
<thead>
<tr>
<th>Salesperson</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comissioned : Boolean</td>
</tr>
<tr>
<td>issueCommissionPayment ()</td>
</tr>
</tbody>
</table>

Assume that Mary.Commissioned is false. What does Mary.issueCommissionPayment do?
Removal of Mixed-Instance

Salesperson

Comissioned

NonComissioned
Mixed-domain

• A class with mixed-domain cohesion contains an element that directly encumbers the class with an extrinsic class of a different domain.

• The class B is extrinsic to A if A can be fully defined with no notion of B.
Example of Mixed-Domain

<table>
<thead>
<tr>
<th>Real</th>
</tr>
</thead>
<tbody>
<tr>
<td>arctan () : Angle</td>
</tr>
<tr>
<td>euroAmt () : Eruo</td>
</tr>
</tbody>
</table>
Removal of Mixed-Domain

\[
\text{tan (argname : argtype = default) : Real} \\
\text{arctan(x:Real) : Angle}
\]

\[
\text{value} \\
\text{Real}
\]

\[
\text{Money} \\
\text{USDollars} \\
\text{Euros}
\]

\[
\text{Real}
\]
Mixed-role

• A class with *mixed-role* cohesion contains an element that directly encumbers the class with an extrinsic class that lies in the same domain.

<table>
<thead>
<tr>
<th>Person</th>
<th>Why not cats?</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>numOfDogsOwned</code></td>
<td>Boats? Cars?</td>
</tr>
</tbody>
</table>
Design B

Solves the mixed-role cohesion.
To find how many dogs Fred owns

\[ \text{DogOwnership}::\text{HowmanyDogs}(Fred) \]

Potential efficiency issue.
What if Fred’s dog dies?  
What if he acquires a cat?
Design D

Can be easily extended to accommodate cats, boats, etc.

Efficiency issues.
- Uses multiple objects.
- Uses message forwarding.
Ideal Class Design

• No mixed-instance
• No mixed-domain
• No mixed-role
• May not be possible or reasonable
• No one correct answer