Design Patterns and Responsibility Assignment

CMPT 371
Fall 2004
J.W. Benham
Origins of design patterns

• Christopher Alexander, et.al. *A Pattern Language*
  – “Each pattern describes a problem that occurs over and over… and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice.”
  – Writing about architecture (for buildings, etc.)

• Gamma, Helm, Johnson, Vlissides (Gang of Four): *Design Patterns: Elements of Reusable Object-Oriented Software*
What are design patterns?

- Pattern name
- Problem
  - Describes when to apply pattern
- Solution
  - Elements of design, their relationships, responsibilities, and collaborations
- Consequences
  - Results and trade-offs of applying pattern
GRASP Patterns

• General Responsibility Assignment Software Patterns

• Fundamental principles of object design and responsibility assignment

• These principles are not new
Responsibility Assignment

• Responsibility – “a contract or obligation of a classifier”

• Responsibilities to know
  – A precedence table should know comparative precedence of two tokens.
  – A search string syntax should know the supported operations and their names.

• Responsibilities to do
  – A lexical analyzer should be able to extract find lexemes in a source string and determine the associated token.
Four GRASP Patterns

• Information Expert
• Creator
• High Cohesion (evaluative)
• Low Coupling (evaluative)
Information Expert

• Problem: “What is a general principle of assigning responsibilities to objects?”
• Solution: Assign responsibility to the class that has the information needed to fulfill the responsibility
    – Responsibility to know terminal nearest top of stack – ParsingStack
    – Responsibility to compare precedence of two tokens – PrecedenceTable
    – Responsibility to determine structure of source string??
Creator

• Problem: Who should be responsible for creating a new instance of a given class?
• Solution: Assign to class B the responsibility to create an instance of class A if one or more of the following holds:
  – B aggregates A objects
  – B contains A objects
  – B records instances of A objects
  – B closely uses A objects
  – B has the initializing data needed to create an A object
Creator Examples

• Who should create a *PrecedenceTable*?
• Who should create a *ParsingStack*?
• Who should create a *Token*?
• Who should create a *LexicalAnalyzer*?
  – TranslatorFrontEnd?
  – TranslationEnvironment?
  – SearchStringSyntax?
Low Coupling

- **Coupling** measures how strongly two elements (objects, components, modules) are connected?
- Problem: How does one design software that supports low dependency, low change impact, and increased reuse?
- Solution: Assign a responsibility so that coupling remains low
- An *evaluative* principle – used to compare two or more possible solutions.
Types of Coupling

• Content coupling
• Common coupling
• Control coupling
• Stamp coupling
• Data coupling

From Shari Lawrence Pfleeger’s Software Engineering: Theory and Practice, pp220-223
Low Coupling Examples

• Who should be responsible for storing a *TranslationEnvironment*?

• An old example revisited: Who should create a *LexicalAnalyzer*?
High Cohesion

- **Cohesion** measures how strongly related the responsibilities of a component are.
- Problem: How can one keep complexity manageable?
- Solution: Assign responsibility so that cohesion remains high.
Types of cohesion

- Coincidental cohesion
- Logical cohesion
- Temporal cohesion
- Procedural cohesion
- Communicational cohesion
- Sequential cohesion
- Functional cohesion

From Pfleeger, pp 223-225
High cohesion examples

• Who should maintain information about which operations and features are supported in source string?
• An old example revisited: Who should create a LexicalAnalyzer?
• Another old example: Who should store environments?
Coupling and Cohesion: The Yin and Yang of Software Engineering

• Interdependence
  – Low cohesion often results in high coupling also, and vice versa

• Example: *TranslationEnvironments* responsible for their own persistent storage