Thinking in design patterns

Damayanthi Herath

GRASP

Design Patterns

“Learning from your mistakes makes you smart. Learning from other people’s mistakes makes you a genius.”

Rick, why didn't you just say you were using the **Observer** Pattern?
Exactly. If you communicate in patterns, then other developers know immediately and *precisely* the design you're describing. Just don't get Pattern Fever... you'll know you have it when you start using patterns for Hello World...

Responsibility Driven Design

Responsibility Driven Design

Responsibility:

Definition: A contract or obligation of a classifier

Doing responsibilities include:

- directly—e.g. create object, perform calculation
- initiate action in other objects
- control and coordinate activities in other objects

Knowing responsibilities include knowledge of:

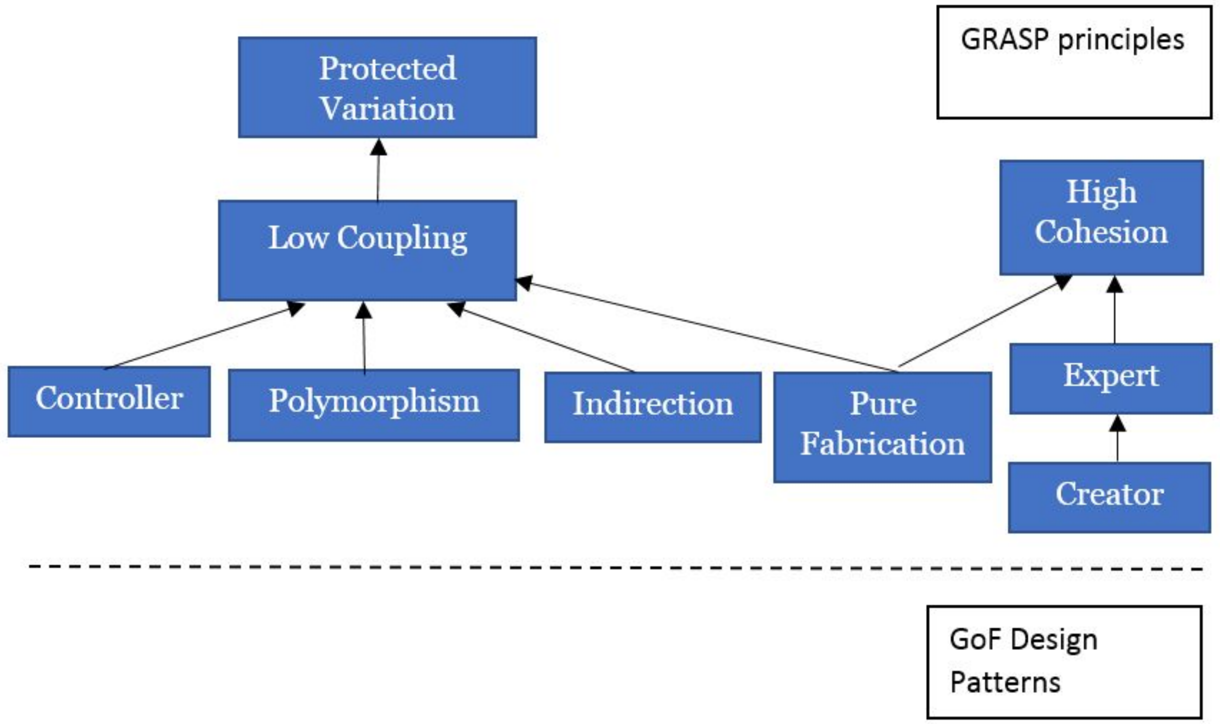
- Private encapsulated data
- Related objects
- Derivable or calculable items

Responsibility Driven Design (RDD)

- RDD sees an OO Design as a community of collaborating responsible objects.
- RDD involves assigning responsibilities to classes which should be based on proven principles.

GRASP:
General Responsibility Assignment Software Patterns (or Principles):

Design patterns Overload?



GRASP: General Responsibility Assignment Software Patterns (or Principles):

- **Information Expert**
- **Creator**
- **Low coupling**
- **Controller**
- **High Cohesion**

Problem
What is a general principle of assigning responsibilities to objects?

Design model:
- May have 100s or 1000s of software classes
- May have 100s or 1000s of responsibilities
- Useful to have a general principle to guide choice of assignment

Solution

Information Expert

Assign responsibility to the information expert—the class that has the information necessary to fulfil the responsibility.

Problem

Who should be responsible for creating a new instance of some class?

Solution

Creator

Assign class B responsibility to create instances of class A if one of these is true (the more the better):

- B “contains” or compositely aggregates A.
- B records A.
- B closely uses A.
- B has the initializing data for A that will be passed to A when it is created. Thus B is an Expert with respect to creating A.

Problem

How to support low dependency, low change impact, and increased reuse?

Coupling: Measure of how strongly one element is connected to, has knowledge of or relies on others.
Problems for a class with high coupling:

- Forced changes: result of changes in related classes
- Harder to understand in isolation

Solution

Low Coupling

Assign responsibility so that coupling remains low.
Use this principle to evaluate alternatives.

Problem
What first object beyond the UI layer receives and coordinates (“controls”) a system operation?

Solution

Controller

Assign responsibility to a class representing one of:

- the overall “system”, a “root” object, a device the software is running within, or a major subsystem
- a use case scenario that deals with the event, e.g. use case or session controller

Problem
How to keep objects focussed, understandable, and manageable, and as a side effect, support Low Coupling?

(functional) cohesion:
A measure of how strongly (functionally) related and focussed the responsibilities of an element are

Solution

High Cohesion

Assign a responsibility so that cohesion remains high. Use this to evaluate alternatives.

Class with low cohesion:

- Hard to comprehend
- Hard to reuse
- Hard to maintain
- Delicate; constantly affected by change