L16 - Responsibility-driven design and detailed design patterns

CS3028 - Principles of Software Engineering

Ernesto Compatangelo

Department of Computing Science

UNIVERSITY OF ABERDEEN

16.1 Reminding past issues and mapping them to current topics

Where are we now?

$\Rightarrow \cdots\cdots$

$\Rightarrow$ Elaboration (second UP phase)

$\Rightarrow \cdots\cdots$

$\Rightarrow$ Elaboration Design

$\Rightarrow$ Architectural design & patterns

$\Rightarrow$ Detailed design

$\Rightarrow$ Detailed design patterns

$\Rightarrow \cdots\cdots$

16.2 From mid-level design to responsibility-driven design

The focus of object-oriented mid-level design

- **Reuse**, aiming to identify (i) *off-the-shelf components* and (ii) *design patterns* to get the benefits of using existing solutions
- **Service specification**, aiming to describe each class interface *precisely*
- **Class & object model restructuring**, aiming to transform the detailed design model to improve its basic and constructive quality attributes (*e.g.*, understandability, extensibility)
- **Class & object model optimisation**, aiming to transform the detailed design model to improve its performance criteria (*e.g.*, response time, memory usage)

Responsibility-Driven Design (RDD)

A popular way of looking at mid-level, object-oriented design is in terms of *responsibilities*, *roles*, and *collaborations*

- A **responsibility**, *i.e.*, an abstraction of what an object does, is a contract or obligation
- Responsibilities are related to the obligations or to the behaviour of an object in terms of its **role**
- Responsibilities are implemented by means of methods that either act alone or have a **collaboration** with other roles

Responsibility types

Responsibilities are of two kinds:

- **Doing**
  - doing something itself, such as creating an object or doing a calculation
  - initiating actions in other objects
  - controlling and coordinating activities in other objects
- **Knowing**
  - knowing about private encapsulated data
  - knowing about related objects
  - knowing about things it can derive or calculate

The GRASP approach to detailed design

- GRASP = General Responsibility Assignment Software Patterns
- Focuses on *mid-level*, *object-oriented* design
- Views an object-oriented design as a *community of collaborating responsible objects*
- Is based on *patterns of assigning responsibilities*

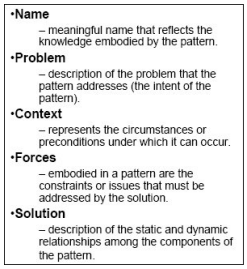
16.3 Detailed design patterns

Detailed design patterns

- Outline **reusable solutions** to general subsystem or unit-level problems, expressing them in terms of a few objects and their collaboration
- Address **changeability** issues and involve maintainability, extensibility, restructuring, and portability aspects of software
- Identify **relationships among software components**
- Describe the **responsibilities**, the **collaborations** and the **structural relationships** between these components
- Also capture practices that are demonstrably bad, documenting attempted solutions to recurring problems that proved unsuccessful (**antipatterns**)

Pattern structure and template

- Patterns are three-part rules that express a relation between:
  - a context
  - a *system of forces* which occurs repeatedly in that context
  - a *software configuration* allowing these forces to resolve themselves
- Templates are used to document patterns
- Different template styles emphasise different pattern aspects
- No consensus view on most appropriate template!
- There is agreement on a minimal template:



A classification of detailed design patterns

- Patterns first 'formalised' and classified in 1995 by Gamma, Helm, Johnson, & Vlissides (Gang of Four); subsequently known as the Gang of Four (GoF) patterns
- Patterns currently classified as:
  - **Creational**: concerned with the construction of object instances
  - **Structural**: concerned with the way classes and objects are organised
  - **Behavioural**: addressing the problems that arise when assigning responsibilities to classes and when designing algorithms

Principles underlying patterns

- abstraction
- encapsulation
- information hiding
- modularisation
- separation of concerns
- coupling and cohesion
- sufficiency
- completeness and primitiveness
- separation of policy and implementation
- separation of interface and implementation
- single point of reference
- divide and conquer

Pattern families

- GRASP patterns (9):
  - Creator
  - Information Expert
  - Low Coupling
  - Controller
  - High Cohesion
  - Polymorphism
  - Indirection
  - Pure Fabrication
  - Protected Variations
- GoF (Gang of Four) patterns:
  - A series of 23 advanced patterns
  - Only 15 of them are widely used

16.3.1 When to use a pattern

1. Read the pattern to get a complete overview
2. Analyse pattern structure, participants and collaborations
3. Examine the sample code to see an example of the pattern in use
4. Answer to the following questions:
   - Is there a pattern that addresses a similar problem?
   - Does the pattern trigger a more acceptable alternative solution?
   - Is there a simpler solution? If so, don't use the pattern!
   - Is the context of the pattern consistent with that of the problem?
   - Are the consequences of using the pattern acceptable?
   - Are there constraints imposed by the software environment that would conflict with the use of the pattern?

Example: the GRASP *creator* pattern

Name: Creator

Problem: Who should be responsible for creating a new instance of a class `A`?

Solution: Assign class `B` the responsibility to create an instance of class `A` if one of these is true (the more the better) -

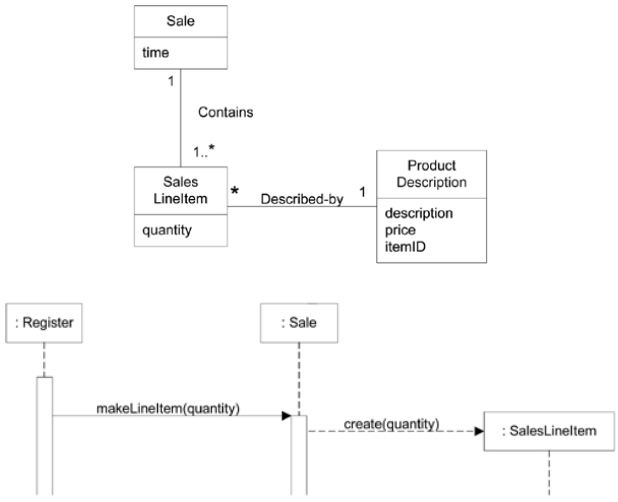
- `B` 'contains' or compositely aggregates `A`.
- `B` records `A`.
- `B` closely uses `A`.
- `B` has the initialising data for `A` that will be passed to `A` when it is created. Thus `B` is an *Expert* with respect to creating `A`.

`B` is a *creator* of `A` objects.

Note: This pattern focuses on responsibility assignment

Creator pattern application

Who creates a New `SalesLineItem` while the supermarket till generates a `Sale` receipt?



16.4 Preparing for the topic ahead

Next week. . .

A catalogue of patterns

We will focus on:

- GRASP patterns
- GoF patterns