2. Introduction to Design Patterns

*SW Design Patterns*,
by Boyan Bontchev,
FMI - Sofia University
2006/2011
Annotation

- Pattern origins and history
- Definitions
- Properties
- Types of patterns
- Describing design patterns
- Pattern language, system, catalog
- Reorganizing an object-oriented application using design patterns
- GoF patterns
References

- Gamma, Helm, Johnson, Vlissides (*"Gang of Four“ - GoF*) *Design Patterns: Elements of Reusable Object-Oriented Software*, 1995
- *THE DESIGN PATTERNS JAVA COMPANION*, by James W. Cooper, Addison-Wesley, October 2, 1998
- *Design Patterns Explained*, by Allan Shalloway and James Trott, Prentice Hall, 2001
- *Thinking in Patterns*, by Bruce Eckel, Revision 0.9, 5-20-2003
Pattern origins and history 1/3

- writings of architect Christopher Alexander (coined this use of the term "pattern" ca. 1977-1979) - books on the topic as it relates to urban planning and building architecture
- Kent Beck and Ward Cunningham, Textronix, OOPSLA'87 (used Alexander's "pattern" ideas for Smalltalk GUI design)
Pattern origins and history 2/3

- Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides ("Gang of Four" - GoF): *Design Patterns: Elements of Reusable Object-Oriented Software*, 1991-1994

- by Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal (Gang of Five): *Pattern-Oriented Software Architecture: A System of Patterns* (*also called the POSA book*)

- Pattern Languages of Program Design – with selected papers from the first and second conferences on Patterns Languages of Program Design (PLoP), by Addison-Wesley
Pattern origins and history 3/3

- J2EE Design Patterns
- .NET Design Patterns
- SOA Patterns - http://www.soapatterns.org
Definitions 1/2

- ... a design pattern is a general reusable solution to a commonly occurring problem in design. A design pattern is not a finished design that can be transformed directly into realization…[Wiki]

- ... describes a problem which occurs over and over again in our environment, 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 [Alexander]

- ... the abstraction from a concrete form which keeps recurring in specific non-arbitrary contexts [Riehle&Zullighoven, Understanding and Using Patterns in Software Development]
Definitions 2/2

- ...both a thing and the instructions for making the thing [Coplien, Software Design Patterns: Common questions and Answers]

- ...pattern can be viewed as a particular prose form of recording design information such that designs which have worked well in the past can be applied again in similar situations in the future [Beck & Coplien, Industrial experience with design patterns]

- ...a literary format for capturing the wisdom and experience of expert designers, and communicating it to novices
Architectural patterns

Structures may look different but still solve a common problem
What Is a SW OO Design Pattern

A design pattern is a description of communicating objects and classes that are customized to solve a general design problem in a particular context.

A pattern is made by four elements:

- name
- problem
- solution
- consequences
Name of Design Pattern

- Describes a design problem and its solutions in a word or two
- Used to talk about design pattern with our colleagues
- Used in the documentation
- Increase our design vocabulary
- Have to be coherent and evocative
Problem

- Describes when to apply the patterns
- Explains the problem and its context
- Sometimes includes a list of conditions that must be met before it makes sense to apply the pattern
- Has to occur over and over again in our environment
Solution

- Describes the elements that make up the design, their
  - relationships,
  - responsibilities and, as well,
  - collaborations
- Does not describe a concrete design or implementation
- Has to be well proven in some projects
Consequences

- Results and trade-offs of applying the pattern
- Helpful:
  - for describing design decisions and
  - for evaluating design alternatives
- Benefits of applying a pattern
- Impacts on a system’s *flexibility*, *extensibility* and/or *portability*
The 7 levels of SW architecture*

Global architecture

Enterprise architecture

System architecture

Application architecture

Macro-architecture

Micro-architecture

Objects

OO architecture

Subsystem

Frameworks

Design patterns

OO programming

* Mowbray and Malveau, 1997
How patterns do arise?

- Problem
- Context
- Solution
- Benefits
- Consequences
- Related Patterns
- Forces
Properties – patterns do:

- provide “a common vocabulary for designers to communicate, document, and explore design alternatives” (Gamma’93)
- describe software abstractions, ensuring wide applicability, and may provide hints about potential implementation issues provide “shorthand” for effectively communicating complex principles
- help document software architecture
- capture essential parts of a design in compact form
- show more than one solution
- describe software abstractions
Properties – patterns do not:

- provide an implementation
- provide an exact solution
- solve all design problems
- only apply for object-oriented design
Patterns everywhere

- Patterns cover all aspects of software engineering including: development organization, software process, project planning, requirements engineering, and software configuration, management...
Types of software patterns

- **design patterns (software design)**
  - architectural (systems design)
  - design (micro-architectures) [Gamma-GoF]
  - idioms (low level)

- **analysis patterns (recurring & reusable analysis models) [Flower]**

- **organization patterns (structure of organizations/projects)**

- **process patterns (software process design)**

- **domain-specific patterns**
A possible classification of patterns

- **Idioms** = implementation patterns

Ingredients

Pattern

Context

a design situation giving rise to a design problem

Problem

a set of forces (goals and constraints) occurring in that context

Solution

a form or rule that can be applied to resolve these forces

Example – window place

forces

- he wants to sit down and be comfortable
- he is drawn toward the light

solution

- in every room, make at least one window into a “window place”
Alexandrian form (canonical form) (1/2)

Name
meaningful name

Problem
the statement of the problem

Context
a situation giving rise to a problem (preconditions under which the problem and its solution seem to recur)

Forces
a description of relevant forces (goals) and constraints

Solution
proven solution to the problem

Examples
sample applications of the pattern
Alexandrian form (canonical form) (2/2)

**Resulting context** (force resolution)
the state of the system after pattern has been applied

**Rationale**
explanation of steps or rules in the pattern

**Related patterns**
static and dynamic relationship

**Known use**
occurrence of the pattern and its application within existing system
GoF format (1/2)

Pattern name and classification

Intent
what does pattern do / when the solution works

Also known as
other known names of pattern (if any)

Motivation
the design problem / how class and object structures solve the problem

Applicability
situations where pattern can be applied

Structure
a graphical representation of classes in the pattern
GoF format (2/2)

Participants
the classes/objects participating and their responsibilities

Collaborations
of the participants to carry out responsibilities

Consequences
trade-offs, concerns

Implementation
hints, techniques

Sample code
code fragment showing possible implementation

Known uses
patterns found in real systems

Related patterns
closely related patterns
Pattern templates

IF you find yourself in CONTEXT for example EXAMPLES, with PROBLEM, entailing FORCES THEN for some REASONS, apply DESIGN FORM AND/OR RULE to construct SOLUTION leading to NEW CONTEXT and OTHER PATTERNS

http://g.oswego.edu/dl/pd-FAQ/pd-FAQ.html

More pattern templates:
- http://hillside.net/patterns/template.html
- http://www.paterndepot.com/pages (Templates)
Pattern language [Coplien]

- A pattern language defines a *collection of patterns and the rules* to combine them into an *architectural style*, or...
- …is a structured collection of patterns that build on each other to transform needs and constraints into an architecture [Software Design Patterns: Common Questions and Answers]
- …describe software frameworks or families of related systems [Patterns Home Page -> Patterns Definitions]
Pattern catalogs and systems

[Buschmann, the POSA (Pattern-Oriented-Software-Architecture) book]

- **pattern catalog**
  
  …a collection of related patterns, where patterns are subdivided into small number of broad categories…

- **pattern system**
  
  …a cohesive set of related patterns, which work together to support the construction and evolution of the whole architecture…
Pattern design principles [4] (1/3)

- **Principle Of Least Astonishment (POLA)** - when two elements of an interface conflict, or are ambiguous, the behavior should be that which will least surprise user when the conflict arises.

- **Make common things easy, and rare things possible**

- **Consistency** - the more random rules you pile onto the programmer, rules that have nothing to do with solving the problem at hand, the slower the programmer can produce. And this does not appear to be a linear factor, but an exponential one.
Pattern design principles [4] (2/3)

- **Law of Demeter (Principle of Least Knowledge):** “Don’t talk to strangers.” An object should only reference itself, its attributes, and the arguments of its methods:
  - Each unit should have only limited knowledge about other units: only units "closely" related to the current unit.
  - Each unit should only talk to its friends; don't talk to strangers.

- **Subtraction:** a design is finished when you cannot take anything else away.
Simplicity before generality - “the simplest working solution is the best”; the best route to generality is through understanding well-defined specific examples.

Reflexivity - one abstraction per class, one class per abstraction. Might also be called Isomorphism.

Independence or Orthogonality - express independent ideas independently. This complements Separation, Encapsulation and Variation, and is part of the Low-Coupling-High-Cohesion message.

Once and once only - avoid duplication of logic and structure where the duplication is not accidental, i.e. where both pieces of code express the same intent for the same reason.
Reorganization using patterns

- See “Experiences using Design Patterns to Reorganize an Object-Oriented Application”, by Walter Zimmer
- Goal of reorganization - to eliminate deficiencies in the design and implementation; better maintenance

Diagram:

- Existing hypermedia application
- MacApp
- Application specific parts
- Generic, reusable parts
- Reorganized hypermedia application
- Reusable framework layer
- MacApp
Steps in reorganization (1/2)

PRELIMINARY STEPS

1. documentation
2. finding starting points
   - identification of classes / subsystems with design goals similar to ones of design patterns
   - experiences and future scenarios
   - metrics / design rules
   - analyzing the application for existing patterns

Hint for course projects
Steps in reorganization (2/2)

REORGANIZATIONAL STEPS

1. finding and exploring suitable design patterns
2. reconstructing and documentation
   - application classes corresponding to the design pattern
   - incorporate names of the application classes to the classes in the design pattern (e.g., LinkStrategy, HyperMediaDecorator)
Results and experiences

RESULTS

- drastic reduction of dependencies between subsystems
- short design documentation

EXPERIENCES

- common vocabulary - main advantage
- reorganization is time-intensive task
- good knowledge of design patterns needed
- combination of several design patterns required
The learning process [4]

There are found learning Design patterns is a multiple step process.

1. Acceptance - you accept the premise that design patterns are important in your work

2. Recognition - you recognize that you need to read about design patterns in order to know when you might use them

3. Internalization - you internalize the patterns in sufficient detail that you know which ones might help you solve a given design problem
Three most popular design patterns groups

*Design Patterns* is a catalog of 23 generally useful patterns for writing object-oriented software. 3 groups:

- **Creational patterns** are ones that create objects for you, rather than having you instantiate objects directly. This gives your program more flexibility in deciding which objects need to be created for a given case.

- **Structural patterns** help you compose groups of objects into larger structures, such as complex user interfaces or accounting data.

- **Behavioral patterns** help you define the communication between objects in your system and how the flow is controlled in a complex program.
Classification of Design Patterns

*By scope* - whether the pattern applies primarily on classes or objects:

- **Class patterns** deal with relationships between classes and their sub-classes. These relationships are established through inheritance, so they are static and focused on class relationships.

- **Object patterns** deal with object relationships, which can be changed at run-time and are more dynamic.
Classification of Design Patterns

By purpose - what a pattern does

- **Creational patterns**
  - Abstract the instantiation process
  - Make a system independent to its realization
  - Class Creational use inheritance to vary the instantiated classes
  - Object Creational delegate instantiation to an another object
Classification of Design Patterns

- **Structural patterns**
  - Class Structural patterns concern the aggregation of classes to form largest structures
  - Object Structural patterns concern the aggregation of objects to form largest structures
Classification of Design Patterns

- **Behavioral patterns**
  - Concern with algorithms and assignment of responsibilities between objects
  - Describe the patterns of communication between classes or objects
  - Behavioral class pattern use inheritance to distribute behavior between classes
  - Behavioral object pattern use object composition to distribute behavior between classes
## Design pattern catalog - GoF

<table>
<thead>
<tr>
<th>Scope</th>
<th>Class</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Creational</td>
<td>Structural</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Factory Method</td>
<td>Adapter</td>
</tr>
<tr>
<td></td>
<td>Abstract Factory</td>
<td>Bridge</td>
</tr>
<tr>
<td></td>
<td>Builder</td>
<td>Composite</td>
</tr>
<tr>
<td></td>
<td>Prototype</td>
<td>Decorator</td>
</tr>
<tr>
<td></td>
<td>Singleton</td>
<td>Facade</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Flyweight</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Proxy</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Let’s follow a paper for homework

- **Non-Software Examples of Software Design Patterns**, by Michael Duell, in AG Communication Systems e-zine

  [http://62.44.125.62/sdp/homework/Patterns%20Non-Software%20Examples%20of%20Software%20Design%20Patterns%20-%20AGCS.htm](http://62.44.125.62/sdp/homework/Patterns%20Non-Software%20Examples%20of%20Software%20Design%20Patterns%20-%20AGCS.htm)