Crash course on design patterns

Yann-Gaël Guéhéneuc
guehene@emn.fr

From Olivier Motelet's course (2001/10/17)

École des Mines
de Nantes, France

Object Technology
International, Inc., Canada

Design patterns

=

Tools for

object-oriented

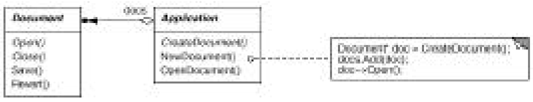
designers

Content

- Feeling
  - The *Factory Method* design pattern
- Seeing
  - Origins
  - Definition
  - Structure
- Touching
  - When and how to use design patterns
  - Tools supporting design patterns

Factory Method (1/3)

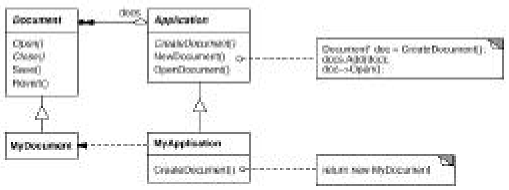
- **Need**: A framework for programs that support multiple kinds of documents
- **Problem**: A program only knows *when* it must create a new document, not *what* kind of document to create



```
Document* doc = CreateDocument();
docs.Add(doc);
doc->Open();
```

Factory Method (2/3)

- **Solution**: Isolate the knowledge of which document to create and move this knowledge out of the framework

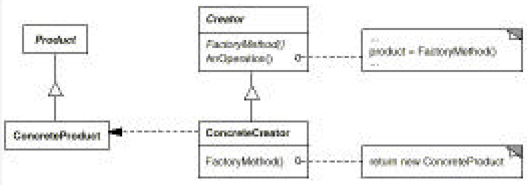


```
Document* doc = CreateDocument();
docs.Add(doc);
doc->Open();
```

```
return new MyDocument
```

Factory Method (3/3)

- Abstraction?
- Quality characteristics?



```
...
product = FactoryMethod()
...
```

```
return new ConcreteProduct
```

Origins (1/4)

"Each pattern 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 way that you can use this solution a million times over, without ever doing it the same way twice."

"Each pattern is a three part rule, which express a relation between a context, a problem, and a solution."

Christopher Alexander [Alexander, 1977]

Origins (2/4)

"The strict modeling of the real world leads to reflect today's realities but not necessarily tomorrow's. The abstractions that emerge during design are key to making a design flexible."

Erich Gamma [Gamma *et al*., 1994]

Origins (3/4)

- Complex systems involve several classes and their instances
- Towards the reuse of more than one class: Sets of collaborating classes
- What is a *good* design style?

Origins (4/4)

- A pattern is a named, reusable solution to a recurrent problem in a particular context

Definition (1/2)

“The description of communicating objects and classes customized to solve general design problem in a particular context.”

“Each design pattern lets some aspect of system structure vary independently of other aspects, thereby making a system more robust to a particular kind of change.”

[Gamma *et al.*, 1994]

Definition (2/2)

- A way to *enhance the reusability*
- A way to *encapsulate design experience*
- A *common vocabulary* among designers

Structure (1/3)

- Name
- Problem
- Solution
- Consequences

And much more (2/3)

- Problem + Consequence = Context
  - Intent, applicability, consequences
- Solution + Consequence = Strategies
  - Structure, participants, collaborations
- Understanding
  - Motivation, related patterns, known uses
- Use
  - implementation and sample code

But (3/3)

- Scattered information
  - Informal text
- A general example rather than a general rule

Interpreting them all...

When to use design patterns?

- When encountering complex problems?
  - Numerous design patterns (is there any complete list out there?)
  - Granularity
    - Requirements, analysis, architecture
    - Design, implementation (idioms)
    - Refactoring, testing
    - …

Knowing them all...

How to use design patterns?

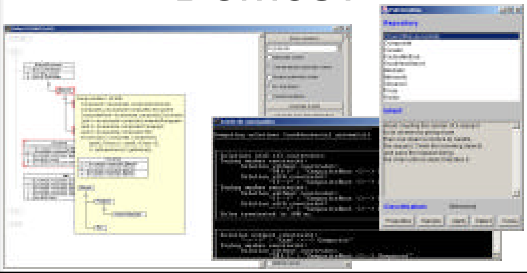
- Iterative induction process
  - From an example to an abstraction to an application to the abstraction to an application…
  - Validation process?
- Categories
  - Behavioural
  - Creational
  - Structural

Tools supporting design patterns

- "GoF" book
  - Lists, classifications, relationships
  - [Gamma *et al.*, 1996]
- CASE Tools
  - Fragments [Florijn *et al.*, 1997]
  - *PatternsBox* and *Ptidej* [Albin *et al.*, 2001]
- Navigation
  - *Tutor* [Motelet, 2000]

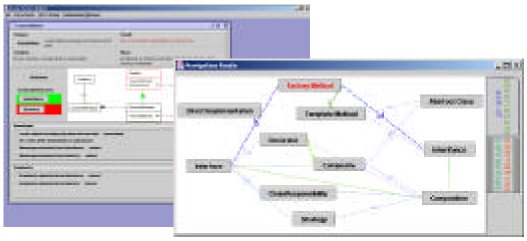
PatternsBox and *Ptidej*

Demos?



Tutor

Demos?



References

- [Alexander, 1977] Christopher Alexander ; *A Pattern Language* ; New York Oxford University Press, 1977.
- [Gamma et al., 1994] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides ; *Design Patterns – Elements of Reusable Object-Oriented Software* ; Addison Wesley 1994.
- [Florijn *et al.*, 1997] Gert Florijin, Marco Meijers, and Pieter van Winsen ; *Tool Support for Object-Oriented Pattern* ; Proceedings of ECOOP, 1997.
- [Albin *et al.*, 2001] Hervé Albin-Amiot, Pierre Cointe, Yann-Gaël Guéhéneuc, and Narendra Jussien ; *Instantiating and Detecting Design Patterns: Putting Bits and Pieces Together* ; Proceedings of ASE, 2001.
- [Motelet, 2000] Olivier Motelet ; *A Contextual Help System for Assisting Object-Oriented Software Designers in using Design Patterns* ; EMOOSE Master Thesis, 2000

That’s all folks!

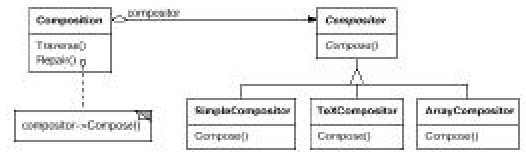
- Thank you so much for your attention!

- Questions?
- Comments?

Workshop on design patterns

- Three design patterns
  - Strategy
  - Decorator
  - Composite
- Purpose
  - Questions
  - Ideas
  - Explanations

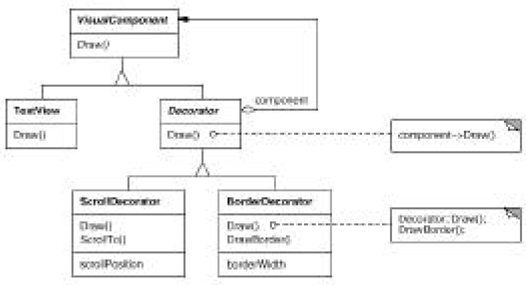
Strategy (1/2)



Strategy (2/2)

- What happens when a system has an explosion of Strategy objects? Is there some way to better manage these strategies?
- In the implementation section of this pattern, the authors describe two ways in which a strategy object can get the information it needs to do its job. One way describes how a strategy object could get a reference from the context object, thereby giving it access to context data. But is it possible that the data required by the strategy are not available from the context interface. How could you solve this problem?

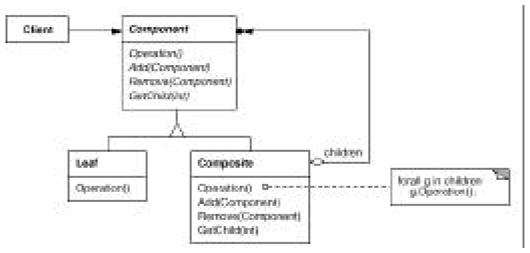
Decorator (1/2)



Decorator (2/2)

- The implementation section of this pattern states that a decorator object's interface must conform to the interface of the object it decorates. Consider an object *O*, that is decorated with an object *D*. Object *D* shares an interface with object *O* because object *D* « decorates » object *O*. If some instance of this decorator attempts to call a method *m* that is not part of *O*'s interface, does it mean that the object is no longer a decorator? Why is it important that a decorator object 's interface conforms to the interface of the object it decorates?

Composite (1/2)



Composite (2/2)

- How does the *Composite* design pattern help to consolidate system–wide conditional logic?
- Would you use this pattern if you do not have a part–whole hierarchy? In other words, if only a few objects have children and almost everything else in your collection is a leaf (a leaf has no children), would you still use this pattern to model these objects?

Relationships

- *Decorator – Strategy*
- *Decorator – Composite*
- *Composite – Decorator*

- Others
  - *Abstract Factory – Singleton*
  - …