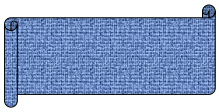


Design is not just what it looks like and feels like.
Design is how it works. –Steve Jobs

CSE403: Software Engineering

David Notkin
Winter 2009

Chunking

- Advanced chess players are in part superior because they don't see each piece individually
  - Instead, they chunk groups of them together
  - This reduces the search space they need to assess in deciding on a move
- This notion of chunking happens in almost every human endeavor
- Such chunking can lead to the use of idioms

High Level Languages

- High level programming languages can be viewed as providing idioms that have proven generally useful
- These high level constructs are sometimes more, constraining the ability to see the pieces
- Structures
  - Grouping together heterogeneous elements
- Structured loops
  - Grouping together disciplined uses of comparisons and branches
- Procedure call
  - Saving & restoring registers, jumps, …

Design patterns

- "a 'well-proven generic scheme' for solving a recurring design problem"
- Idioms intended to be "simple and elegant solutions to specific problems in object-oriented software design"
- A key to design patterns is that they are drawn from examples in existing systems
  - Not proposed solutions to possible problems, but real solutions to real problems

Language independent

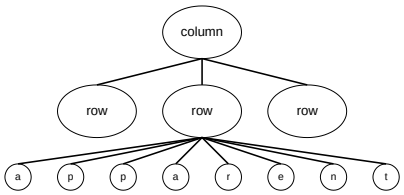
- They are language-independent
  - Although some language support is starting to exist in some cases
- Again, there is an analogy to high-level control structures
  - Knuth's 1974 article ("Structured Programming with go to Statements") shows that this is not a language issue alone

Low-level

- Patterns are a collection of "mini-architectures" that combine structure and behavior
- They are closely linked to the programming level
  - Information hiding is a (higher-level) design notion, which is often supported in programming languages
  - Layering has little direct link to the programming level

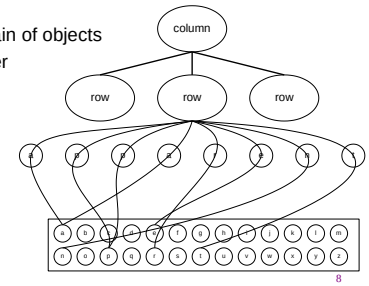
Example: flyweight pattern

- What happens when you try to represent lots of small elements as full-fledged objects?
- It's often too expensive
- And it's pretty common

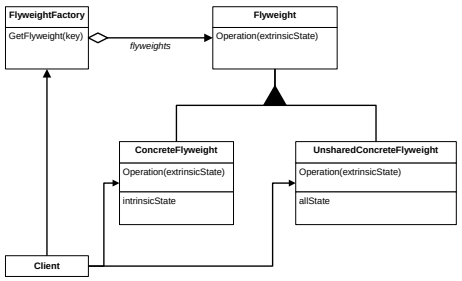


An alternative approach

- Use sharing to support many fine-grained objects efficiently
  - Fixed domain of objects
  - Maybe other constraints



Flyweight structure



Participants

- Flyweight (`glyph` in text example)
  - Interface through which flyweights can receive and act on extrinsic state
- ConcreteFlyweight (`character`)
  - Implements flyweight interface, shareable, only intrinsic state (independent of context)
- UnsharedConcreteFlyweight (`row`, `column`)
- FlyweightFactory
  - Creates and manages flyweight objects

Sample code

```
class Glyph {
public:
  virtual ~Glyph();virtual
  void Draw(…);
  virtual void SetFont(…);
  …
}
class Character : public Glyph {
  Character(char);
  virtual void Draw(…);
private:
  char _charcode;
};
```

- The code itself is in the domain (glyphs, rows, etc.)
- But it's structured based on the pattern
- The client interacts with `Glyph`, `Character`

A little more code

```
Character* GlyphFactory::CreateCharacter(char c)
  {
  if (!_character[c]) {
      _character[c] = new Character();
  }
  return _character[c];
}
```

- Explicit code for each of the elements in the flyweight structure

Defining a pattern

- Name and classification
- Intent
- Also known as
- Motivation
- Applicability
- Structure
- Participants
- Collaborations
- Consequences
- Implementation
- Sample code
- Known uses
- Related patterns

Classification of patterns

- Creational
  - Abstract factory, builder, factory method, prototype, singleton
- Structural
  - Adapter, bridge, composite, decorator, façade, flyweight, proxy
- Behavioral
  - Chain of responsibility, command, interpreter, iterator, mediator, memento, observer, state, strategy, template method, visitor

- Original GoF patterns

An historical aside

- The Gang of Four loosely based their initial work on that of architect Christopher Alexander
  - Not a systems or software architect, but an architecture architect (with planning, too)
  - *The Timeless Way* trilogy
    - *The Timeless Way of Building* (1979), *A Pattern Language: Towns, Buildings, Construction* (1977), *The Oregon Experiment* (1975)
- Not surprisingly, a focus on idiomatic solutions to common design problems

A little more

- Alexander and his influence on CS
  - `www.math.utsa.edu/sphere/salingar/Chris.text.html`
- Too much can be (and is) made of the connection to Alexander
  - In particular, Alexander takes the "big" view of architecture and patterns
  - In software, it is important but still the "little" view

An enlightening experience

- I had an experience with two of the Gang of Four
- They sat down with Griswold and me to show how to use patterns to (re)design a software design we had published
  - The rate of communication between these two was unbelievable
  - Much of it was understandable to us without training (a good sign for a learning curve)

The real thing

- Design patterns are not a silver bullet
- But they are impressive, important and worthy of attention
  - I think that some of the patterns have and more will become part and parcel of designers' vocabularies
  - This will improve communication and over time improve the designs we produce
  - The relatively disciplined structure of the pattern descriptions may be a plus

Final reminder

- Design patterns are highly unlikely to be on your mind now
- They are lower-level than the design you're thinking about at this stage
- They are probably elements that each of you, even within a team, can choose to use (or not) on an individual basis
- The idea is easy, but there is a learning curve

Questions?