CSC 335: Object-Oriented Programming and Design

Object-Oriented Design Patterns

Outline

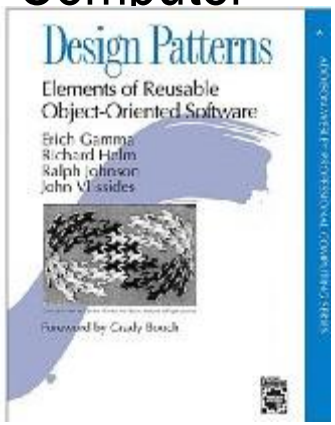
- Overview of Patterns
- Iterator
- Strategy

The Beginning

- Christopher Alexander, architect
  - A Pattern Language--Towns, Buildings, Construction
  - Timeless Way of Building (1979)
  - "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 a way that you can use this solution a million times over, without ever doing it the same way twice."
- Other patterns: novels (tragic, romantic, crime), movies genres,

“Gang of Four” (GoF) Book

- Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley Publishing Company, 1994
- Written by this "gang of four"
  - Dr. Erich Gamma, then Software Engineer, Taligent, Inc.
  - Dr. Richard Helm, then Senior Technology Consultant, DMR Group
  - Dr. Ralph Johnson, then and now at University of Illinois, Computer Science Department
  - Dr. John Vlissides, then a researcher at IBM
    - Thomas J. Watson Research Center
  - See WikiWiki tribute page http://c2.com/cgi/wiki?JohnVlissides



Patterns

- This book defined 23 patterns in three categories
  - *Creational patterns* deal with the process of object creation
  - *Structural patterns*, deal primarily with the static composition and structure of classes and objects
  - *Behavioral patterns*, which deal primarily with dynamic interaction among classes and objects

Documenting Discovered Patterns

- Many other patterns have been introduced documented
  - For example, the book **Data Access Patterns** by Clifton Nock introduces 4 decoupling patterns, 5 resource patterns, 5 I/O patterns, 7 cache patterns, and 4 concurrency patterns.
  - Other pattern languages include telecommunications patterns, pedagogical patterns, analysis patterns
  - Patterns are mined at places like Patterns Conferences

ChiliPLoP

- Recent patterns books work shopped at ChiliPLoP, Carefree Arizona
  - Patterns of Enterprise Application Arhitecture Martin Fowler
  - Patterns of Fault Tolerant Software, Bob Hamner
  - Patterns in XML Fabio Arciniegas
  - Patterns of Adopting Agile Development Practices Amr Elssamadisy
  - 2010: Patterns of Parallel Programming, Ralph Johnson
    - 16 patterns and one Pattern Language work shopped
      - Will be yet another book worksopped at ChiliPLoP

GoF Patterns

- *Creational Patterns*
  - Abstract Factory
  - Builder
  - Factory Method
  - Prototype
  - Singleton
- *Structural Patterns*
  - Adapter
  - Bridge
  - Composite
  - Decorator
  - Façade
  - Flyweight
  - Proxy
- *Behavioral Patterns*
  - Chain of Responsibility
  - Command
  - Interpreter
  - **Iterator**
  - Mediator
  - Memento
  - Observer
  - State
  - **Strategy**
  - Template Method
  - Visitor

Why Study Patterns?

- Reuse tried, proven solutions
  - Provides a head start
  - Avoids gotchas later (unanticipated things)
  - No need to reinvent the wheel
- Establish common terminology
  - Design patterns provide a common point of reference
  - Easier to say, “We could use Strategy here.”
- Provide a higher level prospective
  - Frees us from dealing with the details too early

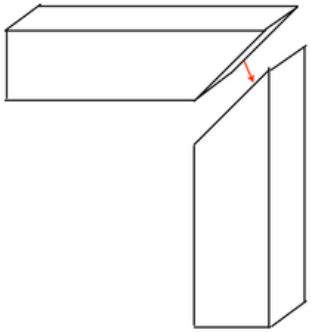
Carpenter's Conversation

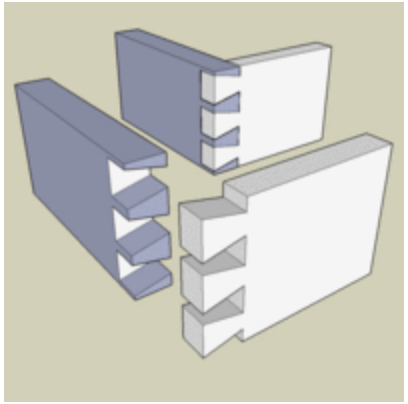
Adapted from Ralph Johnson

- How should we build the cabinet drawers?
- Cut straight down into the wood, cut back up 45 degrees a specific length, then go straight back down a specific length, the cut back up at 45 degrees .....

A Higher Level Discussion

- A high level discussion could have been:
  - "Should we use a miter joint or a dovetail joint?"
  - This is a higher, more abstract level
  - Avoids getting bogged down in details
- Which level of detail is more efficient?s





Consequences of which joint

- Dovetail joints
  - are more complex, more expensive to make
  - withstands climate conditions – dovetail joint remains solid as wood contracts and expands
  - independent of fastening system
  - more pleasing to look at
- Thoughts underneath this question are
  - Should we make a beautiful durable joint or a cheap and dirty one that lasts until the check clears?

http://www.infoq.com/interviews/Ralph-Johnson-Parallel-Programming-Patterns

Consequences

- Carpenters, patterns writers, and software developers discuss consequences
  - consequences simply refer to cause and effect
    - If we do this, what will happen – both good and bad
  - also known as the **forces** that patterns consider
    - Example: If we use Mediator to add and drop courses
      - Add an extra type that needs references to several objects
      - All of the logic and process is confined to one class so any change to the "rules" would be handled there
      - Reduces dependencies between others objects (simpler design when student does NOT tell the scheduled course to change)

Other advantages

- Most design patterns make software more modifiable, less brittle
  - we are using time tested solutions
- Using design patterns makes software systems easier to change
- Helps increase the understanding of basic object-oriented design principles
  - encapsulation, inheritance, interfaces, polymorphism

Style for Describing Patterns

- We will use this structure:
  - *Pattern name*
  - *Recurring problem:* what problem the pattern addresses
  - *Solution:* the general approach of the pattern
  - *UML for the pattern*
    - *Participants:* a description of the classes in the UML
  - *Use Example(s):* examples of this pattern, in Java

A few Patterns

- Coming p: Two OO Design Patterns
  - **Iterator** Design Pattern
    - access the elements of an aggregate object sequentially without exposing its underlying representation
  - **Strategy**
    - A means to define a family of algorithms, encapsulate each one as an object, and make them interchangeable

Pattern: *Iterator*

- Name: Iterator (a.k.a Enumeration)
- Problem: How can you loop over all objects in any collection. You don't want to change client code when the collection changes. You also want the same interface (methods)
- Solutions: 1) Have each class implement an interface. 2) Have an interface that works with all collections
- Consequences: Can change collection class details without changing code to traverse the collection

GoF Version of Iterator page 257

ListIterator
First() Next() IsDone() CurrentItem()

```
// Imaginary code
ListIterator<Employee> itr = list.iterator();
for(itr.First(); !itr.IsDone(); itr.Next()) {
    cout << itr.CurrentItem().toString();
```

Java version of Iterator

interface Iterator

boolean hasNext()

Returns true if the iteration has more elements.

Object next()

Returns the next element in the iteration and updates the iteration to refer to the next (or have hasNext() return false)

void remove()

Removes the most recently visited element

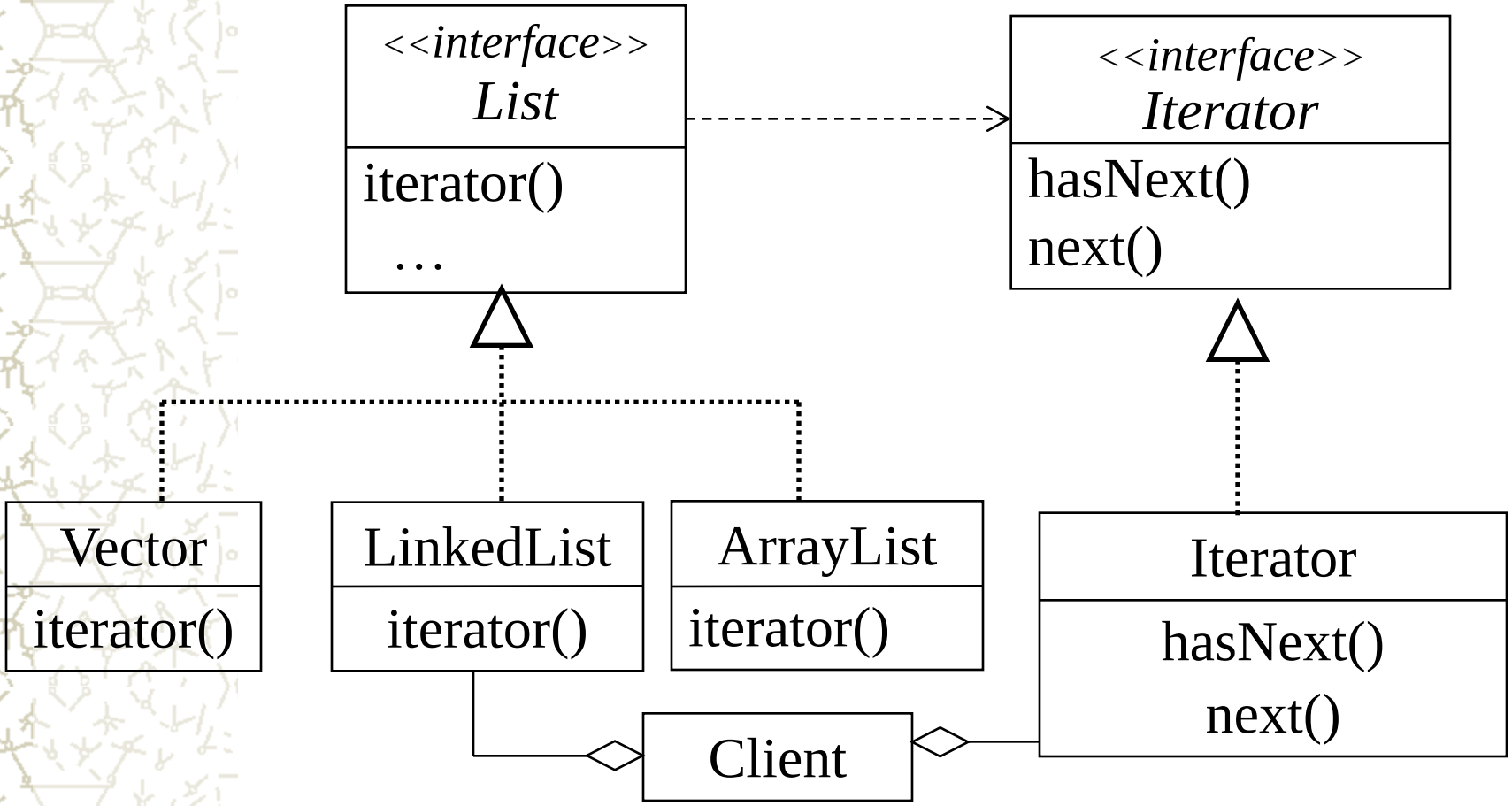
The Iterator interface in use

```
// The Client code
List<BankAccount> bank =
                    new ArrayList<BankAccount>();
bank.add(new BankAccount("One", 0.01) );

// ...
bank.add(new BankAccount("Nine thousand", 9000.00));

String ID = "Two";
Iterator<BankAccount> i = bank.iterator();
while(i.hasNext())  {
   if(i.next().getID().equals(searchAcct.getID()))
     System.out.println("Found " + ref.getID());
}
```

UML Diagram of Java's Iterator and Collections



http://download.oracle.com/javase/6/docs/api/java/util/List.html

Code Demo

Iterate over two different data structures

See iterators.zip on code demos page

Strategy Design Pattern

Strategy

Pattern: *Strategy*

- **Name**: Strategy (a.k.a Policy)
- **Problem**: You want to encapsulate a family of algorithms and make them interchangeable. Strategy lets the algorithm vary independently from the clients that use it (GoF)
- **Solution**: Create an abstract strategy class (or interface) and extend (or implement) it in numerous ways. Each subclass defines the same method names in different ways

Design Pattern: *Strategy*

- Consequences:
  - Allows families of algorithms.
- Known uses:
  - Layout managers in Java
  - Different Poker Strategies in a 335 Project
  - Different PacMan chase strategies in a 335 Project
  - TextField validators in dBase and Borland OWL:
    - Will use different algorithms to verify if the user input is a valid integer, double, string, date, yes/no.
    - Eliminates conditional statements

Java Example of Strategy

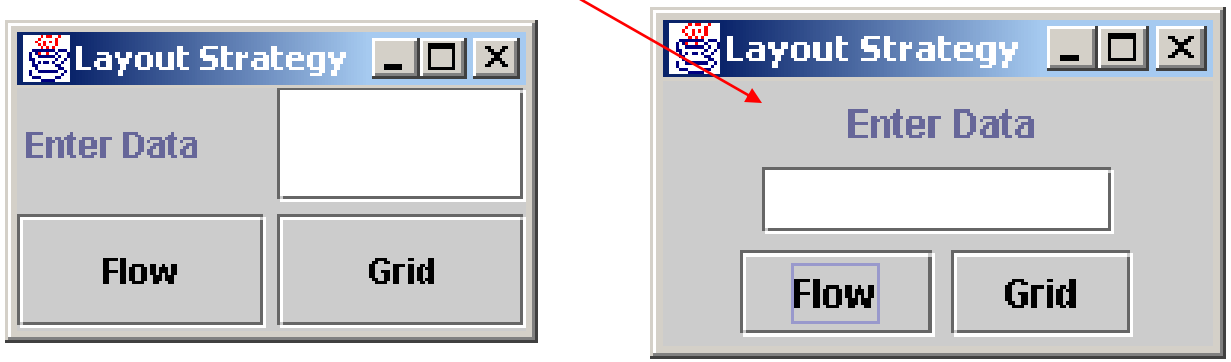
```
this.setLayout(new FlowLayout());
this.setLayout(new GridLayout());
```

- In Java, a container HAS-A layout manager
  - There is a default
  - You can change a container's layout manager with a **`setLayout`** message

Change the stategy at runtime

- Demonstrate LayoutControllerFrame.java

```
private class FlowListener
        implements ActionListener {
 // There is another ActionListener for GridLayout
 public void actionPerformed(ActionEvent evt) {
  // Change the layout strategy of the JPanel
  // and tell it to lay itself out
  centerPanel.setLayout(new FlowLayout());
  centerPanel.validate();
 }
}
```

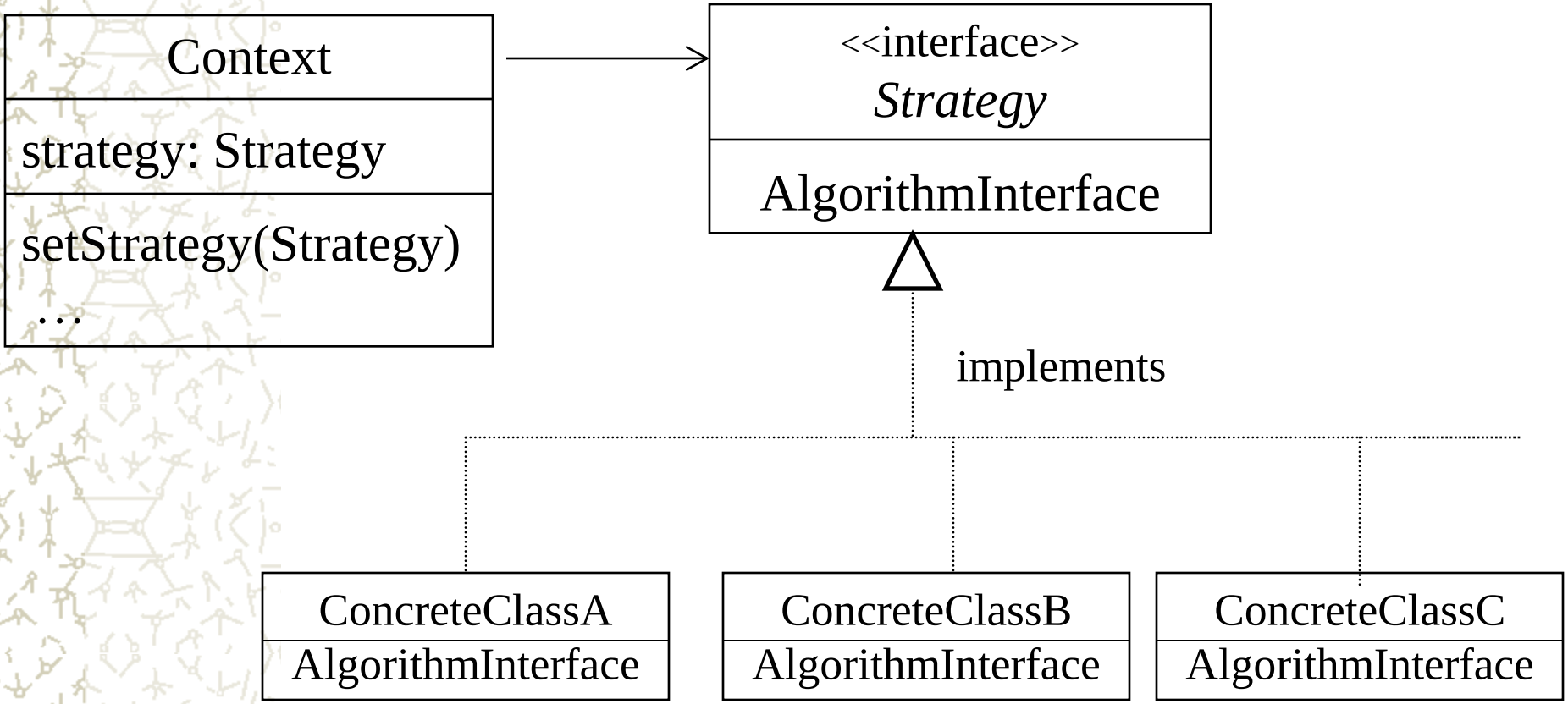


interface LayoutManager

- Java has interface java.awt.LayoutManager
- Known Implementing Classes
  - GridLayout, FlowLayout, ScrollPaneLayout
- Each class implements the following methods

```
addLayoutComponent(String name, Component comp)
layoutContainer(Container parent)
minimumLayoutSize(Container parent)
preferredLayoutSize(Container parent)
removeLayoutComponent(Component comp)
```

UML Diagram of Strategy General Form



Specific UML Diagram of LayoutManager in Java