CIS 211

Design Patterns

Based on: http://csis.pace.edu/~bergin/patterns/strategydecorator.html
By Joseph Bergin, Pace University

Thinking Big

Typical CIS 211 assignment: 100 lines of code
Windows XP: 45 million lines of code

How do you scale up to big problems?

Break the problem up into pieces of code that are small and easy to change.

(See also CIS 422/522, which talks all about how to build software.)

How to break things into pieces?

- Loose coupling: Limit the dependencies among different classes so that you can change one without changing the others (very much).

- A class should be responsible for just one thing. Example: Don’t buy a combo TV/DVD player.

- High-level classes should not depend on low-level details. Example: Use List instead of ArrayList.

- Keep interfaces as simple as possible. Example: List, Comparable, ActionListener.

- Sub-classes should work in place of the super-class. (Liskov substitution principle.)

There is much, much more to be said on object-oriented design!

Design Patterns

- Origin: “Design Patterns: Elements of Reusable Object-Oriented Software” by Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides (Gang of Four) 1994.

- Common idioms for designing classes that work well together

- A Few Examples:
  - Abstract factory
  - Factory method
  - Singleton
  - Observer
  - Iterator
  - Strategy
  - Decorator
  - Composite
  - Adapter
### Iterator Pattern

- **Purpose:** Access elements in a data structure sequentially.
- **How It Works:** Helper class works with the data structure internals, and exposes a common interface.
- **Example:**

```java
void printAll(Collection<String> stuff) {
    Iterator<String> i = stuff.iterator();
    while (i.hasNext()) {
        System.out.println(i.next());
    }
}
```

(C++ collections are also accessed with iterators.)

### Iterator Pattern

- **Purpose:** Access elements in a data structure sequentially.
- **How It Works:** Helper class works with the data structure internals, and exposes a common interface.
- **Example:**

```java
void printNames(List<String> firstNames, List<String> lastNames) {
    Iterator<String> firsti = firstNames.iterator();
    Iterator<String> lasti = lastNames.iterator();
    while (firsti.hasNext() && lasti.hasNext()) {
        System.out.println(firsti.next() + " " + lasti.next());
    }
}
```

### Observer Pattern

- **Purpose:** Allow multiple clients to be notified about updates to a particular object.
- **How It Works:** The observed object class implements a "subscribe" method that accepts a client, and each client implement a notification method.
- **Example:**

```java
public class Noisy implements ActionListener {
    public void actionPerformed(ActionEvent e) {
        System.out.println("Button was clicked!");
    }
}

JButton b = new JButton("Click me!");
b.addActionListener(new Noisy());
```

### Strategy Pattern

- **Purpose:** Separate out an algorithm that may vary from instance to instance or need to be selected at runtime.
- **How It Works:** Encapsulate algorithm as classes with a common interface to allow plug-and-play programming!
- **Example:** Suppose you want to read in a list of words and then print only...
  - Palindromes ("racecar", "bob", "madam")
  - or: Words that begin with 't' ("terrific", "tree", "Tarzan")
  - or: Words that have exactly four letters ("bugs", "love", "code")

How can we minimize redundant code?

*Example by: Joseph Bergin, Pace University, jbergin@pace.edu*
Case 1: Palindromes

```java
public void printWhenPalindromes(String filename) throws IOException {
    Scanner infile = new Scanner(new FileReader(filename));
    // Read in lines one at a time
    while (scanner.hasNextLine()) {
        String currLine = scanner.nextLine();
        StringTokenizer words = new StringTokenizer(currLine);
        while (words.hasMoreTokens()) {
            String word = words.nextToken();
            // Print line only if it contains a matching word
            if (isPalindrome(word)) {
                System.out.println(currLine);
                break;
            }
        }
    }
}
```

Case 2: Begins with ‘t’

```java
public void printWhenBeginsWithT(String filename) throws IOException {
    Scanner infile = new Scanner(new FileReader(filename));
    // Read in lines one at a time
    while (scanner.hasNextLine()) {
        String currLine = scanner.nextLine();
        StringTokenizer words = new StringTokenizer(currLine);
        while (words.hasMoreTokens()) {
            String word = words.nextToken();
            // Print line only if it contains a matching word
            if (word.charAt(0) == 't' || words.charAt(0) == 'T') {
                System.out.println(currLine);
                break;
            }
        }
    }
}
```

Case 3: Four letter words

```java
public void printFourLetterWords(String filename) throws IOException {
    Scanner infile = new Scanner(new FileReader(filename));
    // Read in lines one at a time
    while (scanner.hasNextLine()) {
        String currLine = scanner.nextLine();
        StringTokenizer words = new StringTokenizer(currLine);
        while (words.hasMoreTokens()) {
            String word = words.nextToken();
            // Print line only if it contains a matching word
            if (word.length() == 4) {
                System.out.println(currLine);
                break;
            }
        }
    }
}
```

Attempt 1

```java
public void printWhen(String filename, int choice) throws IOException {
    Scanner infile = new Scanner(new FileReader(filename));
    // Read in lines one at a time
    while (scanner.hasNextLine()) {
        String currLine = scanner.nextLine();
        StringTokenizer words = new StringTokenizer(currLine);
        while (words.hasMoreTokens()) {
            String word = words.nextToken();
            // Print line only if it contains a matching word
            if (choice == 1) {
                if (isPalindrome(word)) {
                    System.out.println(currLine);
                    break;
                }
            } else if (choice == 2) {
                if (word.charAt(0) == 't' || words.charAt(0) == 'T') {
                    System.out.println(currLine);
                    break;
                }
            } else {
                if (words.length() == 4) {
                    System.out.println(currLine);
                    break;
                }
            }
        }
    }
}
```
Strategy Pattern

```java
public void printWhen(String filename, CheckStrategy strategy) throws IOException {
    Scanner infile = new Scanner(new FileReader(filename));
    // Read in lines one at a time
    while (scanner.hasNextLine()) {
        String currline = scanner.nextLine();
        StringTokenizer words = new StringTokenizer(currline);
        while (words.hasMoreTokens()) {
            String word = words.nextToken();
            // Print line only if it contains a matching word
            if (strategy.check(word)) {
                System.out.println(currline);
                break;
            }
        }
    }
}
```

Strategy Implementation (1)

```java
public interface CheckStrategy {
    public boolean check(String s);
}

public class StartsWithT implements CheckStrategy {
    public boolean check(String s) {
        return s.charAt(0) == 't' || s.charAt(0) == 'T';
    }
}

public class FourLetters implements CheckStrategy {
    public boolean check(String s) {
        return s.length == 4;
    }
}

printWhen("hamlet.txt", new StartsWithT());
printWhen("hamlet.txt", new FourLetters());
```

Strategy Implementation (2)

```java
public interface CheckStrategy {
    public boolean check(String s);
}

public class NumLetters implements CheckStrategy {
    private int numLetters;
    public NumLetters(int numLetters) {
        this.numLetters = numLetters;
    }
    public boolean check(String s) {
        return s.length == numLetters;
    }
}

printWhen("hamlet.txt", new NumLetters(5));
```

Decorator Pattern

- **Purpose:** Add functionality to an existing class.
- **How It Works:** Contains the original object, but adds some new functionality.

Example by: Joseph Bergin, Pace University, jbergin@pace.edu
Decorator Pattern

• **Purpose:** Add functionality to an existing class.
• **How It Works:** Contains the original object, but adds some new functionality.
• **Example:** Suppose you want to count the words while checking to see if they satisfy the criterion. How can we accomplish this?

• Other Examples:
  – Adding a peek() method to a Queue class
  – Adding scroll bars to a Window class

Example by: Joseph Bergin, Pace University, jbergin@pace.edu

Decorator Example

```java
class CounterDecorator implements CheckStrategy {
    private int count = 0;
    private CheckStrategy checker;
    public CounterDecorator(CheckStrategy check) {
        checker = check;
    }
    public boolean check(String s) {
        boolean result = checker.check(s);
        if (result)
            count++;
        return result;
    }
    public int count() { return count; }
}

CounterDecorator counter = new CounterDecorator(new NumLetters(5));
printWhen("hamlet.txt", counter);
System.out.println("Hamlet has "+ counter.count() + " lines with 5-letter words.");
```

Composite Pattern

```java
Suppose we want 5-letter-words starting with t:
class AndStrategyComposite implements CheckStrategy {
    private List<CheckStrategy> tests = new LinkedList();
    public void addStrategy(CheckStrategy s) {
        tests.add(s);
    }
    public boolean check(String s) {
        for (CheckStrategy strategy: tests) {
            if(!strategy.check(s)) return false;
        }
        return true;
    }
}
CheckStrategy tAndFive = new AndStrategyComposite();
tAndFive.add(new BeginsWithT());
tAndFive.add(new NumLetters(5));
printWhen("hamlet.txt", tAndFive);
```
• **Question:** Suppose we don’t have a PriorityQueue\(<E>\) class and want to create one using a TreeMap\(<E>\). How?

• **Answer:** Use a TreeMap\(<E>\) internally to implement the PriorityQueue\(<E>\) methods (add, remove).

• **Alternate Answer:** Inherit from TreeMap\(<E>\) and implement Queue interface.