Patterns, Vectors, Sequences

October 17th 2002

Overview

- Today:
  - Software Design Patterns
  - Vectors
  - Sequences
  - BubbleSort algorithm

Patterns

Design Patterns

?Today: What is a Design Pattern?
  - ? What can they be used for?
  - ? What does it look like?

?Literature:

?“Design Patterns”, Gamma et al, Addison Wesley,1995. *the classic*

?“Software Architecture”, Shaw & Garlan, Prentice Hall, 1996.

?“Object-oriented Design and Patterns” W. Pree, Addision-Wesley, 1995.

?Advanced software engineering courses such as *Software Architecture* course taught by Hausi Mueller.

Definition of Patterns

- A *Design Pattern* is a description of a standard solution for a standard problem in design.
  - Goal: Reuse of design information
  - A pattern must not be “new”!
- What do we want from a design?
  - reusability
  - general design concepts
  - a specific implementation
  - flexibility
  - reliability
  - comprehensibility

Why Design Patterns?

- Classes used to be the main element in the design of a software system
  - now, they are more implementation specific.
  - for example, is it simple to ‘re-use’ a class?
  - what are some areas that might run into problems re-using classes, even in the ‘cross-platform’ Java language?
- Instead, shift to a collection of classes and objects which address a particular domain and problem
  - we term this a framework
  - a design pattern can describe how to apply and construct this framework

Motivation for Patterns

- Design Patterns seek to avoid the *tyranny of the new*
  - the tendency of people to always seek new solutions to old problems.
  - Design Patterns codify previous experience and 'hair-pulling' so others can benefit.
  - develop a community of practice for software engineers.
- Major types of Patterns:
  - Solution Patterns, which show how to solve a problem
    - structural, behavioral, creational...
  - Problem Patterns or Anti-Patterns
    - these address things to avoid or to watch for.
    - example: transitioning from procedural code (C, Pascal) to object-oriented paradigms (e.g. the Blob).

Pattern Formats

- Patterns are described in numerous places: web repositories, textbooks, papers
  - problem of capturing patterns
- Several different formats for describing patterns
- Example format:
  - **Name**: to increase the vocabulary
  - **Problem**: when to apply this pattern (and when not to)
  - **Solution**: how to implement, as a template, not as specific directions. Includes the basic elements of the design.
  - **Consequences**: results, trade-offs, cost, etc.

Example of a Pattern

- **Name**: Adapter (p. 91)
- **Problem**: we want to reuse functionality from one class in another, but with slight differences
- **Solution**: incorporate that other class as member and modify the operations on that member in the *adapted class*
- ***Consequences**: some code is duplicated. Readability is improved. Implements information hiding principle encapsulation.*
- *AddressBook example from Eclipse...*

Vectors

text pp. 185-193

What is a Vector

? Skipped vectors last time we met
  - I thought they were superseded in the Java API
  - turns out they've been updated

? Can be very useful, particularly in their simplicity

? Think of them as extended, resizeable arrays
  - plus easy to do insertions

? Compare Vectors, ArrayList, and LinkedList

Type	**Get**	**Iteration**	**Insert**	**Remove**
array	*1430*	*3850*	*n/a*	*n/a*
Vector	*4890*	*16250*	*550*	*46850*
ArrayList	*3070*	*12200*	*500*	

Vector Operations

- Notion of rank of an object:
  - the *rank of an object e in a sequence S is the number of elements which are before e in S*
- Book operations are slightly different
- the Java API method names
  - size()
  - isEmpty()
  - get(rank)
  - set(rank, element) //replace
  - add(rank, element) //insert
  - remove(rank)

Vector Implementation

- Extended Array
  - simple to *get or set*
  - requires shifting elements when inserting or removing at a rank
    - we have to remove the element then repack the array
    - if we overflow the array, we must resize it
      - but we can't resize arrays?
      - double the array size and copy the old data, then add the element.
  - *running times?*

Sequences

pp. 206-211

Sequences

- Abstract Data Type which merges Lists and Vectors
  - i.e. implements methods of both.
  - think of it as an Adapter for the two.
  - recall Lists refer to positions or nodes – the location of an element in a list irrespective of rank
  - a Sequence is basically the combination of List and Vector into one class
    - therefore there is an array implementation, and a doubly-linked list implementation
    - tradeoffs?

Bubblesort example

- Bubblesort – a slow sorting algorithm
  - but it works!
  - Formally: consider a sequence of *n elements that can be compared using an order relation. Sorting this sequence means to order it such that the elements are in non-decreasing order.*
- Our implementation:
  - uses an array of integers to sort
  - use the Comparator interface to sort other classes
    - compareTo(), isEqual()

Basic idea

- make a series of passes over the array, swapping elements which are out of order.
  - (i.e. a > a+1)
- implications: on the *n*th pass, *the* largest element is always moved to the position (number of elements) – n
- we need at most how many passes?
- and what is the big-O running time?