Design Patterns Introduction

Jim Fawcett

CSE776 – Design Patterns

Fall 2017

Design Patterns Class

- Run in Seminar Style
  - Each class consists of student presentations of design patterns
  - Each presentation is followed by a discussion of the material presented
- We will cover all 23 patterns in the text plus additional material.
- I will serve as moderator and organizer.
- You will do all of the work preparing for and delivering presentations on each pattern
- I will present a few of the patterns.
  - First pattern today
  - About 20 % of the remainder

Your Responsibilities as Presenters

- Prepare a Power-Point presentation for delivery in class following the pattern format discussed in the text.

- Prepare two pieces of C++, C#, or Java code.
  - an operational skeleton which compiles, links, and runs, but has no more code than absolutely necessary to illustrate how the pattern works
  - a more detailed example, just large enough to show how the pattern is used in a realistic context

- Deliver your presentation using guidelines discussed in subsequent slides today.

- Lead a discussion of the pattern.
  - prepare a list of questions, assertions, and issues to use for this part -- don't make slides for this part, just use the notes to help you organize and lead the discussion
  - I will help, but it's your responsibility to get the discussion going.

Grades

- **Grades are based on:**
  - your presentation
    - but not affected by your command of the English language
  - your code samples
    - these will be put on a class directory, accessible from ECS cluster: www.ecs.syr.edu/faculty/fawcett/handouts/CSE776
  - the closing discussion you lead, focused on your pattern
  - Your participation in discussions of patterns presented by other students
- Most patterns will be presented by three or four students.
- Usually two students prepare and present the pattern-based part of the presentation.
- Twp other members of the team prepare and present code examples.
- The specifics of how you do this is up to you and your presentation partner(s) .

Patterns

- A pattern is a model of a software component which has a specific structure allowing it to successfully solve some set of design problems.

- Patterns convey their message with text and diagram descriptions of a specific design idiom at the architectural and implementation levels.
  - Architecture is shown using class diagrams and object relationships in the OMT notation.
  - Implementation is shown with sample code fragments.

- A pattern provides:
  - a name which, given a catalog of patterns, allows designers to communicate precisely about their designs.
  - a statement which describes a design problem the pattern is trying to handle.
  - a solution in terms of architecture and implementation.
  - a brief description of each of the collaborators in the pattern.
  - optionally, a critique describing the strengths and weaknesses of the pattern.

Uses

- Patterns are used to describe:
    - software architectural components
      (class text)
    - language specific patterns and design idioms
    - frameworks
      (like Microsoft Foundation Classes - MFC)
    - software architectures
        - systems and distributed processing
    - process and organization
    - business objects

- Sources of patterns:
    - Design Patterns, Gamma, Helm, Johnson, Vlissides
      Addison-Wesley, 1995
    - Pattern Languages of Program Design (PLOP)
      conference proceedings, vol. 1, 2, 3, or 4
      on restricted hold in Sci Tech Library, Carnagie
    - Books held in Sci-Tech Library
    - C++ Report, many articles in back issues

Pattern Grammar

- Intent
  - the purpose of this pattern
- Motivation
  - an example application
- Forces (not in class text)
  - conflicting forces and constraints
- Applicability
  - when would you expect to use this pattern
- Structure
  - what is the static structure of this pattern
  - expressed using class diagram(s) to show logical structure
- Participants
  - name each component and tell what it does - this text goes along with the diagram shown in Structure.
- Collaborations
  - how do the participants interact with each other and with the client – text, often accompanied by a sequence diagram.
- Consequences
  - what are the advantages and disadvantages of using this pattern
- Implementation
  - code fragments showing how to use the pattern
  - a complete example
- known uses
  - where has this pattern been used before?
- related patterns

Presentation Pattern

- The next few slides are concerned with making good presentations.

- They are presented in the pattern format, using pattern grammar.

- They are not an ideal example of how patterns are used, but will get us used to the pattern ideas and grammar.

- Later today I will present the first of the patterns covered in this course.
    - Intended to introduce you to patterns
    - I'll make some remarks about how to present patterns as that presentation unfolds.
    - My presentation should be a good example of how to present in this class.

Presentation Pattern

- Intent:
  - help you avoid common idiocies most of us fall into from time to time as we prepare and give technical presentations
- Motivation:
  - The presentations we will all been giving in CSE776 are typical of those you will be required to give early in your professional careers.
- Applicability:
  - You may be asked to give presentations to:
    - report progress
    - communicate bad news
    - sell a product, service or idea
    - recommend a strategy
    - train other technical people
    - make other general presentations

    The pattern structure works for all of these.
- Structure:
  - The “Design Patterns” book uses a very specific structure that we will follow throughout this course.
  - When giving professional presentations this pattern often will fit well – it is always a good place to start.
  - See also the Microsoft suggested Content examples.

Presentation Pattern

- Participants:
  - you and the audience
    - do you tailor your presentation for yourself?
    - or your audience?
- Collaborators:
  - presenter makes eye contact, talks with moderate pace to the back of the room, and frequently asks questions
- your talk is successful if the audience:
  - pays attention
  - asks questions
  - argues
  - gets emotionally involved, e.g., intrigued, excited, angry, pleased
- your talk is not successful if the audience:
  - goes to sleep
  - sits in stony silence
  - carries on parallel conversations
  - gets up and leaves

Presentation Pattern

- Consequences:
  - if successful you get one or more of the following:
    - promoted
    - a salary increase
    - a pat on the back
    - your boss takes the credit
  - if unsuccessful you get one or more of these:
    - no raise
    - your work station replaced with a 486 Windows 95 machine
    - the opportunity to seek new employment

Presentation Pattern Implementation

- Sign post:
  - tell them what you’re going to tell them
  - tell them
  - tell them what you told them
- Limit detail:
  - no more than five items per level
  - usually no more than two levels
- Don’t read your slides:
  - best to leave off details
  - use brief bullets
  - verbalize the details in your own words, using notes if you need to
  - have back up slides with details if your audience asks questions
  - leave unexplored packets of information with your audience
- Never, never, never, never, never apologize.

Presentation Pattern Implementation (continued)

- Encourage questions:
  - stop frequently and ask questions of audience
  - badger them
- say outrageous things
- Pick out three or four people in the audience:
  - speak directly to them in near conversational manner
  - adjust your pace based on their reactions
  - look them in the eyes
- Keep it interesting:
  - Tell one or two really corny jokes.
  - wave you arms, walk around, gesture (politely)
  - vary your pitch and volume
- Be as positive and optimistic as you can be.
- Rules of thumb:
  - allow 3 minutes per slide
  - don’t time yourself during the talk.
  - plan 3 to 5 hours of preparation for each hour of delivery
- Stop before you get boring
  (unless you’re avoiding a quiz).

Presentation Pattern

- Known Uses:
  - You will make many presentations during your career
    - Selling ideas to your boss and customers
    - Providing status reports on the work of your team
    - Participating in interviews
      - As candidate
      - As recruiter
    - Demonstrating finished work
    - Training new hires

Presentation Pattern

End of Pattern

Next - More about Patterns

Pattern Types

- **Creational Patterns**

  defer some part of the object creation process to subclasses or to other objects

  - abstract factory
    provide interface for building related or dependent objects without specifying their concrete classes
  - builder
    separate construction from representation so one construction process can build many representations
  - factory method
    define interface for creating object but let subclasses decide which object to create
  - prototype
    specify object to create using a prototype and construct by cloning
  - singleton
    ensure class has only one object

Pattern Types

- **Structural**

  Describe useful ways of building inheritance hierarchies or assembling objects to deal effectively with some design problem.

  - adapter (wrapper)
    wrap a new interface around an existing class or module
  - bridge (handle/body)
    decouple abstraction from its implementation so that each can change independently
  - composite
    build recursive structure representing part/whole com-positions
  - decorator
    attach responsibilities to an object dynamically
  - facade
    provide one interface for a set of objects with logically connected but different interfaces
  - flyweight
    use state sharing to support use of many fine-grained objects
  - proxy
    provide surrogate object to another to control access to it

Pattern Types

- **Behavioral**

  group classes or objects into patterns which perform some task in a particularly effective way

  - chain of responsibility
    decouple requestor from receiver by allowing more than one object to handle request
  - command
    encapsulate request as object, separating request from execution, and dynamically binding invoker with receiver
  - interpreter
    represent and process a grammar
  - iterator
    access container elements sequentially without breaking encapsulation
  - mediator
    lets objects communicate without knowing about each other explicitly

Pattern Types

- **Behavioral** patterns (continued)
  - memento
    capture and return a state (or partial state) snapshot supporting undo and checkpointing
  - observer
    when one object changes state all dependents are notified and updated
  - state
    represent finite state machine (remember elevator simulation)
  - strategy
    define a public interface for a family of algorithms which will be used interchange-ably
  - template
    factor common steps of a family of algorithms into a base class and define subclasses to complete the family
  - visitor
    represent an operation to be performed on each element of a container

Additional Patterns

- Plop 4 Conference
  - Pattern Languages Of Programming
- Books on hold in Sci-Tech Library
  - Design Patterns, Gamma et. al., Addison-Wesley, 1995
  - Refactoring to Patterns, Kerievsky, Addison-Wesley, 2005
  - Patterns of Enterprise Application Architecture, Fowler, Addison-Wesley, 2003
  - Enterprise Integration Patterns, Hohpe and Woolf, Addison-Wesley, 2004
  - Pattern Oriented Software Architecture, Schmidt, Wiley, 2000
  - Head First Design Patterns, Freeman and Freeman, OReilly,2004

Patterns Support Change

- Design patterns can help to avoid massive redesign when faced with the need for change.
- Creating an object using a class name commits you to an implementation as well as an interface. To avoid this create objects indirectly:

  abstract factory, factory method, prototype

- Specifying a request by name commits you to one specific member operation. You can avoid specific requests by using:

  chain of command, command

- If clients know how an object is represented or implemented, or where it is located, then the client may need to change if the object changes. This kind of information can be hidden using the patterns:

  abstract factory, bridge, memento, proxy

- Objects that depend on algorithms have to change when the algorithm changes. The algorithms can be isolated using:

  builder, iterator, strategy, template, visitor

Patterns Support Change

- Tightly coupled classes mean you can't remove or change a class without understanding and changing many other classes. Loose coupling is supported by the patterns:

  abstract factory, bridge, chain of responsibility, command, facade, mediator, and observer

- Extending functionality by subclassing is not always easy. Object composition and delegation provide an alternate flexible means for extending functionality. Many of the patterns, discussed in the class text, allow you to customize by defining one subclass and composing its objects with existing classes. See:

  bridge, chain of responsibility, composite, decorator, observer, strategy

- If you need to modify behavior of a class, but can't directly do so conveniently (perhaps there are too many subclasses) try some of the patterns:

  adapter, decorator, visitor

A Surprise

- You may be surprised that the Design Patterns book does not use C++ templates
  - Templates were just being introduced at the time the book was published.
  - Very few compilers implemented them correctly, if at all, at that time.
  - You are encouraged to implement patterns in your demonstrations using templates.

Design Patterns Course

- There are no examinations.
- You can expect to make several presentations.
- Only small amounts of coding are required.
  - The smallest demo you can devise that implements the pattern, using all the participant names
  - A slightly larger example that shows how the pattern could support some application
- Leading effective presentations and contributing during other’s is an important part of this class.

End of Presentation