Lecture 41: Summary

CSCI 62
Fall, 2017

Kim Bruce & Alexandra Papoutsaki

Rest of Semester

- No quiz Friday!
- Review session Friday afternoon at 2 p.m. (here)
- Final exam Monday from 9 a.m. until noon. *Please contact us if you have accommodations requiring extra time.*
- Finish design patterns!

Visitor Pattern

- Problem: want to implement multiple analyses on the same kind of object data
  - Spellchecking and Hyphenating Glyphs
  - Generating code for and analyzing an Abstract Syntax Tree (AST) in a compiler
- Flawed solution: implement each analysis as a method in each object
  - Follows idea objects are responsible for themselves
  - But many analyses will occlude the objects' main code
  - Result is classes hard to maintain

Visitor Pattern

- We define each analysis as a separate Visitor class
  - Defines operations for each element of a structure
- A separate algorithm traverses the structure, applying a given visitor
  - But, like iterators, objects must reveal their implementation to the visitor object
- Separates structure traversal code from operations on the structure
  - Observation: object structure rarely changes, but often want to design new algorithms for processing

Visitor Pattern

- One class hierarchy for object structure
  - AST in compiler
- One class hierarchy for each operation family, called visitors
  - One for typechecking, code generation, pretty printing in compiler

Visitor Pattern Consequences

- Gathers related operations into one class
- Adding new analyses is easy
  - New visitor for each one
  - Easier than modifying the object structure
- Adding new concrete elements is difficult
  - must add a new method to each concrete Visitor subclass

Visitor Traversal Choices

- Traversal in object structure (typical)
  - Define operation that performs traversal while applying visitor object to each component
- Traversal implemented in visitor itself
  - E.g., perform processing at this node, then pass visitor to children nodes.
- Traversal code replicated in each concrete visitor
  - External Iterator

See ParserVisitor5

Designing with Patterns

- How do you know which patterns to use?
- What if you choose the wrong pattern?
  - I.e. your code doesn't evolve the way you thought it would.
- What if all your work to make things extensible via patterns never pays off?
  - I.e. your code doesn't change in the way you thought it would.
- Choosing the right pattern implies prognostication

Designing with Patterns

- Some design patterns are immediately useful
  - Observer, Decorator
- Some are not immediately useful, but you think they might be
  - You anticipate changing things later -- prognostication
- Recently popular philosophy: XP (*now called agile*)
  - Design for your immediate needs
  - When needs change, redesign your code to match
  - Use extensive testing to validate frequent changes

Topics

- Object-oriented Programming & Design in Java
  - Encapsulation, information hiding for flexibility!
- Proofs by induction for correctness & complexity *(more in Discrete Math)*
- Big-O complexity – performance
  - Sorting & searching: selection, merge, heapsort, binary search, tree & graph algos
- Java graphics, GUI programming

More Topics

- Basic Data Structures including alternate implementations:
  - Lists
  - Stacks
  - Queues
  - Trees - including (balanced) binary search trees
  - Maps & Dictionaries (including hash tables)
  - *Graphs, including sophisticated algorithms*
  - *Understand trade-offs in selection of data structures*

Topics in italics covered since last midterm

More Topics

- *Parallelism & Concurrency*
- *OO Design*
- *Design Patterns*

Place of CS 62

- Last core course with focus on teaching to program.
  - Though will learn other languages later.
  - Further courses focus on core topics & applications
- Assume now comfortable in creating medium sized programs
  - There are courses, e.g. CS 121 Software Design & CS 181 Software Engineering, that focus on designing large programs

Goals from Syllabus

- Good understanding of the object-oriented design, coding, and debugging of programs in Java.
- Good understanding of how one might analyze programs for correctness and efficiency
- Understand the trade-offs involved in selections of different data structures and algorithms to solve computational problems.

Choice of Language

- What is important?
  - If programmer time: Use high-level garbage-collected language like Java, C#, Python, ML, Haskell, Scala, Javascript, etc.
  - If execution time (and need access to low-level details): Systems language like C, Objective C or Swift, or C++.
- Students taking 105 (Systems) and graphics will be learning C.

Final Exam

- Monday from 9 a.m. to noon.
- Roughly 7 to 10 questions (some w/many parts)
  - several will involve coding Java
  - Lots of analysis of data structures, descriptions/ analysis of algorithms covered in class -- including graphs!
  - More emphasis on items since second midterm
    - but cumulative!

Bruce Office Hours

- Today: 11 to noon & 1:30 to 3 p.m.
- Thursday, 1 to 2 p.m.
- Friday: review session at 2 p.m. (here?)
- Final Grade Calculation:
  - 15% each midterm plus 25% final
  - 35% programming assignments
  - 10% labs + quizes

Questions?