CS 2110
Lecture 18
Wrapping Up
Logistics

- Test3 Scores Out
- Please turn in A5 today/tomorrow [Check the FAQ post]
- Please check your participation scores on cmsx
- Final: 8:30-11AM, August 1st, Philips 219
  - https://sce.cornell.edu/courses/roster/cs-2110
Design patterns
Reusable design patterns

• Design templates that solve recurring problems in a variety of different systems
• Popularized by "Gang of Four"
  • E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994
• Avoid reinventing the wheel; adopt proven solutions with known tradeoffs
• When developers are familiar with design patterns, they can be used to quickly communicate complex relationships between classes
Properties of patterns

• Meaningful name
• Description of the problem setting
  • Explains where pattern may be applied
• Description of solution
  • Not a library, but a "design template"; can be instantiated in different ways
  • Often expressed graphically
• Statement of consequences
  • Results and tradeoffs of applying the pattern in the problem setting
Iterator pattern: problem setting

• What can you do with the contents a bag?

• Often, just want to do “something” for each element. But…
  • Can’t loop over positions
  • Copying to an array is wasteful

• Awkward mixture of concerns
  • Client controls the loop, but data structure controls the order
Where else have we seen this?

*Situation:* client wants to iterate over elements, but data structure determines order

- **Tree traversals**
  - Preorder, inorder, postorder, level order
- **Graph traversals**
  - BFS, DFS
Iterator pattern: solution

• **Iterable** objects can produce an **Iterator** for each supported traversal order

• **Iterators** provide a `next()` operation so clients can get the next element in the order
  • **Iterators** are one-time-use objects

Client code:

```java
Iterator<E> it = tree.preorderIterator();
while (it.hasNext()) {
    E value = it.next();
    // Do something with value
}
```
Example: array iterator

class ArrayIterator<E> implements Iterator<E> {
    private E[] array;
    /** Index of next element to be yielded. */
    private int nextIndex;

    ArrayIterator(E[] array) {
        this.array = array;
        nextIndex = 0;
    }

    boolean hasNext() {
        return nextIndex < array.length;
    }

    // TODO: NoSuchElementException
    E next() {
        E next = array[nextIndex];
        nextIndex += 1;
        return next;
    }
}
Iterator pattern: example instantiations

• Java collections define interfaces for this pattern
  • Iterable<E>
  • Iterator<E>, Enumeration<E>

• DSAJ defines iterators for its data structures and traversal orders

• So important, Java 5 added a new language feature for iterating over Iterables: the enhanced for-loop

• Topic of Discussion 12
Iterator pattern: limitations

• Awkward to implement recursive traversals
  • Must store state in fields, not on call stack (typically by allocating a Stack object)
• Iterators are easier in languages with coroutines (e.g. Python generators; see CS 1110)

• Preorder traversal with coroutines (not Java):
  preorder(Node n) {
    if (n==null) {return;}
    yield n.value;
    preorder(n.left);
    preorder(n.right);
  }
Design patterns in Java’s libraries

• Stack Overflow: [Examples of GoF Design Patterns in Java's core libraries](#)
Software forges
Programming requires more than code

• Implementation comments
• Specifications
• Test cases

• These are just the tip of the iceberg…
Software development activities

- Project management
- Tasks (“tickets”)
- Development
  - Version control
  - Quality assurance
    - Testing
    - Code review
  - Branch management
  - User documentation
  - User feedback
- Not to mention…
  - Software architecture & design
  - Interface design and validation
  - Monitoring
Software development loop

- Ticket
  - Validation

- Feature branch

- Merge request
  - Integration testing
  - Peer review

- Merge & close
  - Post-merge verification

- Development & debugging

- Post-merge verification
Version control systems

• Who wrote this code?
• Why did they write this code?
• How was feature X implemented?
• What was changed when X stopped working?
• Everything’s broken! Let’s go back to when things worked.
• How can we (=100s of devs) work in parallel on the same code?

• Like backups, but better
Development history is stored as a DAG

- Code changes are bundled into **commits**
  - Commits are documented with a **commit message**
- Most commits have one parent (think **linked list**)
- Multiple commits may share the same parent (think **tree**)
  - Each such commit forms a **branch**
- **Merge** commits have two parents (think **graph**)

![Diagram of commit history as a DAG](image-url)
Branch management

- Dev branch
- Trunk
- Release branch

Long-lived development branches

Other dev branch

Potential conflict
Kinds of merges

- Rebase
- Merge
- Back-merge
Here is a problem, can you solve it?

Object-Oriented Programming and Data Structures

Course homepage (Summer 2023)
Welcome to the homepage for Cornell's intermediate-level course on computer programming and software design. The majority of materials used in the course will be available on this publicly-accessible website.

See Cornell's class roster for official meeting times and locations. Lectures and discussion sections will be delivered exclusively in person.

Prerequisites
We assume that all students in the class have prior programming experience with a general-purpose procedural language (e.g., Python, Java, ...). Suitable courses offered by Cornell include CS 1110 (this is also offered this summer) and CS 1112. Credit for CS 1110 is also offered to students who scored a 5 on the "Computer Science A" AP exam, passed the CASE exam during orientation, or took an equivalent course at another university. If you are not familiar with recursion or reference semantics (i.e., objects), then your prior experience is likely insufficient, and we recommend taking one of Cornell's introductory offerings.

The language used in this course is Java. Knowledge of Java is not a prerequisite, but we do assume that you will be able to adapt your knowledge of other languages to this setting quickly. CS 2110 focuses on generalizable design principles, algorithms, and data structures, not on the syntax and quirks of a particular language, so be prepared to do additional reading and practice at the outset if the language is new to you.

Related courses
CS 2110 is cross-listed as ENGRD 2110. These are the same exact course (same lecture, same discussion sections); it makes no difference which one you enroll in. The ENGRD label means this course can count towards the engineering distribution requirement for students in the College of Engineering (if taken for a letter grade). Whenever course staff or materials refer to "CS 2110", they also apply to "ENGRD 2110".

An honors course on object-oriented programming and data structures is also usually offered in the fall semester as CS 2112. That course covers topics in more depth, and its assignments entail writing significantly more code.
The language used in this course is Java. Knowledge of Java is not a prerequisite, but we do assume that you will be able to adapt your knowledge of other languages to this setting quickly. CS 2110 focuses on generalizable design principles, algorithms, and data structures, not on the syntax and quirks of a particular language, so be prepared to do additional reading and practice at the outset if the language is new to you.

Related courses

CS 2110 is cross-listed as ENGRD 2110. These are the exact same course (same lecture, same discussion sections); it makes no difference which one you enroll in. The ENGRD label means this course can count towards the engineering distribution requirement for students in the College of Engineering (if taken for a letter grade). Whenever course staff or materials refer to “CS 2110”, they also apply to “ENGRD 2110”.

An honors course on object-oriented programming and data structures is also usually offered in the fall semester as CS 2112. That course covers topics in more depth, and its assignments entail writing significantly more code.
How do I do this?

........... ??

CS2110
How do I do this?

Hmm ..... ??
Maybe I should just go get ice cream ....
Course Themes

Programming languages and paradigms

Producing correct & maintainable software

Organizing information in memory

Comparing algorithm performance
Some personal advice

- I need to understand every little detail on the screen immediately
  - Bad Idea!!!

- I will get everything right away
  - Most probably not, even if you think you do, you could be wrong
    - Always test yourself with example test cases!!!
      - Unlike other courses, you can actually run your code with a single button, so don’t be shy.
Go Do Amazing Things With Your Life
Review
Review
Review Grades for Test3

- Minimum: 14.0
- Median: 34.0
- Maximum: 45.0
- Mean: 33.04
- Std Dev: 8.24