CSC207.01 2014S, Class 23: OOD in Practice: Designing a List Interface

Overview

- Preliminaries.
  - Admin.
  - Questions.
- Topics.
  - The design of ADTs, revisited.
  - Quick notes on implementation.
  - A short motivating example.
  - Exercise: Designing a list ADT.

Preliminaries

Upcoming Work

- Readings for Friday to be posted Thursday night.
- Homework 5 is due *next* Wednesday (March 5), but you may need two weeks to get it done.
- The exam makeup is due Sunday at 10:30 p.m. There will be no extensions.

Admin

- Earnest will be running class this Friday and next TWF.
- Review session tomorrow at 10 am.
- Extra credit:
  - CS Extras, Thursday at 4:30 p.m.: The new CS Curriculum.
  - Next week’s convocation on Wednesday.
  - "No Chance Harris" Study Break on Friday.
- Other cool things on campus
  - Feminist film scholar tomorrow

Questions on the Makeup Exam

Questions on the Homework

The design of ADTs, revisited

ADTs

- Big picture: What does it do/what do we want it to do? "Philosophy"
- What *methods* achieve that goal?
- How do we *use* the ADT?
    - What can it represent?
- General design questions:
    - Mutable or immutable?
    - Are there common methods this should include/exclude?
    - Minimalist or maximalist?

Data Structures

- What’s the "big picture" of the underlying implementation
    - Shove it in an array
        - In the natural order
        - With some additional ordering dataa
    - Use small things connected with pointers/references
- Fields
- Implement all of the methods.
- Analyze the implementation: How efficient is it

Methods

- What do they do?
- What type should they return?
- What parameters should they take?
- What exceptions should they throw?
- Design questions
    - Can I generalize this? *

Thinking about lists:

- Big picture philosophy: A dynamic/mutable ordered set of data
    - Some number of elements
    - Put in order by the client
    - Ordered: Come in sequence; connected; linear
    - "iterable" - we can step through the elements
- Uses:
    - Store collections of similar things
    - Mutable strings
    - Represent strange mathematical objects, such as classes
- Methods:

    - find
    - iterate (may be multiple methods)
    - remove
    - add

Example: `insert`

```
values.insert("the answer");

Parameters: object to insert, and the place
What’s a place?
   An integer index - but that can lead to inefficient implementation
```

Exercise: Designing a list ADT

- What parameters would you give to each variation of the `remove` method?
- What parameters would you give to each variation of the `insert` method?
- What would `iterate` look like?
- What do we mean by "a position"?
- Can you design some of these in such a way that they permit O(1) (that is, constant time) implementations in, say, linked lists.

How do we represent positions?

- "Each list has a *current* element."
    - Read "Lists with current considered harmful."
    - Only one position limits you.
- So make a "Position" object
    - We can have either List l; Position p = l.getPosition(); l.advance(p); p.advance(); // 1
    - l.insertAt(p, value);
        - vs. p.insert(value)
    - l.deleteAt(p);
        - vs. p.delete()
    - l.get(p)
        - vs. p.get()