CSC207.01 2013F, Class 23: A List Interface, Continued

Overview

- Preliminaries.
  - Admin.
  - About exam 1.
- A list ADT, continued.

Admin

- Lea says that you all have difficulty reading.
- Read the documentation for java.util.AbstractList and java.util.ListIterator for tomorrow's class.
- Also read [Lists with Current Considered Harmful] (http://csis.pace.edu/~bergin/papers/ListsWithCurrent.html)
- The titles of individual classes may not follow the actual content - Our goal is to think about design.
- Exam 1 is ready, although in draft form.
- Upcoming extra credit opportunities
  - Road to Rio, Tuesday 7:00 p.m., Natatorium.
  - CS Extras, Thursday: Grad School
  - Learning from Aluni, Thursday: Tony Stubblebine '00 - CEO at Lift
  - Codebreaker Friday night at 7pm in Harris.
  - CS Table Friday: Hopper

Exam 1

- Standard Sam policies (although mental health option is gone; we can talk about that)
- Five questions.
- Question 1: Predicates
  - Functions are not first class objects in Java (yet)
  - So we simulate with objects with one method interface Predicate { public boolean test(T val) }
  - In Scheme, (define isEven (lambda (x) (= (mod x 2) 0)))
  - In Java class Even implements Predicate { public boolean test(Integer i) { ... } }
  - Using this predicate Predicate even = new Even(); if (even.test(42)) { pen.println("*The* answer is even"); } pen.println("even.test(i): " + even.test(i));q if (even.test(expt)) { return square(pow(val, expt/2)); }
    - Building new predicates from old
    - In Scheme: (define negate (lambda (pred) (lambda (x) (not (pred x))))) (define negate (lambda (pred) (o not pred))) (define negate (l-s o not))

A list ADT, continued

- Create
- Add/insert
- Delete
- Iterate - look at the values one by one
- Swap

- Big-picture things - sort, shuffle, reverse, etc.

public interface ListOfStrings { // Constructors

```
// Adders


// Deleters


// Iterate stuff - Want to go through the list
/**
 * Create a new position at the beginning of the list
 */
ListPosition front();


/**
 * Given a current location in the list, get the value at the
 * position.
 *
 * @pre
 *    We must have an element at the current position.
 * @pre
 *    ListPostion must be associated with this list.
 */
String get(ListPosition p);


/**
 * Advance to the next position.
 * @pre
 *    hasNext(p)
 */
void advance(ListPosition p);


/**
 * Determine if a ListPostion has a next element.
 *
 * @pre
 *    ListPostion must be associated with this list.
 */
boolean hasElement(ListPosition p);
```

```
/**
 * Determine if one position precedes another.  MAXIMALIST
 */
```

boolean precedes(ListPosition p1, ListPosition p2);

```
// Swap
/**
 * Swap two elements of the list.
 *
 * @pre
 *   p1 and p2 are associated with the list
 *   hasElement(p1), hasElement(p2)
 * @post
 *   The values at the positions
 */
public void swap(ListPostition p1, ListPosition p2);
```

} // interface ListOfStrings

Make the list generic

(Maybe) some notes on implementation