CSC207.01 2013F, Class 24: Generics

Overview

- Preliminaries.
  - Admin.
  - Questions on exam 1.
- ADT Design, considered.
- A list ADT, continued.
- Making the list "generic".

Admin

- Readings for Friday
  - http://docs.oracle.com/javase/7/docs/technotes/guides/collections/overview.html
  - http://docs.oracle.com/javase/tutorial/collections/intro/index.html
  - http://docs.oracle.com/javase/tutorial/collections/interfaces/index.html
- Upcoming extra credit opportunities
  - CS Extras, Thursday: Grad School
  - Learning from Aluni, Thursday: Tony Stubblebine ’00 - CEO at Lift
  - Codebreaker Friday night at 7pm in Harris.
  - Codebreaker discussion after the movie.
  - Wit starts Thursday
  - Debate about need-blind admissions tonight
- 10/10 is this weekend. Please behave responsibly. Please take care of yourself and each other.
- Sam will only be availble intermittently via email this weekend.

Exam 1

- What will go in a typical class that implements the Predicate interface?
  - the test method
  - maybe some public constructors
  - maybe some other private/package/protected methods
  - maybe some private/package/protected fields
- You can provide corrections to the exam starting Thursday at noon.

Our List ADT

```
public interface ListOfStrings {
    // Constructors

    // Adding Elements
```

```
    // Removing Elements

    // Iterating Lists
    /**
     * Get the position of the front of the list.
     *
     * @throws Exception
     *   If the list is empty.
     */
    public ListPosition front() throws Exception;

    /**
     * Advance to the next element.
     *
     * @pre
     *   The list has a next element.
     * @throws Exception
     *   If there is no next element.
     */
    public void advance(ListPosition pos) throws Exception;

    /**
     * Get the element at a particular position.
     *
     * @pre
     *   pos is valid and associated with this list.
     * @throws Exception
     *   If the preconditions are not met.
     */
    public String get(ListPosition pos) throws Exception;

    /**
     * Determine if it's safe to advance to the next position.
     *
     * @pre
     *   pos is valid and associated with the list.
     */
    public boolean hasNext(ListPosition pos);

    // Other operations

    /**
     * Swap the elements at positions p1 and p2.
     *
     * @pre
     *   Both p1 and p2 are valid and associated with this list.
     *   v1 = get(p1), v2 = get(p2)
     * @post
     *   p1 and p2 are unchanged.
     *   v1 = get(p2), v2 = get(p1).
     */
    public void swap(ListPosition p1, ListPosition p2);

} // interface ListOfStrings
```

ADT Design, considered

As you've already started to see, there are a huge number of choices that you make in ADT design, some subtle, some not so subtle.

- How do you figure out if you've made the right decision? Usually, you write client code (or, better yet, have other people write client code).
- And there are often multiple correct decisions.
- Sometimes naming can make a difference. What you think is a clear name, someone else might interpret differently (e.g., our nextValue)
- Here are some related names. Does it matter which we use?
  - Position
  - Cursor
  - Iterator
- Here's a design decision we didn't yet consider: Are positions static (e.g., once you have a position it's always at the same place) or mutable (e.g., you move through the list).

A list ADT, continued

```
/**
 * Lists have cursors/iterators, which fall between elements (or before
 * the first element or after the last element).
 */
public interface ListOfStrings {
    // Adding Elements

    /**
     * Insert an element at the location of the cursor (between two
     * elements).
     *
     * @pre
     *   lit must be associated with the list and in the list.
     *
     * @throws Exception
     *   If the precondition is not met.
     * @throws Exception
     *   If there is no memory to expand the list.
     *
     * @post
     *   The previous elemetn to the iterator remains the same
     *   str is immediately after the iterator
     *   The element that previously followed the iterator follows str
     *   And writing postconditions is a PITN
     */
    public void insert(String str, ListIterator lit) throws Exception;

    /**
     * Add an element to the end of the list.  (Creates a one-element
     * list if the list is empty.)
     *
     * @throws Exception
     *   If there is no memory to expand the list.
```

```
 */
public void append(String str) throws Exception;

/**
 * Add an element to the front of the list.  (Creates a one-element
 * list if the list is empty.)
 *
 * @throws Exception
 *   If there is no memory to expand the list.
 */
public void prepend(String str) throws Exception;

// Removing Elements
/**
 * Delete the element immediately after the iterator.
 *
 * @post
 *    The remaining elements retain their order.
 * @post
 *    The iterator is at the position
 *    The successor of the element immediately before the iterator
 *      is the successor of the now-deleted element.
 */
public void delete(ListIterator lit);

// Iterating Lists
/**
 * Get an iterator right before the front of the list.
 *
 * @throws Exception
 *   If the list is empty.
 */
public ListIterator front() throws Exception;

/**
 * Advance to the next position between elements
 *
 * @pre
 *   The list has a next element.
 * @throws Exception
 *   If there is no next element.
 */
public void advance(ListIterator it) throws Exception;

/**
 * Get the element immediately following this iterator.
 *
 * @pre
 *   it is valid and associated with this list.
 * @throws Exception
 *   If the preconditions are not met.
 */
public String get(ListIterator it) throws Exception;

/**
 * Get the element immediately before this iterator.
 */
```

```
    public String getPrev(ListIterator it) throws Exception;

    /**
     * Determine if it's safe to advance to the next position.
     *
     * @pre
     *   pos is valid and associated with the list.
     */
    public boolean hasNext(ListIterator it);

    // Other operations

    /**
     * Swap the elements at the positions the corresopnd to it1 and it2.
     *
     * @pre
     *   Both it1 and it2 are valid and associated with this list.
     *   v1 = get(it1), v2 = get(it2)
     * @post
     *   it1 and it2 are unchanged.
     *   v1 = get(it2), v2 = get(it1)
     */
    public void swap(ListIterator it1, ListIterator it2);

    /**
     * Search for a value, moving the iterator to that value.
     *
     * @return true, if the value was found
     * @return false, if the value was not found
     *
     * @post If the value is not found, the iterator has not moved.
     * @post IF the value is found, get(it) is value
     */
    public boolean search(ListIterator it, String val);

    /**
     * Grab a sublist.  (Detailed discussion not included.)
     *
     * @pre
     *    Valid iterators.
     *    start precedes end.
     * @throws Exception
     *    If the iterators are invalid.
     */
    public ListOfStrings subList(ListIterator start, ListIterator end)
       throws Exception;

    /**
     * Determine if one iterator precedes another iterator.
     */
    public boolean precedes(ListIterator it1, ListIterator it2);
} // interface ListOfStrings
```

Making the list generic