CSC207.01 2013F, Class 25: The Collections API

Overview

- Preliminaries.
  - Admin.
  - About the exam.
- Making our list interface generic.
- Beyond our own design: The collections API.

Admin

- No readings for Monday. Work on the exam!
- Upcoming extra credit opportunities
  - Codebreaker Friday night at 7pm in Harris.
  - Codebreaker discussion after the movie.
- Drake Library book sale this weekend
- I already have corrections to the exam, but have not made them. Keep sending them in.
- I should have the repo up soon.
- Cool booksale this weekend.
- 10/10 is this weekend. Please behave responsibly. Please take care of yourself and each other.
  - And lock your doors
  - And help the people on Cowles and Younker 1st recover
- EC for going to Wartburg this weekend and cheering on Xcountry

Exam Questions

- What's the name of the class that sorts for DNF (DNF.dnf).
- Can DNF.dnf throw exceptions? No. If you fail to meet preconditions, it can do whatever it wants.
- Do we have to deal with incorrect inputs for DNF.dnf? No. You just want to make sure that it works correctly with correct inputs.
- For problems 4 and 5, do we just have to implement the STUBs, or add procedures.
  - Just finish the STUBs.
- Can you explain a bit more about what you want for loop invariants?
  - A loop invariant is a condition/assesrtion that, if holds at beginning of the loop, also holds at the end.
  - Specific enough that it helps you understand the problem.
  - General enough that you can guarantee that it holds.
  - Note: The invariant can be temporarily invalidated in the middle
  - For this problem, the only things we know about are:
    - The total number of beans in the jar
    - The number of dark beans

- The number of light beans

Making our list interface generic

- How do we generalize the following so that it works for Integers or UshahidiIncidents, or BigDecimals, or whatever?
- Strategy one (early Java): Use Objects
  - Yay polymorphism! We can put Strings or UshahidiIncidents or ... into the list.
  - And our lists can be heterogeneous - We can have an Integer and a String
- But heterogeneity can be problematic. How do you map or sort or .... a heterogeneous list?
  - Java philosophy: Catch possible type errors at compile time rather than run time Scheme: (define whatever (x) (* x x)) ... (define morestuff (fun y) (whatever (fun y))) Java wants to know before you run the program whether you'll have type errors.
- Java redesign: Allow "generic" structures that still do some kind of type checking, so that we can enforce type safety.

- Soln': Parameterize a class definition ("Generics") class ListOf we can plug in type variables, much like we plug in variables elsewhere

  ListOf grades; ListOf csc207; ListOf randomCrapInSamsOffice; and ...

  /**

  - Lists have cursors/iterators, which fall between elements (or before

  - the first element or after the last element). */ public interface ListOf { // 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(Type 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(Type 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(Type 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 Type get(ListIterator it) throws Exception;

/**

- Get the element immediately before this iterator. */ public Type 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, Type val);

/**

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

/**

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

Continuing the example

```
ListOf<Integer> grades;
ListOf<Student> csc207;
ListOf<Object> randomCrapInSamsOffice;
...
grades.prepend(5);
csc207.prepend(5); // COMPILATION ERROR!  5 is not of the appropriate type
Professor SamR = ...;
csc207.prepend(SamR); // COMPILATION ERROR!  SamR is not a student

ListOf<Person> grinnellcs;
grinnellcs.prepend(SamR);  // OKAY, Profesor is a subtype of person, whether or
                           //   not most students believe that claim
grinnellcs.prepend(new Student("A", "A", "A");
```

Beyond our own design: The collections API

- At some point, the designers of Java said "Everyone is going to build these ADTs, so let's just put them in the language".
- Benefits
  - Standardized: Easier for someone joining a project to understand the interface
  - Programmers become more efficient. (Of course, good programmers already have their libraries that the plug in to whatever project the use.)
  - Subclassing might allow you to customize.
  - Likely to be well tested and implemented.
- Disadvantages
  - You don't know what's going on behind the scenes - efficiency issues
  - You might not understand the documentation (because you and the documenters think differently)
  - If you only use prebuilt ADTs and switch to a new language that doesn't, you'll be clueless as to how to design your own.
  - Sometimes you will have to change your client code to match the standard ADT.
