Containers, Array Lists, and Java

CSE 373
Data Structures
Winter 2006

Agenda

- Overview of containers (ADTs) and implementations
- First example - list implemented with arrays (review)
- Java best practices
  - › Interfaces and classes
  - › JavaDoc
  - › Iterators

Types and Implementations

- Common collection types
  - › List, queue, stack, set, bag (multiset), priority queue, map/dictionary, graph
- Variations: sorted or not (sets, maps, others)
- Implementation techniques
  - › Array, linked list (many variations), hashing, trees/ graphs (many, many variations), heaps
- Is it a collection or an implementation technique? Might be either depending on context, e.g., trees, graphs

First Example: Lists (review)

- An ordered collection, position matters
- Operations
  - › Constructor: create a properly initialized empty list
  - › Modifications: clear, add/remove element at end or at position, change element
  - › Queries: size, isEmpty, get element
  - › Processing: iterator

Java

- CSE373 is about data structures, not Java, but…
- Java and the culture around it capture many "best practices", so…
- We'll learn those practices and focus on things that will have value in other settings

Collections (and other Abstractions) in Java

- Every interface and class defines a type
- Conventions
  - › Define *every* important type with an interface
  - › Provide implementations as appropriate
  - › Client code should use the interface type name instead of a specific implementation unless there is a good reason not to
    - Promotes generality and reusability

Today's Example

- Interfaces: BasicList, BasicListIterator
  - › Specifies list operations essentially the same as ones in Java collection classes
- Implementation: BasicArrayList
  - › A particular implementation using an array as the backing store
- Sample code on the web (and basis of hw1)

BasicArrayList Representation

- Representation is an array and count of number of items currently stored

  private Object[ ] items;

  private int nItems;
- Invariant
  - References to objects in the collection are stored in items[0..nItems-1]
  - › Check invariants while coding – powerful bug avoidance tool

Comments

- Java comments

  // to end of line

  /* c-style */

  /** JavaDoc */
- All comments should capture "why" that is not apparent from the "how" of the code
- JavaDoc – particular style of comments that can be automatically processed to create documentation

JavaDoc

- Can put almost any html between /** and */
- Place right before interface/class or method definitions (and elsewhere, but these are the main uses)
- Special tags to identify particular things

  @author, @version – primarily for classes/ interfaces

  @param, @return, @throws – primarily methods

Using JavaDoc

- Every class/interface should have a summary JavaDoc comment at the beginning
- Every public method (visible outside the class) should use JavaDoc to explain *all* parameters, return values, exceptions that are part of the method contract
- Exception: JavaDoc automatically copies comments from interfaces to doc pages for implementing classes – no need to duplicate

Exceptions

- Problem: a collection (or other object) may be in a position to detect an error but not know how best to handle it
- Solution: throw an exception object that can be caught to handle the error or, if not caught, will terminate the program

  throw new IndexOutOfBoundsException();

Exception Guidelines

- Extensive hierarchy of exception types in Java standard library – use one of these if appropriate; define your own if library ones don't meet your needs
- Throw the most specific exception appropriate to the error, e.g., IllegalArgumentException(…) instead of Exception(…)
- Optional argument: string that provides detail

```
throw new IllegalArgumentException("null not allowed…");
```


Processing Collection Contents

- To process an *ordered* collection we can access the elements by position

```
for (int k = 0; k < size; k++)
   do something with things.get(k)
```

- But
  - › This may be inefficient if access by position is not guaranteed to be fast
  - › Likely impossible (get(k) not implemented) for unordered collections (sets, maps)

Iterators – General Solution

- Every Java collection can provide an iterator that can be used to access its contents

```
Iterator it = things.iterator();
while (it.hasNext()) {
   Object item = it.next();
   process item
}
```


Standard Iterator Methods

- Forward access
  - hasNext() – true if more elements
  - next() – return next element and advance
- Similar methods for reverse access in some collections (e.g., lists)
- Modification
  - remove() – remove last item returned by next/previous

Iterator Details

- Multiple iterators may be active on a single collection at the same time
- Remove may only be used once per next/ previous, otherwise IllegalStateException thrown
- Collection may not be modified while iteration is in progress except by remove; ConcurrentModificationException thrown if next/remove/previous attempted after other modification, including remove() in other iterator(s)