CS1007: Object Oriented Design and Programming in Java

Lecture #17
Nov 29

Shlomo Hershkop
shlomo@cs.columbia.edu

Outline

- Frameworks
- Design considerations

- Reading: chapter 8.3+

Announcements

- Next week last week of classes
  - Would like to wrap up
  - Have one class of advanced topics
- Final scheduled for 12/20 (Tuesday) 1 – 4pm
  - Will have review
  - Will post sample online

Collections

- Besides basic functionality of a programming language, JAVA includes many bundled libraries
  - Advantage: no need to reinvent the wheel
  - Disadvantage: square wheels ☺

Collections

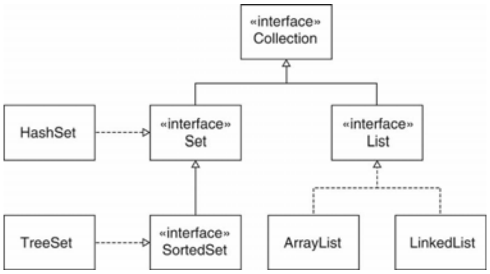
- Java library supplies standard data structures
- Supplies useful services (e.g. Collections.sort, Collections.shuffle)
- Framework: Programmers can supply additional data structures, services
- New data structures automatically work with services
- New services automatically work with data structures (hopefully)

Interfaces

- Collection: the most general collection interface type
- Set: an unordered collection that does not permit duplicate elements
- SortedSet: a set whose elements are visited in sorted order
- List: an ordered collection

Some samples

- HashSet: a set implementation that uses hashing to locate the set elements
- TreeSet: a sorted set implementation that stores the elements in a balanced binary tree
- LinkedList and ArrayList: two implementations of the List interface type



Collection interface

```
boolean add(E obj)
boolean addAll(Collection c)
void clear()
boolean contains(E obj)
boolean containsAll(Collection c)
boolean equals(E obj)
int hashCode()
boolean isEmpty()
Iterator iterator()
boolean remove(E obj)
boolean removeAll(Collection c)
boolean retainAll(Collection c)
int size()
E[] toArray()
E[] toArray(E[] a)
```

Iterator interface

- Iterator traverses elements of collection

```
boolean hasNext()
E next()
void remove()
```

Abstract collection

- Collection is a hefty interface
- Convenient for clients, inconvenient for implementers
- Many methods can be implemented from others (Template method!)
- Example: toArray

```
public E[] toArray()
{
   E[] result = new E[size()];
   Iterator e = iterator();
   for (int i = 0; e.hasNext(); i++)
      result[i] = e.next();
   return result;
}
```

Extend abstract collection

- Can't place template methods in interface
- Place them in AbstractCollection class
- AbstractCollection convenient superclass for implementors
- Only two methods undefined: size,iterator

Adding something new

- Use queue from chapter 3
- Supply an iterator (with do-nothing remove method)
- add method always returns true

```
Ch8/queue/Queue.java
Ch8/queue/QueueTester.java
```

Sets

- Set interface has no methods !!!!
  Why?
- Conceptually, sets are a subtype of collections
- Sets don't store duplicates of the same element
- Sets are unordered

Lists

- Lists are ordered
- Each list position can be accessed by an integer index
- Subtype methods:

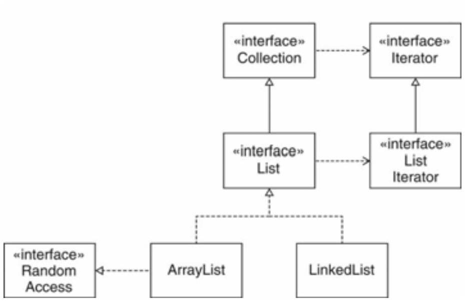
```
boolean add(int index, E obj)
boolean addAll(int index, Collection c)
E get(int index)
int indexOf(E obj)
int lastIndexOf(E obj)
ListIterator listIterator()
ListIterator listIterator(int index)
E remove(int index)
E set(int index, int E)
List subList(int fromIndex, int toIndex)
```

Iterating lists

```
int nextIndex()
int previousIndex()
boolean hasPrevious()
E previous()
void set(E obj)
```

List Classes

- ArrayList
- LinkedList
- Indexed access of linked list elements is possible, but slow
- Problem/weakness in the design
- Partial fix in Java 1.4: RandomAccess interface



Options

- Many operations tagged as "optional"
- Example: Collection.add, Collection.remove
- Default implementation throws exception
- Why have optional operations?

Views

- View = collection that shows objects that are stored elsewhere
- Example: Arrays.asList
- String[] strings = { "Kenya", "Thailand", "Portugal" };
  List view = Arrays.asList(strings)
- Does not copy elements!
- Can use view for common services

otherList.addAll(view);

Views

- get/set are defined to access underlying array
- Arrays.asList view has no add/remove operations
- Can't grow/shrink underlying array
- Several kinds of views:

read-only
modifyable
resizable
. . .

- Optional operations avoid inflation of interfaces
- Controversial design decision

Graphs

- Nodes/vertices
- Edges

Graphs

- Entire branch of Computer Science
  - lots of fun!
  - Lots of math
  - Very relevant

Graph Editor Framework

- Problem domain: interactive editing of diagrams
- Graph consists of nodes and edges
- Class diagram:

nodes are rectangles
edges are arrows

- Electronic circuit diagram:

nodes are transistors, resistors
edges are wires

Graph editor framework

- Traditional approach: programmer starts from scratch for every editor type
- Framework approach: Programmer extends graph, node, edge classes
- Framework handles UI, load/save, ...
- Our framework is kept simple
- Violet uses extension of this framework

Requirements

- What are the GUI requirements?
- What are the programming requirements?

UI

- Toolbar on top
- Grabber button for selecting nodes/edges
- Buttons for current node/edge type
- Menu
- Drawing area

Next time

- Wrap up the framework example
- Over view of software engineering related to OOD.
- Reading: finish chapter 8.