Java Collection Framework

- A collection is an object that represents a group of objects
- A collections framework is just a unified design used for representing and manipulating collections

What does the Collections Framework consist of?

- Collection interfaces
- General-purpose implementations (primary implementations)
- Legacy implementations (collection classes for earlier releases)
- Wrapper implementations
- Convenience implementations
- Abstract implementations
- Algorithms (static methods that perform useful functions such as sorting)
- Infrastructure
- Array Utilities

Core Collection Interfaces
Collection

- The most basic interface in the collections framework
- Set, List and SortedSet all extend Collection
- Map and SortedMap do not extend Collection

Collection Interface - methods

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

List Interface

- A list is an ordered collection or a sequence.
- ArrayList implements the List interface
- The user of this interface will have control over where in the list each element is inserted.
- The user can access elements by their integer index (position in the list), and search for elements in the list.

ArrayList

Resizable array implementation of the List interface
Methods

- void `add`(int index, `Object` x)
- boolean `add`(Object x)
- Object `set`(int index, Object x)
- Object `remove`(int index)
- int `size`()
- Object `get`(int index)

How the methods work

**add:**
- boolean `add`(Object x) – *inserts* the Object x at the end of the list (size increases by 1), returns true
- void `add`(int index, Object x) – *inserts* the Object x at the given index position (elements will be shifted to make room and size increases by 1)

**get:**
- returns the Object at the specified index
- should cast when using value returned
- throws `IndexOutOfBoundsException` if index<0 or index=size

**set**
- replaces value of Object parameter at the given index
- size is not changed
How the methods work

- **remove**
  - removes the element at the specified index
  - throws IndexOutOfBoundsException if index<0 or index>=size
  - size will be decreased by 1
  - returns Object removed

Examples

```java
ArrayList club = new ArrayList();
club.add("Spanky");
club.add("Darla");
club.add("Buckwheat");
System.out.print(club);
Displays: [Spanky, Darla, Buckwheat]

//using club from previous slide
club.set(1, "Mikey");
System.out.print(club);
Displays: [Spanky, Mikey, Buckwheat]

//using club from previous slide
club.add(0, club.remove(club.size()-1));
System.out.print(club);
Displays: [Buckwheat, Spanky, Mikey]
```
ArrayLists only contain Objects!!

```java
ArrayList odds = new ArrayList();
for(int i=1; i<10; i+=2)
    odds.add(new Integer(i));
System.out.println(odds);
```

Displays:

```
[1, 3, 5, 7, 9]
```

Linked Lists

- LinkedList class in the java.util package
- Each element is “linked” by a reference to the next element:

![Linked List Diagram](image)

Some Methods in LinkedList

- `void addFirst(Object obj)`
- `void addLast(Object obj)`
- `Object getFirst()`
- `Object getLast()`
- `Object removeFirst()`
- `Object removeLast()`

Middle of List??

- ListIterator
- Used as a pointer between two links
- Initially it points to the first element
- EX:
  ```java
  LinkedList list = new LinkedList();
  ListIterator iterator = list.listIterator();
  ```
- To get the next element:
  ```java
  if (iterator.hasNext()) //make sure there is a next!
      iterator.next();
  ```

**the next method will throw a NoSuchElementException if there is not a next value**
Traversing the whole list

```java
while (iterator.hasNext()) {
    Object obj = iterator.next();
    //do something with obj
}

next returns the object of the link it just passed
```

Adding Elements

- The add method adds an object after the iterator then moves the iterator position past the new element

```java
iterator.add(“Bob”);
```

Removing Elements

```java
while (iterator.hasNext()) {
    Object obj = iterator.next();
    if ( //obj meets some condition)
        iterator.remove();
}
```

Iterators

- **Iterator** – Similar to Enumeration interface but more powerful
- **ListIterator** – iterator for use with lists.
  - All functionality of Iterator
  - Supports bi-directional iteration
  - Supports element replacement
  - Supports element retrieval
  - Supports index retrieval
java.util
Interface Iterator

- Iterators take the place of Enumerations for Collections
- Iterators differ from Enumerators –
  - Iterators allow the caller to remove elements from the collection during the iteration

Methods

- hasNext
- next
- remove

public boolean hasNext()

- Returns true if the iteration has more elements
- Ex:
  ```java
  while(it.hasNext())
  { //do something
  ```

public Object next()

- Returns the next element in the iteration
- Each time this method is called the iterator “moves”
- Ex:
  ```java
  while(it.hasNext())
  {
      Object obj = it.next();
      if( //obj meets some condition)
      { //do something
  ```
public void remove()  

- Removes from the collection the last element returned by the iterator  
- Can be called only once per call to next

```java
while (it.hasNext())  
{  
    Object obj = it.next();  
    if ( //obj meets some condition)  
        it.remove();  
}
```

Example

```java
ArrayList list = new ArrayList();  
Random r = new Random();  
for (int i = 0; i < 10; i++)  
    list.add(new Integer(r.nextInt(30)));  
Iterator it = list.iterator();  
while (it.hasNext())  
{  
    System.out.print(it.next()+ " ");  
}
```

ListIterator additional methods

- add  
- hasPrevious  
- nextIndex  
- previous  
- previousIndex  
- set

ListNode  

- Class provided by College Board  
- Will be in the reference materials given to students during the exam
Creating a LinkedList class with ListNode

- Constructors
- addFirst
- addLast
- add
- remove - don’t forget special cases!
- print

Getting Started

public class LLList
{
    private ListNode front;
    private ListNode rear;
    public LLList() //constructor
    {
        front = null;
        rear = null;
    }
    public void addFirst(Object obj)
    public void addLast(Object obj)
    public Object removeFirst()
    public Object removeLast()
    public String toString()
}

Big-Oh for ArrayList

<table>
<thead>
<tr>
<th>Methods</th>
<th>expected run-time</th>
<th>worst-case run-time</th>
</tr>
</thead>
<tbody>
<tr>
<td>add(obj)</td>
<td>O(1)</td>
<td>O(1)</td>
</tr>
<tr>
<td>add(index, obj)</td>
<td>O(n)</td>
<td>O(n)</td>
</tr>
<tr>
<td>get(index)</td>
<td>O(1)</td>
<td>O(1)</td>
</tr>
<tr>
<td>set(index, obj)</td>
<td>O(1)</td>
<td>O(1)</td>
</tr>
<tr>
<td>remove(index)</td>
<td>O(n)</td>
<td>O(n)</td>
</tr>
</tbody>
</table>
### Big-Oh for LinkedList

<table>
<thead>
<tr>
<th>Methods</th>
<th>expected run-time</th>
<th>worst-case run-time</th>
</tr>
</thead>
<tbody>
<tr>
<td>add(obj)</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>add(index, obj)</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>get(index)</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>set(index, obj)</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>remove(index)</td>
<td>$O(n)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>addFirst/addLast</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>getFirst/getLast</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>removeFirst/removelast</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
</tbody>
</table>

### Big-Oh for Iterator

#### expected run-time only

<table>
<thead>
<tr>
<th>Methods</th>
<th>ArrayList</th>
<th>LinkedList</th>
</tr>
</thead>
<tbody>
<tr>
<td>next()</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>remove()</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
</tr>
</tbody>
</table>

### Big-Oh for ListIterator

#### expected run-time only

<table>
<thead>
<tr>
<th>Methods</th>
<th>ArrayList</th>
<th>LinkedList</th>
</tr>
</thead>
<tbody>
<tr>
<td>add(obj)</td>
<td>$O(n)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>set(obj)</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
</tbody>
</table>