Objectives

- Implement the ADT dictionary using
  - Either an array
  - Or a chain of linked nodes

// Listing 19-1, pp. 476-477, Textbook
// Recall that keys are unique within a dictionary
import java.util.Iterator;
public interface DictionaryInterface < K, V > {
    public V add (K key, V value);
    public V remove (K key);
    public V getValue (K key);
    public boolean contains (K key);
    public Iterator < K > getKeyIterator ();
    public Iterator < V > getValueIterator ();
    public boolean isEmpty ();
    public int getSize ();
    public void clear ();
}
Challenges, Design Decisions

- Trade-off between
  - Search: `getValue()`, `contains()`
  - Updates: `add()`, `remove()`

- Design Choices
  - List: array or linked-node chain
    - Sorted or unsorted
    - Search: linear, linear (early exit) or binary
  - Array: How many arrays?
  - Array empty space management:
    - Unsorted Array:
    - Sorted: `Shift`

```java
// Listing 19-1, pp. 476-477, Textbook
// Recall that keys are unique within a dictionary
import java.util.Iterator;
public interface DictionaryInterface <K, V> {
  public V add (K key, V value);
  public V remove (K key);
  public V getValue (K key);
  public boolean contains (K key);
  public Iterator <K> getKeyIterator ();
  public Iterator <V> getValueIterator ();
  public boolean isEmpty ();
  public int getSize ();
  public void clear ();
}
```
Dictionary Implementations: Contents

- Array-Based Implementations
  - An **Unsorted** Array-Based Dictionary
  - A **Sorted** Array-Based Dictionary
- Linked Implementations
  - An **Unsorted** Linked Dictionary
  - A **Sorted** Linked Dictionary

<table>
<thead>
<tr>
<th>Dictionary Implementations</th>
<th>Unsorted</th>
<th>Sorted</th>
</tr>
</thead>
</table>
| Array-based                 | - Linear Search  
+ no shift needed for add(), remove() | + Binary Search  
- Shift entries for add(), remove() |
| Chain of linked Nodes       | - Linear Search  
+ no shift needed for add(), remove() | - Linear Search (early exit)  
+ no shift needed for add(), remove() |
Decision 1: How many Unsorted Arrays?

Figure 20-1: Two possible ways to represent dictionary entries
(a) an array of objects,
   - each object is a pair <search key, corresponding value>
   - Listing 20-1: Class ArrayDictionary, Inner class Entry
(b) parallel arrays of search keys and values
Decision 2: Keep empty space to right?

Figure 20-2 Adding a new entry to an unsorted array-based dictionary

Shift is not needed!

Figure 20-3 Removing an entry from an unsorted array-based dictionary

Shift is not needed! Instead bring last entry to vacated slot!
Worst Case Complexity: Unsorted Array Based Dictionary

• **Exercise:** Determine worst-case for add(), remove(), contains(), getKeyIterator() with retrieval of all pairs, ...

• Worst case efficiencies
  - Addition $O(n)$
  - Removal $O(n)$
  - Retrieval $O(n)$
  - Traversal $O(n)$

• Q? Why is add() $O(n)$?
  - Duplicate elimination

• Occasional overhead
  - for enlarging array

```java
// Listing 19-1, pp. 476-477, Textbook
import java.util.Iterator;
public interface DictionaryInterface < K, V > {
    public V add (K key, V value);
    public V remove (K key);
    public V getValue (K key);
    public boolean contains (K key);
    public Iterator < K > getKeyIterator ();
    public Iterator < V > getValueIterator ();
    public boolean isEmpty ();
    public int getSize ();
    public void clear ();
}
```
Dictionary Implementations: Contents

- **Array-Based Implementations**
  - An *Unsorted* Array-Based Dictionary
  - A *Sorted* Array-Based Dictionary

- **Linked Implementations**
  - An *Unsorted* Linked Dictionary
  - A *Sorted* Linked Dictionary

<table>
<thead>
<tr>
<th>Dictionary Implementations</th>
<th>Unsorted</th>
<th>Sorted</th>
</tr>
</thead>
<tbody>
<tr>
<td>Array-based</td>
<td>- Linear Search + no shift needed for add(), remove()</td>
<td>+ Binary Search - Shift entries for add(), remove()</td>
</tr>
<tr>
<td>Chain of linked Nodes</td>
<td>- Linear Search + no shift needed for add(), remove()</td>
<td>- Linear Search (early exit) + no shift needed for add(), remove()</td>
</tr>
</tbody>
</table>
A Sorted Array-Based Dictionary

- Search keys belong to a class that implements interface `Comparable`
- Some of implementation for unsorted dictionary can still be used
- View outline of class, Listing 20-2
- Decision 1: How many sorted arrays?
  - One array, where each entry is a `<key, value>` pair object
- Decision 2: How to Keep empty space to right?
  - `Add()`, `Remove()` require shift!
Figure 20-4 Adding an entry to a sorted array-based dictionary

(a) search

Search from the beginning to find the correct position for a new entry

(b) make room

After locating the correct position for the insertion, shift the contents of subsequent array locations toward the end of the array in the order indicated

(c) insert

Complete the insertion
FIGURE 20-5 Removing an entry from a sorted array-based dictionary:

(a) search

Search from the beginning to find the entry to remove

(b) shift entries

To remove this entry, shift the contents of subsequent array locations toward the beginning of the array in the order indicated
A Sorted Array-Based Dictionary

- **Exercise:** Determine worst-case for
  - add(), remove(), contains(),
  - getKeyIterator() with retrieval of all pairs, …
  - …

- Worst-case efficiencies when `locateIndex` uses a binary search,
  - Addition O(n)
  - Removal O(n)
  - Retrieval O(log n)
  - Traversal O(n)

- Q? Why is add() O(n)?
  - Shift to maintain order

```java
// Listing 19-1, pp. 476-477, Textbook
import java.util.Iterator;
public interface DictionaryInterface < K, V > {
    public V add (K key, V value);
    public V remove (K key);
    public V getValue (K key);
    public boolean contains (K key);
    public Iterator < K > getKeyIterator ();
    public Iterator < V > getValueIterator ();
    public boolean isEmpty ();
    public int getSize ();
    public void clear ();
}
```
Dictionary Implementations: Contents

- Array-Based Implementations
  - An Unsorted Array-Based Dictionary
  - A Sorted Array-Based Dictionary
- Linked Implementations
  - An Unsorted Linked Dictionary
  - A Sorted Linked Dictionary

<table>
<thead>
<tr>
<th>Dictionary Implementations</th>
<th>Unsorted</th>
<th>Sorted</th>
</tr>
</thead>
<tbody>
<tr>
<td>Array-based</td>
<td>- Linear Search</td>
<td>+ Binary Search</td>
</tr>
<tr>
<td></td>
<td>+ no shift needed for add(), remove()</td>
<td>- Shift entries for add(), remove()</td>
</tr>
<tr>
<td>Chain of linked Nodes</td>
<td>- Linear Search</td>
<td>- Linear Search (early exit)</td>
</tr>
<tr>
<td></td>
<td>+ no shift needed for add(), remove()</td>
<td>+ no shift needed for add(), remove()</td>
</tr>
</tbody>
</table>
Linked Implementations

- Chain of linked nodes
- Chain can provide as much storage as necessary
- Encapsulate the two parts of an entry into an object
Figure 20-6: 3 possible ways to use linked nodes to represent a dictionary:

(a) a chain of nodes that each reference an entry object;

(b) parallel chains of search keys and values;

(c) a chain of nodes that each reference a search key and a value.
Figure 20-7 Adding to an unsorted linked dictionary
An Unsorted Linked Dictionary

- Exercise: Determine worst-case for 
  \texttt{add()}, \texttt{remove()}, \texttt{contains()}, 
  \texttt{getKeyIterator()} with retrieval of all pairs, 
  ...

- For this implementation, worst-case efficiencies of operations
  - Addition $O(n)$
  - Removal $O(n)$
  - Retrieval $O(n)$
  - Traversal $O(n)$

- Q? Why is \texttt{add()} $O(n)$?
  - Duplicate elimination

// Listing 19-1, pp. 476-477, Textbook
import java.util.Iterator;
public interface DictionaryInterface < K, V > {
    public V add (K key, V value);
    public V remove (K key);
    public V getValue (K key);
    public boolean contains (K key);
    public Iterator < K > getKeyValueIterator ();
    public Iterator < V > getValueIterator ();
    public boolean isEmpty ();
    public int getSize ();
    public void clear ();
}
Dictionary Implementations: Contents

• **Array-Based Implementations**
  - An *Unsorted* Array-Based Dictionary
  - A *Sorted* Array-Based Dictionary

• **Linked Implementations**
  - An *Unsorted* Linked Dictionary
  - A *Sorted* Linked Dictionary

<table>
<thead>
<tr>
<th>Dictionary Implementations</th>
<th>Unsorted</th>
<th>Sorted</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Array-based</strong></td>
<td>- Linear Search + no shift needed for add(), remove()</td>
<td>+ Binary Search - Shift entries for add(), remove()</td>
</tr>
<tr>
<td><strong>Chain of linked Nodes</strong></td>
<td>- Linear Search + no shift needed for add(), remove()</td>
<td>- Linear Search (early exit) + no shift needed for add(), remove()</td>
</tr>
</tbody>
</table>
A Sorted Linked Dictionary

- Adding a new entry requires sequential search of chain
  - Do not have to look at the entire chain, only until pass node where it should have been
- **Listing 20-5**, class `SortedLinkedDictionary`
- Note private inner class for iterator, **Listing 20-6**
A Sorted Linked Dictionary

- Exercise: Determine worst-case for `add()`, `remove()`, `contains()`, `getKeyIterator()` with retrieval of all pairs, ...
- ...

- Worst case efficiencies of dictionary operations for sorted linked implementation
  - Addition $O(n)$
  - Removal $O(n)$
  - Retrieval $O(n)$
  - Traversal $O(n)$
- Q? Why is `add()` $O(n)$?
  - Duplicate elimination

// Listing 19-1, pp. 476-477, Textbook
import java.util.Iterator;
public interface DictionaryInterface < K, V > {
    public V add (K key, V value);
    public V remove (K key);
    public V getValue (K key);
    public boolean contains (K key);
    public Iterator < K > getKeyIterator ();
    public Iterator < V > getValueIterator ();
    public boolean isEmpty ();
    public int getSize ();
    public void clear ();
}
Exercise: Review Table on pp. 518 in Chapter Summary. It provides worst-case complexity for a few operations on four different dictionary implementations. Which implementation will you recommend in following two case? Why?

Case A: Search is the most frequent operation

Case B: Updates (e.g., add, remove) are most frequent operations