Dictionaries

- References:
  - Text book : Chapter 19 and Chapter 20

1. Specification of ADT Dictionary

   - Contains entries that each have two parts
     A key word or search key
     A value associated with that key

   - Also called map, table or associative array

   - Examples: (customer name, phone#), (student id, student info), (employee id, employee record) etc

   - Assume distinct search keys
     Possible variations: duplicate search keys; secondary search keys

   - Data
     Pairs of objects (key, value)
     Number of pairs in the collection

   - Java interface

```
/** A dictionary with distinct search keys. */
import java.util.Iterator;
public interface DictionaryInterface<K, V>
{
  /** Task: Adds a new entry to the dictionary. If the given search
   *       key already exists in the dictionary, replaces the
   *       corresponding value.
   *  @param key    an object search key of the new entry
   *  @param value  an object associated with the search key
   *  @return either null if the new entry was added to the dictionary
   *          or the value that was associated with key if that value
   *          was replaced */
  public V add(K key, V value);
```

```
/** Task: Removes a specific entry from the dictionary.
 *  @param key  an object search key of the entry to be removed
 *  @return either the value that was associated with the search key
 *          or null if no such object exists */
public V remove(K key);

/** Task: Retrieves the value associated with a given search key.
 *  @param key  an object search key of the entry to be retrieved
 *  @return either the value that is associated with the search key
 *          or null if no such object exists */
public V getValue(K key);

/** Task: Sees whether a specific entry is in the dictionary.
 *  @param key  an object search key of the desired entry
 *  @return true if key is associated with an entry in the
 *          dictionary */
public boolean contains(K key);

/** Task: Creates an iterator that traverses all search keys in the
 *        dictionary.
 *  @return an iterator that provides sequential access to the search
 *          keys in the dictionary */
public Iterator<K> getKeyIterator();

/** Task: Creates an iterator that traverses all values in the
 *        dictionary.
 *  @return an iterator that provides sequential access to the values
 *          in the dictionary */
public Iterator<V> getValueIterator();

/** Task: Sees whether the dictionary is empty.
 *  @return true if the dictionary is empty */
public boolean isEmpty();

/** Task: Gets the size of the dictionary.
 *  @return the number of entries (key-value pairs) currently
 *          in the dictionary */
public int getSize();

/** Task: Removes all entries from the dictionary. */
public void clear();
} // end DictionaryInterface
```

- Iterators

  Note that getKeyIterator and getValueIterator return iterators

  Possible to traverse:

  All search keys in dictionary without traversing values
  All values without traversing search keys
  All search keys and values in parallel

2. Application : Frequency of words

Data:

- wordTable is a SortedDictionary where each entry is (word, count)
- word = String
- count = Integer

Tasks:

- Read words from a file and store into wordTable
- Display frequency of words from wordTable

Driver class

```
import java.util.Scanner;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;
public class Driver
{
   public static void main (String [] args)
   {
      FrequencyCounter wordCounter = new FrequencyCounter ();
      String fileName = "Data.txt"; // or file name could be read
      try
      {
         Scanner data = new Scanner (new File (fileName));
         wordCounter.readFile (data);
      }
      catch (FileNotFoundException e)
      {
         System.out.println ("File not found: " + e.getMessage ());
      }
```

```
        catch (IOException e)
        {
           System.out.println ("I/O error " + e.getMessage ());
        }
        wordCounter.display ();
        System.out.println ("Bye!");
    } // end main
} // end Driver
```

Output for input "row, row, row your boat" :

```
boat 1
row 3
your 1
```

Frequency Counter class:

```
import java.util.Iterator;
import java.util.Scanner;
public class FrequencyCounter
{
   private DictionaryInterface < String, Integer > wordTable;
   public FrequencyCounter ()
   {
      wordTable = new SortedDictionary < String, Integer > ();
   } // end default constructor


   /** Task: Reads a text file of words and counts their frequencies
   *of occurrence.
   *@paramdataatext scanner for the text file of data */
   public void readFile (Scanner data)
   {
      data.useDelimiter ("\\W+"); // skip non letter/digit/underscore chars
      while (data.hasNext ())
      {
         String nextWord = data.next ();
         nextWord = nextWord.toLowerCase ();
         Integer frequency = wordTable.getValue (nextWord);
         if (frequency == null)
         { // add new word to table
            wordTable.add (nextWord, new Integer (1));
         }
         else
         { // increment count of existing word; replace wordTable entry
            frequency++;
```

```
            wordTable.add (nextWord, frequency);
         } // end if
      } // end while
      data.close ();
   } // end readFile

   /** Task: Displays words and their frequencies of occurrence. */
   public void display ()
   {
      Iterator < String > keyIterator = wordTable.getKeyIterator ();
      Iterator < Integer > valueIterator = wordTable.getValueIterator ();
      while (keyIterator.hasNext ())
      {
         System.out.println (keyIterator.next () + " " +
               valueIterator.next ());
      } // end while
   } // end display

} // end FrequencyCounter
```

3. Java Class Library: The Interface Map
   http://download.oracle.com/javase/6/docs/api/java/util/Map.html

Package java.util contains interface: Map <K, V>
Similar to our ADT dictionary. Here are some methods:

void	clear() Removes all of the mappings from this map (optional operation).
boolean	containsKey(Object key) Returns true if this map contains a mapping for the specified key.
boolean	containsValue(Object value) Returns true if this map maps one or more keys to the specified value.
V	get(Object key) Returns the value to which the specified key is mapped, or null if this map contains no mapping for the key.
boolean	isEmpty() Returns true if this map contains no key-value mappings.
V	put(K key, V value) Associates the specified value with the specified key in this map (optional operation).
V	remove(Object key) Removes the mapping for a key from this map if it is present (optional operation).
int	size() Returns the number of key-value mappings in this map.

The Java platform contains three general-purpose Map implementations: HashMap, TreeMap (Sorted), and LinkedHashMap. Will cover them later.

```
// Frequency count from input argument
    import java.util.*;

    public class Freq {
       public static void main(String[] args) {
          Map<String, Integer> m = new HashMap<String, Integer>();

          // Initialize frequency table from command line
          for (String a : args) {
             Integer freq = m.get(a);
             m.put(a, (freq == null) ? 1 : freq + 1);
          }

          System.out.println(m.size() + " distinct words:");
          System.out.println(m);
       }
    }
```

Sample run:

```
java Freq if it is to be it is up to me to delegate
```

The program yields the following output.

```
8 distinct words:
{to=3, delegate=1, be=1, it=2, up=1, if=1, me=1, is=2}
```

4. Basic Dictionary Implementations

    a. Each element in dictionary is an instance of class Entry
    b. Entry class has 2 private generic data: Key, Value
    c. Entry will be private and internal to the dictionary class

- The worst case performance on each operations

	Addition	Removal	Retrieval	Traversal
Unsorted array based	O(1)	O(n)	O(n)	O(n)
Unsorted linked based	O(1)	O(n)	O(n)	O(n)
Sorted array based	O(n)	O(n)	O(log n)	O(n)
Sorted linked based	O(n)	O(n)	O(n)	O(n)
Binary Search Tree (balanced)	O(log n)	O(log n)	O(log n)	O(n)

- Advanced implementations: hash tables and balanced trees (will cover in next few chapters)