Dictionaries

Chapter 19

Data Structures and Abstractions with Java, 4e, Global Edition
Frank Carrano

Examples

- When you want to look up …
  - The meaning of a word
  - An address
  - A phone number
  - A contact on your phone
- These can be implemented in an ADT Dictionary

Specifications for the ADT Dictionary

- Synonyms for ADT Dictionary
  - Map
  - (Index) Table
  - Associative array
- An entry contains
  - Keyword, search key
  - Value

Specifications for the ADT Dictionary

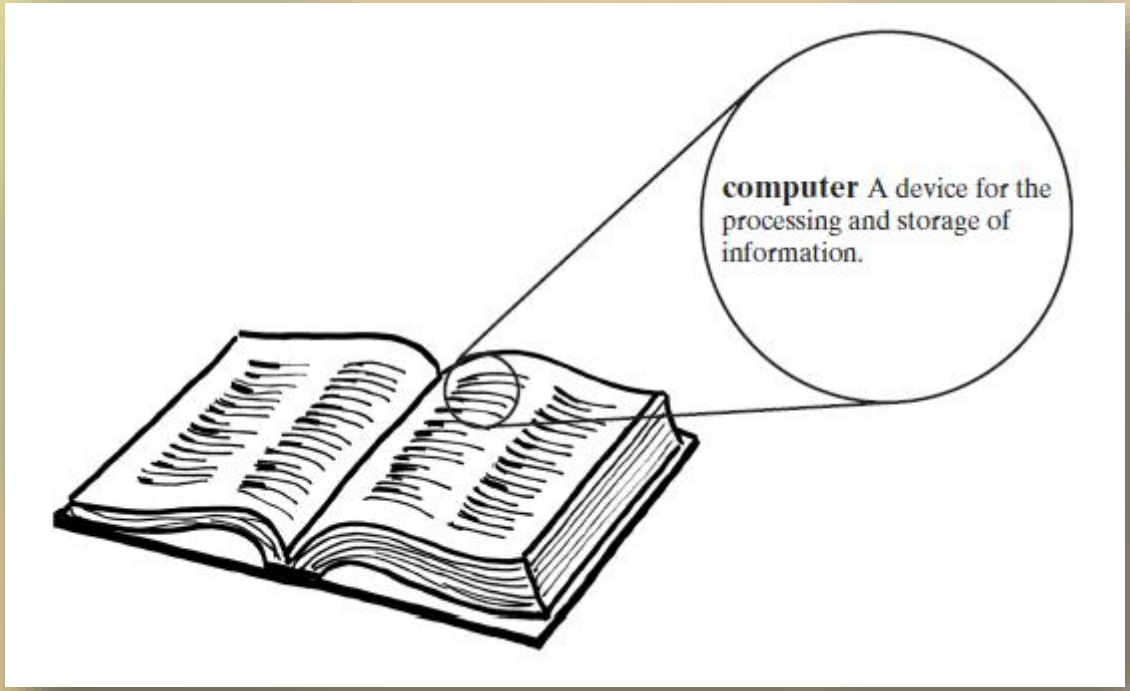

FIGURE 19-1 An English dictionary

Specifications for the ADT Dictionary

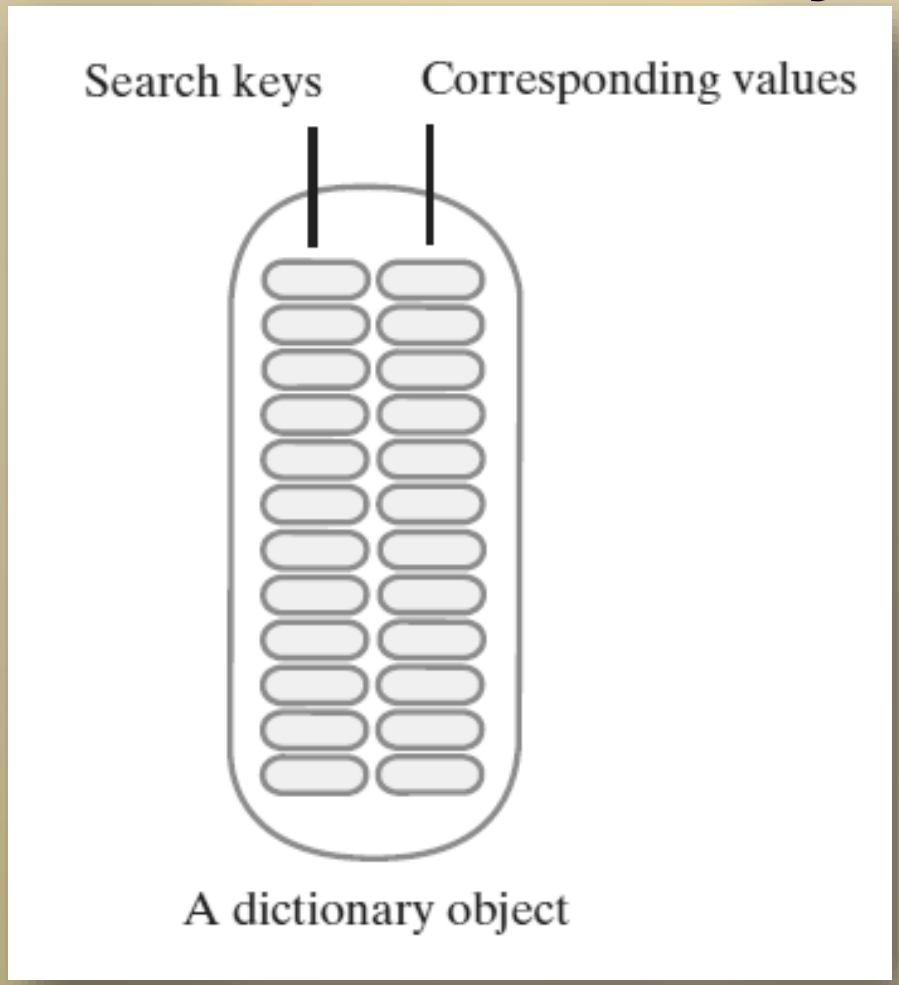

FIGURE 19-2 An instance of ADT dictionary has search keys paired with corresponding values

Specifications for the ADT Dictionary

Data

- Collection of pairs (k, v) of objects k and v,
  - k is the search key
  - v is the corresponding value
- The number of pairs in the collection

Specifications for the ADT Dictionary

Operations

- `add(key, value)`
- `remove(key)`
- `getValue(key)`
- `contains(key)`
- `getKeyIterator()`
- `getValueIterator()`
- `isEmpty()`
- `getSize()`
- `clear()`

Implementation of the ADT 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();
} // end DictionaryInterface
```


Interface for a dictionary

```java
import java.util.Iterator;
/**
    An interface for a dictionary with distinct search keys.
    @author Frank M. Carrano
*/
public interface DictionaryInterface<K, V>
{
   /** Adds a new entry to this 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);

   /** Removes a specific entry from this 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);

   /** Retrieves from this dictionary the value associated with a given
       search key.
```


LISTING 19-1 An interface for the ADT dictionary

A Java Interface

```
public V add(K key, V value);

/** Removes a specific entry from this 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);

/** Retrieves from this dictionary 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);

/** Sees whether a specific entry is in this dictionary.
    @param key  An object search key of the desired entry.
    @return  True if key is associated with an entry in the dictionary.
```


LISTING 19-1 An interface for the ADT dictionary

```
   public boolean contains(K key);

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

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

   /** Sees whether this dictionary is empty.
       @return  True if the dictionary is empty. */
   public boolean isEmpty();

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

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


LISTING 19-1 An interface for the ADT dictionary

Iterators

```
Iterator<String> keyIterator = dataBase.getKeyIterator();
Iterator<Student> valueIterator = dataBase.getValueIterator();
```

Creation of iterators.

Can use each of these iterators either separately or together to traverse:

- All search keys in a dictionary without traversing values
- All values without traversing search keys
- All search keys and all values at the same time

Iterators

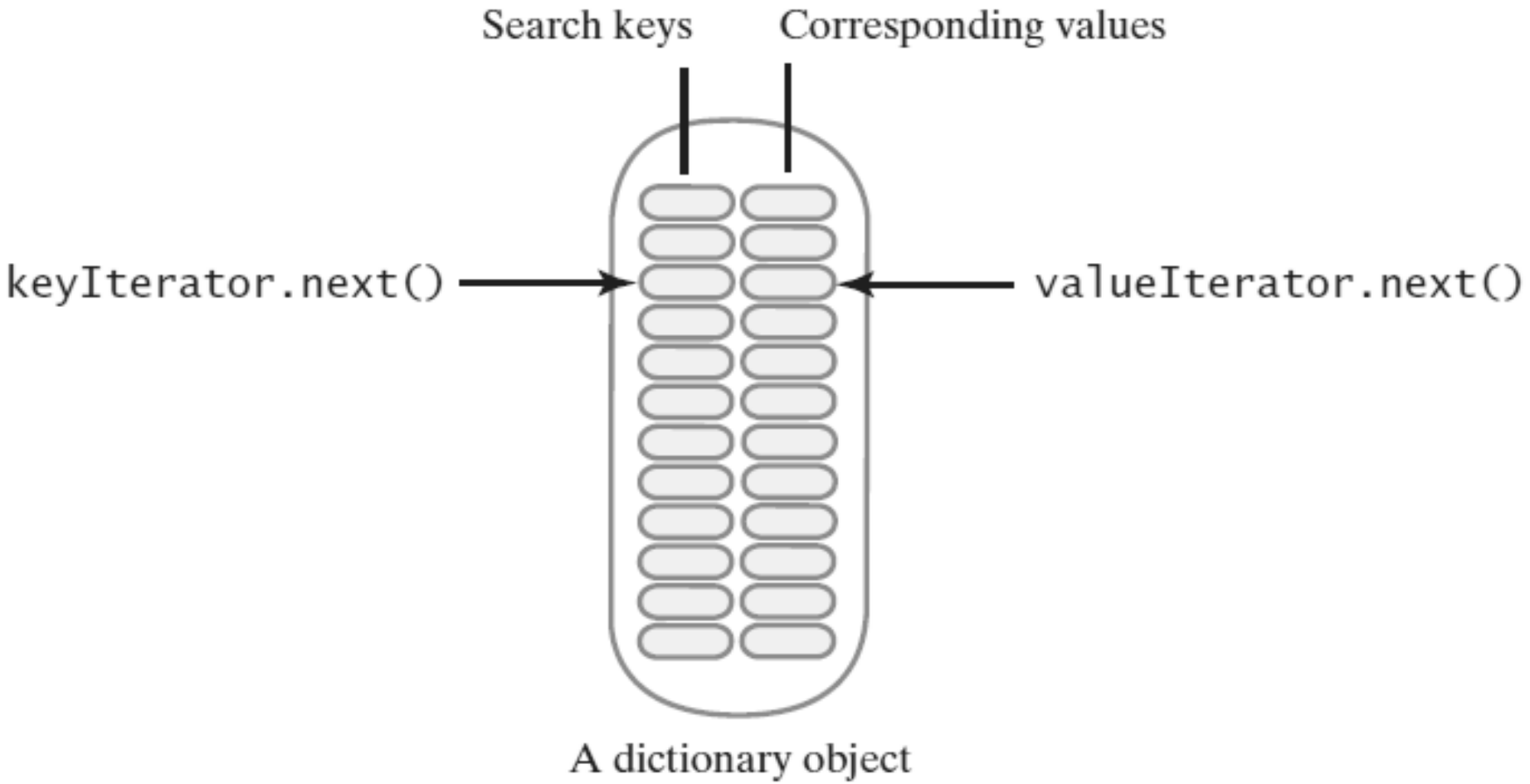

FIGURE 19-3 Two iterators that traverse a dictionary's keys and values in parallel

A Directory of Telephone Numbers

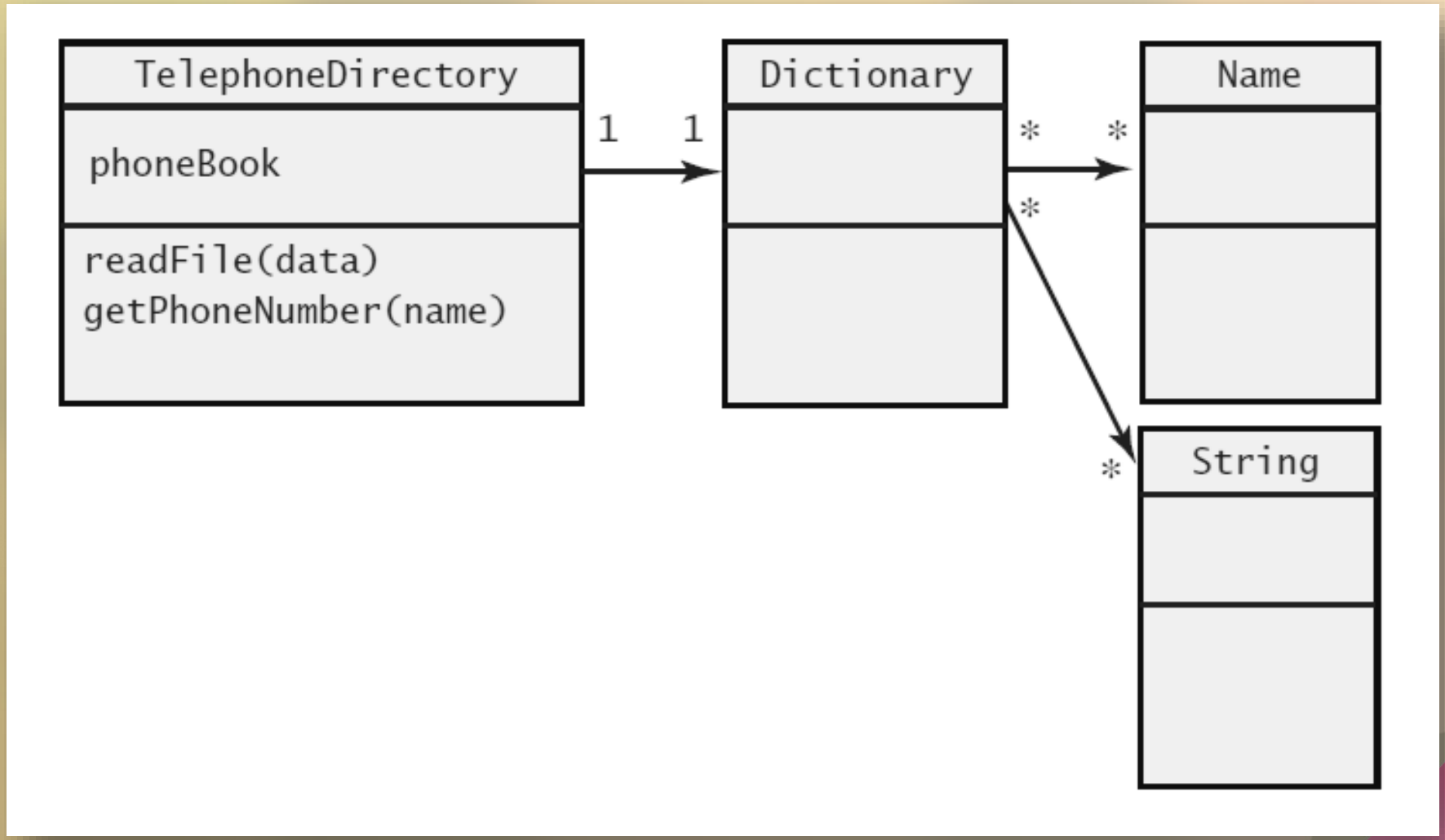

FIGURE 19-4 A class diagram for a telephone directory

```
1  import java.util.Scanner;
2  import java.io.File;
3  import java.io.FileNotFoundException;
4  public class Driver
5  {
6     private static final Name INPUT_ERROR = new Name("error", "error");
7     private static final Name QUIT = new Name("quit", "quit");
8
9     public static void main(String[] args)
10    {
11       TelephoneDirectory directory = new TelephoneDirectory();
12       String fileName = "data.txt"; // Or file name could be read
13
14       try
15       {
16          Scanner data = new Scanner(new File(fileName));
17          directory.readFile(data);
18       }
19       catch (FileNotFoundException e)
20       {
21          System.out.println("File not found: " +  e.getMessage());
22       }
23
24       Name nextName = getName();     // Get name for search from user
25       while (!nextName.equals(QUIT))
```


LISTING 19-2 A client of the class **TelephoneDirectory**

```
24       Name nextName = getName();  // Get name for search from user
25       while (!nextName.equals(QUIT))
26       {
27          if (nextName.equals(INPUT_ERROR))
28             System.out.println("Error in entering name. Try again.");
29          else
30          {
31             String phoneNumber = directory.getPhoneNumber(nextName);
32             if (phoneNumber == null)
33                System.out.println(nextName + "  is not in the directory.");
34             else
35                System.out.println("The phone number for " +  nextName +
36                                   " is " +  phoneNumber);
37          } // end if
38
39          nextName = getName();
40       } // end while
41       System.out.println("Bye!");
42    } // end main
43
44    // Returns either the name read from user, INPUT_ERROR, or QUIT.
45    private static Name getName()
46    {
47       Name result = null;
48       Scanner keyboard = new Scanner(System.in);
```

LISTING 19-2 A client of the class `TelephoneDirectory`

```
50        System.out.print("Enter first name and last name, " +
51                         "or quit to end: ");
52        String line = keyboard.nextLine();
53
54        if (line.trim().toLowerCase().equals("quit"))
55           result = QUIT;
56        else
57        {
58           String firstName = null;
59           String lastName = null;
60           Scanner scan = new Scanner(line);
61
62           if (scan.hasNext())
63           {
64              firstName = scan.next();
65              if (scan.hasNext())
66                 lastName = scan.next();
67              else
68                 result = INPUT_ERROR;
69           }
70           else
71              result = INPUT_ERROR;
72
73           if (result == null)
74              // First and last names have been read
```


LISTING 19-2 A client of the class **TelephoneDirectory**

```
            if (result == null)
               // First and last names have been read
               result = new Name(firstName, lastName);
            } // end if

            return result;
         } // end getName
} // end Driver
```

Output

```
Enter first name and last name or quit to end: Maria Lopez
The phone number for Maria Lopez is 401-555-1234
Enter first name and last name or quit to end: Hunter
Error in entering name. Try again.
Enter first name and last name or quit to end: Hunter Smith
Hunter Smith is not in the directory.
Enter first name and last name or quit to end: quit
Bye!
```


LISTING 19-2 A client of the class `TelephoneDirectory`

A Directory of Telephone Numbers

```
import java.util.Iterator;
import java.util.Scanner;
public class TelephoneDirectory
{
   private DictionaryInterface<Name, String> phoneBook;

   public TelephoneDirectory()
   {
      phoneBook = new SortedDictionary<>();
   } // end default constructor

   /** Reads a text file of names and telephone numbers.
       @param data  A text scanner for the text file of data. */
   public void readFile(Scanner data)
   {

      <See Segment 19.10.>
```


LISTING 19-3 An outline of the class **TelephoneDirectory**

A Directory of Telephone Numbers

```
17        . . . < See Segment 19.10. >
18
19     } // end readFile
20
21     /** Gets the phone number of a given person. */
22     public String getPhoneNumber(Name personName)
23     {
24
25        . . . < See Segment 19.11. >
26
27     } // end getPhoneNumber
28     . . .
```

LISTING 19-3 An outline of the class **`TelephoneDirectory`**

A Directory of Telephone Numbers

```
public void readFile(Scanner data)
{
   while (data.hasNext())
   {
      String firstName   = data.next();
      String lastName    = data.next();
      String phoneNumber = data.next();

      Name fullName = new Name(firstName, lastName);
      phoneBook.add(fullName, phoneNumber);
   } // end while

   data.close();
} // end readFile
```

Definition of method **`readFile`**

A Directory of Telephone Numbers

```
public String getPhoneNumber(Name personName)
{
   return phoneBook.getValue(personName);
} // end getPhoneNumber
```

```
public String getPhoneNumber(String firstName, String lastName)
{
   Name fullName = new Name(firstName, lastName);
   return phoneBook.getValue(fullName);
} // end getPhoneNumber
```

Two versions of method **`getPhoneNumber`**

The Frequency of Words

```
1  import java.util.Scanner;
2  import java.io.File;
3  import java.io.FileNotFoundException;
4
5  public class Driver
6  {
7     public static void main(String[] args)
8     {
9        FrequencyCounter wordCounter = new FrequencyCounter();
10       String fileName = "Data.txt"; // Or file name could be read
11
12       try
13       {
14          Scanner data = new Scanner(new File(fileName));
15          wordCounter.readFile(data);
16       }
```

To provide a count of the number of times each word occurs in a document … LISTING 19-4 A client of the class **FrequencyCounter**

The Frequency of Words

```
16          }
17          catch (FileNotFoundException e)
18          {
19             System.out.println("File not found: " +  e.getMessage());
20          }
21
22          wordCounter.display();
23       } // end main
24    } // end Driver
```

Output

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

To provide a count of the number of times each word occurs in a document … LISTING 19-4 A client of the class `FrequencyCounter`

The Frequency of Words

```
import java.util.Iterator;
import java.util.Scanner;
public class FrequencyCounter
{
   private DictionaryInterface<String, Integer> wordTable;

   public FrequencyCounter()
   {
      wordTable = new SortedDictionary<>();
   } // end default constructor

   /** Reads a text file of words; counts their frequencies of occurrence.
       @param data  A text scanner for the text file of data. */
   public void readFile(Scanner data)
   {

      < See Segment 19.16 >
```

LISTING 19-5 An outline of the class

FrequencyCounter

The Frequency of Words

```
16
17          . . . < See Segment 19.16. >
18
19      } // end readFile
20
21      /** Displays words and their frequencies of occurrence. */
22      public void display()
23      {
24
25          . . . < See Segment 19.17. >
26
27      } // end display
28  } // end FrequencyCounter
```

LISTING 19-5 An outline of the class `FrequencyCounter`

The Frequency of Words

```
/** Reads a text file of words and counts their frequencies of occurrence.
    @param data  A text scanner for the text file of data. */
public void readFile(Scanner data)
{
   data.useDelimiter("\\W+");
   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
```

The Frequency of Words

```
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
```


Method to display the dictionary

A Concordance of Words

```
1  import java.util.Iterator;
2  import java.util.Scanner;
3
4  public class Concordance
5  {
6     private DictionaryInterface<String, ListWithIteratorInterface<Integer>>
7                                                                   wordTable;
8
9     public Concordance()
10    {
11       wordTable = new SortedDictionary<>();
12    } // end default constructor
13
14    /** Reads a text file of words and creates a concordance.
15        @param data  A text scanner for the text file of data. */
16    public void readFile(Scanner data)
17    {
18
19       . . . < See Segment 19.20. >
20
21    } // end readFile
```

A concordance provides location (page or line number) of word. LISTING 19-6 An outline of the class **Concordance**

A Concordance of Words

```
14      /** Reads a text file of words and creates a concordance.
15          @param data  A text scanner for the text file of data. */
16      public void readFile(Scanner data)
17      {
18
19          . . . < See Segment 19.20. >
20
21      } // end readFile
22
23      /** Displays words and the lines in which they occur. */
24      public void display()
25      {
26
27          . . . < See Segment 19.21. >
28
29      } // end display
30  } // end Concordance
```

A concordance provides location (page or line number) of word. LISTING 19-6 An outline of the class **Concordance**

A Concordance of Words

```
public void readFile(Scanner data)
{
   int lineNumber = 1;

   while (data.hasNext())
   {
      String line = data.nextLine();
      line = line.toLowerCase();

      Scanner lineProcessor = new Scanner(line);
      lineProcessor.useDelimiter("\\W+");
      while (lineProcessor.hasNext())
      {
         String nextWord = lineProcessor.next();
         ListWithIteratorInterface<Integer> lineList =
                                              wordTable.getValue(nextWord);

         if (lineList == null)
         {  // Create new list for new word; add list and word to index
            lineList = new LinkedListWithIterator<>();
```

Method **`readFile`** reads the text file and uses the dictionary **`wordTable`** to create the concordance.

A Concordance of Words

```
        String nextWord = lineProcessor.next();
        ListWithIteratorInterface<Integer> lineList =
                                              wordTable.getValue(nextWord);

        if (lineList == null)
        {  // Create new list for new word; add list and word to index
           lineList = new LinkedListWithIterator<>();
           wordTable.add(nextWord, lineList);
        }  // end if

        // Add line number to end of list so list is sorted
        lineList.add(lineNumber);
     } // end while

     lineNumber++;
   } // end while

   data.close();
} // end readFile
```

Method **`readFile`** reads the text file and uses the dictionary **`wordTable`** to create the concordance.

A Concordance of Words

```
public void display()
{
   Iterator<String> keyIterator = wordTable.getKeyIterator();
   Iterator<ListWithIteratorInterface<Integer>> valueIterator =
                                        wordTable.getValueIterator();

   while (keyIterator.hasNext())
   {
      // Display the word
      System.out.print(keyIterator.next() + "  ");

      // Get line numbers and iterator
      ListWithIteratorInterface<Integer> lineList = valueIterator.next();
      Iterator<Integer> listIterator = lineList.getIterator();
```


Method to display the concordance.

A Concordance of Words

```
        // Display line numbers
        while (listIterator.hasNext())
        {
            System.out.print(listIterator.next() + "  ");
        } // end while
        System.out.println();
    } // end while
} // end display
```


Method to display the concordance.

Java Class Library: The Interface Map

```
public V put(K key, V value);
public V remove (Object key);
public V get(Object key);
public boolean containsKey(Object key);
public boolean containsValue(Object value);
public Set<K> keySet();
public Collection<V> values();
public boolean isEmpty();
public int size();
public void clear();
```

Method headers for a selection of methods in Map like those seen in this chapter … highlighted where they differ from our methods.

End

Chapter 19