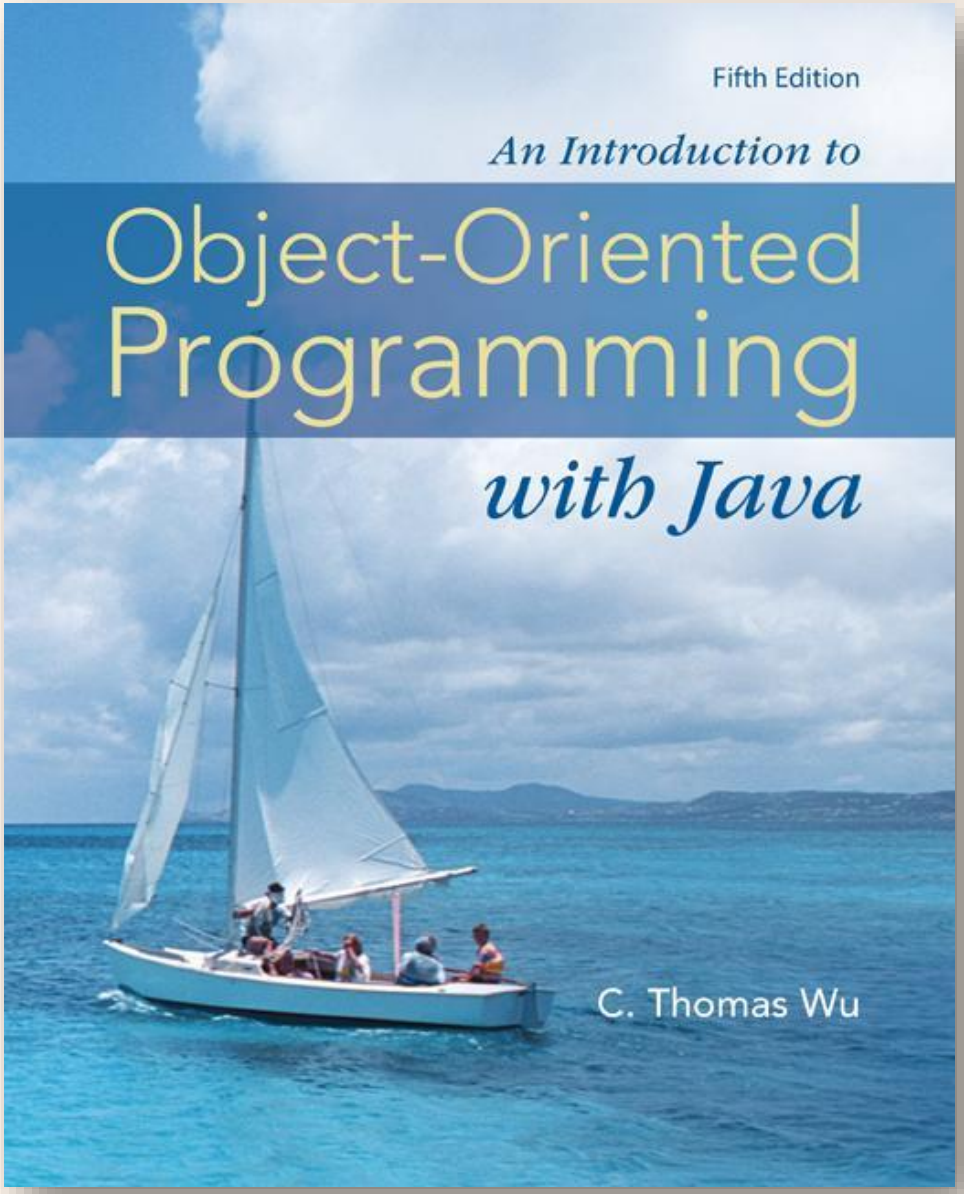


Chapter 10

Arrays and Collections

Animated Version

Collection Classes: Lists and Maps

- The **java.util** standard package contains different types of classes for maintaining a collection of objects.
- These classes are collectively referred to as the *Java Collection Framework (JCF).*
- JCF includes classes that maintain collections of objects as sets, lists, or maps.

Java Interface

- A Java interface defines only the behavior of objects
  - It includes only public methods with no method bodies.
  - It does not include any data members except public constants
  - No instances of a Java interface can be created

JCF Lists

- JCF includes the **List** interface that supports methods to maintain a collection of objects as a linear list

$$L = (l_0, l_1, l_2, \ldots, l_N)$$

- We can add to, remove from, and retrieve objects in a given list.
- A list does not have a set limit to the number of objects we can add to it.

List Methods

- Here are five of the 25 list methods:

`boolean add ( E o )`
Adds an object o to the list
`void clear ( )`
Clears this list, i.e., make the list empty
`E get ( int idx )`
Returns the element at position idx
`boolean remove ( int idx )`
Removes the element at position idx
`int size ( )`
Returns the number of elements in the list

E is a generic class.
Replace E with a concrete class.

Using Lists

- To use a list in a program, we must create an instance of a class that implements the List interface.
- Two classes that implement the **List** interface:
  - **ArrayList**
  - **LinkedList**

- The **ArrayList** class uses an array to manage data.
- The **LinkedList** class uses a technique called *linked-node representation*.

Homogeneous vs. Heterogeneous Collections

- Heterogeneous collections can include any types of objects (Person, Integer, Dog, etc.)
- Homogenous collections can include objects from a designated class only.
  - Designate the class in the collection declaration.
  - For example, to declare and create a list (ArrayList) of Person objects, we write

```
List<Person> friends;
…
friends = new ArrayList<Person>( ) ;
```

Sample List Usage

- Here's an example of manipulating a list of Person objects:

```
import java.util.*;

List<Person>  friends;
Person        person;

friends = new ArrayList<Person>( );

person = new Person("jane", 10, 'F');
friends.add( person );
person = new Person("jack",  6, 'M');
friends.add( person );

Person p = friends.get( 1 );
```

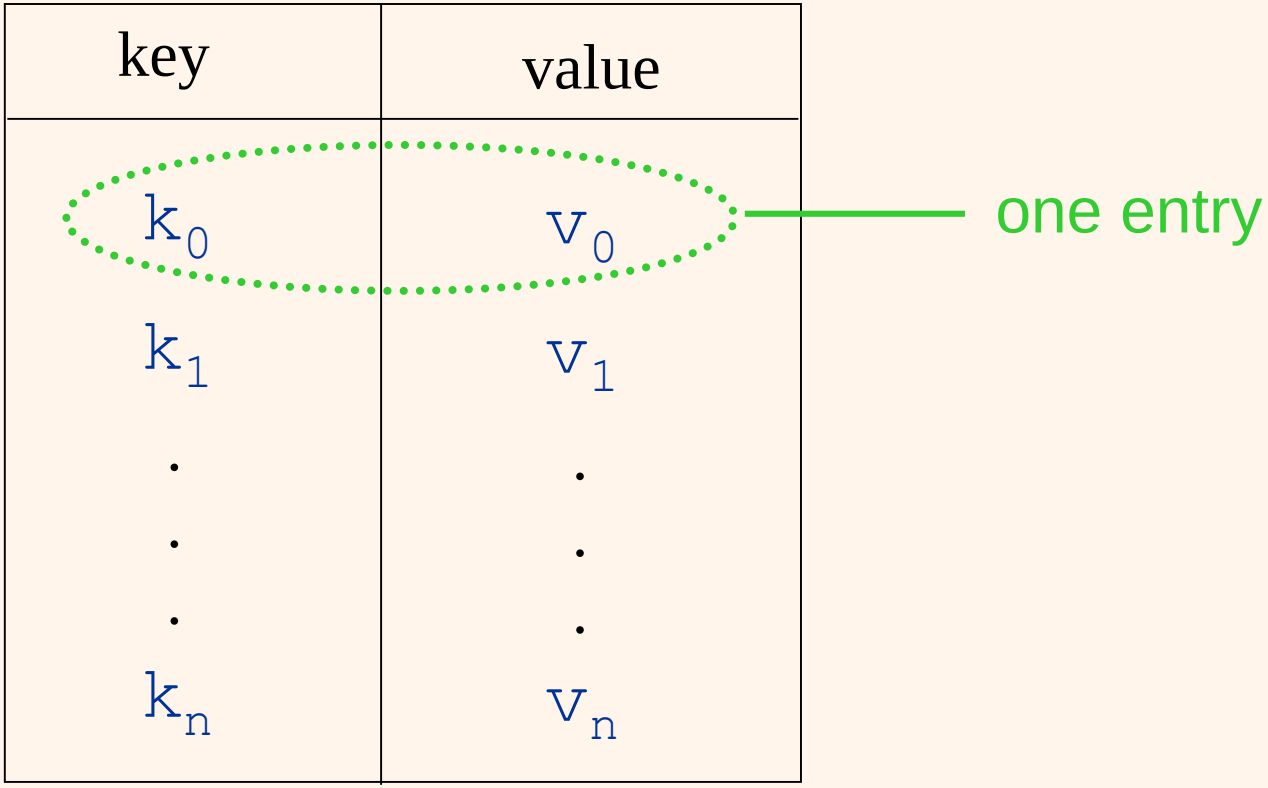
Main Point

An Array List encapsulates the random access behavior of arrays, and incorporates automatic resizing and optionally may include support for sorting and searching. Using a style of sequential access instead, Linked Lists improve performance of insertions and deletions, but at the cost of losing fast element access by index.

Random and sequential access provide analogies for forms of gaining knowledge. Knowledge by way of the intellect is always sequential, requiring steps of logic to arrive at an item of knowledge. Knowing by intuition, is knowing the truth without steps – a kind of “random access” mode of gaining knowledge.

JCF Maps

- JCF includes the **Map** interface that supports methods to maintain a collection of objects (key, value) pairs called map entries.



Map Methods

- Here are five of the 14 list methods:

`void clear ( )`
Clears this list, i.e., make the map empty
`boolean containsKey ( Object key )`
Returns true if the map contains an entry with a given key
`V put (K key, V value)`
Adds the given (key, value) entry to the map
`V remove ( Object key )`
Removes the entry with the given key from the map
`int size ( )`
Returns the number of elements in the map

Using Maps

- To use a map in a program, we must create an instance of a class that implements the Map interface.
- Two classes that implement the **Map** interface:
  - **HashMap**
  - **TreeMap**

Sample Map Usage

- Here's an example of manipulating a map:

```
import java.util.*;

Map     catalog;
catalog = new TreeMap<String, String>( );

catalog.put("CS101", "Intro Java Programming");
catalog.put("CS301", "Database Design");
catalog.put("CS413", "Software Design for Mobile Devices");

if (catalog.containsKey("CS101")) {
    System.out.println("We teach Java this semester");
} else {
    System.out.println("No Java courses this semester");
}
```

Problem Statement

Write an AddressBook class that manages a collection of Person objects. An AddressBook object will allow the programmer to add, delete, or search for a Person object in the address book.

(Working code using Array is already provided)

(Replace array with List)

(Solve this using Map)

Overall Plan / Design Document

- Since we are designing a single class, our task is to identify the public methods.

Public Method	Purpose
`AddressBook`	A constructor to initialize the object. We will include multiple constructors as necessary.
`add`	Adds a new Person object to the address book.
`delete`	Deletes a specified Person object from the address book.
`search`	Searches a specified Person object in the address book and returns this person if found.

Development Steps

- We will develop this program in five steps:

1. Implement the constructor(s).
2. Implement the add method.
3. Implement the search method.
4. Implement the delete method.
5. Finalize the class.

Step 1 Design

- Start the class definition with two constructors
- The zero-argument constructor will create an array of default size
- The one-argument constructor will create an array of the specified size

Step 1 Code

Program source file is too big to list here. From now on, we ask you to view the source files using your Java IDE.

Directory: Chapter10/Step1

Source Files: AddressBook.java

Step 1 Test

- The purpose of Step 1 testing is to verify that the constructors work as expected.

Argument to Constructor	Purpose
`Negative numbers`	Test the invalid data.
`0`	Test the end case of invalid data.
`1`	Test the end case of valid data.
`>= 1`	Test the normal cases.

Step 2 Design

- Design and implement the add method
- The array we use internal to the AddressBook class has a size limit, so we need consider the overflow situation
  - Alternative 1: Disallow adds when the capacity limit is reached
  - Alternative 2: Create a new array of bigger size
- We will adopt Alternative 2

Step 2 Code

Directory: Chapter10/Step2

Source Files: AddressBook.java

Step 2 Test

- The purpose of Step 2 test is to confirm that objects are added correctly and the creation of a bigger array takes place when an overflow situation occurs.

Test Sequence	Purpose
`Create the array of size 4`	Test that the array is created correctly.
`Add four Person objects`	Test that the Person objects are added correctly.
`Add the fifth Person object`	Test that the new array is created and the Person object is added correctly (to the new array).

Step 3 Design

- Design and implement the search method.

```
loc = 0;
while ( loc < count &&
          name of Person at entry[loc] is not equal to
          the given search name ) {
    loc++;
}
if (loc == count) {
    foundPerson = null;
} else {
    foundPerson = entry[loc];
}

return foundPerson;
```

Step 3 Code

Directory: Chapter10/Step3

Source Files: AddressBook.java

Step 3 Test

- To test the correct operation of the search method, we need to carry out test routines much more elaborate than previous tests.

Test Sequence	Purpose
`Create the array of size 5 and add five Person objects with unique names.`	Test that the array is created and set up correctly. Here, we will test the case where the array is 100 percent filled.
`Search for the person in the first position of the array`	Test that the successful search works correctly for the end case.
`Search for the person in the last position of the array`	Test another version of the end case.
`Search for a person somewhere in the middle of the array.`	Test the normal case.
`Search for a person not in the array.`	Test for the unsuccessful search.
`Repeat the above steps with an array of varying sizes, especially the array of size 1.`	Test that the routine works correctly for arrays of different sizes.
`Repeat the testing with the cases where the array is not fully filled, say, array length is 5 and the number of objects in the array is 0 or 3.`	Test that the routine works correctly for other cases.

Step 4 Design

- Design and implement the delete method.

```
boolean status;
int       loc;
loc = findIndex( searchName );

if ( loc is not valid ) {
     status = false;
} else { //found, pack the hole
     replace the element at index loc+1 by the last element
     at index count;

     status = true;

     count--;      //decrement count, since we now have one less element
     assert 'count' is valid;
}
return status;
```

Step 4 Code

Directory: Chapter10/Step4

Source Files: AddressBook.java

Step 4 Test

- To test the correct operation of the delete method, we need to carry out a detailed test routine.

Test Sequence	Purpose
`Create the array of size 5 and add five Person objects with unique names.`	Test the array is created and set up correctly. Here, we will test the case where the array is 100 percent filled.
`Search for a person to be deleted next.`	Verify that the person is in the array before deletion.
`Delete the person in the array`	Test that the delete method works correctly.
`Search for the deleted person.`	Test that the delete method works correctly by checking the value null is returned by the search.
`Attempt to delete a nonexisting person.`	Test that the unsuccessful operation works correctly.
`Repeat the above steps by deleting persons at the first and last positions.`	Test that the routine works correctly for arrays of different sizes.
`Repeat testing where the array is not fully filled, say, an array length is 5 and the number of objects in the array is 0 or 3.`	Test that the routine works correctly for other cases.

Step 5: Finalize

- Final Test
  - Since the three operations of add, delete, and search are interrelated, it is critical to test these operations together. We try out various combinations of add, delete, and search operations.
- Possible Extensions
  - One very useful extension is scanning. Scanning is an operation to visit all elements in the collection.
  - Scanning is useful in listing all Person objects in the address book.