Chapter 10: An Array Instance Variable

Asserting Java

Rick Mercer

A Collection Class

- Programmers often use *collection* classes
  - classes with the main purpose of storing a collection of elements that can be treated as a single unit
    - For example, class Bank might store a collection of BankAccount objects and provide appropriate access to the individual elements
  - A Few Java collection class names:

    `Stack` *a collection of objects with access to the one on top*

    `ArrayList` *a collection of objects with the List operations*

    `TreeSet` *a collection of unique objects*

    `HashMap` *collection to provide fast adds, removes, finds*

Characteristics of a collection class

- Main responsibility: store a collection of objects
- Can add and remove objects from the collection
- Provides access to individual elements
- May have search-and-sort operations
- Some collections allow duplicate elements (ArrayList), other collections do not (class Set for example)
- Some collections have a natural ordering--OrderedSet--other collections do not--ArrayList

What about arrays?

- Is an array a collection?
  - arrays are objects, they do have similar characteristics, however
    - subscripts are needed to access individual elements
    - programmers have to spend a lot of time implementing array based adds, removes, sorts, and searches
    - arrays are lower level
    - Arrays provides programmers with the opportunity to make more more errors and spend more time than using a collection class

class StringBag

- The StringBag class
  - represents a mathematical bag or multi-set
  - is a simple collection class
  - will have an array instance variable
  - is a collection capable of a storing only strings elements *actually references to string objcets*
  - is not a "standard", but this allows us to see the inner working of the class

A StringBag class continued

- A StringBag object
  - stores a collection of String elements that
    - are not in any particular order
    - are not necessarily unique
    - understands these messages

```
add  occurencesOf remove
```

StringBag with no implementation

```
// A class for storing a multi-set (bag) of String elements.
public class StringBag {

  // Construct an empty StringBag object (no elements yet)
  public StringBag() {
  }

  // Add a string to the StringBag in no particular place.
  public void add(String stringToAdd) {
  }

  // Return how often element equals one in this StringBag
  public int occurencesOf(String element) {
    return 0;
  }

  // remove first element that equals elementToRemove
  public boolean remove(String elementToRemove) {
    return false;
  }
}
```

A test method for add and occurencesOf

```
@Test
public void testAddAndOccurencesOf() {
  StringBag sb = new StringBag();
  sb.add("Marlene");
  sb.add("Eric");
  sb.add("Marlene");
  sb.add("Eric");
  sb.add("Marlene");
  assertEquals(3, sb.occurencesOf("Marlene"));
  assertEquals(2, sb.occurencesOf("Eric"));
  assertEquals(0, sb.occurencesOf("Not here"));
}
```

Implement StringBag methods

- The constructor creates an empty StringBag
  - no elements in it, size is 0

```
public StringBag()  {
   size = 0;
   data = new String[20];
}
```

StringBag add

- The **`StringBag.add`** operation adds all new elements to the "end" of the array if there is "room"

```
public void add(String stringToAdd) {
  // If there is no more room, do nothing



  // Otherwise, place at end of array



}
```

  - *could we have added stringToAdd at the beginning?____?*

StringBag occurencesOf

- The **`occurencesOf`** method returns how often the argument equals a StringBag element

```
public int occurencesOf (String value) {



}
```

StringBag remove

- **`StringBag remove`** uses sequential search to find the element to be removed (arbitrallily use the first when occurencesOf > 1
- If the element is found,
  - move the last element **`data[size-1]`** into the location of the removal element
  - place null into where the last element was
    - done to release the memory for garbage collection
  - decrease **`size`** by 1

State of s1 before removing "Jignesh"

Array Data Field	State
`data[0]`	`"Kelly"`
`data[1]`	`"Jignesh"`
`data[2]`	`"Kristen"`
`data[3]`	`"Maya"`
`data[4]`	`null`
`...`	`null`
`size`	`4`

local objects in **StringBag remove**	
`removalCandidate`	`"Jignesh"`
`subscript`	`1`

The state after removing "another string"

1. Find removalCandidate in data[1]
2. Overwrite data[1] with "Maya" (the last element and decrease size by 1

StringBag remove

calls a private helper method indexOf

```
public boolean remove(String stringToRemove) {
   boolean result = false;
   // Get subscript of stringToRemove or -1 if not found
   int subscript = indexOf(stringToRemove);
   if(subscript != -1) {
     // Move the last string in the array to
     // where stringToRemove was found.
     data[subscript] = data[size-1];
     // Release that memory to be reused for any object
     data[size-1] = null;
     // And then decrease size by one
     size--;
     // return true to where the message was sent
     result = true;
   }
   return result;
 }
```

Code Demo: Complete StringBag remove that shifts all elements

```
public boolean remove(String stringToRemove) {
    // Get subscript of stringToRemove or -1 if not found
    int subscript = indexOf(stringToRemove);
    if(subscript < 0)
     return false;
    else {

      // Shift all elements left so this array
{ "Kelly", "Jignesh", "Kristen", "Maya", null, null}
      // would change to this
{ "Kelly", "Kristen", "Maya", null, null, null}
      // don't forget to return true and reduce size


    }
 }
```