

Array-Based Implementations

Chapter 3

The Approach

- An ADT is
 - A collection of data ... and ...
 - A set of operations on that data
- Specifications indicate
 - What ADT operations do
 - But not how to implement
- First step for implementation
 - Choose data structure

The Approach

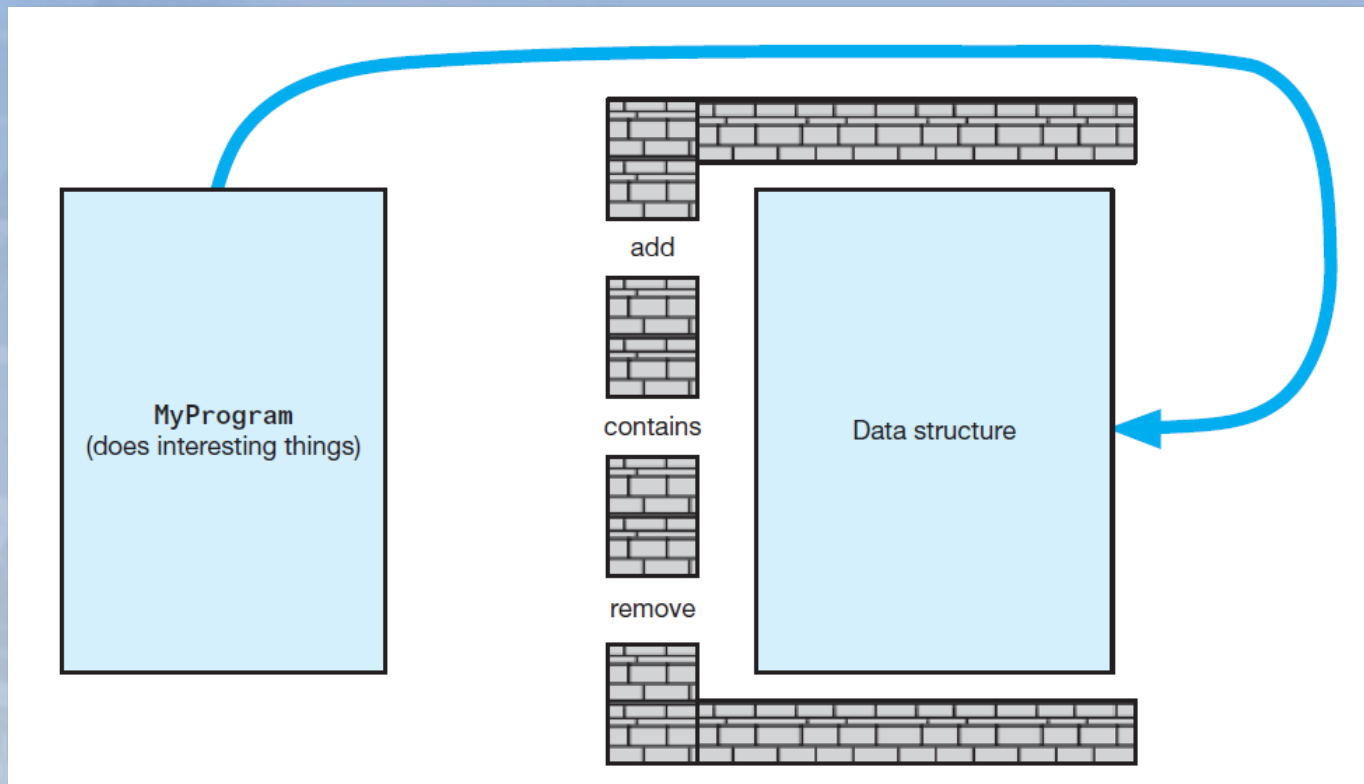


FIGURE 3-1 Violating the wall of ADT operations

Core Methods

- Poor approach
 - Define entire class and attempt test
- Better plan – Identify, then test basic (core) methods
 - Create the container (constructors)
 - Add items
 - Display/list items
 - Remove items

Using Fixed-Size Arrays

- Must keep track of array elements used, available
- Decide if first object goes in element 0 or 1
- Consider if the *add* method places elements in consecutive elements of array
- What happens when *add* method has used up final available element?

Array-Based Implementation of ADT Bag

```
+getCurrentSize(): integer  
+isEmpty(): boolean  
+add(newEntry: ItemType): boolean  
+remove(anEntry: ItemType): boolean  
+clear(): void  
+getFrequencyOf(anEntry: ItemType): integer  
+contains(anEntry: ItemType): boolean  
+toVector(): vector
```

Core *ArrayBag* methods

Array-Based Implementation of ADT Bag

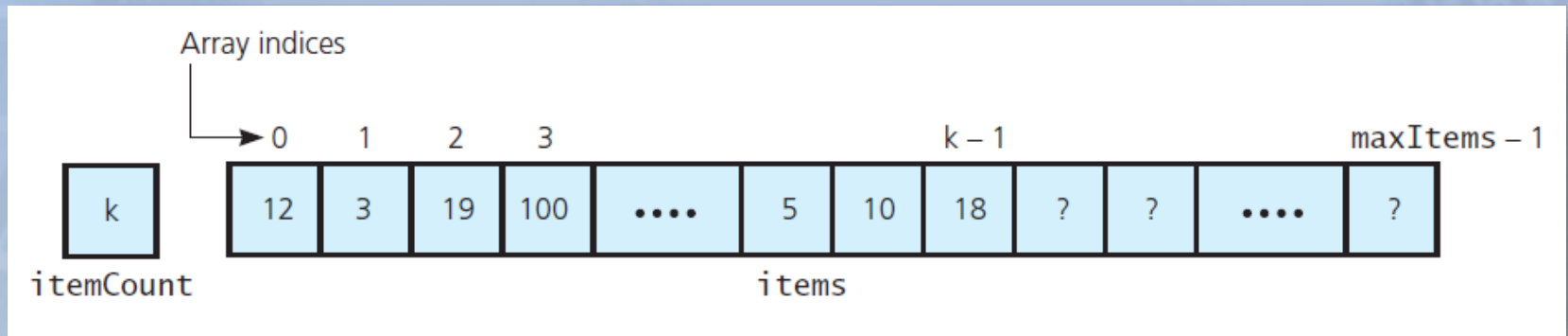


FIGURE 3-2 An array-based implementation of the ADT bag

The Header File

```
1  /** Header file for an array-based implementation of the ADT bag.
2   @file ArrayBag.h */
3
4  #ifndef ARRAY_BAG_
5  #define ARRAY_BAG_
6
7  #include "BagInterface.h"
8
9  template<class ItemType>
10 class ArrayBag : public BagInterface<ItemType>
11 {
12 private:
13     static const int DEFAULT_CAPACITY = 6; // Small size to test for a full bag
14     ItemType items[DEFAULT_CAPACITY];      // Array of bag items
15     int itemCount;                          // Current count of bag items
16     int maxItems;                          // Max capacity of the bag
17
18     // Returns either the index of the element in the array items that
19     // contains the given target or -1 if the array does not contain
```

LISTING 3-1 The header file for the class *ArrayBag*

The Header File

```
17
18     // Returns either the index of the element in the array items that
19     // contains the given target or -1, if the array does not contain
20     // the target.
21     int getIndexOf(const ItemType& target) const;
22
23 public:
24     ArrayBag();
25     int getCurrentSize() const;
26     bool isEmpty() const;
27     bool add(const ItemType& newEntry);
28     bool remove(const ItemType& anEntry);
29     void clear();
30     bool contains(const ItemType& anEntry) const;
31     int getFrequencyOf(const ItemType& anEntry) const;
32     vector<ItemType> toVector() const;
33 }; // end ArrayBag
34
35 #include "ArrayBag.cpp"
36 #endif
```

LISTING 3-1 The header file for the class *ArrayBag*

Defining the Core Methods

```
template<class ItemType>
bool ArrayBag<ItemType>::add(const ItemType& newEntry)
{
    bool hasRoomToAdd = (itemCount < maxItems);
    if (hasRoomToAdd)
    {
        items[itemCount] = newEntry;
        itemCount++;
    } // end if

    return hasRoomToAdd;
} // end add
```

The method *add*

Defining the Core Methods

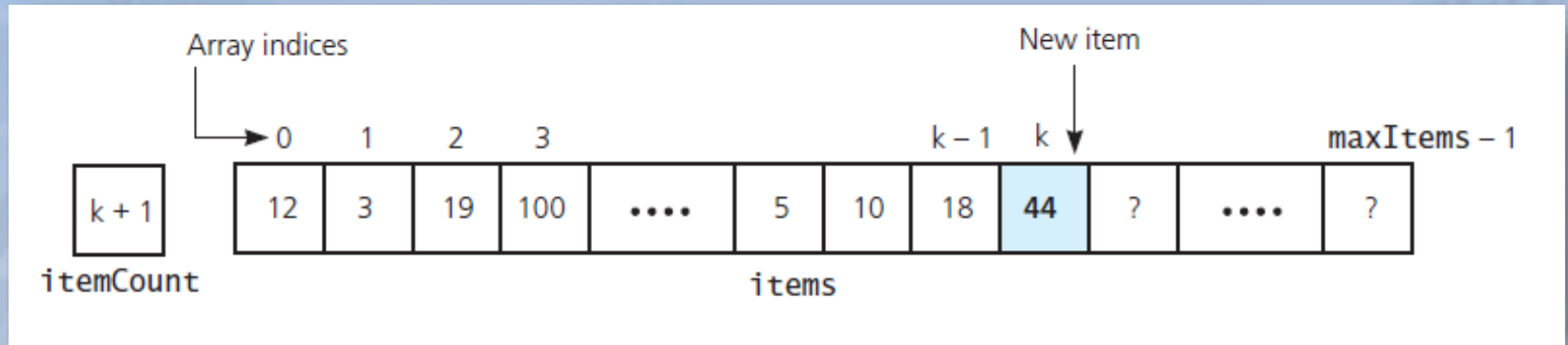


FIGURE 3-3 Inserting a new entry into an array-based bag

Defining the Core Methods

```
template<class ItemType>
vector<ItemType> ArrayBag<ItemType>::toVector() const
{
    vector<ItemType> bagContents;
    for (int i = 0; i < itemCount; i++)
        bagContents.push_back(items[i]);
    return bagContents;
} // end toVector
```

The method *toVector*

Defining the Core Methods

```
template<class ItemType>
int ArrayBag<ItemType>::getCurrentSize() const
{
    return itemCount;
} // end getCurrentSize

template<class ItemType>
bool ArrayBag<ItemType>::isEmpty() const
{
    return itemCount == 0;
} // end isEmpty
```

Methods *getCurrentSize* and *isEmpty*

Testing the Core Methods

```
1  #include <iostream>
2  #include <string>
3  #include "ArrayBag.h"
4
5  using std::cout;
6  using std::endl;
7  void displayBag(ArrayBag<std::string>& bag)
8  {
9      cout << "The bag contains " << bag.getCurrentSize()
10         << " items:" << endl;
11      std::vector<std::string> bagItems = bag.toVector();
12
13      int numberOfEntries = (int)bagItems.size();
14      for (int i = 0; i < numberOfEntries; i++)
15      {
16          cout << bagItems[i] << " ";
17      } // end for
18      cout << endl << endl;
19  } // end displayBag
20
21 void bagTester(ArrayBag<std::string>& bag)
```

LISTING 3-2 A program that tests the core methods of the class *ArrayBag*

Testing the Core Methods

```
21 void bagTester(ArrayBag<std::string>& bag)
22 {
23     cout << "isEmpty: returns " << bag.isEmpty()
24         << "; should be 1 (true)" << endl;
25     displayBag(bag);
26
27     std::string items[] = {"one", "two", "three", "four", "five", "one"};
28     cout << "Add 6 items to the bag: " << endl;
29     for (int i = 0; i < 6; i++)
30     {
31         bag.add(items[i]);
32     } // end for
33
34     displayBag(bag);
35     cout << "isEmpty: returns " << bag.isEmpty()
36         << "; should be 0 (false)" << endl;
37     cout << "getCurrentSize: returns " << bag.getCurrentSize()
38         << "; should be 6" << endl;
39     cout << "Try to add another entry: add(\"extra\") returns "
40         << bag.add("extra") << endl;
41 } // end bagTester
42
43 int main()
```

LISTING 3-2 A program that tests the core methods of the class *ArrayBag*

Testing the Core Methods

```
42
43  int main()
44  {
45      ArrayBag<std::string> bag;
46      cout << "Testing the Array-Based Bag:" << endl;
47      cout << "The initial bag is empty." << endl;
48      bagTester(bag);
49      cout << "All done!" << endl;
50
51      return 0;
52  } // end main
```

Output

```
Testing the Array-Based Bag:
The initial bag is empty.
isEmpty: returns 1; should be 1 (true)
The bag contains 0 items:

Add 6 items to the bag:
The bag contains 6 items:
one two three four five one
```

LISTING 3-2 A program that tests the core methods of the class *ArrayBag*

Implementing More Methods

```
template<class ItemType>
int ArrayBag<ItemType>::getFrequencyOf(const ItemType& anEntry) const
{
    int frequency = 0;
    int curIndex = 0; // Current array index
    while (curIndex < itemCount)
    {
        if (items[curIndex] == anEntry)
        {
            frequency++;
        } // end if

        curIndex++; // Increment to next entry
    } // end while

    return frequency;
} // end getFrequencyOf
```

Method *getFrequencyOf*

Implementing More Methods

```
template <class ItemType>
bool ArrayBag<ItemType>::contains(const ItemType& anEntry) const
{
    bool isFound = false;
    int curIndex = 0; // Current array index
    while (!isFound && (curIndex < itemCount))
    {
        isFound = (anEntry == items[curIndex]);
        if (!isFound)
            curIndex++; // Increment to next entry
    } // end while
    return isFound;
} // end contains
```

Possible implementation of method *contains*

Methods That Remove Entries

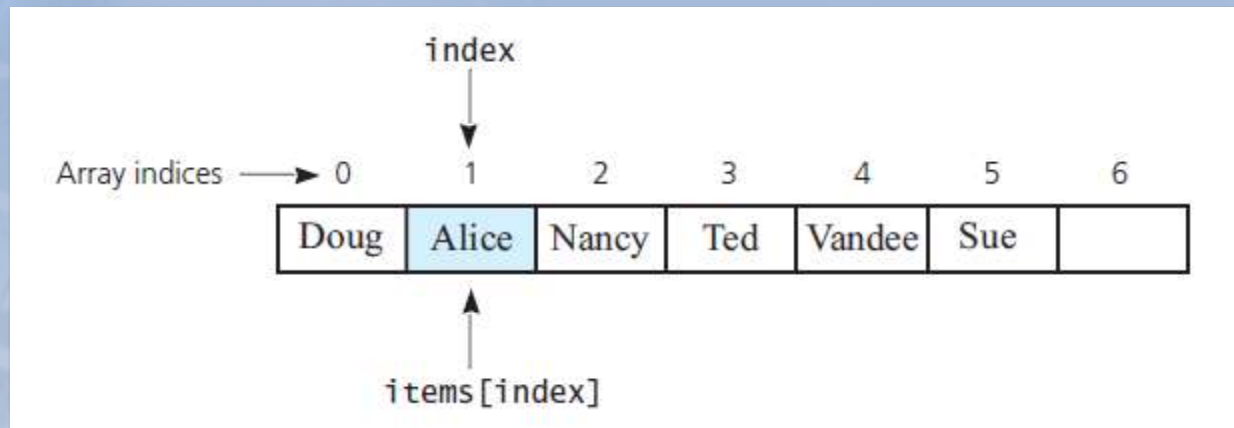


FIGURE 3-4 The array items after a successful search for the string "Alice"

Methods That Remove Entries

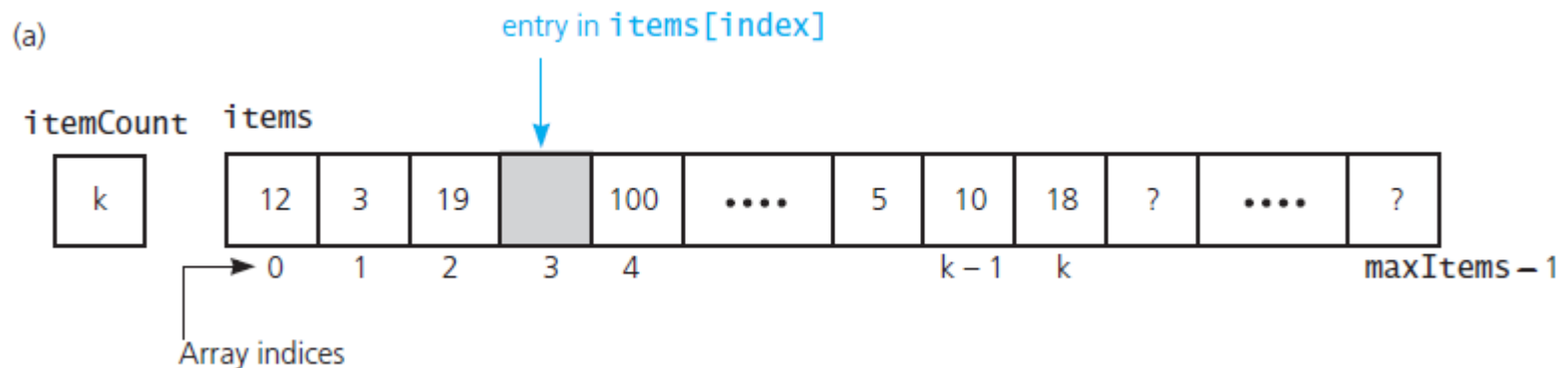


FIGURE 3-5 (a) A gap in the array `items` after the entry in `items[index]` and decrementing `itemCount`;

Methods That Remove Entries

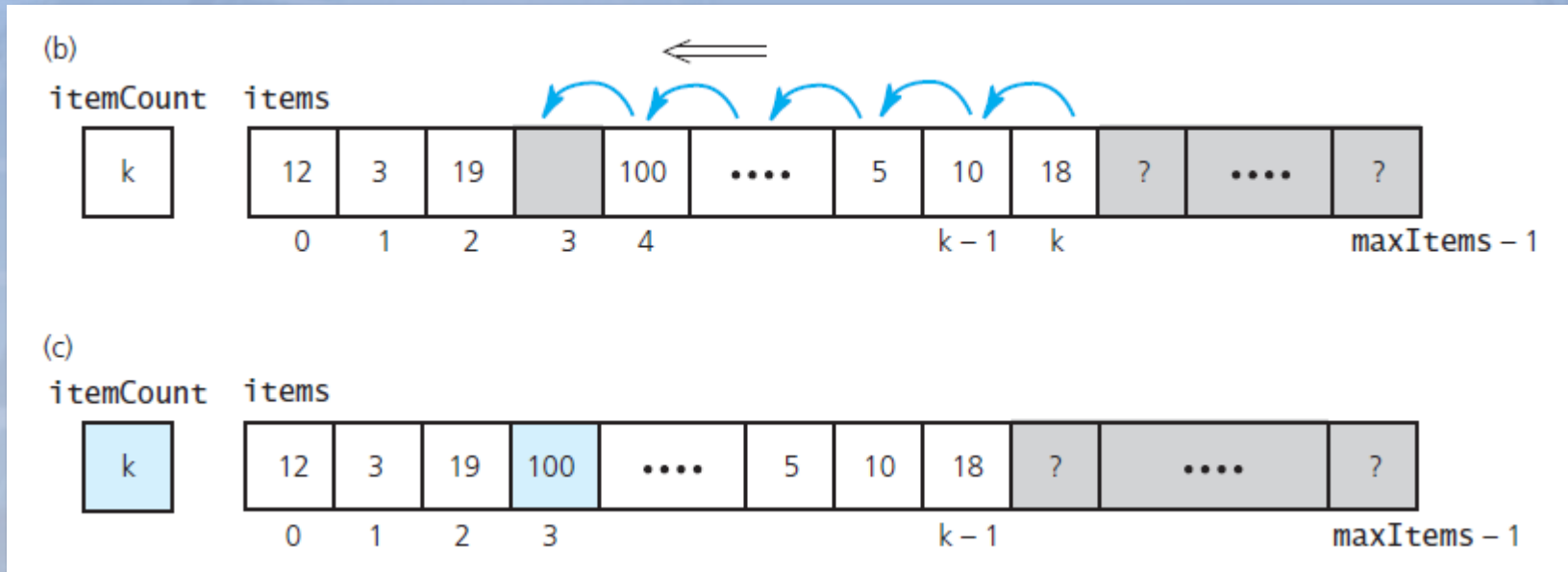


FIGURE 3-5 (b) shifting subsequent entries to avoid a gap;
(c) the array after shifting

Methods That Remove Entries

```
template<class ItemType>
int ArrayBag<ItemType>::getIndexOf(const ItemType& target) const
{
    bool isFound = false;
    int result = -1;
    int searchIndex = 0;

    // If the bag is empty, itemCount is zero, so loop is skipped
    while (!isFound && (searchIndex < itemCount))
    {
        isFound = (items[searchIndex] == target);
        if (isFound)
        {
            result = searchIndex;
        }
        else
        {
            searchIndex++;
        } // end if
    } // end while

    return result;
} // end get IndexOf
```

Method `getIndexOf`

Methods That Remove Entries

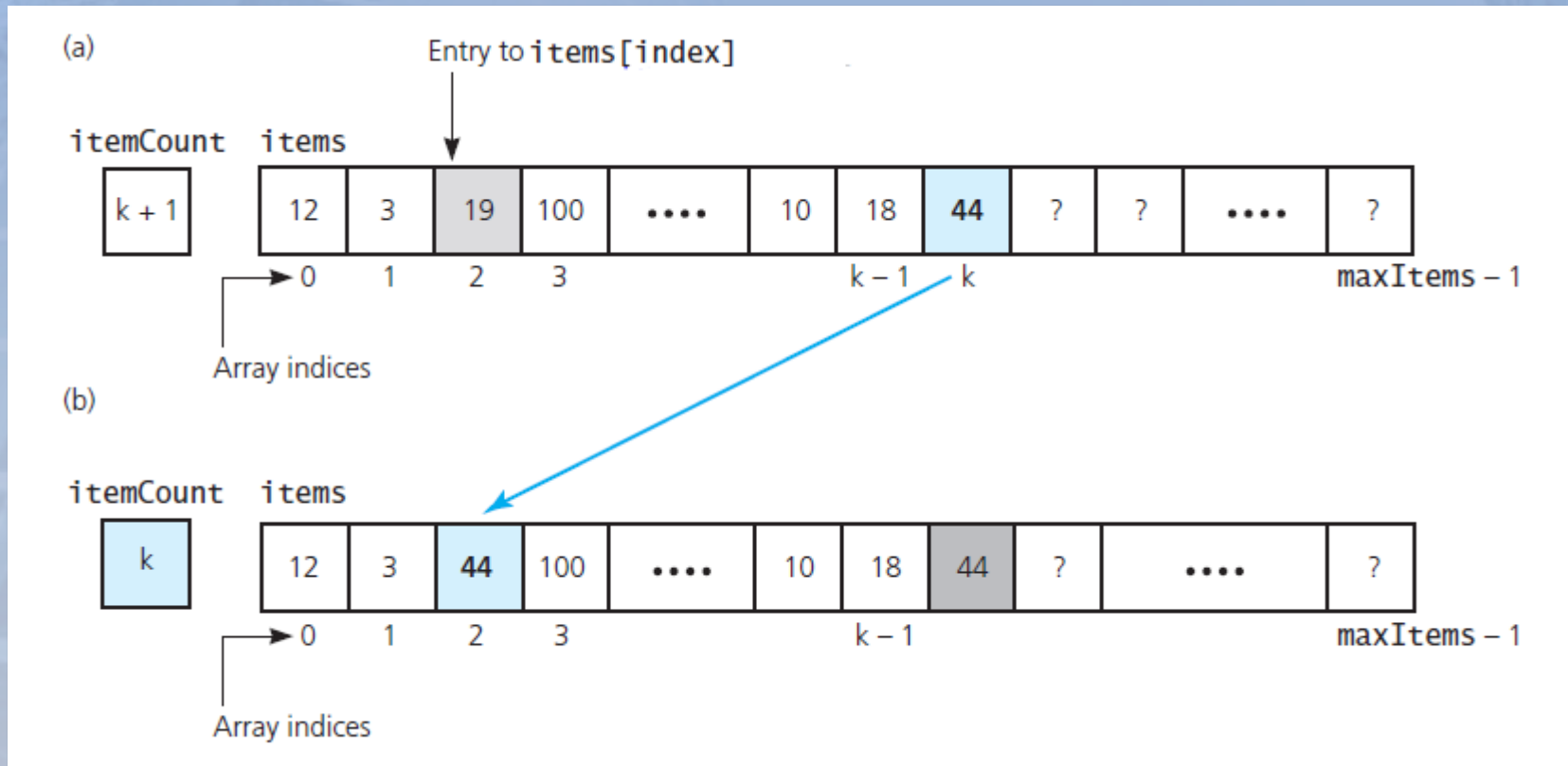


FIGURE 3-6 Avoiding a gap in the array while removing an entry

Methods That Remove Entries

```
template<class ItemType>
bool ArrayBag<ItemType>::remove(const ItemType& anEntry)
{
    int locatedIndex = getIndexOf(anEntry);
    bool canRemoveItem = !isEmpty() && (locatedIndex > -1);
    if (canRemoveItem)
    {
        itemCount--;
        items[locatedIndex] = items[itemCount];
    } // end if

    return canRemoveItem;
} // end remove
```

Method *remove*

Methods That Remove Entries

```
template<class ItemType>
void ArrayBag<ItemType>::clear()
{
    itemCount = 0;
} // end clear
```

Method *clear*

Recursion in the Implementation

```
template<class ItemType>
int ArrayBag<ItemType>::getIndexOf(const ItemType& target, int searchIndex) const
{
    int result = -1;
    if (searchIndex < itemCount)
    {
        if (items[searchIndex] == target)
        {
            result = searchIndex;
        }
        else
        {
            result = getIndexOf(target, searchIndex + 1);
        } // end if
    } // end if

    return result;
} // end getIndexOf
```

Method `getIndexOf`

Recursion in the Implementation

```
template<class ItemType>
int ArrayBag<ItemType>::countFrequency(const ItemType& target,
                                       int searchIndex) const
{
    if (searchIndex < itemCount)
    {
        if (items[searchIndex] == target)
        {
            return 1 + countFrequency(target, searchIndex + 1);
        }
        else
        {
            return countFrequency(target, searchIndex + 1);
        } // end if
    }
    else
        return 0; // Base case
} // end countFrequency
```

Method *countFrequency*

Recursion in the Implementation

```
template<class ItemType>
int ArrayBag<ItemType>::countFrequency(const ItemType& target,
                                       int searchIndex) const
{
    int frequency = 0;
    if (searchIndex < itemCount)
    {
        if (items[searchIndex] == target)
        {
            frequency = 1 + countFrequency(target, searchIndex + 1);
        }
        else
        {
            frequency = countFrequency(target, searchIndex + 1);
        } // end if
    } // end if

    return frequency;
} // end countFrequency
```

Alternative method *countFrequency*



End

Chapter 3