Northeastern University

Lecture 11: Bag Implementations that Use Arrays - 3

Prof. Chen-Hsiang (Jones) Yu, Ph.D.

College of Engineering

Materials are edited by Prof. Jones Yu from

Data Structures and Abstractions with Java, 5th edition. By Frank M. Carrano and Timothy M. Henry.
ISBN-13 978-0-13-483169-5

Fixed-Size Array

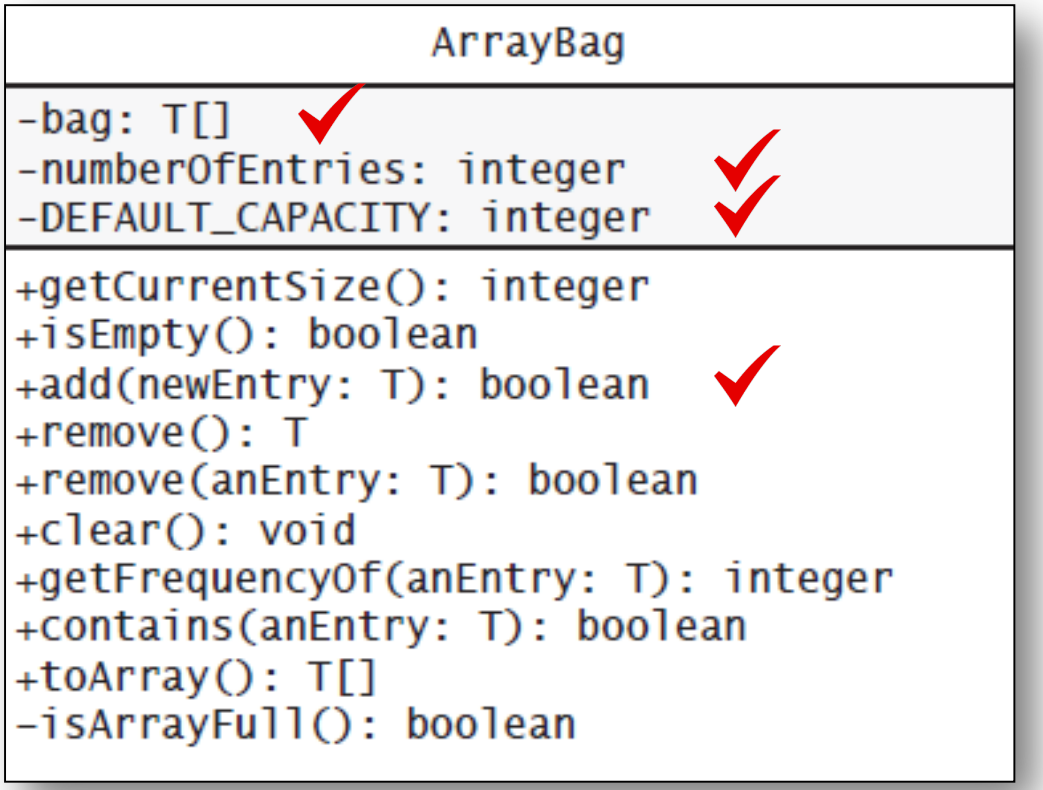

Figure 2-2: UML notation for the class `ArrayBag`, including the class's data fields

Reminder

```
/** Creates an empty bag having a given capacity.
    @param desiredCapacity  The integer capacity desired. */
public ArrayBag(int desiredCapacity)
{

   integrityOK = false;
   if (desiredCapacity <= MAX_CAPACITY)
   {
      // The cast is safe because the new array contains null entries
      @SuppressWarnings("unchecked")
      T[] tempBag = (T[])new Object[desiredCapacity]; // Unchecked cast
      bag = tempBag;
      numberOfEntries = 0;
      integrityOK = true;
   }
   else
      throw new IllegalStateException("Attempt to create a bag whose" +
                                      "capacity exceeds allowed maximum.");

} // end constructor


...


// Throws an exception if this object is not initialized.
private void checkIntegrity()
{
   if (!integrityOK)
      throw new SecurityException("ArrayBag object is corrupt.");
} // end checkIntegrity
```

Implementing More Methods

```
/** Sees whether this bag is empty.
     @return  True if this bag is empty, or false if not. */
public boolean isEmpty()
{
    return numberOfEntries == 0;
} // end isEmpty


/** Gets the current number of entries in this bag.
     @return  The integer number of entries currently in this
     bag. */
public int getCurrentSize()
{
    return numberOfEntries;
} // end getCurrentSize
```

Methods **`isEmpty`** and **`getCurrentSize`**

Implementing More Methods

```
/** Counts the number of times a given entry appears in this bag.
    @param anEntry  The entry to be counted.
    @return  The number of times anEntry appears in this bag. */
public int getFrequencyOf(T anEntry)
{
   checkIntegrity();
   int counter = 0;

   for (int index = 0; index < numberOfEntries; index++)
   {
      if (anEntry.equals(bag[index]))
      {
         counter++;
      } // end if
   } // end for

   return counter;
}
```

```
// Throws an exception if this object is not initialized.
private void checkIntegrity()
{
   if (!integrityOK)
      throw new SecurityException("ArrayBag object is corrupt.");
} // end checkIntegrity
```

Method **`getFrequencyOf`**

Implementing More Methods

```
/** Tests whether this bag contains a given entry.
    @param anEntry  The entry to locate.
    @return  True if this bag contains anEntry, or false otherwise. */
public boolean contains(T anEntry)
{
   checkIntegrity();
   boolean found = false;
   int index = 0;
   while (!found && (index < numberOfEntries))
   {
      if (anEntry.equals(bag[index]))
      {
         found = true;
      } // end if
      index++;
   } // end while
   return found;
} // end contains
```

Method **contains**

Methods That Remove Entries

```
/** Removes all entries from this bag. */
public void clear()
{
    while (!isEmpty())
        remove();
} // end clear
```

The method **clear**

Remove Entries

Methods That Remove Entries

```
/** Removes one unspecified entry from this bag, if possible.
     @return  Either the removed entry, if the removal
              was successful, or null. */
public T remove()
{
   checkIntegrity();
    T result = null;

   if (numberOfEntries > 0))
   {
       result = bag[numberOfEntries - 1];
       bag[numberOfEntries - 1] = null;
       numberOfEntries--;
   } // end if

   return result;
} // end remove
```

The method **remove**

Question

- Why does the method `remove()` replace the entry removed from the array bag with `null`?

Answer

- By setting `bag[numberOfEntries]` to `null`, the method causes the memory assigned to the deleted entry to be recycled, unless another reference to that entry exists in the client.

Methods That Remove Entries

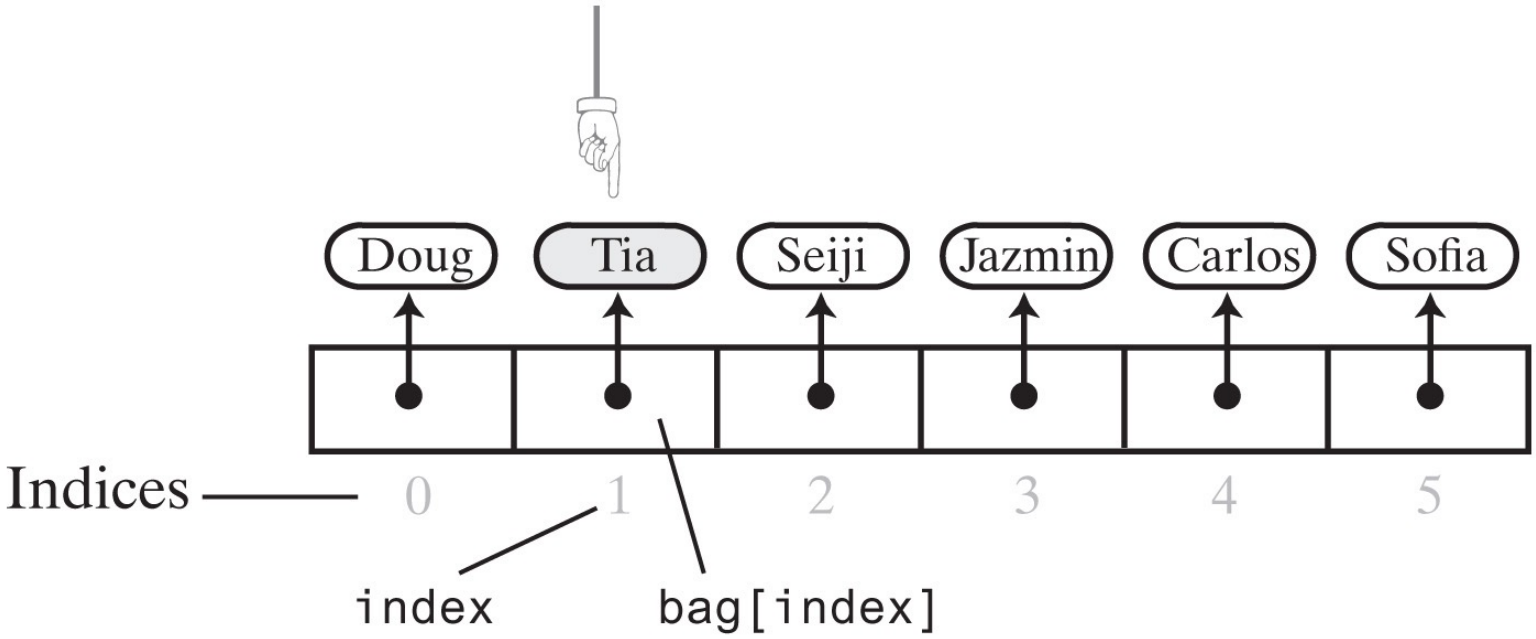

Figure 2-4: The array bag after a successful search for the string “Tia"

Methods That Remove Entries

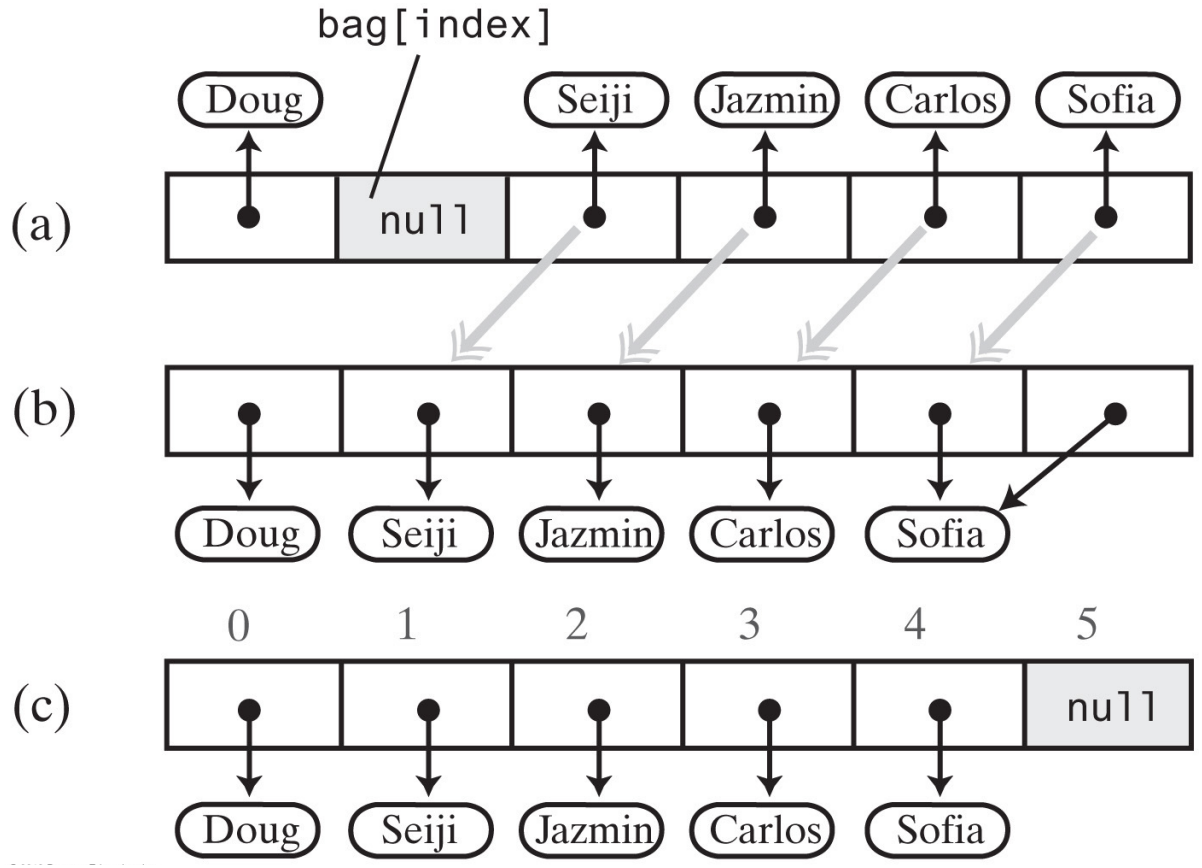

Figure 2-5: (a) A gap in the array bag after setting the entry in `bag[index]` to `null`; (b) the array after shifting subsequent entries to avoid a gap

Methods That Remove Entries

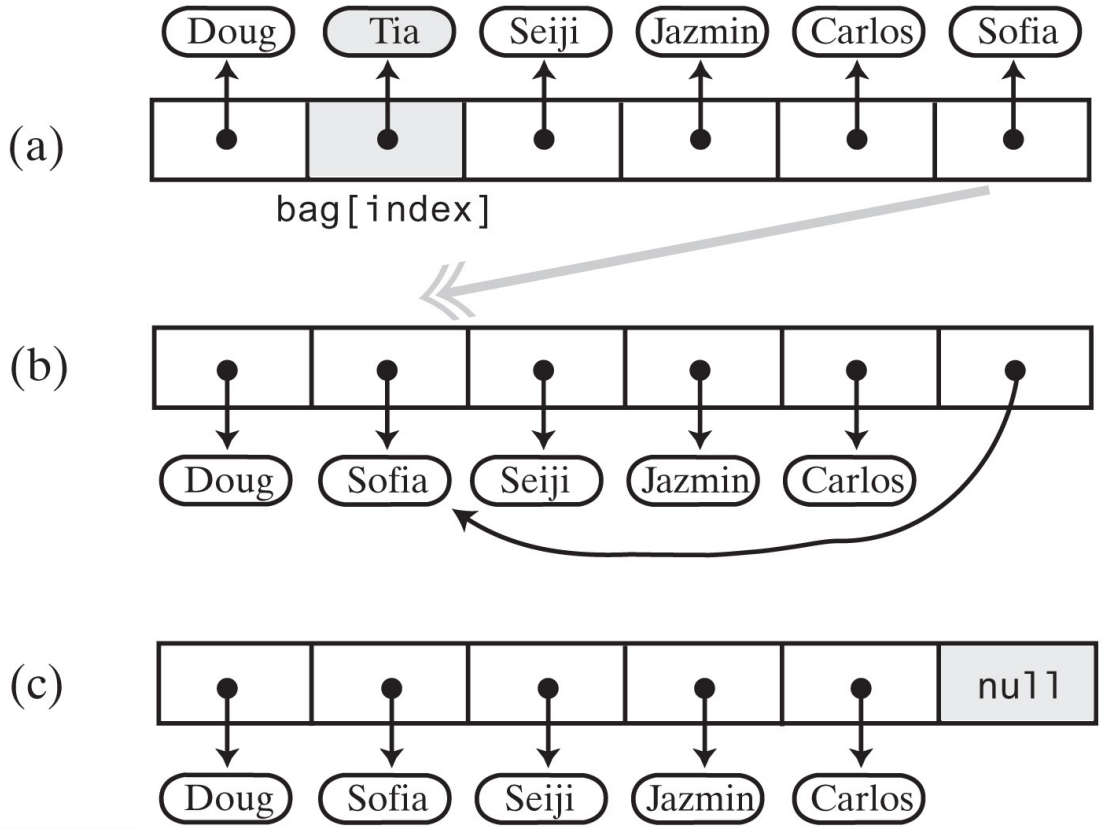

Figure 2-6: Avoiding a gap in the array while removing an entry

Methods That Remove Entries

```
/** Removes one unspecified entry from this bag, if possible.
    @return  Either the removed entry, if the removal was successful,
             or null otherwise. */
public T remove()
{
   checkIntegrity();
   T result = removeEntry(numberOfEntries - 1);
   return result;
} // end remove

/** Removes one occurrence of a given entry from this bag.
 @param anEntry  The entry to be removed.
 @return  True if the removal was successful, or false if not. */
public boolean remove(T anEntry)
{
   checkIntegrity();
   int index = getIndexOf(anEntry);
   T result = removeEntry(index);
   return anEntry.equals(result);
} // end remove
```

`removeEntry` returns the entry it removes or null

The revised **`remove`** method

Methods That Remove Entries

```
// Removes and returns the entry at a given index within the array bag.
// If no such entry exists, returns null.
// Preconditions: 0 <= givenIndex < numberOfEntries;
//                checkIntegrity has been called.
private T removeEntry(int givenIndex)
{

   T result = null;

   if (!isEmpty() && (givenIndex >= 0))
   {
       result = bag[givenIndex];                    // Entry to remove
       bag[givenIndex] = bag[numberOfEntries - 1]; // Replace entry with last entry
       bag[numberOfEntries - 1] = null;             // Remove last entry
       numberOfEntries--;
   } // end if

   return result;

} // end removeEntry
```

The **`removeEntry`** method

Question

```
public boolean remove(T anEntry)
{
   checkIntegrity();
   int index = getIndexOf(anEntry);
   T result = removeEntry(index);

   return anEntry.equals(result);
}
```

- As shown above, can the return statement in the `remove` method to be written as follows?

```
a. return result.equals(anEntry);

b. return result != null;
```

Answer

a. `return result.equals(anEntry);` 

b. `Return result != null;` 

Methods That Remove Entries

```
public boolean contains(T anEntry)
{

   checkIntegrity();
   return getIndexOf(anEntry) > -1; // or >= 0

} // end contains
```

Revised definition for the method **contains**

Methods That Remove Entries

```
// Locates a given entry within the array bag.
// Returns the index of the entry, if located, or -1 otherwise.
// Precondition: checkIntegrity has been called.
private int getIndexOf(T anEntry)
{
   int where = -1;
   boolean found = false;
   int index = 0;

   while (!found && (index < numberOfEntries))
   {
      if (anEntry.equals(bag[index]))
      {
         found = true;
         where = index;
      } // end if
      index++;
   } // end while

   // Assertion: If where > -1, anEntry is in the array bag, and it
   // equals bag[where]; otherwise, anEntry is not in the array

   return where;
} // end getIndexOf
```

Definition for the method **`getIndexOf`**

Fixed-Size Array

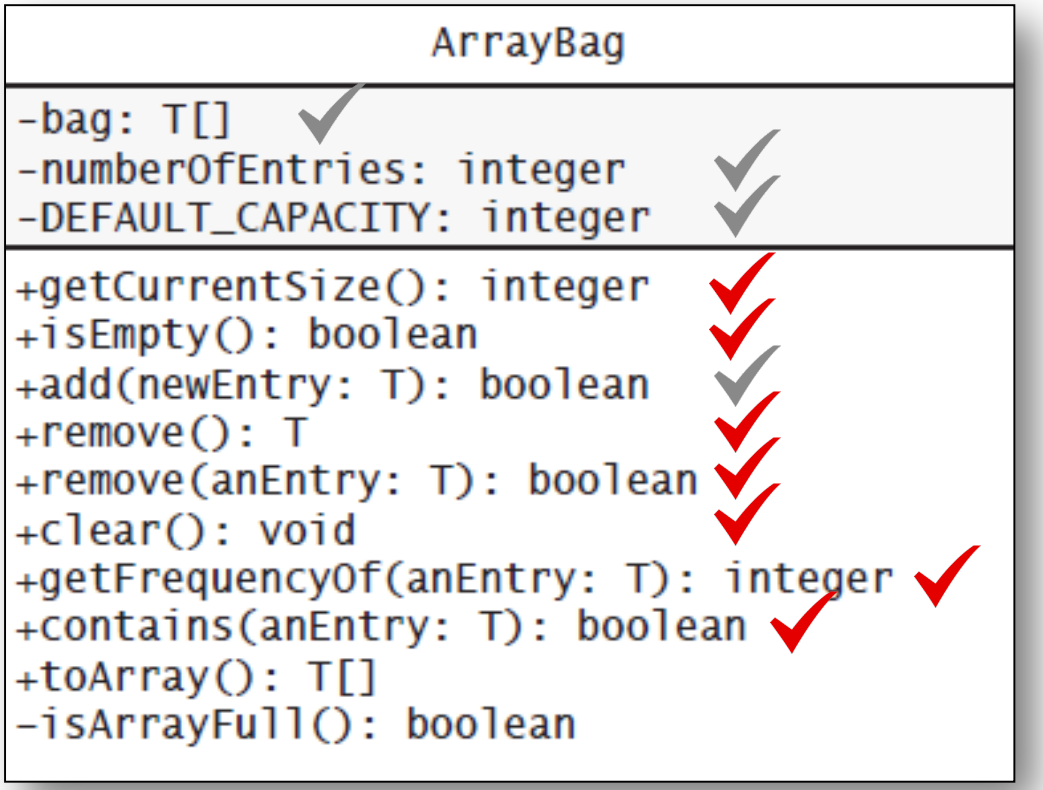

Figure 2-2: UML notation for the class **`ArrayBag`**, including the class's data fields

Question

- Please revise the definition of the method `getIndexOf()` so that it does not use a Boolean variable.

```
private int getIndexOf(T anEntry)
{
   int where = -1;
   boolean found = false;
   int index = 0;

   while (!found && (index < numberOfEntries))
   {
      if (anEntry.equals(bag[index]))
      {
         found = true;
         where = index;
      } // end if
      index++;
   } // end while

   return where;
}
```

Answer

```
private int getIndexOf(T anEntry)
{
    int where = -1;
    for (int index = 0; (where == -1) && (index < numberOfEntries); index++)
    {
        if (anEntry.equals(bag[index]))
            where = index;
    } // end for
    return where;
} // end getIndexOf
```

or

```
private int getIndexOf(T anEntry)
{
    int where = numberOfEntries - 1;
    while ((where > -1) && !anEntry.equals(bag[where]))
        where--;
    return where;
} // end getIndexOf
```

Exercise

- Download L11_E1 from “In-class Exercise” on the Brightspace
- Import the project to the Eclipse

- Run “ArrayBagDemo3.java”
- Questions:
  - » Why do you have this result?
  - » What are we doing in testAdd(), testRemove() and testIsEmpty()?

Results

```
==============================
Testing an initially empty bag:

Testing the two remove methods:

Removing a string from the bag:
remove() returns null
The bag contains 0 string(s), as follows:


Removing "B" from the bag:
remove("B") returns false
The bag contains 0 string(s), as follows:

------------------------------
Adding to the bag: A A B A C A
The bag contains 6 string(s), as follows:
A A B A C A
------------------------------

Testing the two remove methods:

Removing a string from the bag:
remove() returns A
The bag contains 5 string(s), as follows:
A A B A C

Removing "A" from the bag:
remove("A") returns true
The bag contains 4 string(s), as follows:
C A B A

Removing "C" from the bag:
remove("C") returns true
The bag contains 3 string(s), as follows:
A A B

Removing "Z" from the bag:
remove("Z") returns false
The bag contains 3 string(s), as follows:
A A B
------------------------------

Clearing the bag:

Testing the method isEmpty with an empty
bag:
isEmpty finds the bag empty: OK.
The bag contains 0 string(s), as follows:

==============================
```

Using Array Resizing to Implement the ADT Bag

Using Array Resizing

- An array has a fixed size. It has pros and cons. (We will summarize it in a later slide)
- Using a fixed size array to implement the ADT bag limits the size of the bag.
- When an array becomes full, you can move its contents to a larger array. This process is called resizing an array or array resizing.

Using Array Resizing

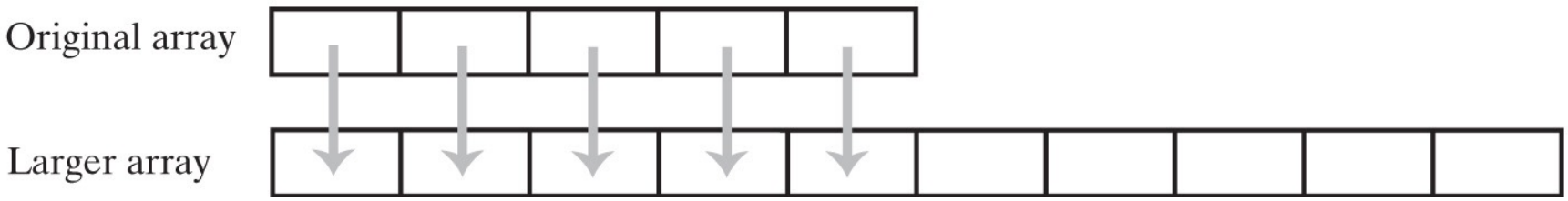

Figure 2-7: Resizing an array copies its contents to a larger second array

Using Array Resizing

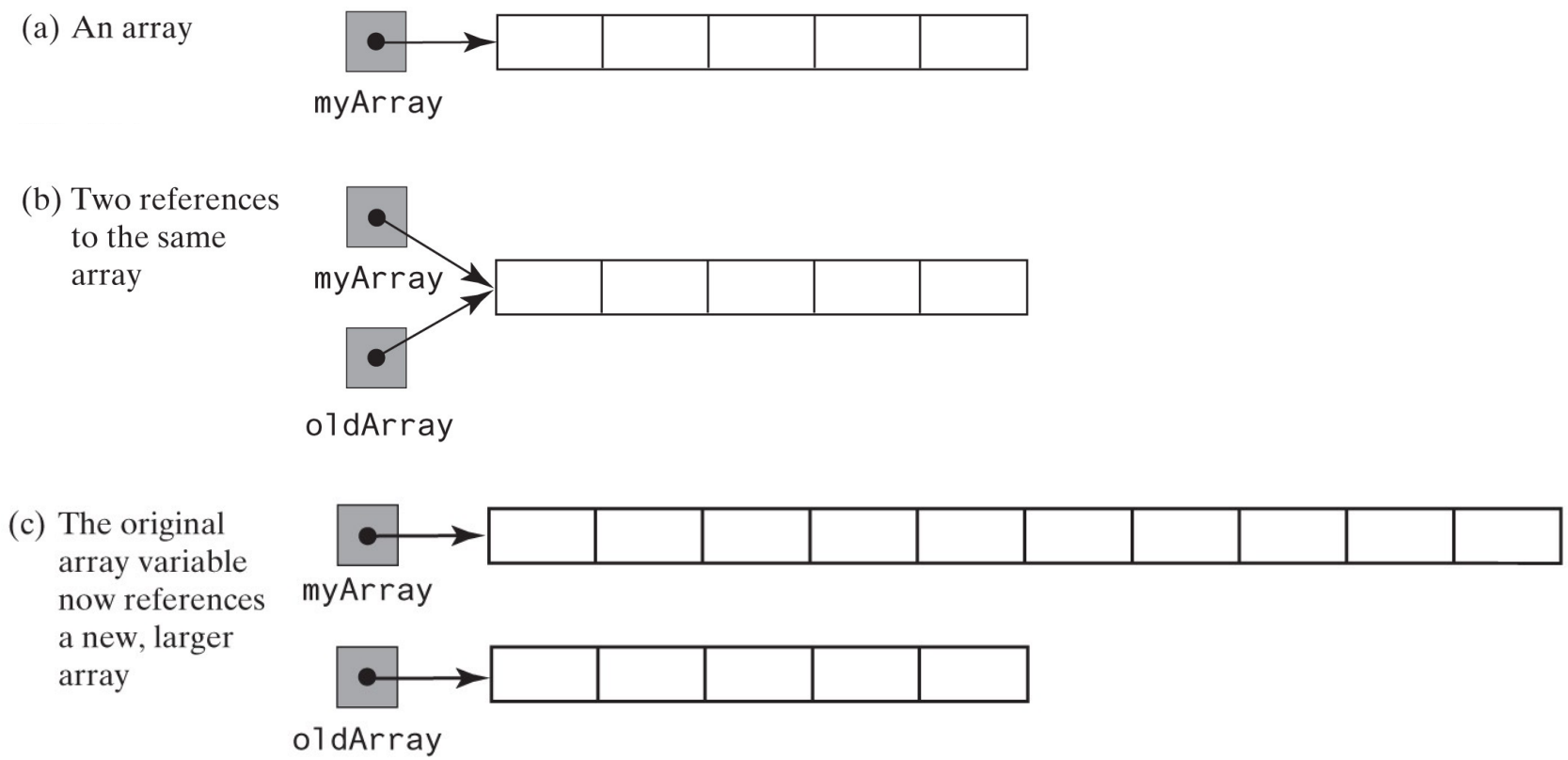

Figure 2-8: (a) An array; (b) two references to the same array; (c) the original array variable now references a new, larger array;

Using Array Resizing

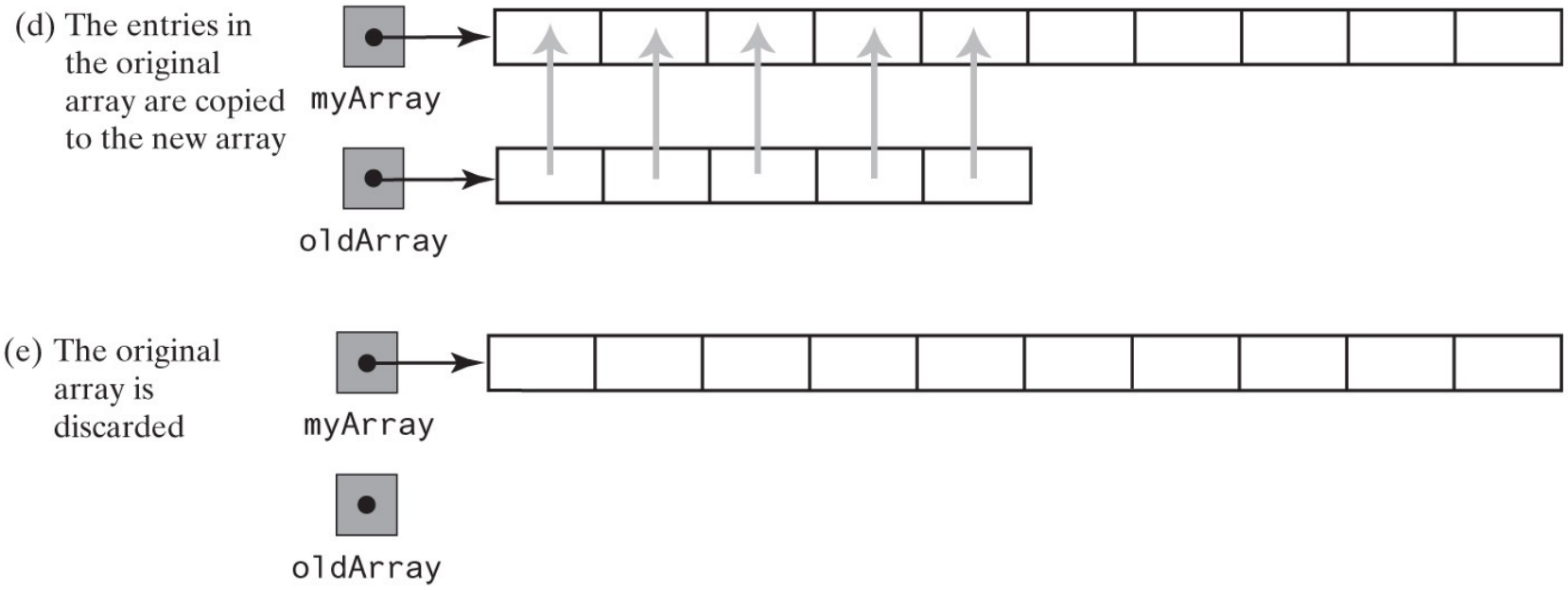

Figure 2-8: (d) the entries in the original array are copied to the new array; (e) the original array is discarded

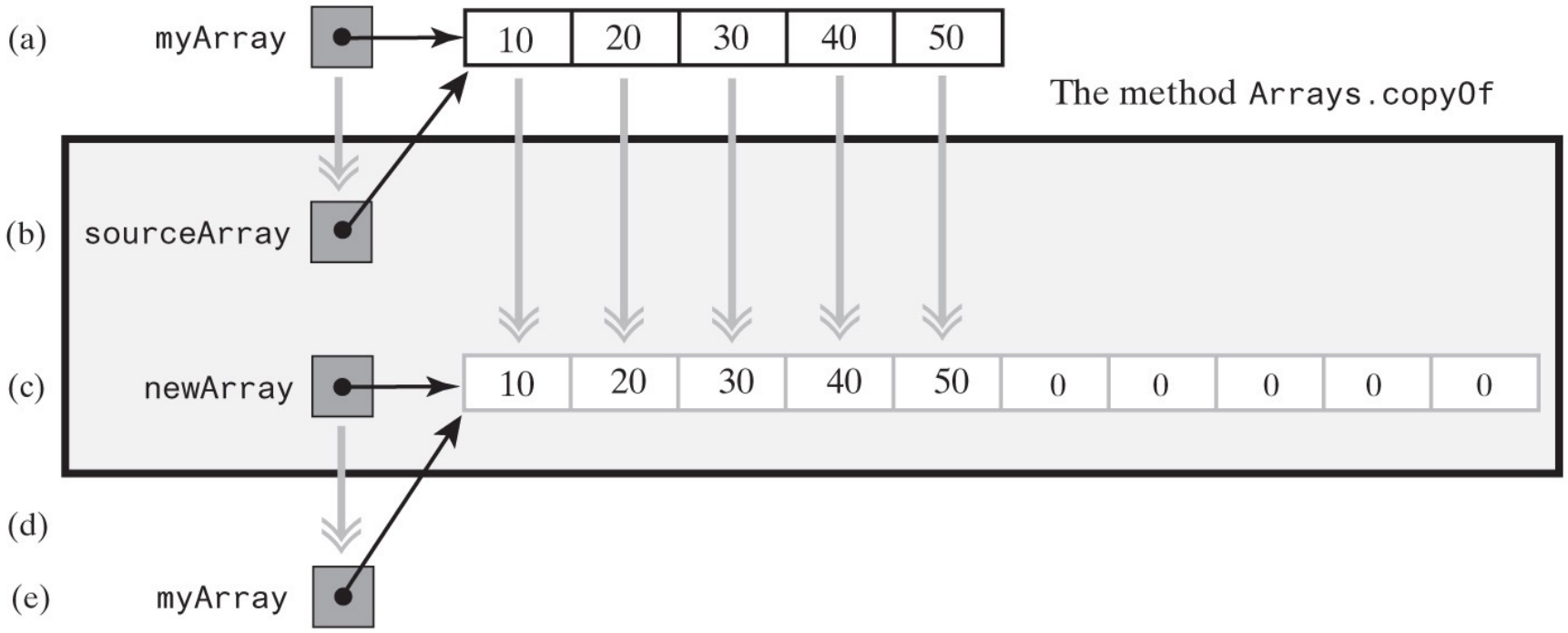

Figure 2-9: The effect of the statement

```
myArray = Arrays.copyOf(myArray, 2 * myArray.length);
```

(a) The argument array; (b) the parameter that references the argument array; (c) a new, larger array that gets the contents of the argument array; (d) the return value that references the new array; (e) the argument variable is assigned the return value

Arrays class: https://docs.oracle.com/javase/8/docs/api/java/util/Arrays.html

Exercise

- Consider the array of strings that the following statement defines:

```
String[] text = {“cat”, “dog”, “bird”, “snake”};
```

What Java statements will increase the capacity of the array `text` by five elements without altering its current contents?

Answer

```
text = Arrays.copyOf(text, text.length + 5);
```

or

```
String[] origText = text;
text = new String[text.length + 5];
System.arraycopy(origText, 0, text, 0, origText.length);
```

New Implementation of a Bag

```
/** Adds a new entry to this bag.
    @param newEntry  The object to be added as a new entry.
    @return  True.  */
public boolean add(T newEntry)
{
   checkIntegrity();
   boolean result = true;
   if (isArrayFull())
   {
      doubleCapacity();
   } // end if

   bag[numberOfEntries] = newEntry;
   numberOfEntries++;

   return true;
} // end add
```

Revised definition of method **add**

New Implementation of a Bag

```
// Throws an exception if the client requests a capacity that is too large.
private void checkCapacity(int capacity)
{
   if (capacity > MAX_CAPACITY)
      throw new IllegalStateException("Attempt to create a bag whose " +
                                      "capacity exeeds allowed " +
                                      "maximum of " + MAX_CAPACITY);
} // end checkCapacity

// Doubles the size of the array bag.
// Precondition: checkIntegrity has been called.
private void doubleCapacity()
{
   int newLength = 2 * bag.length;
   checkCapacity(newLength);
   bag = Arrays.copyOf(bag, newLength);
} // end doubleCapacity
```

The methods **dcheckCapacity** and **doubleCapacity**

Pros and Cons of Using an Array

- Pros:
  - » Adding an entry to the bag is fast
  - » Removing an unspecified entry is fast
- Cons:
  - » Removing a particular entry requires time to locate the entry
  - » Increasing the size of the array requires time to copy its entries

Exercise (offline)

- Continue from L11_E1
- Please copy ArrayBag.java to create a new class, ResizableArrayBag, which can support Array Resizing.
- Please also write a testing program for it.

p.s. Array Resizing happens when you add elements into the bag, which is initialized with limited size.

Answer

- ResizableArrayBag.java (L11_E2_Solution.zip)