University of Pittsburgh

Algorithms and Data Structures 1
CS 0445

Fall 2022

Sherif Khattab

ksm73@pitt.edu

(Slides are adapted from Dr. Ramirez's and Dr. Farnan's CS1501 slides.)

Announcements

- Upcoming Deadlines:
  - Homework 2: this Friday @ 11:59 pm
  - Lab 1: next Monday @ 11:59 pm
  - Programming Assignment 1: Friday Oct. 7th
- Draft slides and handouts available on Canvas
- **Student Support Hours** of the teaching team are posted on the Syllabus page

Previous Lecture …

- ADT Bag Implementations
  - Fixed-size array: ArrayBag
    - add, toArray, size, isFull, isEmpty, and clear methods
    - Making our implementation more secure

Muddiest Points

- **Q: I'm confused about what we mean by physical size vs. logical size**
- A: physical size of an array is the number of items that it can hold; logical size is the actual number of items it holds
- **Q: I still do not understand the specific circumstances one should use a Bag when coding.**
- A: Use a Bag in any application that involves dealing with a set of items whereby the order of the items doesn't matter, and duplicates are allowed.
- **Q: What are the advantages/differences over a simple array?**
- A: Bag hides the details of the implementation. The client code doesn't need to change if one later decides to use a linked implementation instead of an array-based one
- **Q: Mainly just what throwing is and when we have to do it**
- A: Throwing an exception is a way to inform the caller of an error situation
- **Q: what if you have an empty index earlier in the array when the logical size is equal to the physical size? Is it still full?**
- A: The array discipline dictates that filled array entries have to be adjacent with no gaps

Muddiest Points

- **Q: I still am confused as to why wild card type parameters would be needed instead of generic ones**
- A: Wild card type parameters are needed when we want to specify bounds on the acceptable types
  - (e.g., ArrayBag<? extends Number>).
- Wild cards are preferred over type parameters when there is no dependencies between method parameters and/or return type
  - (e.g., void display(ArrayBag<?> bag){ … })

Muddiest Points

- **Q: When you are creating a new exception i.e. "throw new ..." are you creating the name of a new type of exception or using one from a set of already created exception names**.
- A: using an already defined exception class (we can also define our own exception class by extending Exception)
- **Q: Why is ArrayBag<?> the superclass of ArrayBag<Square>?**
- This is by definition.
- **Q: Is ArrayBag<Object> the superclass of ArrayBag<?>**
- A: No! Object is the superclass of ArrayBag<?>

Muddiest Points

- **Q: Creating a deeper copy of the elements in an ArrayBag**

```
//Deep copy

Import java.lang.reflect.Constructor;

    Class<?> c = bag[i].getClass();

    try{

      Constructor<?> copyConstructor = c.getConstructor(c);

      @SuppressWarnings("unchecked")

      T copy = (T) copyConstructor.newInstance(bag[i]);

      result[i] = copy;//deep copy of array elements

    } catch (Exception e){

      result[i] = bag[i]; //fall back to shallow copy if no copy constructor

    }
```

Muddiest Points

- **Q: Everything**
- A: I have personally struggled a lot when I was student in a Data Structures class!
- Please reach out during student support hours and over Piazza!

Today's Agenda

- ADT Bag Implementations
  - Fixed-size array: ArrayBag
    - getFrequencyOf(T), contains(T), remove(), remove(T), copy constructor

A note on toArray()

- Assume
  - ArrayBag<String> bagOfStrings = new ArrayBag<>();
  - bagOfStrings.add(“This”);
  - bagOfStrings.add(“is”);
  - bagOfStrings.add(“fun!”);
- Since toArray is defined to return T[], Java compiler will convert it to
  - Object[] toArray()
  - This is called type erasure
- Java compiler will also insert type casting parentheses in the following statement:
  - String[] strings = bagOfStrings.toArray();
  - So, it becomes:
    - String[] strings = (String[]) bagOfStrings.toArray();
- Since it is not legal to cast Object[] to String[], this will throw a ClassCastException at run-time
- Solution:
  - Object[] strings = bagOfStrings.toArray();

Implementing More Methods

Method `getFrequencyOf`

```
public int getFrequencyOf(T anEntry)
{
  checkInitialization();
  int counter = 0;

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

  return counter;
} // end getFrequencyOf
```

Implementing More Methods

Method `contains`

```
public boolean contains(T anEntry)
{
   checkInitialization();
   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
```

Methods That Remove Entries

The method **clear**

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

Methods That Remove Entries

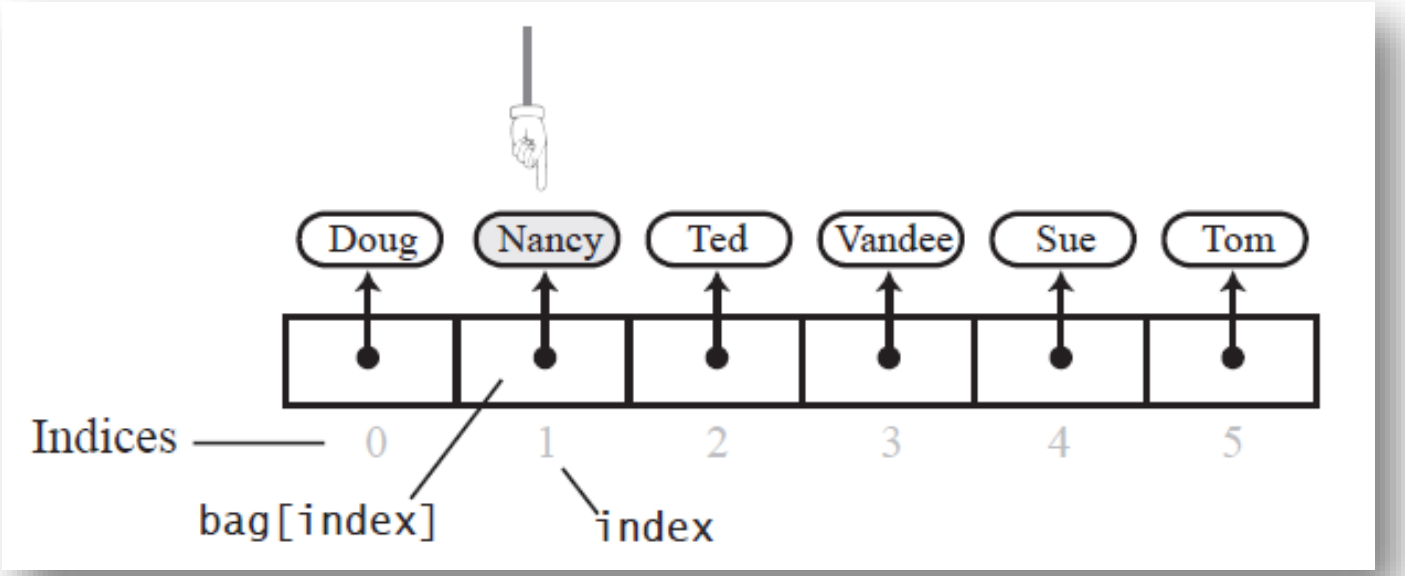
The method **`remove`**

```
public T remove()
{
    checkInitialization();
    T result = null;
    if (numberOfEntries > 0)
    {
        result = bag[numberOfEntries - 1];
        bag[numberOfEntries - 1] = null;
        numberOfEntries--;
    } // end if

    return result;
} // end remove
```

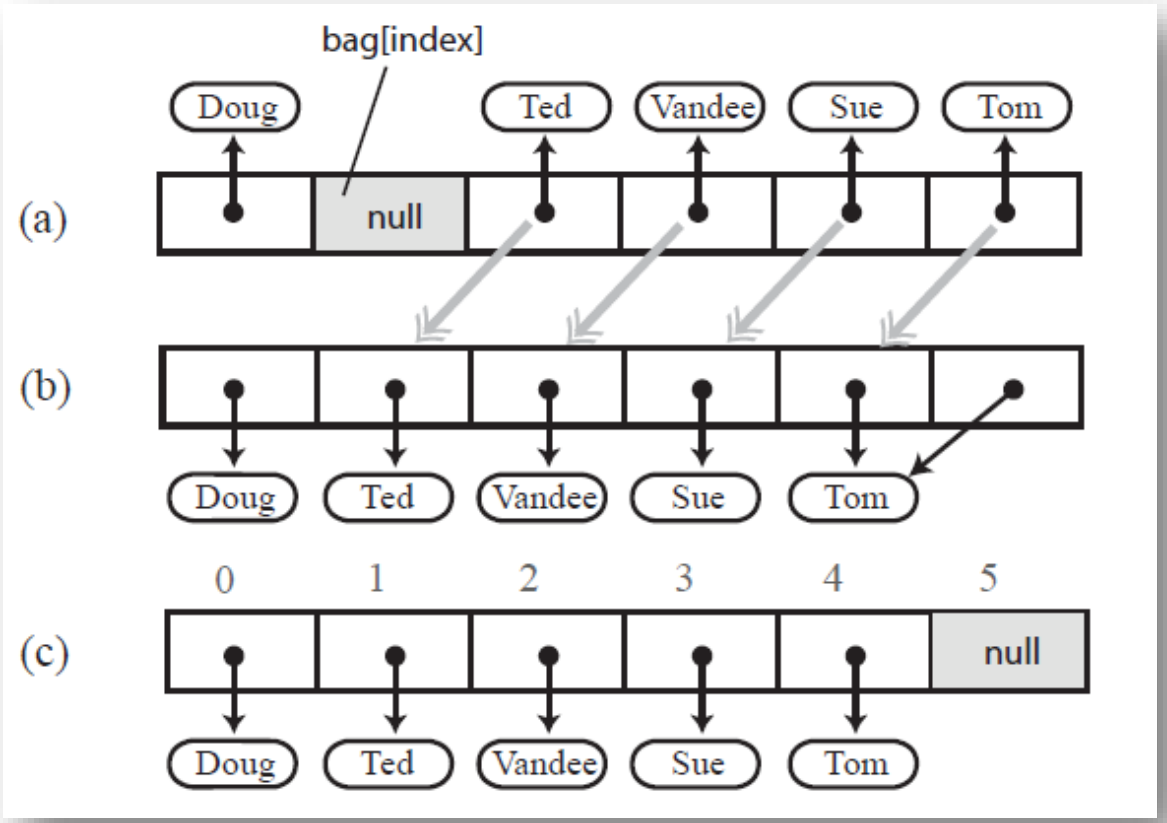
Methods That Remove Entries

The array bag after a successful search for the string “Nancy"



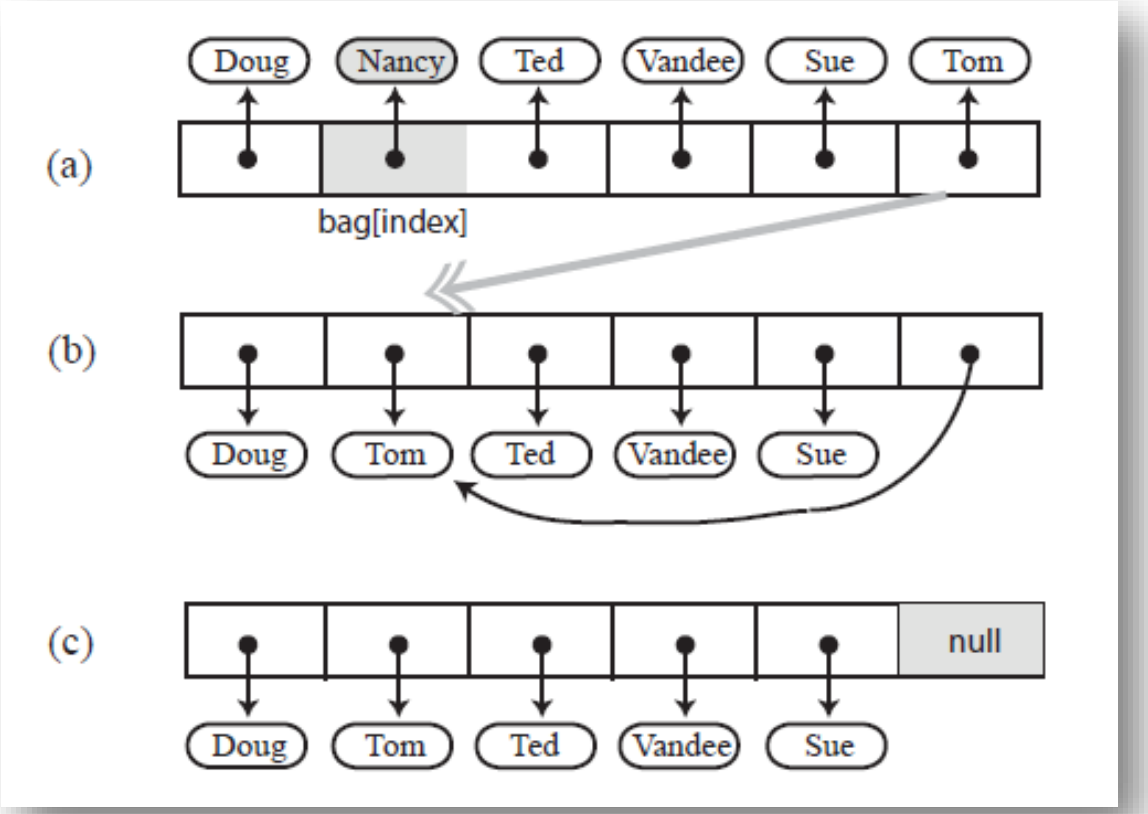
Methods That Remove Entries

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



Methods That Remove Entries

Avoiding a gap in the array while removing an entry



Methods That Remove Entries

New definition of **`remove`**

```
public T remove()
{
    checkInitialization();
    T result = null;
    if (numberOfEntries > 0)
    {
        result = bag[numberOfEntries - 1];
        bag[numberOfEntries - 1] = null;
        numberOfEntries--;
    } // end if

    return result;
} // end remove
```

Methods That Remove Entries

The second **`remove`** method

```
/** 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)
{
   checkInitialization();
   int index = getIndexOf(anEntry);
   T result = removeEntry(index);
   return anEntry.equals(result);
} // end remove
```

Methods That Remove Entries

The **`removeEntry`** method

```
// Removes and returns the entry at a given index within the array bag.
// If no such entry exists, returns null.
// Preconditions: 0 <= givenIndex < numberOfEntries;
//                checkInitialization 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
```

Methods That Remove Entries

Definition for the method `getIndexOf`

```
// Locates a given entry within the array bag.
// Returns the index of the entry, if located, or -1 otherwise.
// Precondition: checkInitialization 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
```

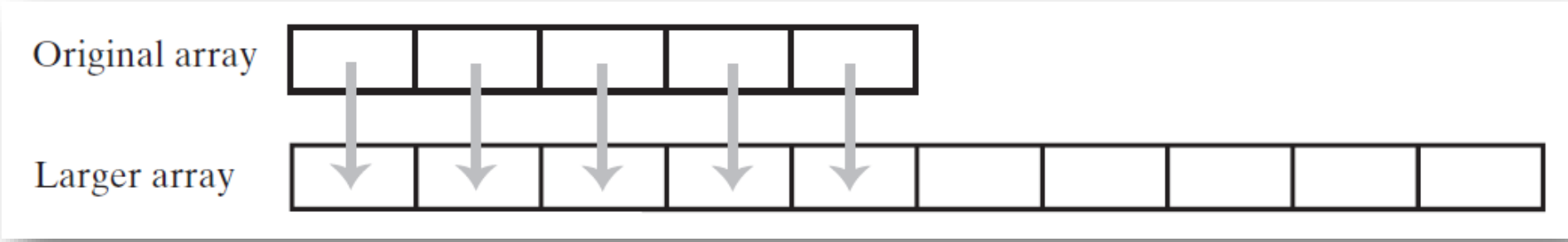
Methods That Remove Entries

Revised definition for the method **`contains`**

```
public boolean contains(T anEntry)
{
    checkInitialization();
    return getIndexOf(anEntry) > -1;
} // end contains
```

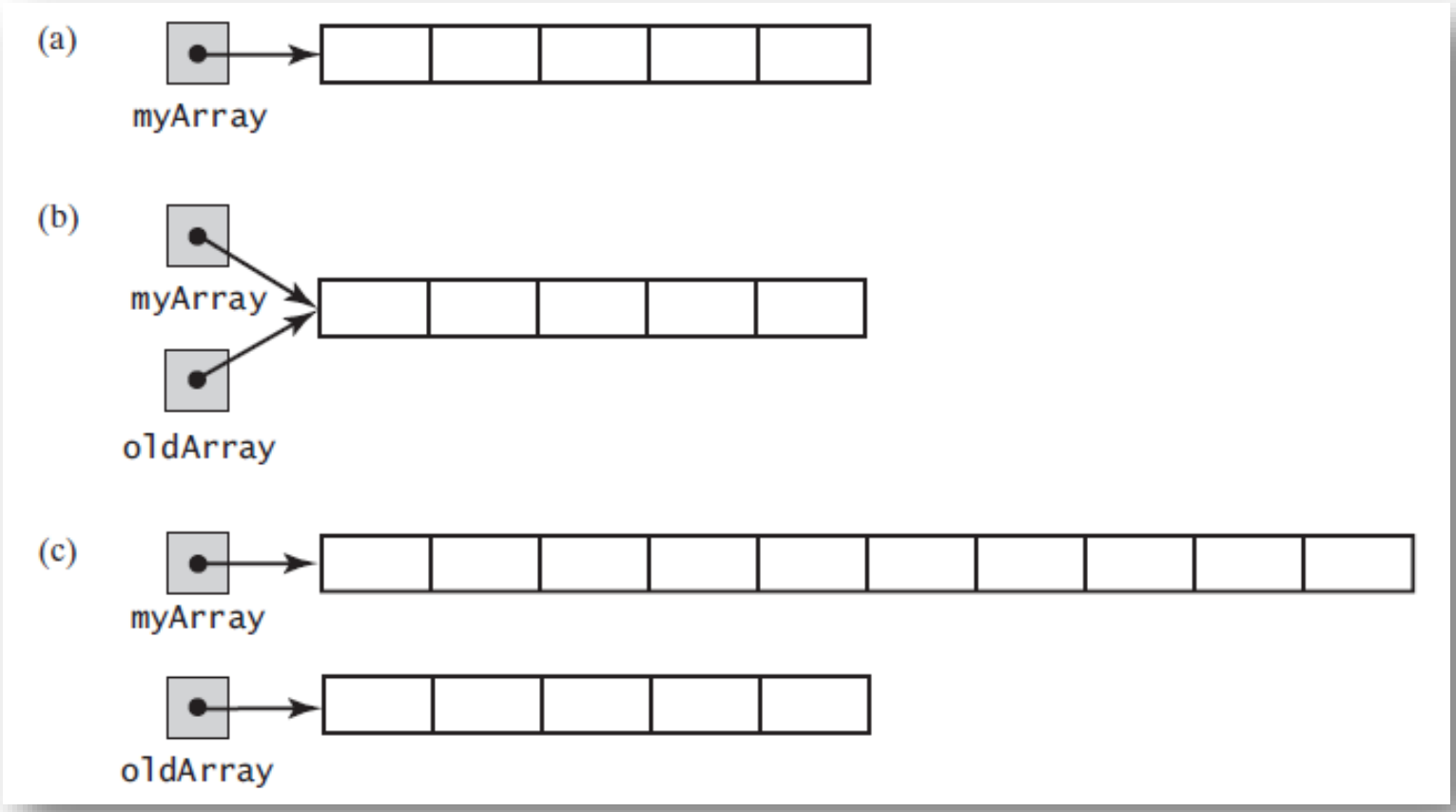
Using Array Resizing

Resizing an array copies its contents to a larger second array



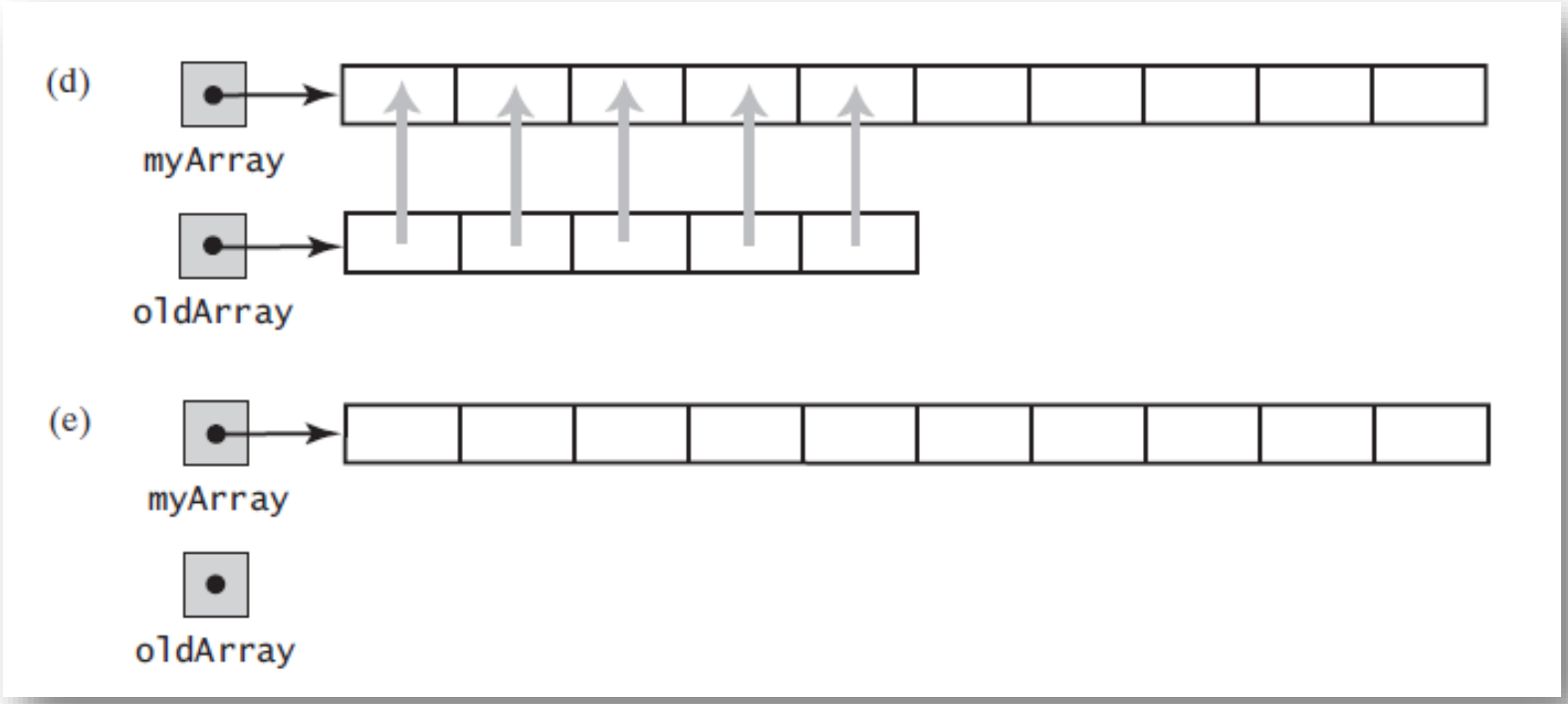
Using Array Resizing

(a) An array; (b) two references to the same array; (c) the original array variable now references a new, larger array;



Using Array Resizing

(d) the entries in the original array are copied to the new array; (e) the original array is discarded

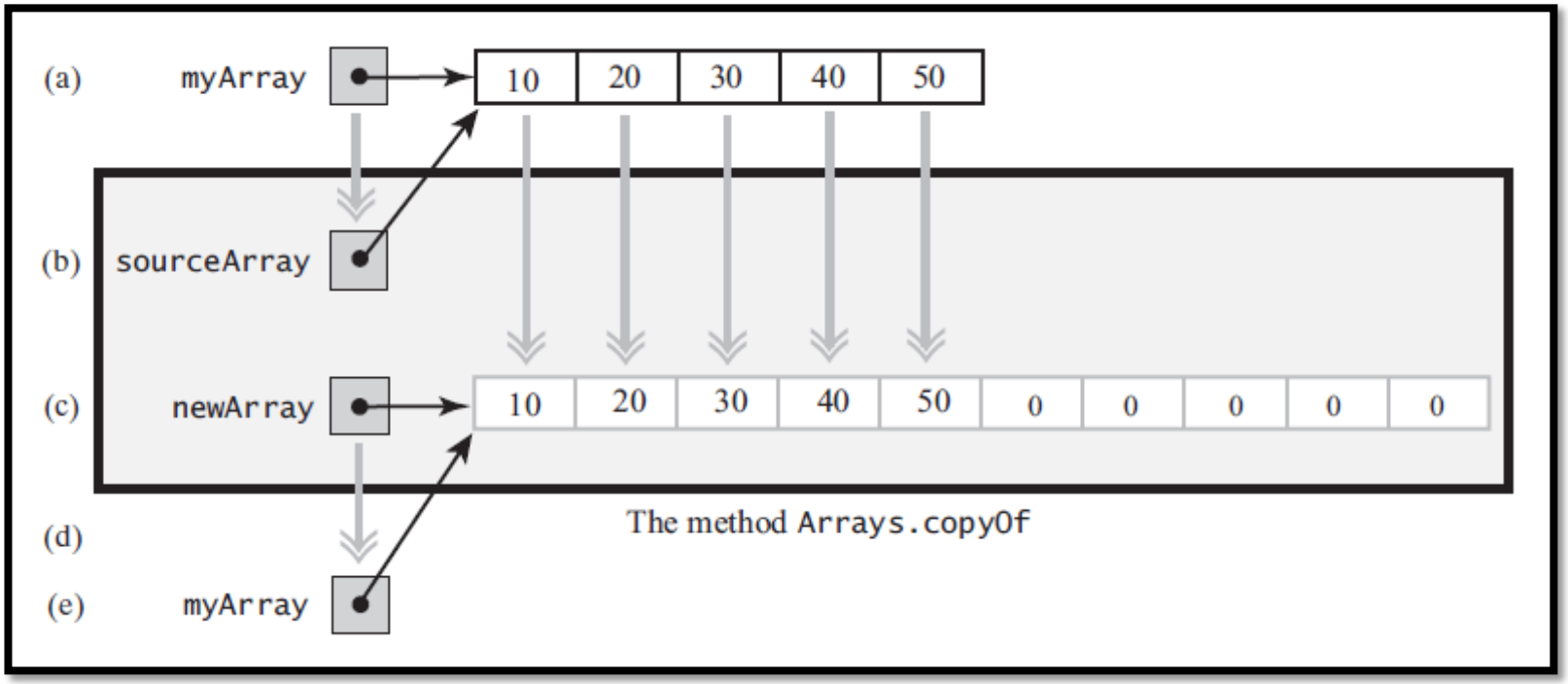


Using Array Resizing

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



New Implementation of a Bag

Previous definition of method **add**

```
public boolean add(T newEntry)
{
    checkInitialization();
    boolean result = true;
    if (isArrayFull())
    {
        result = false;
    }
    else
    { // Assertion: result is true here
      bag[numberOfEntries] = newEntry;
      numberOfEntries++;
    } // end if
    return result;
} // end add
```

New Implementation of a Bag

The method **doubleCapacity**

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

Using a Bag

A Bag is a simple ADT, but it can still be useful

- See examples in text
- Here is another simple one
  - A number of players “shout” Snap! each with a certain probability.
  - Add the player number to a Bag if she shouts.
  - Count the number of shouts in the Bag.

Pros and Cons of Using an Array

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

Problems with Array Implementation

- Array has fixed size
- May become full
- Alternatively may have wasted space
- Resizing is possible but requires time overhead