Northeastern University

Lecture 9: Bag Implementations that Use Arrays - 1

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 to Implement the ADT Bag

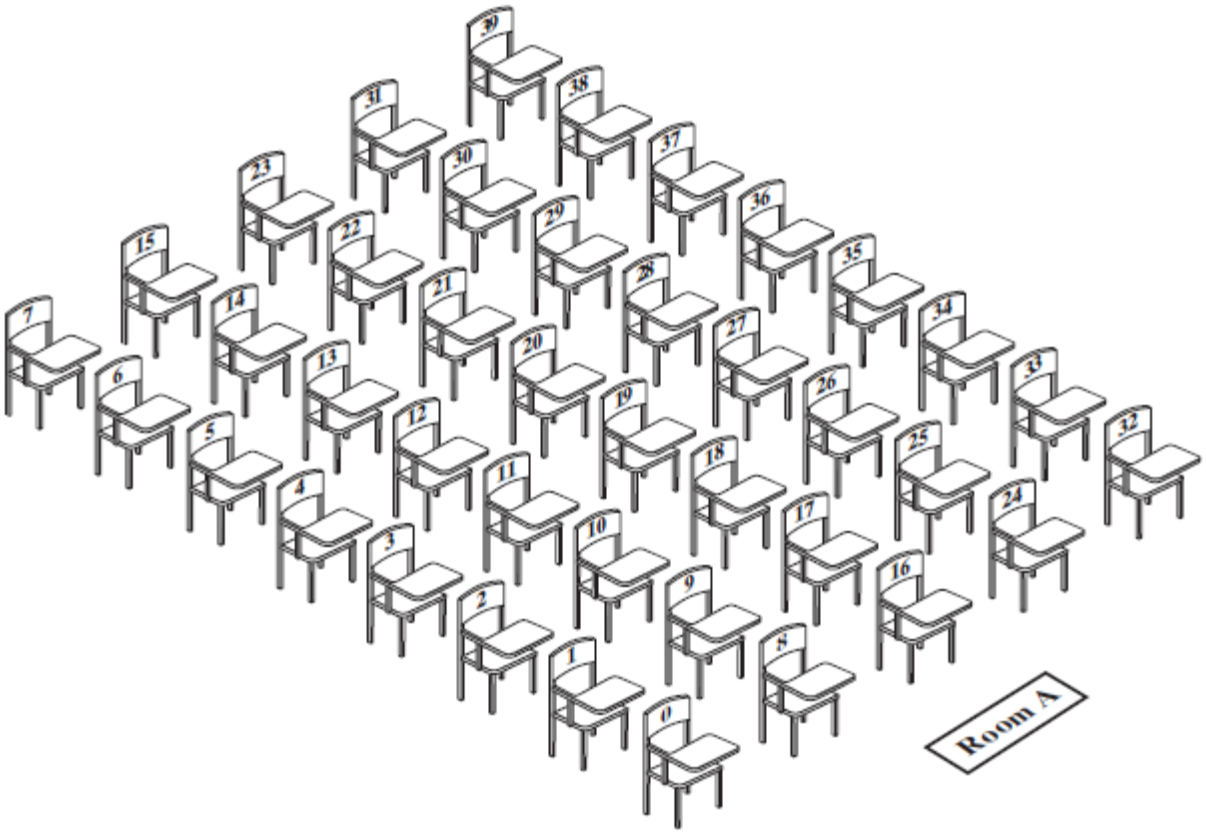

Figure 2-1: A classroom that contains desks in fixed positions

Implementing the Core Methods

Recall the Interface - BagInterface

```java
/**
   An interface that describes the operations of a bag of objects.
   @author Frank M. Carrano
*/
public interface BagInterface<T>
{
   /** Gets the current number of entries in this bag.
       @return  The integer number of entries currently in the bag. */
   public int getCurrentSize();

   /** Sees whether this bag is empty.
       @return  True if the bag is empty, or false if not. */
   public boolean isEmpty();

   /** Adds a new entry to this bag.
       @param newEntry  The object to be added as a new entry.
       @return  True if the addition is successful, or false if r
   public boolean add(T newEntry);

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

   /** Removes one occurrence of a given entry from this bag, if
       @param anEntry  The entry to be removed.
       @return  True if the removal was successful, or false if r
   public boolean remove (T anEntry);
```

```java
   /** Removes one occurrence of a given entry from this bag, if possible.
       @param anEntry  The entry to be removed.
       @return  True if the removal was successful, or false if not. */
   public boolean remove (T anEntry);

   /** Removes all entries from this bag. */
   public void clear();

   /** 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 the bag. */
   public int getFrequencyOf(T anEntry);

   /** Tests whether this bag contains a given entry.
       @param anEntry  The entry to locate.
       @return  True if the bag contains anEntry, or false if not. */
   public boolean contains(T anEntry);

   /** Retrieves all entries that are in this bag.
       @return  A newly allocated array of all the entries in the bag.
                Note: If the bag is empty, the returned array is empty. */
   public T[] toArray();
} // end BagInterface
```

Fixed-Size Array

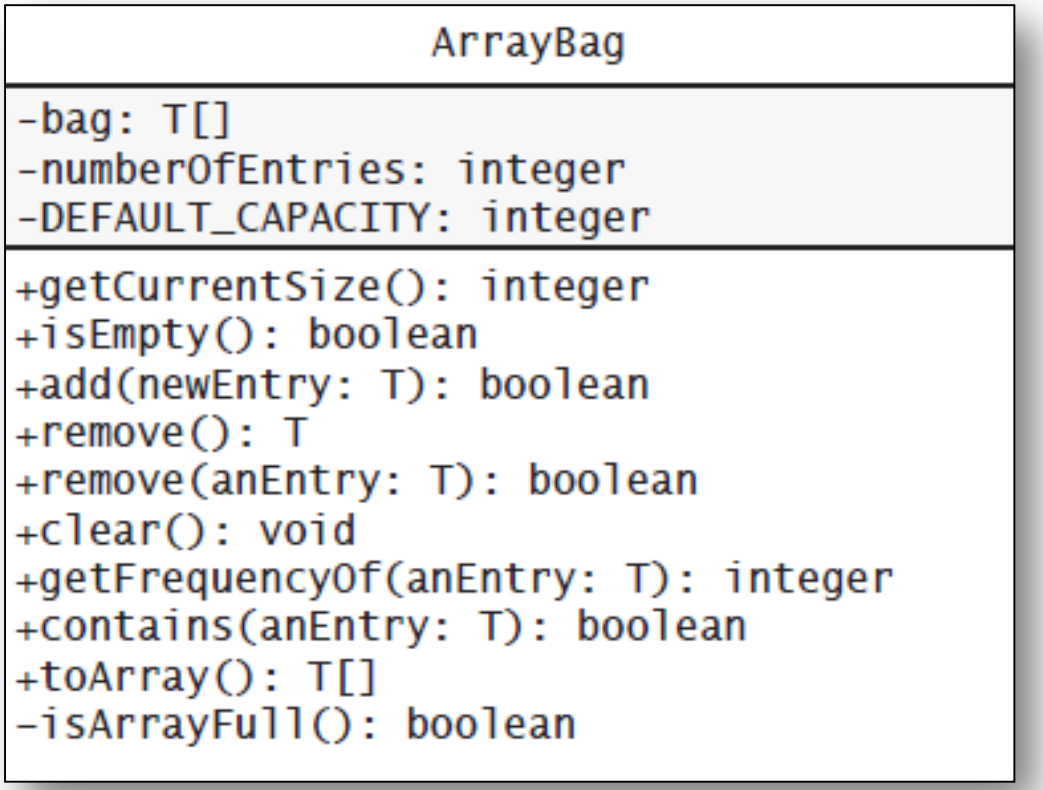

Figure 2-2: UML notation for the class **`ArrayBag`**, including the class's data fields

Fixed-Size Array

final keyword has a numerous way to use:

- A final class cannot be subclassed.
- A final method cannot be overridden by subclasses
- A final variable can only be initialized once

```
/** A class of bags whose entries are stored in a fixed-size array.
    INITIAL, INCOMPLETE DEFINITION; no security checks */

public final class ArrayBag<T> implements BagInterface<T>
{
    private final T[] bag;
    private int numberOfEntries;
    private static final int DEFAULT_CAPACITY = 25;

    /** Creates an empty bag whose initial capacity is 25. */
    public ArrayBag()
    {
       this(DEFAULT_CAPACITY);
    } // end default constructor

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

Unchecked cast means that you are (implicitly or explicitly) casting from a generic type to a nonqualified type or the other way around.

Listing 2-1: An outline of the class ArrayBag

Fixed-Size Array

```
    /** Adds a new entry to this bag.
        @param newEntry  The object to be added as a new entry.
        @return  True if the addition is successful, or false if not. */
    public boolean add(T newEntry)
    {
      // To be defined
    } // end add

    /** Retrieves all entries that are in this bag.
        @return  A newly allocated array of all the entries in this bag. */
    public T[] toArray()
    {
      // To be defined
    } // end toArray

    // Returns true if the array bag is full, or false if not.
    private boolean isArrayFull()
    {
      // To be defined
    } // end isArrayFull

    ...

} // end ArrayBag
```

Listing 2-1: An outline of the class ArrayBag

Fixed-Size Array

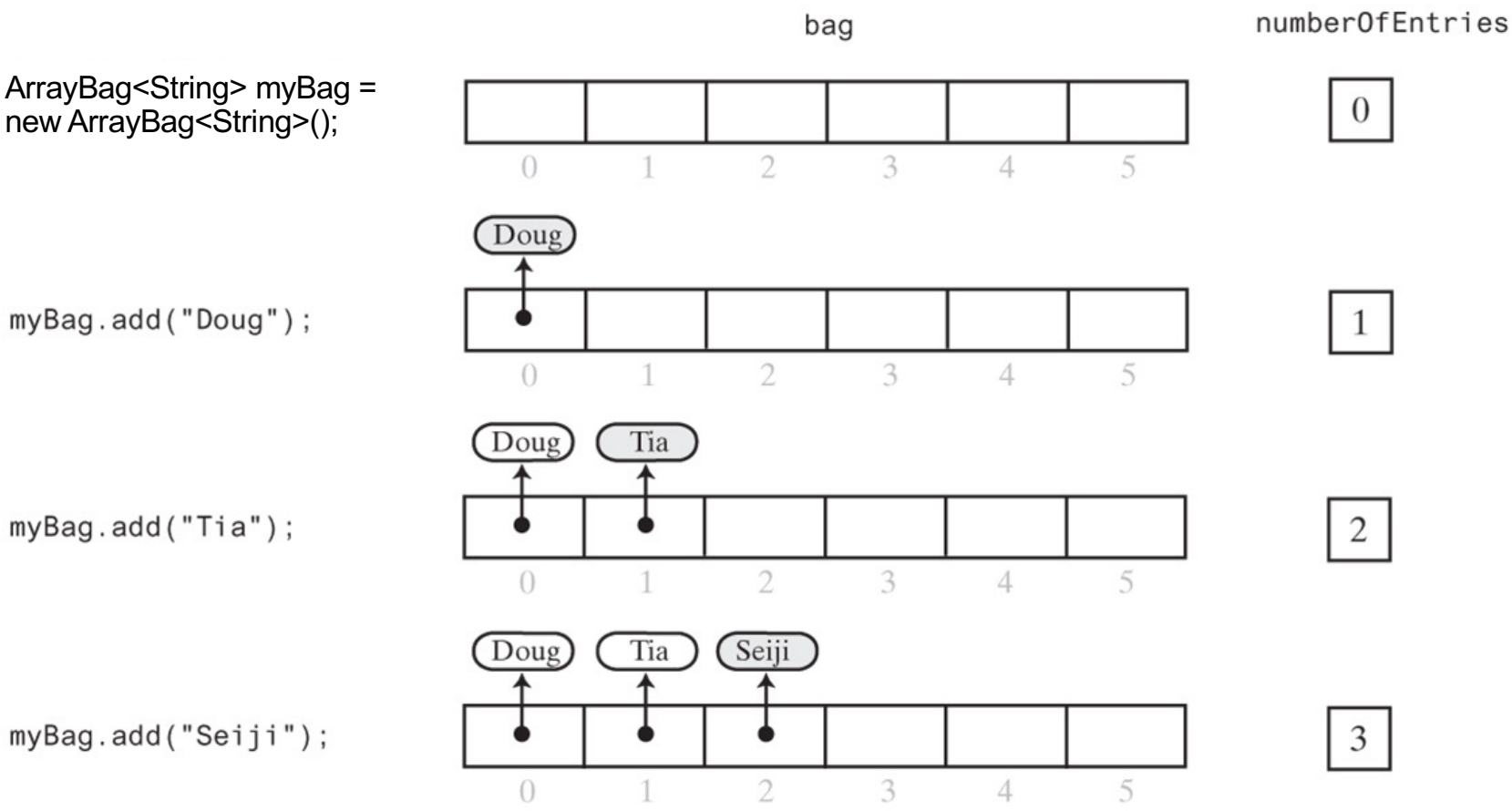

Figure 2-3: Adding entries to an array that represents a bag, whose capacity is six, until it becomes full

Fixed-Size Array

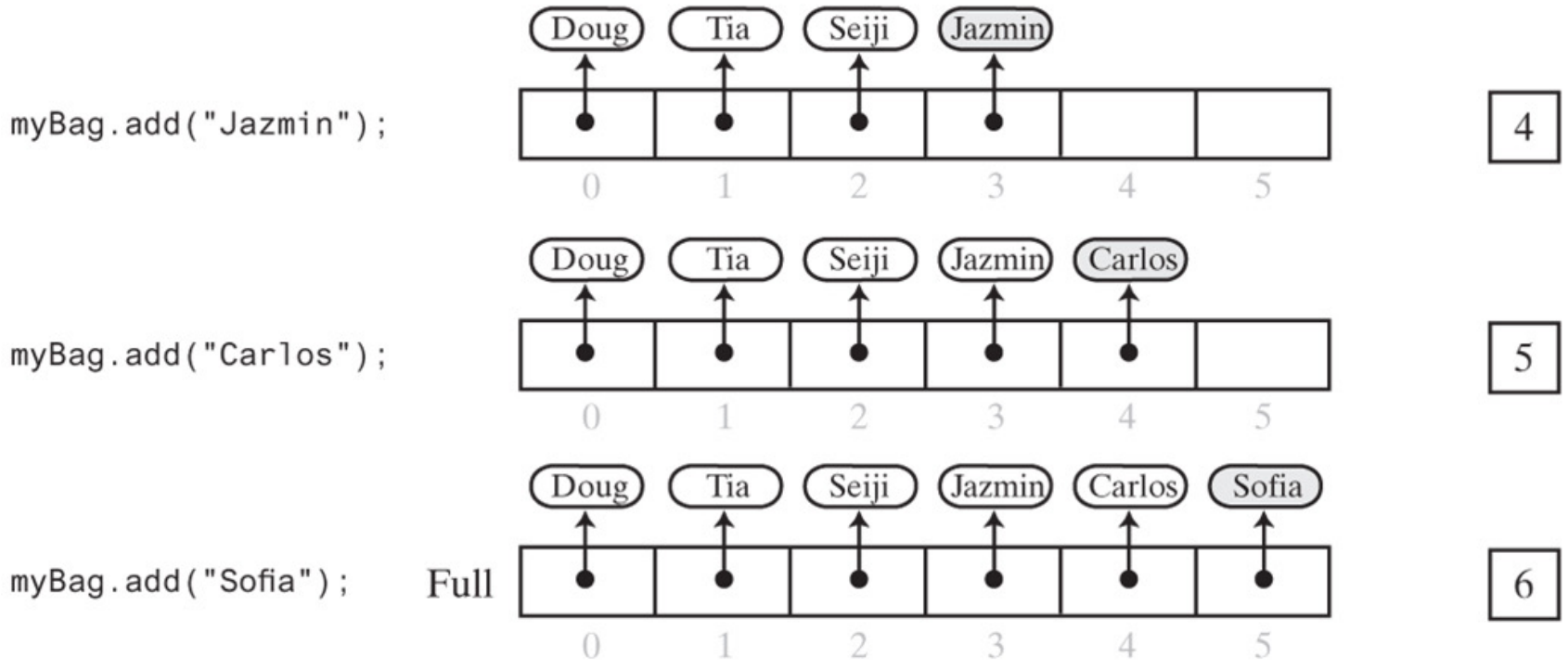

Figure 2-3: Adding entries to an array that represents a bag, whose capacity is six, until it becomes full

Exercise

- Please implement `add()` method for `ArrayBag` class.

```
/** Adds a new entry to this bag.
    @param newEntry  the object to be added as a new entry.
    @return  True if the addition is successful, or false if not.*/
public boolean add(T newEntry)
{


   // ADD YOUR CODE HERE



} // end add
```

Answer

```
/** Adds a new entry to this bag.
    @param newEntry  the object to be added as a new entry.
    @return  True if the addition is successful, or false if not.*/
public boolean add(T newEntry)
{
   boolean result = true;

   if (isArrayFull())
   {
      result = false;
   }
   else
   {  // Assertion: result is true here
      bag[numberOfEntries] = newEntry;
      numberOfEntries++;
   } // end if

   return result;

} // end add
```

Method `add()`

Answer (cont.)

```
// Returns true if this bag is full, or false if not.
private boolean isArrayFull()
{
   return numberOfEntries >= bag.length;
} // end isArrayFull
```

Method `isArrayFull()`

Fixed-Size Array

```
/** Retrieves all entries that are in this bag.
    @return  A newly allocated array of all
                    the entries in the bag. */
public T[] toArray() {

   // The cast is safe because the new array contains null entries.
   @SuppressWarnings("unchecked")
   T[] result = (T[])new Object[numberOfEntries]; // Unchecked cast

   for (int index = 0; index < numberOfEntries; index++){
      result[index] = bag[index];
   } // end for

   return result;

} // end toArray
```

Method **toArray**

Question

- Should the method `toArray` return the array `bag` instead of a copy?

Answer

- No.
- Assume

```
public String[] toArray(){
    return bag;
}
...

String[] bagArray = myBag.toArray();
```

`bagArray` is an alias for the private instance variable `bag` within the object `myBag`.

- A client could change the contents of the bag without calling the class's public methods.

Making the Implementation Secure

Making the Implementation Secure

- Practice fail-safe programming by including checks for anticipated errors
- Validate input data and arguments to a method
- Refine incomplete implementation of `ArrayBag` to make code more secure by adding the following two data fields

```
private boolean integrityOK = false;

private static final int MAX_CAPACITY = 10000;
```

Making the Implementation Secure

```
/** 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
```

Revised constructor

Making the Implementation Secure

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

Method to check initialization