VILLANOVA UNIVERSITY
IGNITE CHANGE. GO NOVA.

CSC 1052 – Algorithms & Data Structures II: Collections

Professor Henry Carter
Spring 2017

Recap

- Queues allow for FIFO behavior in an ADT
- Applications may require atypical functionality or variations on the standard ADT
  - Exception-less queues
  - Glass queues
  - Deques
  - Doubly-linked lists
- Java provides both queue and deque interfaces and implementations
- Using data structures to simulate real-world applications is a valuable application

New ADT: Collections

- Data processing is central to computing
- Storing data in an unordered manner is all that is necessary
  for many applications
- The collection is a generic container for data elements that allows
  access based on *content*

Abstract level: Interface

- We need to add, remove, and check our container for data elements
- Critical methods:
  - add()
  - get()
  - contains()
  - remove()
- Helpers:
  - isFull()
  - isEmpty()
  - size()

Interface

```
package ch05.collections;
public interface CollectionInterface<T>
{
  boolean add(T element);
  // Attempts to add element to this collection.  Returns true if
successful,  false otherwise.

  T get(T target);
  // Returns an element e from this collection such that e.equals(target).
  // If no such element exists, returns null.

  boolean contains(T target);
  // Returns true if this collection contains an element e such that
  // e.equals(target);  otherwise returns false.

  boolean remove (T target);
  // Removes an element e from this collection such that e.equals(target)
  // and returns true. If no such element exists, returns false.

  boolean isFull();
  boolean isEmpty();
  int size();
}
```

Some Ground Rules

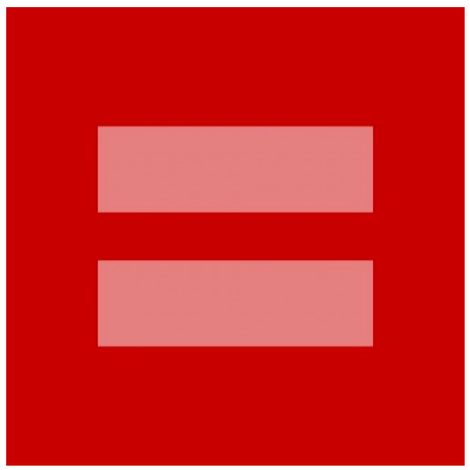
- Duplicate elements ARE allowed
- Null elements are NOT allowed
- Add and remove return boolean indicators of success
  - They do not throw exceptions

Comparison

- Collections vs Arrays
  - Accessed by content rather than index
- Collections vs Stacks and queues
  - The order of arrival does not affect the order that objects are removed
- If items in the collection have no order and no index…
  - What are we assuming about the content?

Equality

- Collection items must allow equality checks
- Recall: what do we use to determine equality?
- Each Class must define this depending on the object contents and application



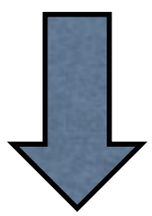
Implementation: Unsorted Array

- Our basic collection uses a bounded-length array
  - Stores elements in any ordering
- No indicator of top/front/rear, etc.
  - How do we track elements?
- How do we avoid searching the entire array?

Adding Elements

numElements: 4

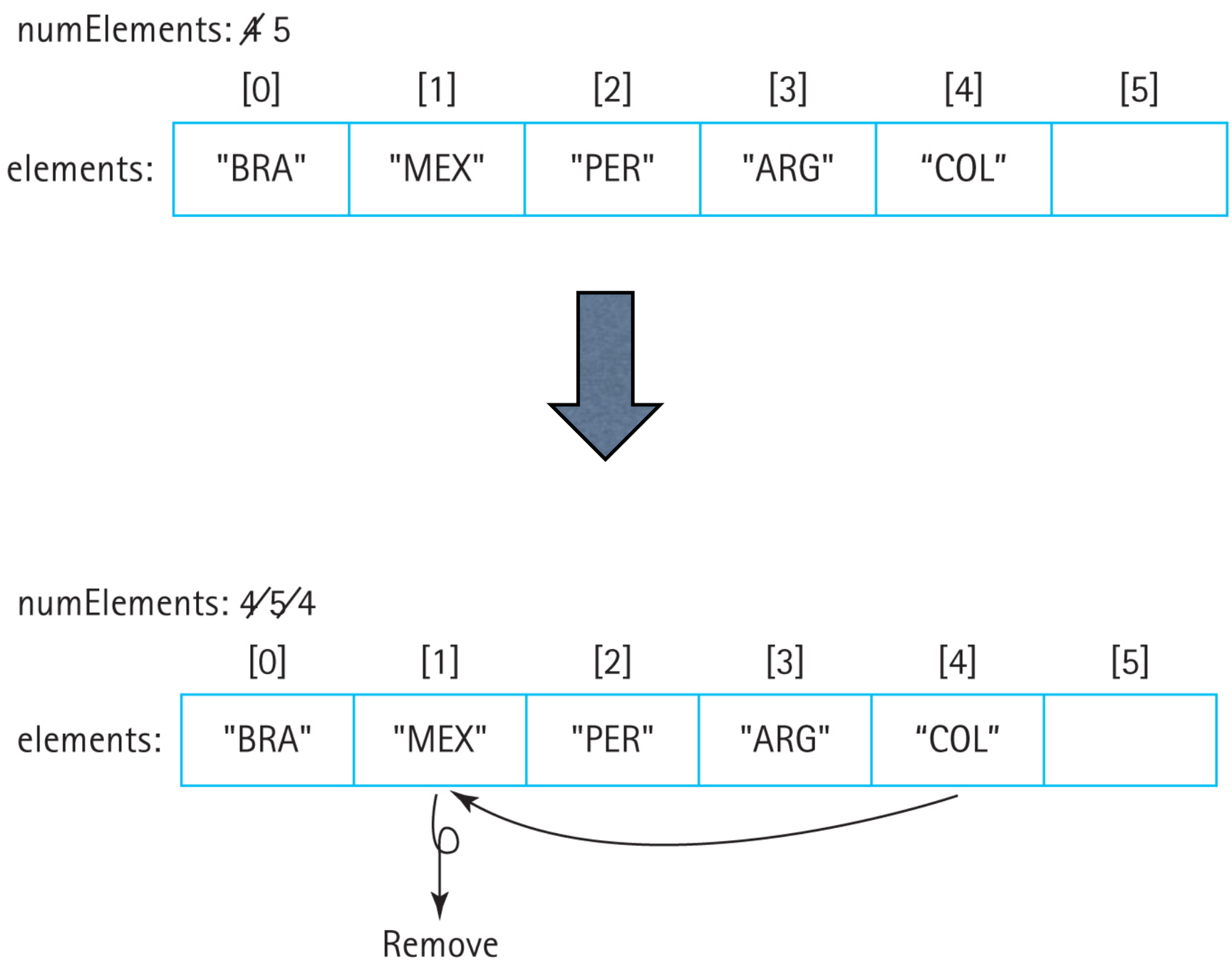
	[0]	[1]	[2]	[3]	[4]	[5]
elements:	"BRA"	"MEX"	"PER"	"ARG"		



numElements: ~~4~~ 5

	[0]	[1]	[2]	[3]	[4]	[5]
elements:	"BRA"	"MEX"	"PER"	"ARG"	"COL"	

Filling Space Without Shifting



Recall Critical Methods

- Add()
- Get()
- Contains()
- Remove()
- What do all of the final three require?

Helper Method: Find

- Implement code to find a matching element once
- Store the results in a protected class variable
- Use this method as a subroutine in get(), contains(), and remove()

Code pt I: Setup

```
package ch05.collections;
public class ArrayCollection<T> implements CollectionInterface<T>
{
  protected final int DEFCAP = 100; // default capacity
  protected T[] elements;           // array to hold collection's elements
  protected int numElements = 0;    // number of elements

  // set by find method
  protected boolean found;  // true if target found, otherwise false
  protected int location;   // indicates location of target if found

  public ArrayCollection()
  {
    elements = (T[]) new Object[DEFCAP];
  }

  public ArrayCollection(int capacity)
  {
    elements = (T[]) new Object[capacity];
  }
. . .
```

Code pt II: Find

```
protected void find(T target)
{
    location = 0;
    found = false;
    while (location < numElements)
    {
        if (elements[location].equals(target))
        {
            found = true;
            return;
        }
        else
            location++;
    }
}
```

Code pt III: Critical Methods

```
public boolean add(T element)
// Attempts to add element to this collection.
// Returns true if successful, false otherwise.
{
    if (isFull())
        return false;
    else
    {
        elements[numElements] = element;
        numElements++;
        return true;
    }
}
```

Code pt III: Critical Methods

```
public boolean remove (T target)
// Removes an element e from this collection
// such that e.equals(target) and returns true;
// if no such element exists, returns false.
{
   find(target);
   if (found)
   {
      elements[location] = elements[numElements - 1];
      elements[numElements - 1] = null;
      numElements--;
   }
   return found;
}
```

Code pt III: Critical Methods

```
public boolean contains (T target)
{
    find(target);
    return found;
}
```

```
public T get(T target)
{
    find(target);
    if (found)
        return elements[location];
    else
        return null;
}
```

Application: Vocabulary Density

- The vocabulary density of a text is the total number of words divided by the number of unique words
  - A measure of how many different words the writer uses
- Requires counting the words in a text and remembering unique words from the text
- How can we use a collection?

$$\rho = \frac{m}{V}$$

Algorithm

- Collections allow duplicates but have methods to help avoid them
- Maintain a collection of unique words
- Count total words as they are read
- Divide the total counter by the size of the collection

Run it!

- Download the program
- Configure the main() arguments
- Test the method on some provided text

Notable tidbits

- Simple approach allowed by the power of the collection
  - Contains() in particular
- Regular expressions used to define complex delimiters

Regex in the program

- `wordsIn.useDelimiter("[^a-zA-Z']+");`

Notable tidbits

- Simple approach allowed by the power of the collection
  - Contains() in particular
- Regular expressions used to define complex delimiters
- Bounded capacity is limiting and may lead to inaccurate measurements
  - Note the two possible print cases at the end

Analyzed Text Table

Magna Carta	Declaration of Independence	Purna Swaraj
3310	1341	594
698	540	288
4.74	2.48	2.06

Is it really useful?

- Searching for objects that you already hold
- Simple, unsorted and bounded implementation very inefficient
- Two things make collections very useful:
  - Partial key information
  - Extending to more sophisticated ADTs

Recap

- Collections store and access data items by their content
  - No indexing, no ordering
- Array-based implementations are simple but inefficient
  - The find() helper makes implementing the rest of the methods easy
- Vocabulary density measures the writer's use of varying words
- Vocabulary density can be measured with the help of a collection
  - Although not required, data elements can be limited to unique items if the application calls for it

Next Time...

- Dale, Joyce, Weems Chapter 5.4-5.5
  - Remember, you need to read it BEFORE you come to class!
- Check the course webpage for practice problems
- Peer Tutors
  - http://www.csc.villanova.edu/help/