BASIC DATA STRUCTURES

CS062

DATA STRUCTURES AND ADVANCED PROGRAMMING

7: ArrayLists

Alexandra Papoutsaki
she/her/hers

Lecture 7: ArrayLists

- ArrayList
- Java Collections

Some slides adopted from Algorithms 4th Edition and Oracle tutorials

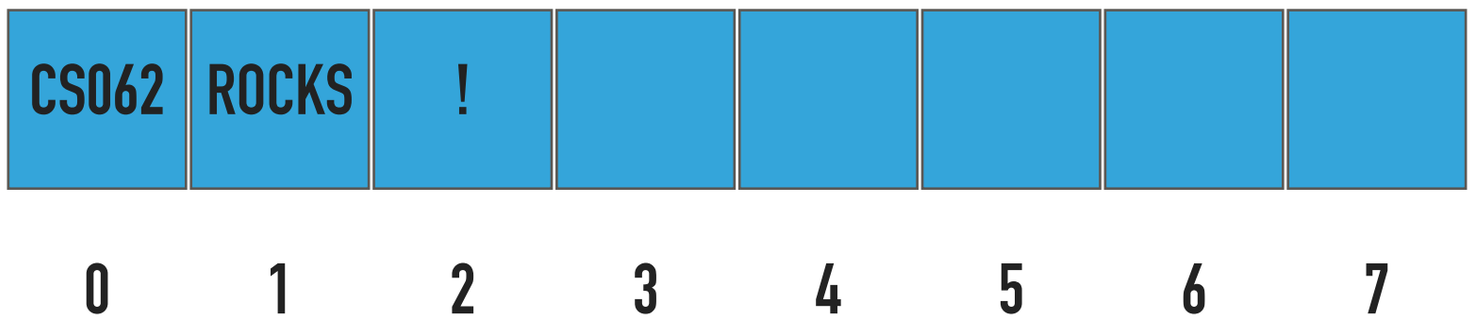
Limitations of arrays

- Fixed-size.
- Do not work well with generics.
  - `E[] myArray = (E[]) new Object[capacity];`
- Limited functionality (Java requires the use of `Arrays` class for printing contents and manipulating arrays, such as sorting and searching).

- We want resizable arrays that support any type of object.

ArrayList (or dynamic/growable/resizable/mutable array)

- Dynamic linear data structure that is zero-indexed.
  - We will use the `List` interface.
- Sequential data structure that requires consecutive memory cells.
- Implemented with an underlying array of a specific capacity.
  - But the user does not see that!



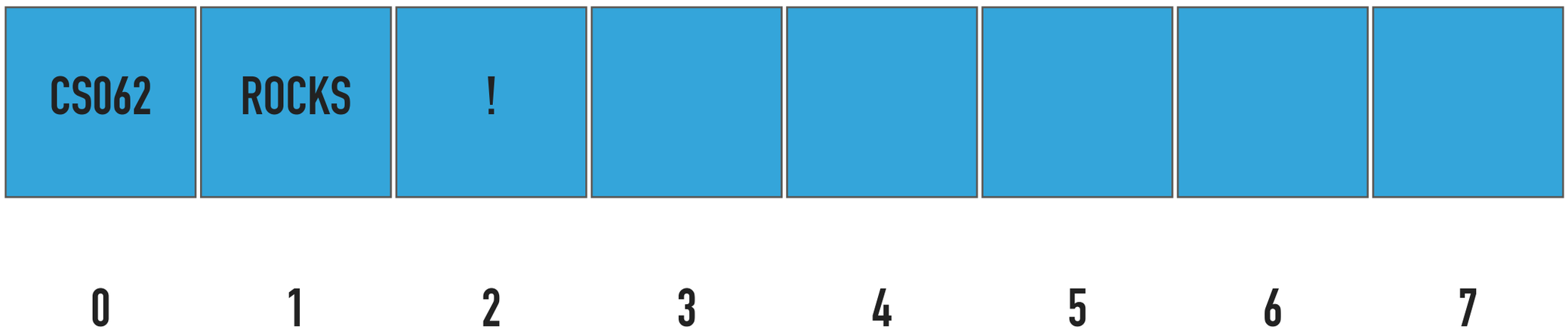
Reminder: Interface List

```
public interface List <E> {
    void add(E element);
    void add(int index, E element);
    void clear();
    E get(int index);
    boolean isEmpty();
    E remove();
    E remove(int index);
    E set(int index, E element);
    int size();
}
```

Standard Operations of `ArrayList<E>` class

- `ArrayList()`: Constructs an empty ArrayList with an initial capacity of 2 (can vary across implementations, another common initial capacity is 10).
- `ArrayList(int capacity)`: Constructs an empty ArrayList with the specified initial capacity.
- `isEmpty()`: Returns true if the ArrayList contains no elements.
- `size()`: Returns the number of elements in the ArrayList.
- `get(int index)`: Returns the element at the specified index.
- `add(E element)`: Appends the element to the end of the ArrayList.
- `add(int index, E element)`: Inserts the element at the specified index and shifts the element currently at that position (if any) and any subsequent elements to the right (adds one to their indices).
- `E remove()`: Removes and returns the element at the end of the ArrayList.
- `E remove(int index)`: Removes and returns the element at the specified index. Shifts any subsequent elements to the left (subtracts one from their indices).
- `E set(int index, E element)`: Replaces the element at the specified index with the specified element and returns the olde element.
- `clear()`: Removes all elements.

ArrayLists



Capacity = 8

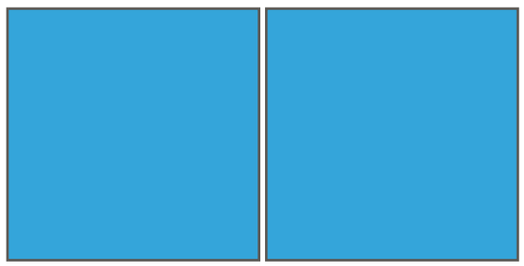
Size = 3

`ArrayList()`: Constructs an ArrayList

What should happen?

```
ArrayList<String> al = new ArrayList<String>();
```

`ArrayList()`: Constructs an ArrayList



0 1

```
ArrayList<String> al = new ArrayList<String>();
```

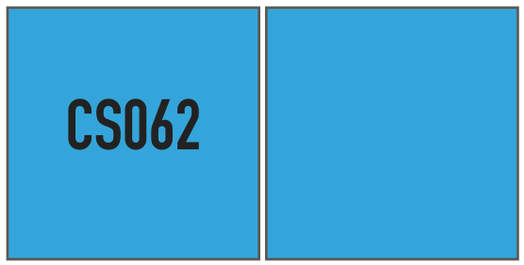
Capacity = 2

Size = 0

What should happen?

```
al.add("CS062");
```

`add(E element)`: Appends the element to the end of the ArrayList




```
al.add("CS062");
```

Capacity = 2

Size = 1

What should happen?

```
al.add("ROCKS");
```

`add(E element)`: Appends the element to the end of the ArrayList


```
al.add("ROCKS");
```

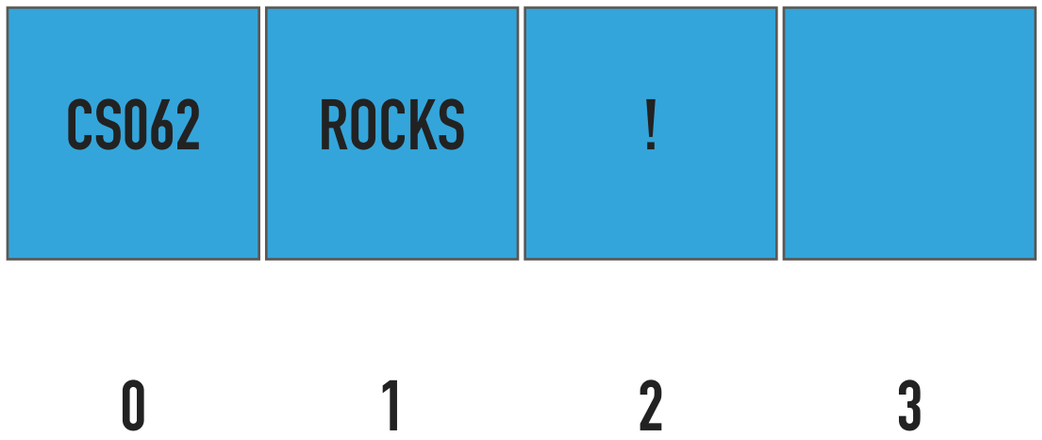
Capacity = 2

Size = 2

What should happen?

```
al.add("!");
```

`add(E element)`: Appends the element to the end of the ArrayList



Capacity = 4

Size = 3

– DOUBLE CAPACITY SINCE IT'S FULL AND THEN ADD NEW ELEMENT

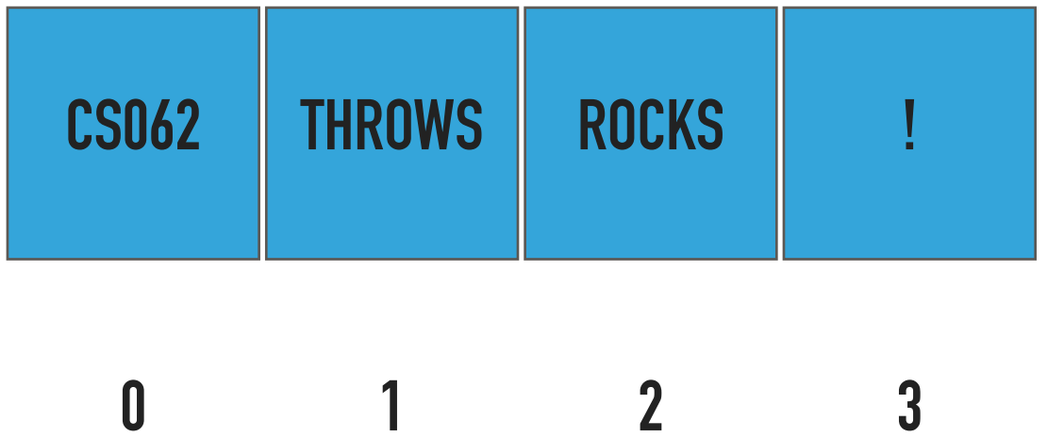
```
al.add("!");
```

What should happen?

```
al.add(1, "THROWS");
```

`add(int index, E element)`: Adds element at the specified index

– SHIFT ELEMENTS TO THE RIGHT




```
al.add(1, “THROWS”);
```

Capacity = 4

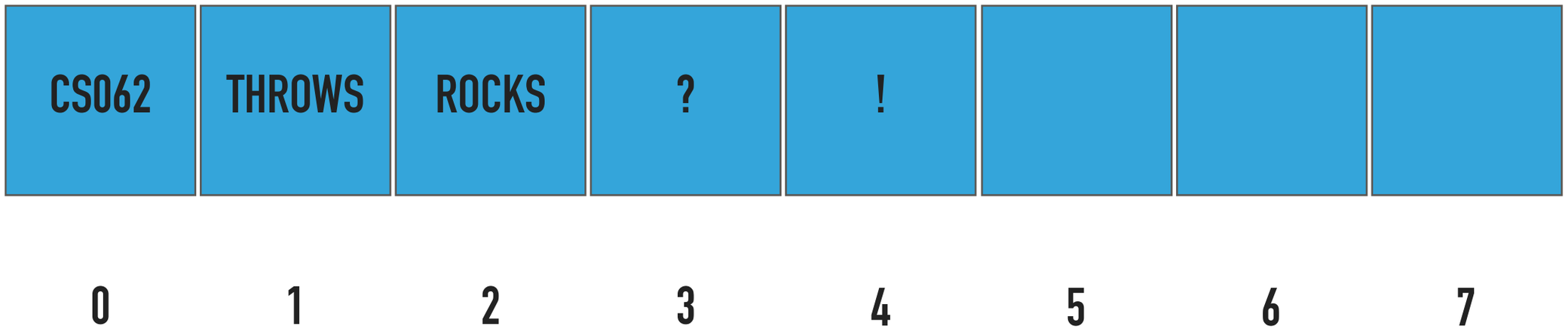
Size = 4

What should happen?

```
al.add(3, “?”);
```

`add(int index, E element)`: Adds element at the specified index

- DOUBLE CAPACITY SINCE IT'S FULL
- SHIFT ELEMENTS TO THE RIGHT



Capacity = 8

Size = 5

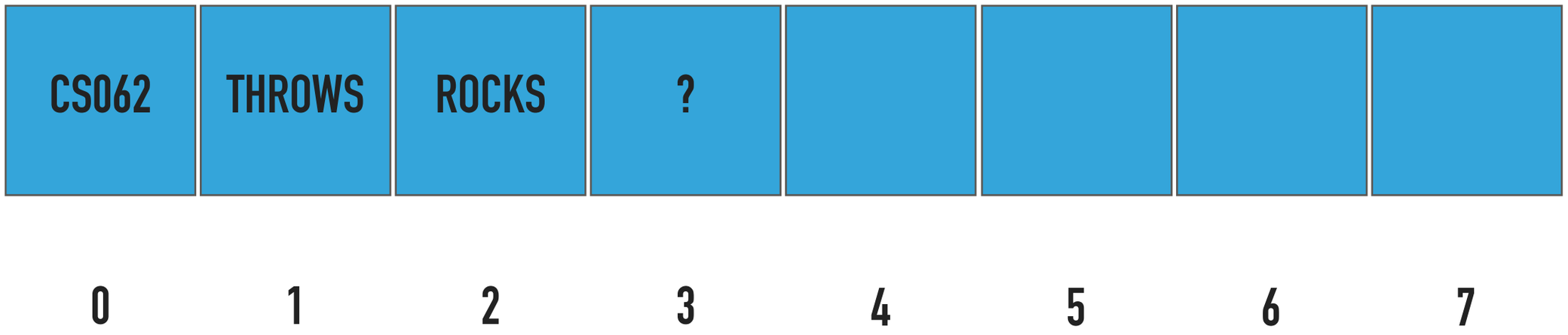
`al.add(3, "?");`

What should happen?

`al.remove();`

`remove()`: Removes and returns element from the end of ArrayList

– REMOVE AND RETURN LAST ELEMENT



Capacity = 8

Size = 4

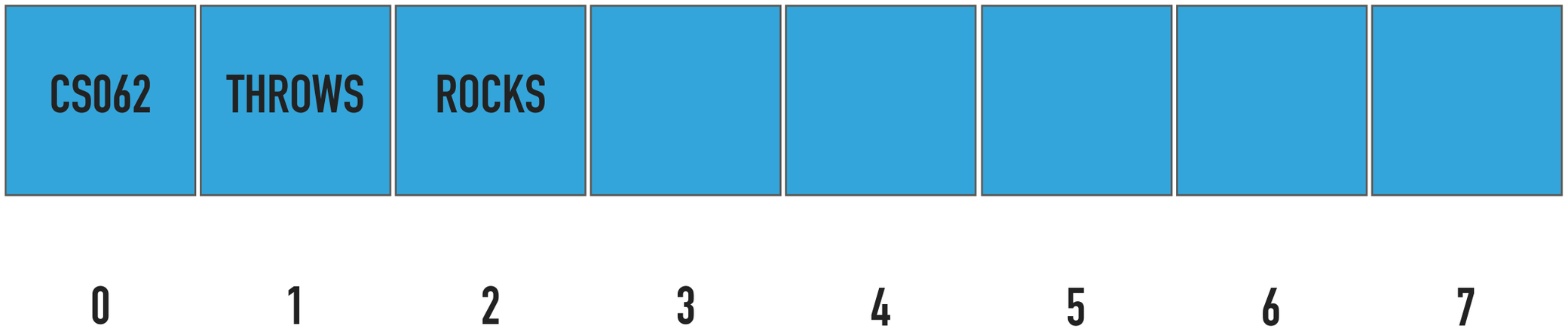
```
al.remove();
```

What should happen?

```
al.remove();
```

`remove()`: Removes and returns element from the end of ArrayList

– REMOVE AND RETURN LAST ELEMENT



Capacity = 8

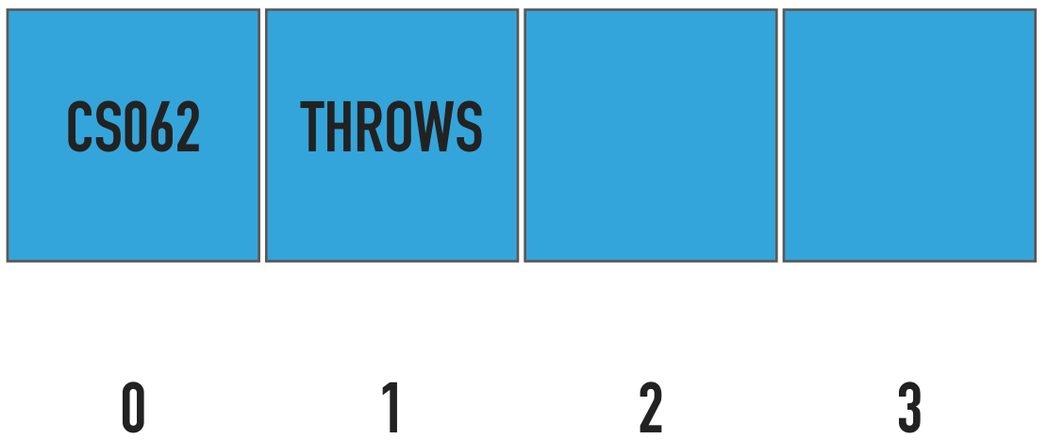
Size = 3

```
al.remove();
```

What should happen?

```
al.remove();
```

`remove()`: Removes and returns element from the end of ArrayList




```
al.remove();
```

Capacity = 4

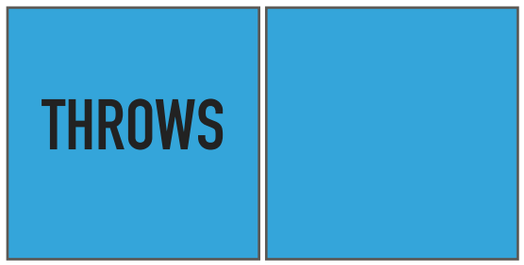
Size = 2

- REMOVE AND RETURN ELEMENT FROM THE END
- HALVE CAPACITY WHEN 1/4 FULL

What should happen?

```
al.remove(0);
```

`remove(int index)`: Removes and returns element from specified index



0 1

Capacity = 2

Size = 1

```
al.remove(0);
```

- REMOVE ELEMENT FROM INDEX
- SHIFT ELEMENTS TO THE LEFT
- HALVE CAPACITY WHEN 1/4 FULL

Our own implementation of ArrayLists

- We will follow the recommended textbook style.
  - It does not offer a class for this so we will build our own. We got to test Java's built-in implementation in lab!
- We will work with generics because we want arrayLists to hold objects of an type.
- We will implement the `List` interface we defined in the last lecture.
- We will use an array and we will keep track of how many elements we have in our ArrayList.

Instance variables and constructors

```
public class ArrayList<E> implements List<E> {
    private E[] data; // underlying array of elements
    private int size; // number of elements in ArrayList

    /**
     * Constructs an ArrayList with an initial capacity of 2.
     */
    @SuppressWarnings("unchecked")
    public ArrayList() {
        data = (E[]) new Object[2];
        size = 0;
    }

    /**
     * Constructs an ArrayList with the specified capacity.
     */
    @SuppressWarnings("unchecked")
    public ArrayList(int capacity) {
        data = (E[]) new Object[capacity];
        size = 0;
    }
```

PRACTICE TIME: Check if is empty and how many elements

```java
/**
 * Returns true if the ArrayList contains no elements.
 *
 * @return true if the ArrayList does not contain any element
 */
public boolean isEmpty() {

}
```

```java
/**
 * Returns the number of elements in the ArrayList.
 *
 * @return the number of elements in the ArrayList
 */
public int size() {

}
```

Check if is empty and how many elements

```java
/**
 * Returns true if the ArrayList contains no elements.
 *
 * @return true if the ArrayList does not contain any element
 */
public boolean isEmpty() {
    return size == 0;
}
```

```java
/**
 * Returns the number of elements in the ArrayList.
 *
 * @return the number of elements in the ArrayList
 */
public int size() {
    return size;
}
```

PRACTICE TIME: Resize underlying array's capacity

```
/**
 * Resizes the ArrayList's capacity to the specified capacity.
 */
@SuppressWarnings("unchecked")
private void resize(int capacity) {
    //reserve a new temporary array with the provided capacity

    //copy all the elements from the old array (data) into the temporary array

    //point data to the new temporary array


}
```

Resize underlying array's capacity

```
/**
 * Resizes the ArrayList's capacity to the specified capacity.
 */
@SuppressWarnings("unchecked")
private void resize(int capacity) {
    //reserve a new temporary array of Es with the provided capacity
    E[] temp = (E[]) new Object[capacity];
    //copy all the elements from the old array (data) into the temporary array
    for (int i = 0; i < size; i++){
        temp[i] = data[i];
    }
    //point data to the new temporary array
    data = temp;
}
```

PRACTICE TIME: Append an element to the end of ArrayList

```
/**
 * Appends the element to the end of the ArrayList. Doubles its capacity if necessary.
 *
 * @param element the element to be inserted
 */
public void add(E element) {
     //check whether ArrayList is full
     //if yes, double in size

     //add the element at the end of the ArrayList and increase the counter by 1

}
```

Append an element to the end of ArrayList

```
/**
 * Appends the element to the end of the ArrayList. Doubles its capacity if necessary.
 *
 * @param element the element to be inserted
 */
public void add(E element) {
     //check whether ArrayList is full
    if (size == data.length){
     //if yes, double in size
        resize(2 * data.length);
    }
     //add the element at the end of the ArrayList and increase the counter by 1
    data[size] = element;
    size++;
}
```

PRACTICE TIME: Add an element at a specified index

```
/**
 * Inserts the element at the specified index. Shifts existing elements
 * to the right and doubles its capacity if necessary.
 *
 * @param index
 *            the index to insert the element
 * @param element
 *            the element to be inserted
 * @pre 0<=index<=size
 */
public void add(int index, E element) {
    //check whether index in range


    //if full double size


    //shift elements to the right



    //set element to position index

    //increase number of elements


}
```

Add an element at a specified index

```
/**
 * Inserts the element at the specified index. Shifts existing elements
 * to the right and doubles its capacity if necessary.
 *
 * @param index
 *            the index to insert the element
 * @param element
 *            the element to be inserted
 * @pre 0<=index<=size
 */
public void add(int index, E element) {
    //check whether index in range
    if (index > size || index < 0){
          throw new IndexOutOfBoundsException("Index " + index + " out of bounds");
     }

    //if full double size
    if (size == data.length){
          resize(2 * data.length);
    }

    //shift elements to the right
    for (int i = size; i > index; i--){
          data[i] = data[i - 1];
    }
    size++;

    //set element to position index
    data[index] = element;
}
```

PRACTICE TIME: Replace an element at a specified index

```
/**
* Replaces the element at the specified index with the specified element.
* @param index
*            the index of the element to replace
* @param element
*            element to be stored at specified index
* @return the old element that was changed.
* @pre 0<=index<size
*/
public E set(int index, E element) {
    //check whether index in range

    //retrieve old element at index

    //update index with new element

    //return old element

}
```

Replace an element at a specified index

```
/**
* Replaces the element at the specified index with the specified element.
* @param index
*            the index of the element to replace
* @param element
*            element to be stored at specified index
* @return the old element that was changed.
* @pre 0<=index<size
 */
public E set(int index, E element) {
    //check whether index in range
     if (index >= size || index < 0){
        throw new IndexOutOfBoundsException("Index " + index + " out of bounds");
      }
    //retrieve old element at index
    E old = data[index];
    //update index with new element
    data[index] = element;
    //return old element
    return old;
}
```

PRACTICE TIME: Retrieve and remove element from the end of ArrayList

```
 /**
 * Removes and returns the element from the end of the ArrayList.
 * @return the removed element
 * @throws NoSuchElementException if ArrayList is empty
 * @pre size>0
 */
public E remove() {
    //if ArrayList is empty throw NoSuchElementException

    //retrieve last element after you reduce number of elements by 1

    //set the position where the removed element is to null


    //shrink in half to save space if number of elements in ArrayList is 1/4 of its capacity


    //return the removed element
}
```

Retrieve and remove element from the end of ArrayList

```
 /**
 * Removes and returns the element from the end of the ArrayList.
 * @return the removed element
 * @throws NoSuchElementException if ArrayList is empty
 * @pre size>0
 */
public E remove() {
    //if ArrayList is empty throw NoSuchElementException
    if (isEmpty()){
        throw new NoSuchElementException("The list is empty”);
    }
    //retrieve last element after you reduce number of elements by 1
    size--;
    E element = data[size];
    //set the position where the removed element is to null
    data[size] = null; // Avoid loitering (see text).

    //shrink in half to save space if number of elements in ArrayList is 1/4 of its capacity
    if (size > 0 && size == data.length / 4)
        resize(data.length / 2);
    //return the removed element
    return element;
}
```

PRACTICE TIME: Retrieve and remove element from a specific index

```
/**
 * Removes and returns the element at the specified index.
 *
 * @param index
 *            the index of the element to be removed
 * @return the removed element
 * @pre 0<=index<size
 */
public E remove(int index) {
    //check whether index in range

    //retrieve element at index

    //reduce number of elements by 1

    //shift all elements from index till the end one position to the left


    //set the last element (since they have been shifted to the left), to null

    //shrink in half to save space if number of elements in ArrayList is 1/4 of its capacity

    //return removed element
}
```

Retrieve and remove element from a specific index

```
/**
 * Removes and returns the element at the specified index.
 *
 * @param index
 *            the index of the element to be removed
 * @return the removed element
 * @pre 0<=index<size
 */
public E remove(int index) {
    //check whether index in range
    if (index >= size || index < 0){
          throw new IndexOutOfBoundsException("Index " + index + " out of bounds");
    }
    //retrieve element at index
     E element = data[index];
    //reduce number of elements by 1
     size--;
     //shift all elements from index till the end one position to the left
     for (int i = index; i < size; i++){
          data[i] = data[i + 1];
     }
    //set the last element (since they have been shifted to the left), to null
     data[size] = null; // Avoid loitering (see text).

    //shrink in half to save space if number of elements in ArrayList is 1/4 of its capacity
     if (size > 0 && size == data.length / 4){
          resize(data.length / 2);
     }
    //return removed element
     return element;
}
```

PRACTICE TIME: Clear all elements

```
/**
 * Clears the ArrayList of all elements.
 */
public void clear() {

    // Go through all elements of the array and set them to null

    // Set number of elements to 0

}
```

Clear all elements

```
/**
 * Clears the ArrayList of all elements.
 */
public void clear() {

    // Go through all elements of the array and set them to null
    for (int i = 0; i < size; i++){
        data[i] = null;
    }
    // Set number of elements to 0
    size = 0;
}
```

Lecture 7: ArrayLists

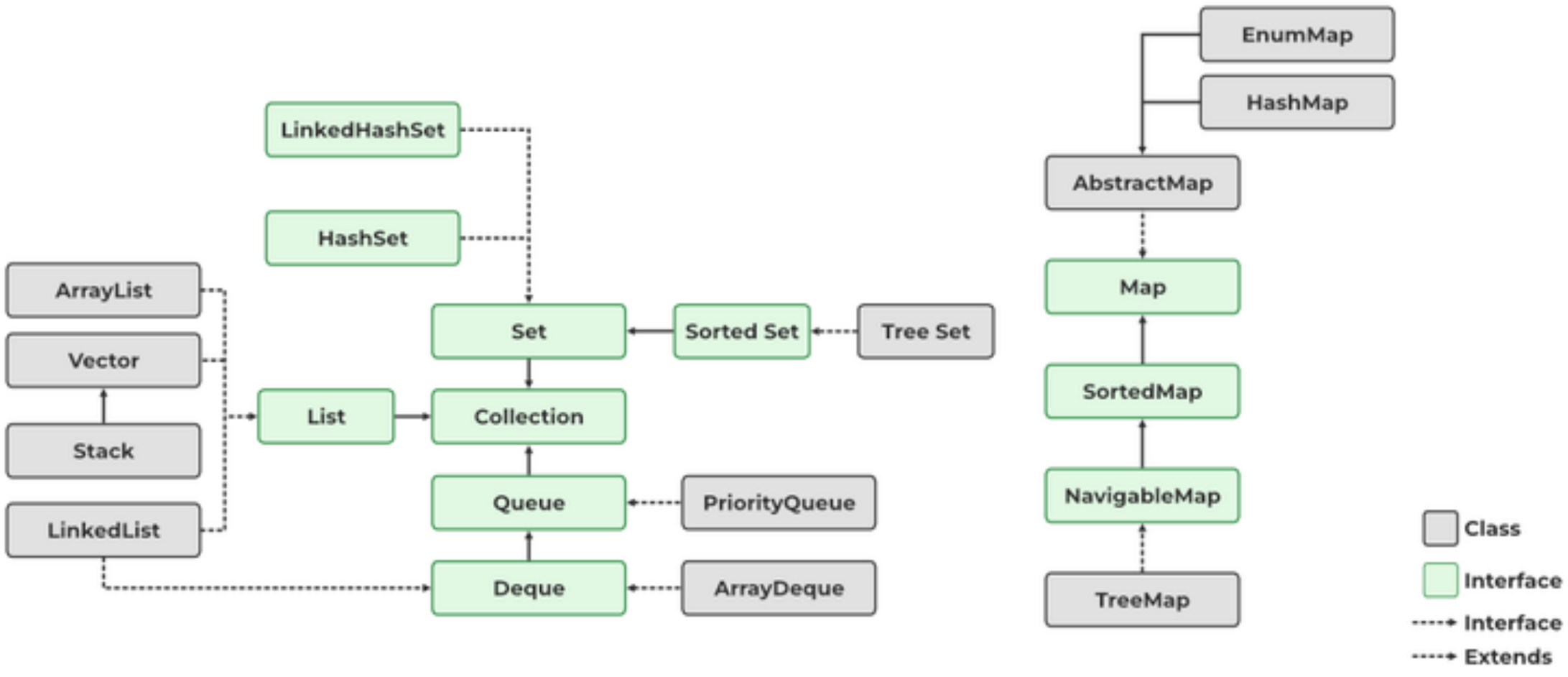
- ArrayList
- Java Collections

The Java Collections Framework

- Collection: an object that groups multiple elements into a single unit, allowing us to store, retrieve, manipulate data.
- Collections Framework:
  - Interfaces: ADTs (abstract data types) that represent collections.
  - Implementations: The actual data structures.
  - Algorithms: methods that perform useful operations, such as searching and sorting.

https://docs.oracle.com/javase/tutorial/collections/intro/index.html

The Java Collections Framework



https://www.geeksforgeeks.org/collections-in-java-2/

List Interface

- A collection storing elements in an ordered fashion.
- Elements are accessed in a zero-based fashion.
- Typically allow duplicate elements and null values but always check the specifications of implementation.

https://en.wikipedia.org/wiki/Java_collections_framework

`ArrayList` in Java Collections

- Resizable list that increases by 50% when full and does NOT shrink.
- Not thread-safe (more in CS105).

```
java.util.ArrayList;

public class ArrayList<E> extends AbstractList<E>
implements List<E>
```

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

Vector in Java Collections

- Java has one more class for resizable arrays.
- Doubles when full.
- Is synchronized (more in CS105).

```
java.util.Vector;
```

```
public class Vector<E> extends AbstractList<E>
implements List<E>
```

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

Lecture 7: ArrayLists

- ArrayList
- Java Collections

Readings:

- Oracle's guides:
  - Collections: https://docs.oracle.com/javase/tutorial/collections/intro/index.html
  - ArrayLists: https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html
- Recommended Textbook:
  - Chapter 1.3 (Page 136–137)
- Recommended Textbook Website:
  - Resizable arrays: https://algs4.cs.princeton.edu/13stacks/

Code

- Lecture 7 code