HW 3: An Array List From Scratch

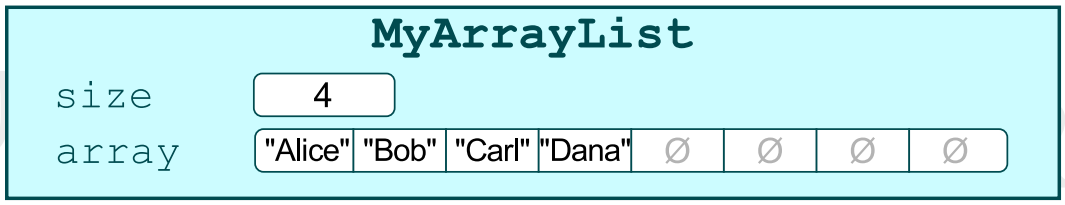
You will create an object called `MyArrayList`, that mimics the `ArrayList` object built into Java. The generic type will be called `E`.

In general, `ArrayList`s automatically resize themselves as needed so that they always have enough space. Your `MyArrayList` will do this as well. To another programmer using your `MyArrayList`, this should all happen automatically and invisibly.

There are two pieces of data that may be thought of as "size": both the length of the internal backing array, and the number of valid elements that have been added to the list. To keep these concepts clear, we will refer to the array's length as the *capacity*, and the number of elements that have been added as the *size*.

Like the `BoundedSet`, this object should have two private fields:

1. the current size of the `MyArrayList`
2. an `E` array holding its contents



In addition, you should have a constant named `DEFAULT_CAPACITY`, which will be the initial size if one is not given. This should be set to `10`.

Unlike the `BoundedSet`, element order matters! When an item is inserted or deleted, the other ones must all shift places to maintain their order. Also, elements may be `null` or duplicates of other elements already inside the list.

You must implement the following public methods:

- `MyArrayList(int initialCapacity)`, which is a constructor in which the initial capacity is specified.
- `MyArrayList()`, which is a another constructor that sets the initial capacity to be `DEFAULT_CAPACITY`.
- `void add(E element)`, which adds the element to the end of the `MyArrayList`. If there is not enough space to add the new element, you must reallocate so that its capacity is doubled.
- `void add(int index, E element)`, which adds the element to the specified position in the `MyArrayList`, shifting other items further down the list. If there is not enough space to add the new element, you must reallocate so that its capacity is doubled.

- `void clear()`, which removes every element of this `MyArrayList`, setting every element in the array to `null` and resetting the capacity to `DEFAULT_CAPACITY` and the size to `0`.
- `boolean contains(E element)`, which returns whether or not the eleement is present inside the `MyArrayList`.
- `void ensureCapacity(int capacity)`, which ensures that the `MyArrayList`'s current capacity is at least the number specified. If it is too small, it will be reallocated. If it is already big enough, nothing will be done.
- `int findIndex(E element)`, which returns the first index at which the element is found (0-indexed), or -1 if it isn't present.
- `E get(int index)`, which returns the element at the given index. This should throw an `IndexOutOfBoundsException` if the given index isn't in the proper range.
- `int getCapacity()`, which returns the current capacity of the `MyArrayList`.
- `int getSize()`, which returns the number of elements currently contained in the `MyArrayList`.
- `boolean isEmpty()`, which returns `true` if this `MyArrayList` is empty; otherwise it returns `false`.
- `void remove(int index)`, which removes the object from the specified index of the `MyArrayList`. All subsequent elements are shifted back one position. This should throw an `IndexOutOfBoundsException` if index isn't in the proper range. If removing this element causes the size to be less than or equal to one quarter of the current capacity, reallocation should occur so that capacity is halved.
- `void set(int index, E element)`, which overwrites the element at the given index. This should throw an `IndexOutOfBoundsException` if `index` isn't in the proper range.
- `String toString()`, which returns a `String` representing this `MyArrayList`. The `String` will consist of a comma-separated list of individual elements, inside a single pair of square brackets (`[]`s). Only items legitimately added to the `MyArrayList` should be included.

In addition, you are responsible for creating your own unit-testing `main()` method. This method must call every one of the above methods at least one time. It must also use at least 2 different `MyArrayLists` of different types. It must also resize a `MyArrayList` at least once on `add()` and at least once on `remove()`.

Keep an eye out for methods that do similar tasks, that can call each other. Remember, try to avoid duplicated code, because it's harder to maintain! Also, this task will be easier if you create a private `reallocateArray()` method that makes a brand new array, copies old elements into it, and replaces the old one.

Finally, remember to comment adequately, and to use Javadoc comments with every public method (except the unit-testing `main()`). Coding style is important!