Implementing Array Lists

What is an array?

- Contiguous area of memory to store data
- Fixed length
- Each item can be directly accessed by its index: 0..length-1
- Less overhead than other data structures

How to define an array?

- double[] collection = new double[SIZE];
- Students[] class = new Student[SIZE];

- String[] dictionary = {“apple”, “banana”, “cherry”, “grape”, “orange”, “plum”};
- int[] bag = {5, 8, 32, 17, 22};

Using arrays

- Access elements
  - double thirdNum = collection[2];
- Write elements
  - collection[0] = 5.0;
- Determine length
  - int lengthOfArray = collection.length;

Using arrays in a for-loop

- Writing the contents of the array:

```
for(int i = 0; i < collection.length; i++)
{
    System.out.println(collection[i]);
}
```

or

```
for(Double n : collection)
{
    System.out.println(n);
}
```

Array list implementation

```
public class MyArrayList
{
        public final int DEFAULT_SIZE = 10;
        private int size;
        private Character[] array;

        public MyArrayList()
        {
                array = new Character[DEFAULT_SIZE];
                size = 0;
        }
        …
```

getSize()

```
public int getSize()
{
        return size;
}
```

set()

```
public void set(int index, Character c)
{
        array[index] = c;
}
```

toString()

```
public String toString()
{
	StringBuilder sb = new StringBuilder();
	sb.append("[");
	toString(0, sb);
	sb.append("]");
	return sb.toString();
}

private void toString(int index, StringBuilder sb)
{
	if (index < getSize() - 1)
	{
		sb.append(array[index] + ", ");
		toString(index + 1, sb);
	}
	else if (index < getSize())
	{
		sb.append(array[index]);
	}
}
```

add() – version 1

```
public void add(int index, Character c)
{
	if (index < 0 || index > getSize())
	{
		throw new IndexOutOfBoundsException();
	}

	// Assumes available space in array.
	if (getSize() == 0  || getSize() == index)
	{
		array[index] = c;
		}
		else
		{
			shiftTowardsEnd(index, getSize() - 1);
			array[index] = c;
		}
		size++;
	}
```

add() – version 1

```
// Shift elements by one towards the end of the array
// Assumes available space in array.
private void shiftTowardsEnd(int index, int current)
{
	if (current >= index)
	{
		array[current + 1] = array[current];
		shiftTowardsEnd(index, current - 1);
	}
}
```

remove()

```
public Character remove(int index)
{
	Character result = array[index];
	shiftTowardsStart(index + 1);
	size--;
	return result;
}

// Shift elements by one towards the start of the array
private void shiftTowardsStart(int current)
{
	array[current - 1] = array[current];
	if (current == getSize() - 1)
	{
		array[current] = null;
	}
	else
	{
		shiftTowardsStart(current + 1);
	}
}
```

add() – version 2

```
public void add(int index, Character c)
{
        // Check for valid index…

        Character[] newArray = new Character[2 * array.length];
        copyArray(array, newArray, 0);
        array = newArray;

        // Same as before…
}

private void copyArray(Character[] from, Character[] to, int index)
{
        if (index < getSize())
        {
                to[index] = from[index];
                copyArray(from, to, index + 1);
        }
}
```

Big-O running time

- getSize():
- set():
- toString():
- add():
- remove():

Big-O running time

- getSize(): O(1)
- set(): O(1)
- toString(): O(n)
- add(): O(n)
- remove(): O(n)

Using generics

- Allows data structures in Java to hold elements of any data type
  - E.g.: examples: List<E>, Map<E>, Set<E>, etc.
- Specific type is indicated during declaration
- MyArrayList
  - public class MyArrayList<E>
  - Replace “Character” with “E”
    - private E[] array
    - array = new E[DEFAULT_SIZE];
    - etc.

Programming exercise

- Recursive version of MyArrayList available on course website
- Re-write the code for MyArrayList to use loops instead of recursion