

# COMP 250

## INTRODUCTION TO COMPUTER SCIENCE

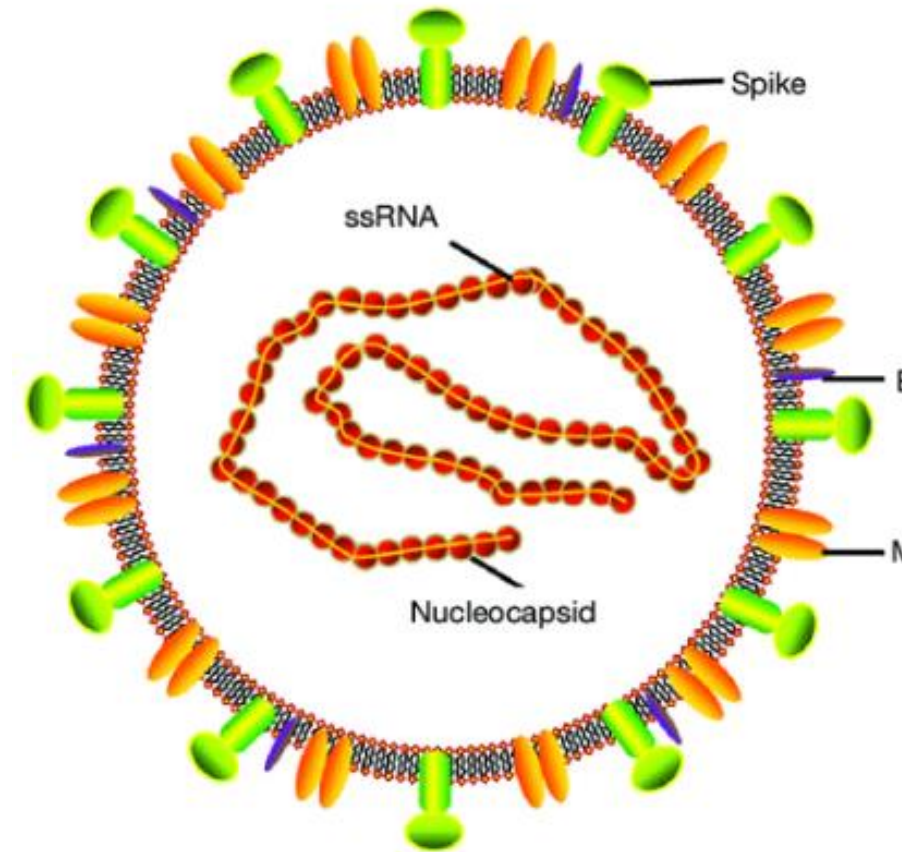
### Lecture 12 – Arrays?

Roman Sarrazin-Gendron, Winter 2020

Slides very much based on Michael Langer and Giulia Alberini

## ABOUT ME

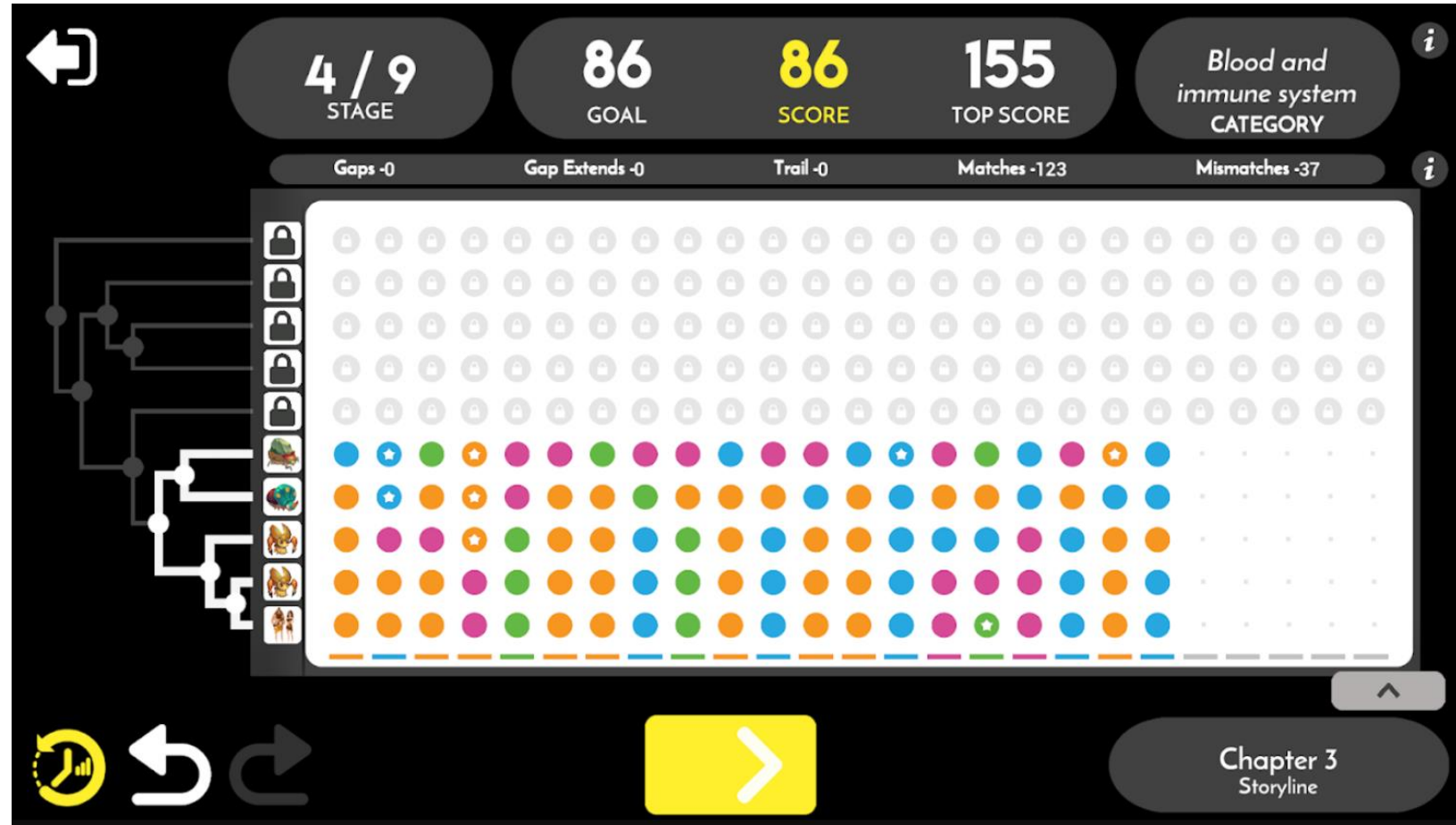
- I am a PhD candidate in Computer Science at McGill
- My research focuses on applying methods/algorithms from Computer Science to the problem of RNA structure prediction.
- RNA is a cousin of DNA, essential to life
- RNA is the final product of 75% of your genes.
- The coronavirus is also RNA!



MERS coronavirus, Y. Wang et al, 2018

## SCIENCE AND VIDEO GAMES

- A lot of *brain computation effort* is spent on games.
- A considerable part is spent on solving problems.
- We can formulate scientific problems as games (puzzles).
- For some specific problems, human intuition can be superior or complementary to algorithms/AI.



## NOT ABOUT ME

- Since the start of the semester, we have been using a bunch of **arrays**.
- In COMP 202, many of you used **arrays**, and then **ArrayLists**.
- In COMP 202 last semester, many of you used **lists** in python.
- What's the difference between these things?
- What is an array?

## LET'S TAKE A STEP BACK

- Since the first week of COMP 202, you might have been thinking about how to solve programming challenges as a coding problem.
  - Solving problems by thinking at a scale of one line.
    - Why does Eclipse tell me this line causes a compiling problem?
    - How can I get the Scanner library thing to work?
- We are now shifting the focus on the course to try to think about problems with more perspective.
  - Solving problems at a scale of one method or one class.
    - What strategy should I use to tackle this problem?
    - How can I make my program more *efficient*?

## ARRAYS IN JAVA

- An array is a data structure holding a set of elements accessible with unique consecutive indexes
- Data structures are particular ways of storing data
- The objective usually is to make the data easy to retrieve
- In Java, arrays contain elements of the same type, and must be initialized with a size.

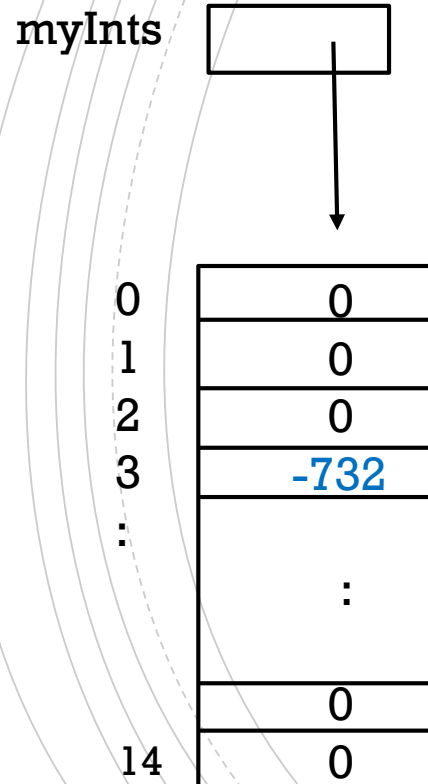
```
int[] myInts = new int[15];
```

```
myInts[3] = -732;
```

Example: array whose elements have a *primitive* type

# ARRAYS IN JAVA

- Arrays in Java are used to store variables at specific places in memory
- Primitives type data is stored directly in memory



```
int[ ] myInts = new int[15];  
myInts[3] = -732;
```

## ARRAYS IN JAVA

Other example : Array whose elements have a ***reference*** type

```
Shape[] shapes = new Shape[428];
```

```
shapes[293] = new Shape(    );
```

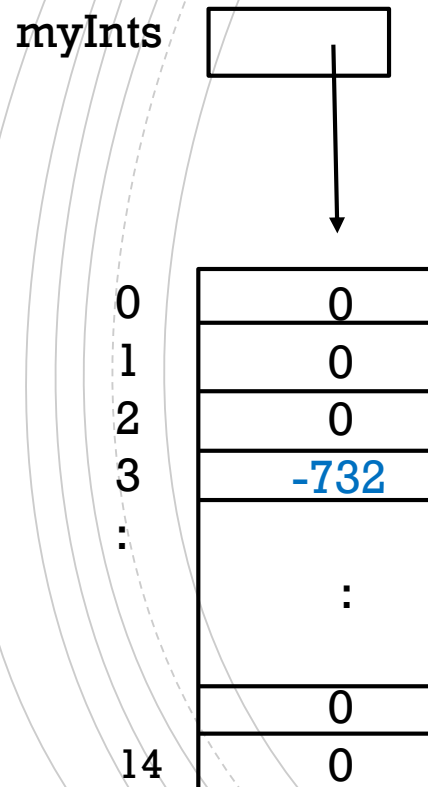


The symbol here corresponds to some arguments that specify a shape.

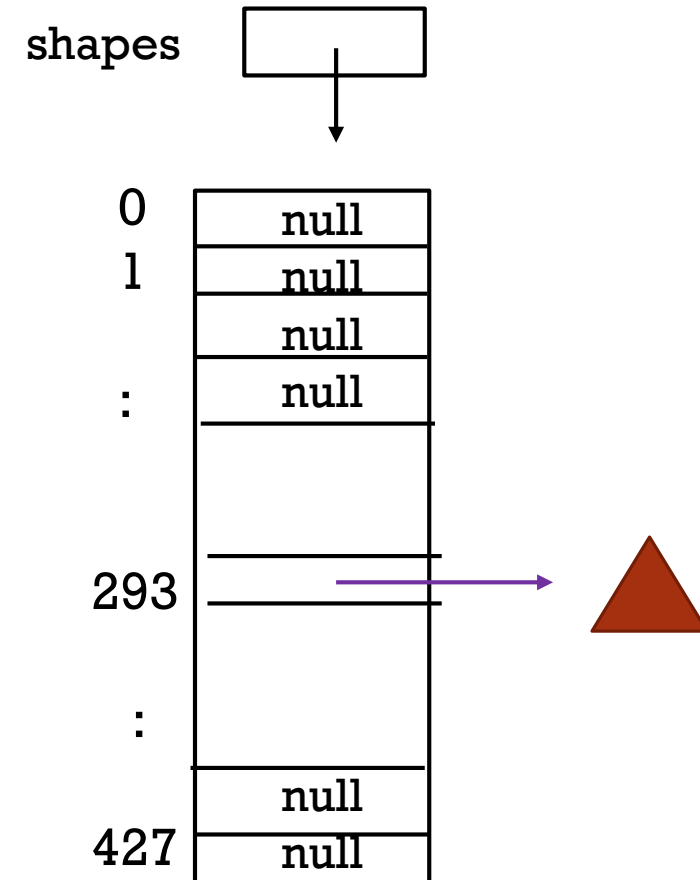


## PRIMITIVE VS REFERENCE TYPES IN ARRAYS

```
int[ ] myInts = new int[15];  
myInts[3] = -732;
```

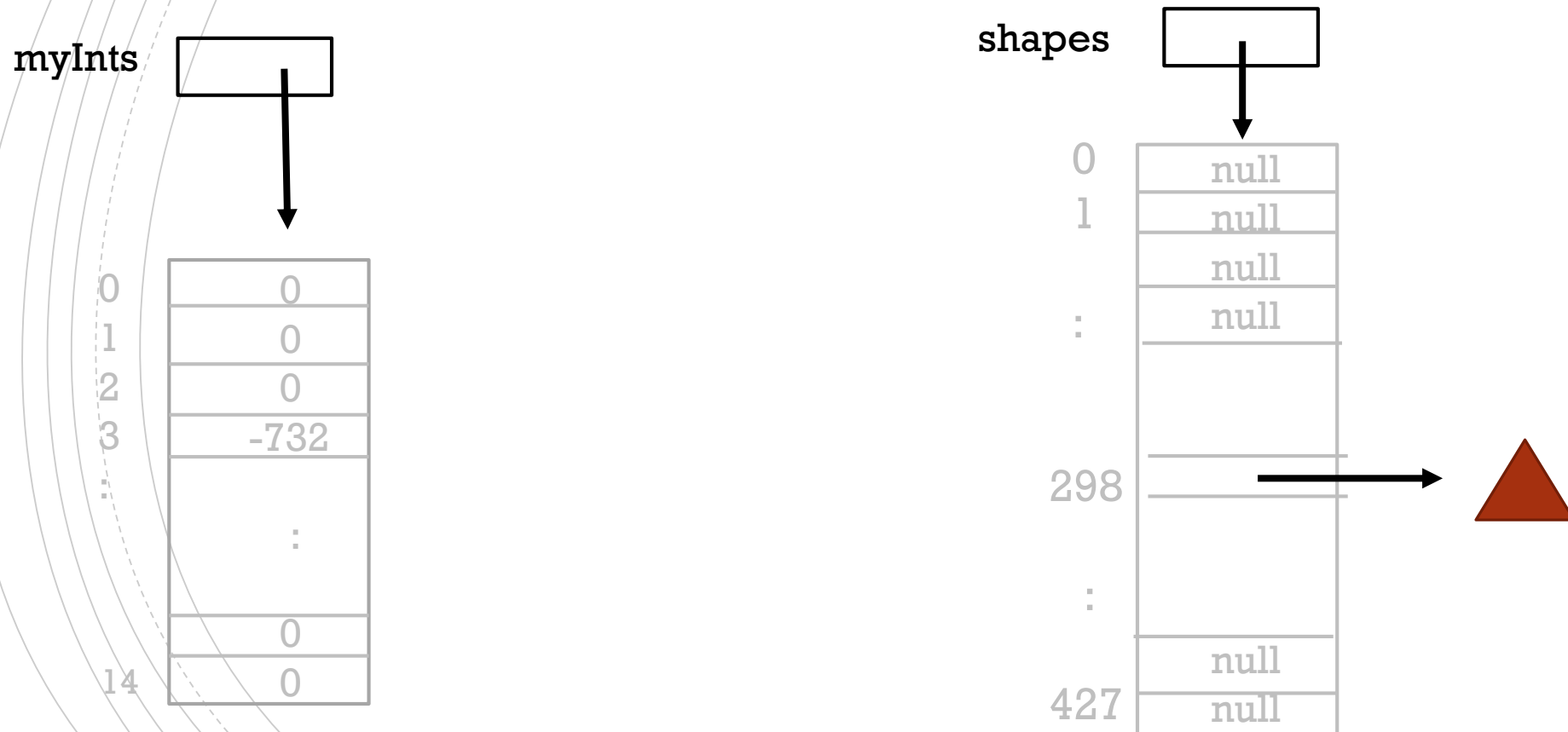


```
Shape[] shapes = new Shape[428];
shapes[293] = new Shape(    );
```



## PRIMITIVE VS REFERENCE TYPES IN ARRAYS

The value of a reference variable is an “*address*” which specifies where an object is in the computer memory. We often represent a reference with an arrow



## ACCESSING ARRAY ELEMENTS

- A computer accesses an element in an array in constant time i.e. constant, independent of the length  $N$  of the array.

```
.... = a[k] ;           // read
```

```
a[k] = .... ;          // write
```

- You will learn more about how this works in COMP 206 and 273.
- Why do we care about time?

## ARRAYS VERSUS 'ARRAY LISTS'

- If arrays can be accessed in constant time no matter the size, why not use them for everything?
- Having to declare the size ahead of time is a major problem because if we end up having more elements than planned, we have to make a new array for each new element.
- This is not a problem with **Lists** in python. Can we make **lists** in Java?

## LIST

**List** : An ordered set of elements

$$a_0, a_1, a_2, a_3, \dots, a_{N-1}$$

$N$  is the number of elements in the list, or **size**.

A list is a sequence. Unlike arrays, it doesn't have empty slots by default.

Which methods do we need in a **List** class?

## A LIST CLASS

What do we need to make our own list class in Java?

<code>get(i)</code>	// Returns the i-th element (but doesn't remove it)
<code>set(i,e)</code>	// Replaces the i-th element with e
<code>add(i,e)</code>	// Inserts element e into the i-th position
<code>remove(i)</code>	// Removes the i-th element from list
<code>remove(e)</code>	// Removes first occurrence of element e from the list if there
<code>clear()</code>	// Empties the list.
<code>isEmpty()</code>	// Returns true if empty, false if not empty.
<code>size()</code>	// Returns number of elements in the list

## HOW DO WE IMPLEMENT LISTS IN JAVA?








- **Array List** : today
- **Array Lists** are a type of list that primarily rely on automatically manipulating arrays so that the user only calls **List** operations.
  - In short, a **resizable array** that behaves like a list.
- **Linked List**: Wednesday

array list of int

0	4
1	-3
2	19
3	-7
4	221
5	0
6	16
7	0
8	0
9	0
10	0

size = 7  
length = 11

array list of Shape

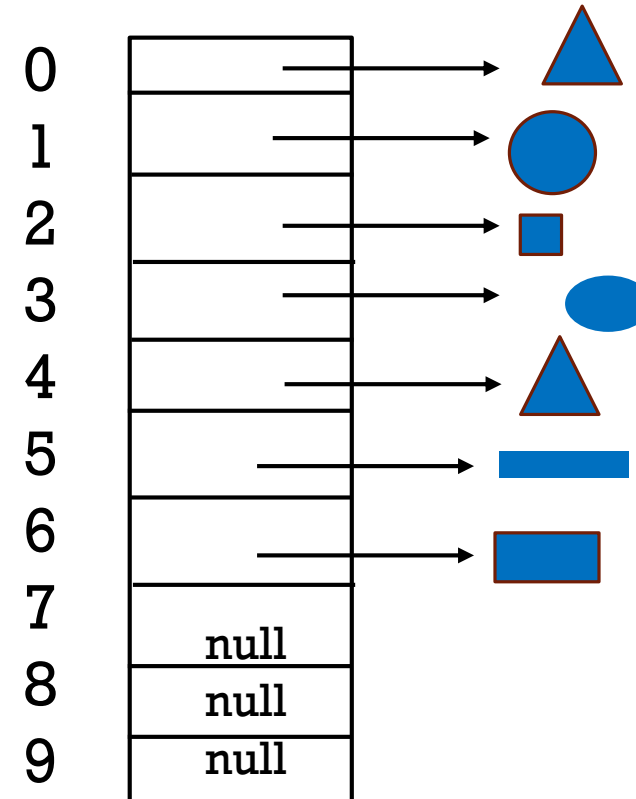
0		→	
1		→	
2		→	
3		→	
4		→	
5		→	
6		→	
7	null		
8	null		
9	null		

size = 7  
length = 10



Let's assume that the array is `a[ ]`.  
How to implement various operations ?

```
get(i) {
    if (i >= 0) & (i < size)
        return a[i]
}
```



size = 7  
length = 10

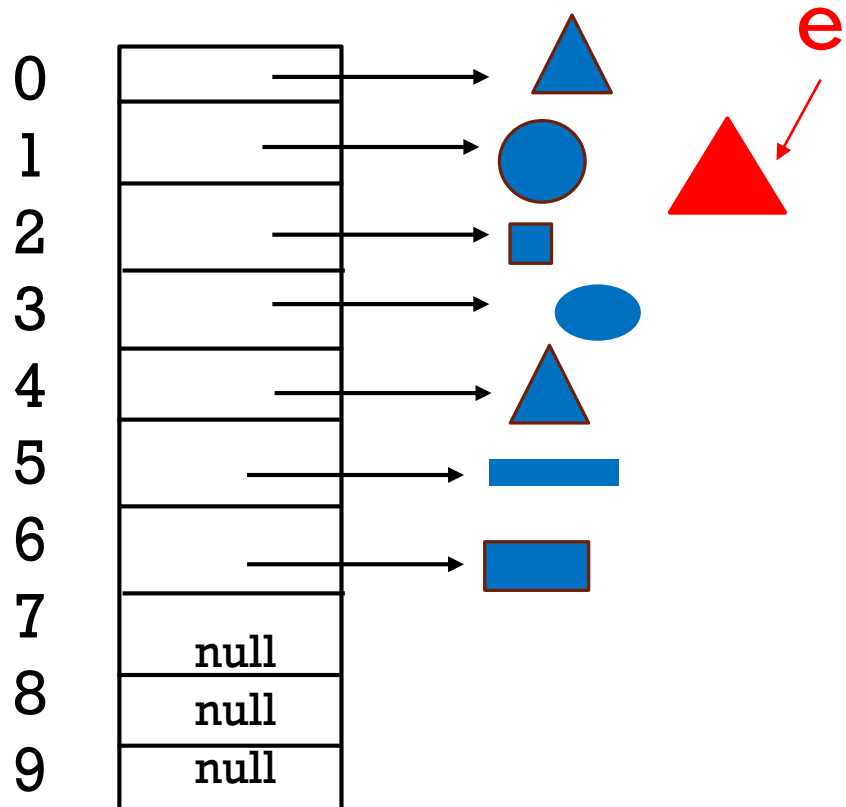
`set(i, e)` { // replaces the object at index i

if (i >= 0) & (i < size)

`a[i] = e`

e.g. `set(4, e)`

}



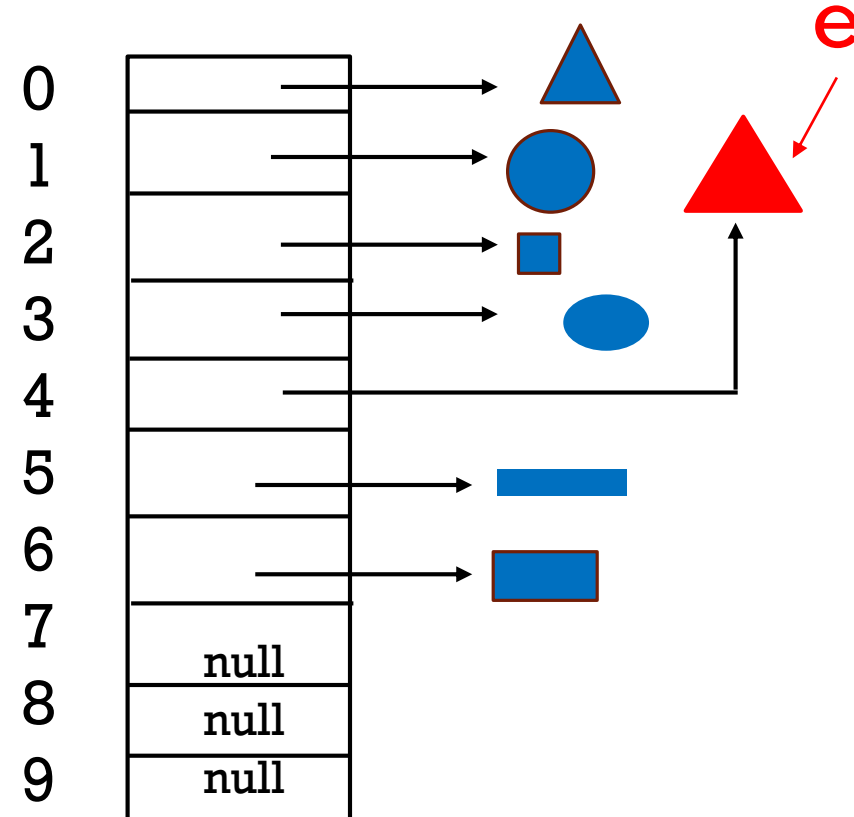
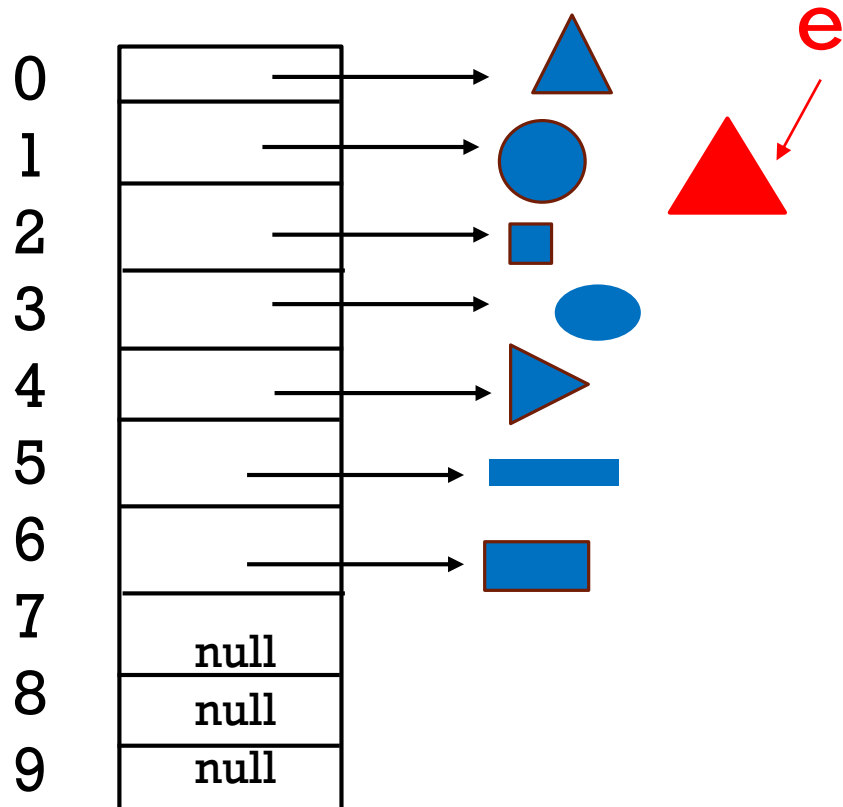
**set(i, e)** { // replaces the object at index i

if (i ≥ 0) & (i < size)

a[i] = e

}

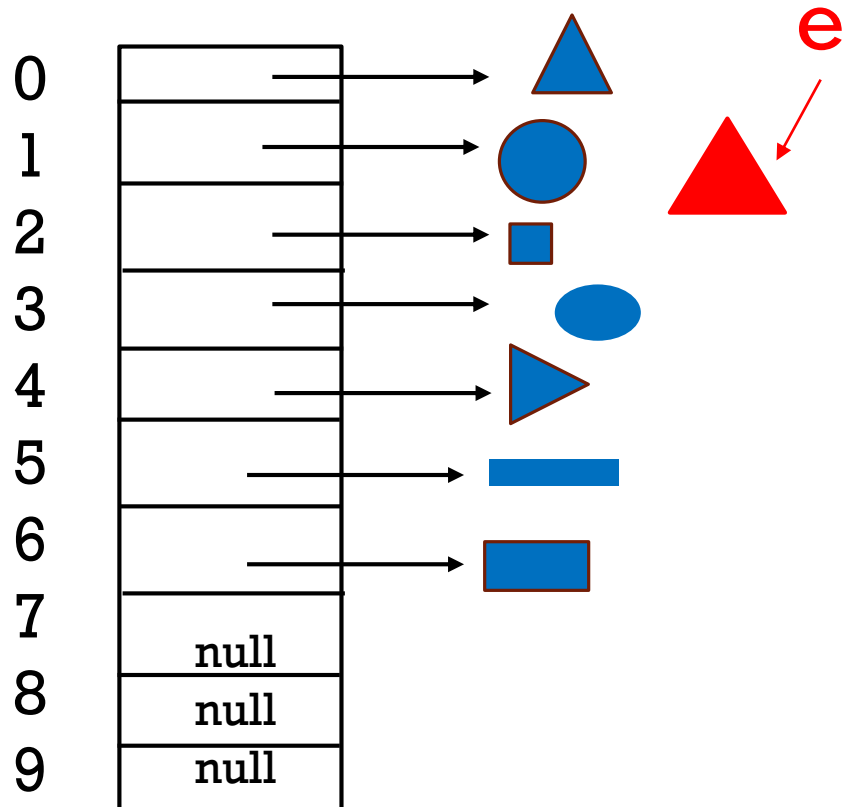
e.g. set(4, e)



add( i, **e** )

Make room by shifting, and then change reference.

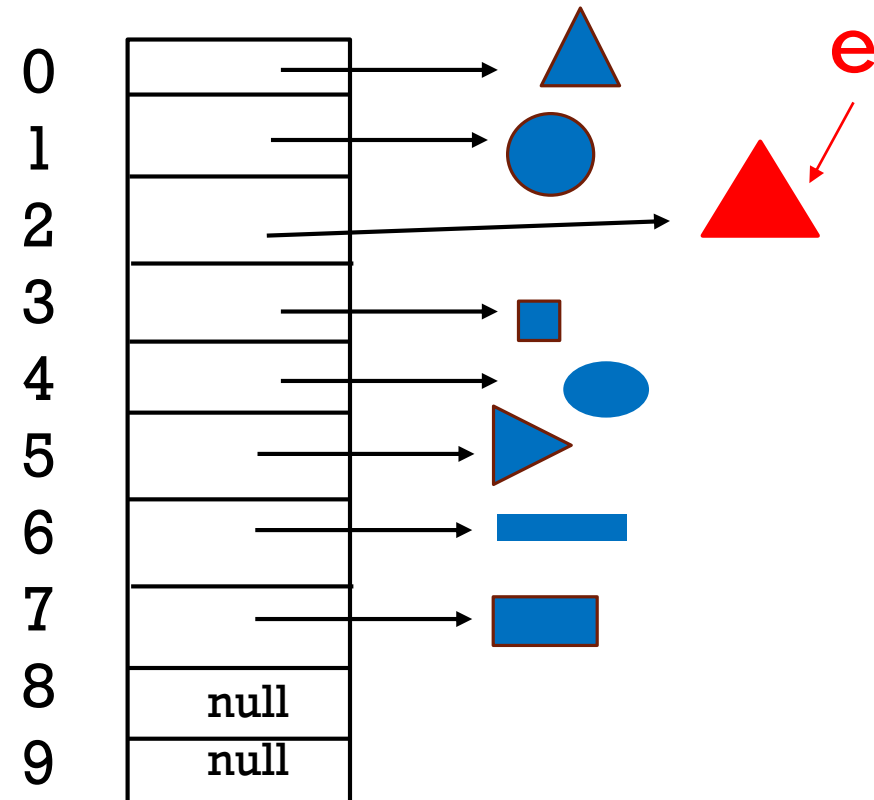
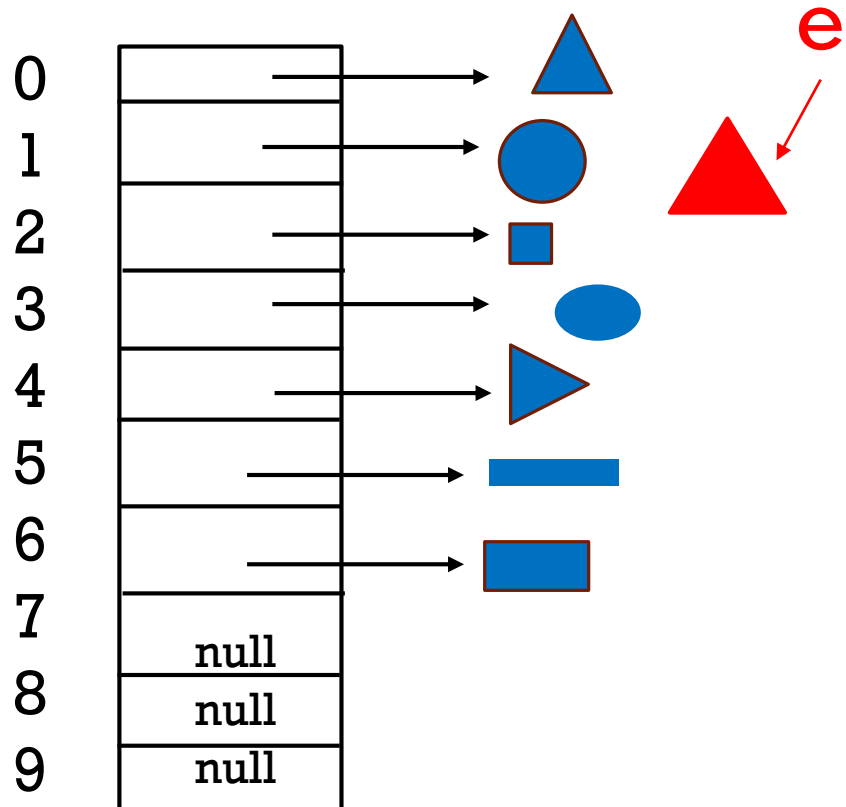
e.g. add(2, **e**)



add( i, e)

Make room by shifting, and then change reference.

e.g. add(2, e)



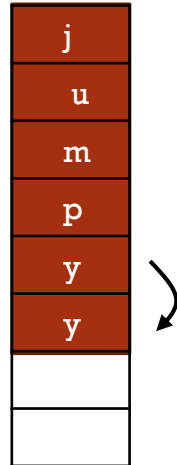
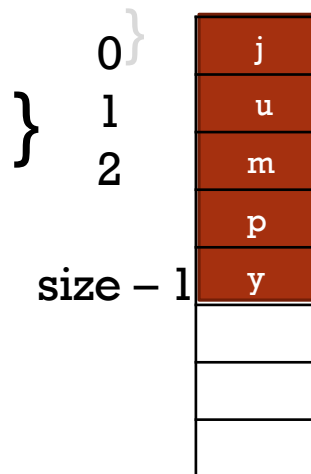
```
add( i, e) {  
  2, e)
```

```
  if (i >= 0) & (i <= size){
```

```
    for (j = size; j > i; j--)  
      a[j] = a[j-1]
```

```
    a[i] = e  
    size = size + 1
```

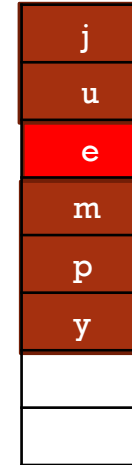
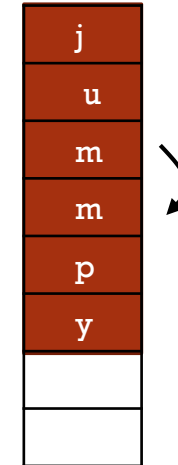
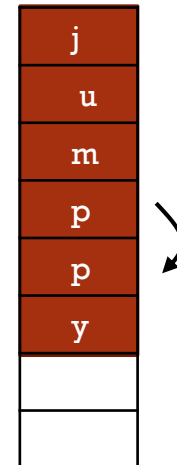
elements



```
  // shift (copy)
```

```
  // replace value
```

```
  // increase number of
```



```
add( i, e) {
```

```
    if (i >= 0) & (i <= size){
```

```
        for (j = size; j > i; j--)  
            a[j] = a[j-1]
```

```
        a[i] = e  
        size = size + 1
```

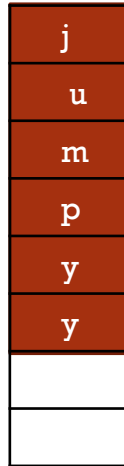
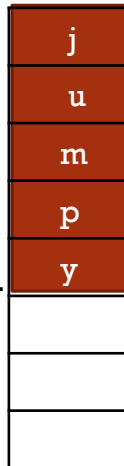
```
    elements
```

```
}
```

```
}
```

0  
1  
2

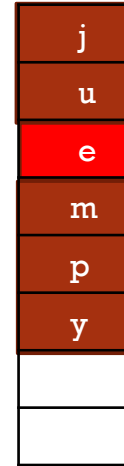
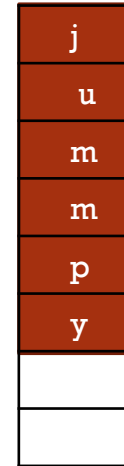
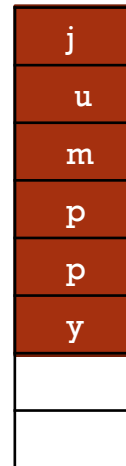
size - 1



```
// shift (copy)
```

```
// replace value
```

```
// increase number of
```



## FULL ARRAYLISTS

Adding elements works great, but what if the array is full?

```
add( i, e) {  
  
    // Create an empty bigger array.  
  
    // Copy all elements to bigger array.  
  
    // Add new element to the bigger array.  
}
```



## FULL ARRAYLISTS

```
add( i, e ) {
```

```
    if (a.size == a.length){  
        make new bigger array b  
        for ( int i=0; i < size; i++)  
            b[i] = a[i]
```

```
// is array full?
```

```
// e.g. b.length = 2*a.length
```

```
// copy elements to b
```

```
    a = b  
}
```

```
    // insert the add( i , e ) code from earlier.
```

```
}
```

## DO I HAVE TO KNOW WHERE?

What if you want to add an element to the list because you don't care where it goes?

Or what if you want to add an element to the end of the list?

The `add(i, e)` code does not allow this.  
Instead we need another method `add(e)`.

## REMINDER : OVERLOADING

**add( e )**      // inserts element e at end of list

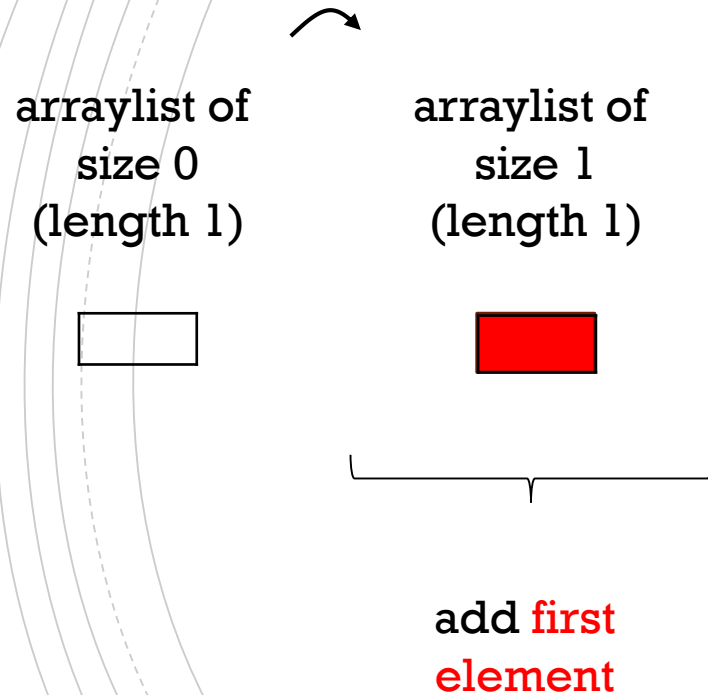
**add( i ,e)**      // Inserts element e into the i-th position

**remove(i)**      // Removes the i-th element from list

**remove(e)**      // Removes first occurrence of element e  
// from the list (if it is there)

## ADDING N ELEMENTS TO AN ARRAY LIST

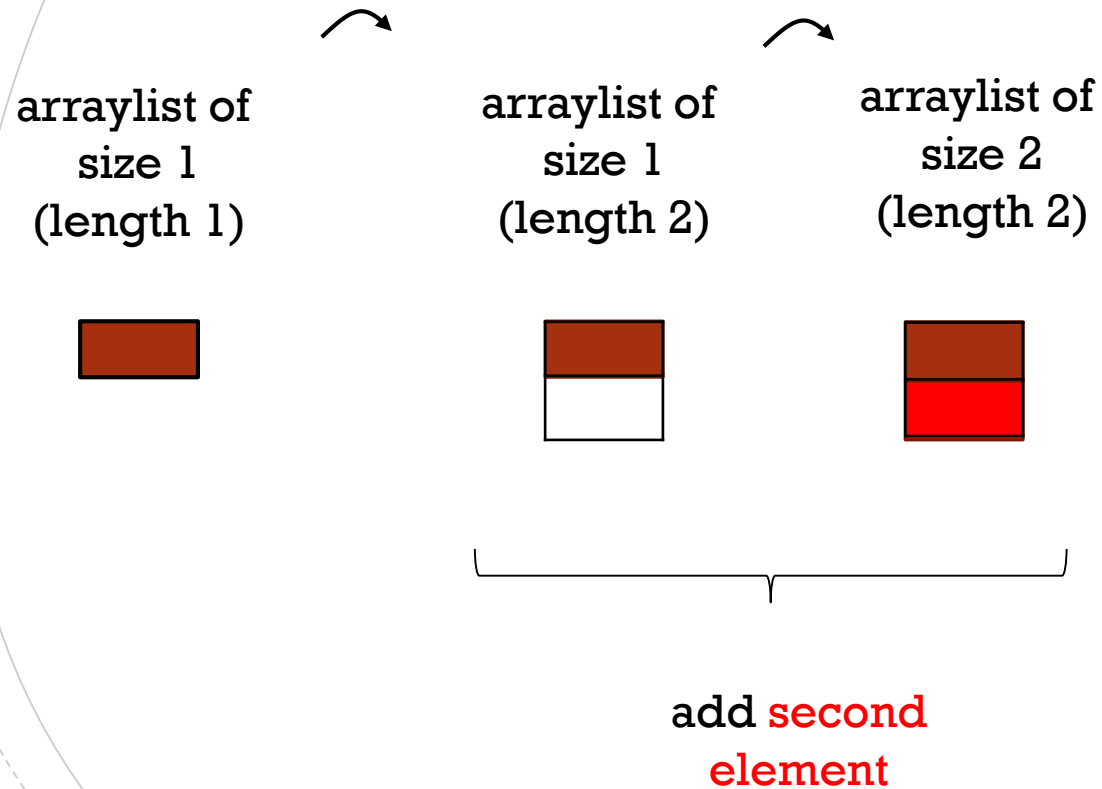
Suppose we initialize an array list with an empty array of length 1. We then add an element.



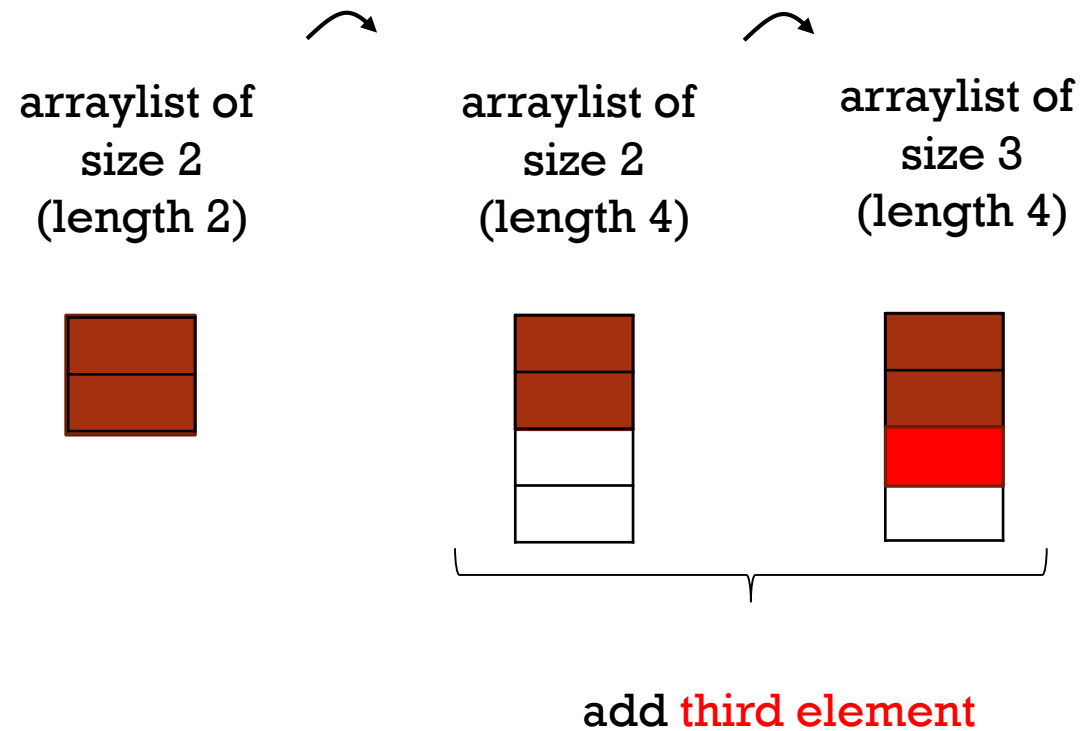
What do we do to add a second element?

## ADDING N ELEMENTS TO AN ARRAY LIST

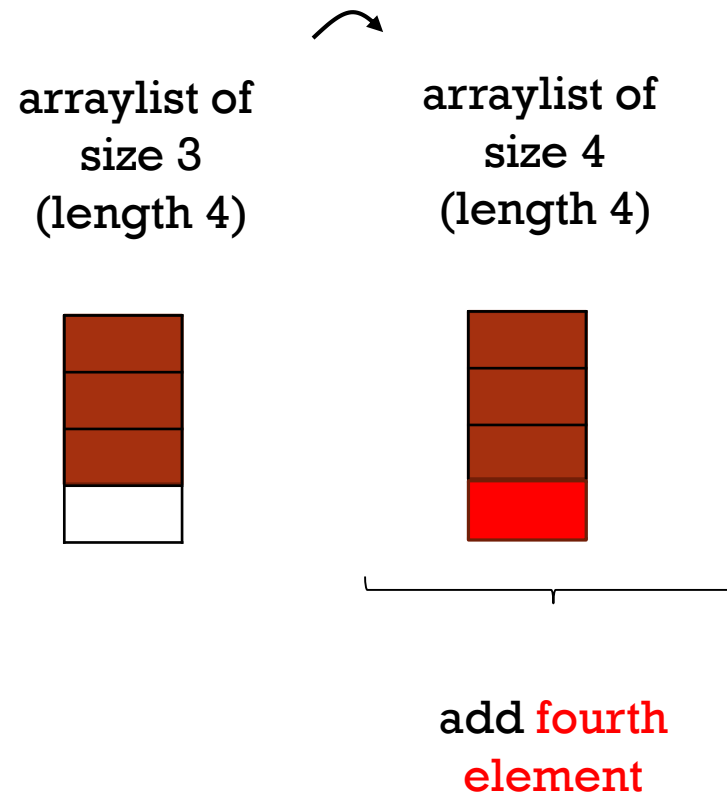
Suppose each time we add to a full array list, we double the length of the array.



# ADDING N ELEMENTS TO AN ARRAY LIST

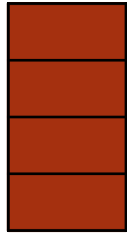


# ADDING N ELEMENTS TO AN ARRAY LIST



# ADDING N ELEMENTS TO AN ARRAY LIST

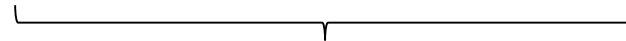
arraylist of  
size 4  
(length 4)



arraylist of  
size 4  
(length 8)



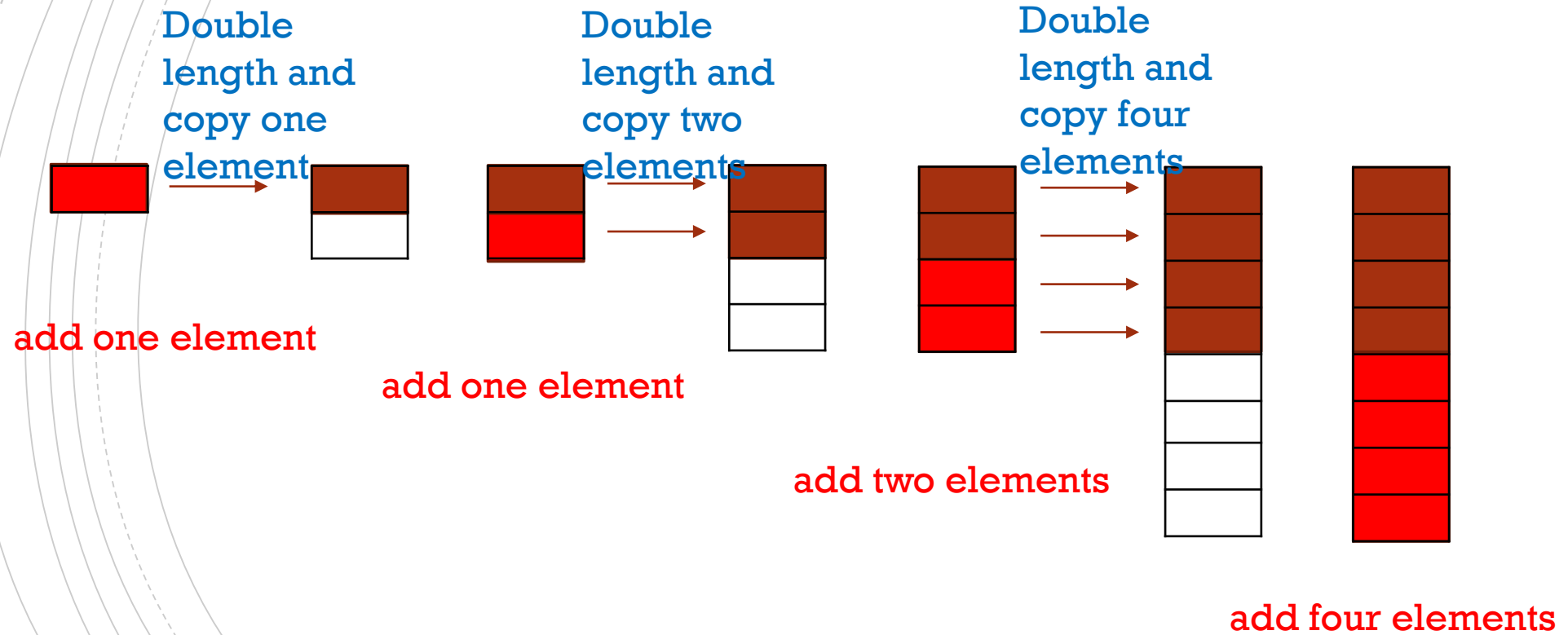
arraylist of  
size 5  
(length 8)



add fifth element



# ADDING ELEMENTS



## ADDING ELEMENTS

**Q:** How many times  $k$  do we need to double the length of the array so that it is of length  $N$  ?

**A:**

**Q:** How many **copy operations** are required to add  $N$  elements to an empty array list ?

**A:**

## ADDING ELEMENTS

Q: How many times  $k$  do we need to double the length of the array so that it is of length  $N$  ?

A:  $2^k = N$ , so  $k = \log_2 N$

Q: How many **copy operations** are required to add  $N$  elements to an empty array list ?

A:  $1 + 2 + 4 + 8 + \dots + 2^{k-1} = 2^k - 1 = N - 1$

## ADDING ELEMENTS

- We now have a full method for adding elements.
- It works; when we type the code, it will perform the operations.
- Would you say it is an efficient way of doing it if we are working for a dataset that does not significantly vary in size?
- What about for a dataset that is constantly expanding?

## REMOVING AN ELEMENT

```
get(i)
set(i,e)
add(i,e)
remove(i)      // Removes the i-th element from list
remove(e)      // Removes element e from the list (if it is there)
clear()        // Empties the list.
isEmpty()      // Returns true if empty, false if not empty.
size()         // Returns number of elements in the list
:
```

## REMOVING ELEMENTS

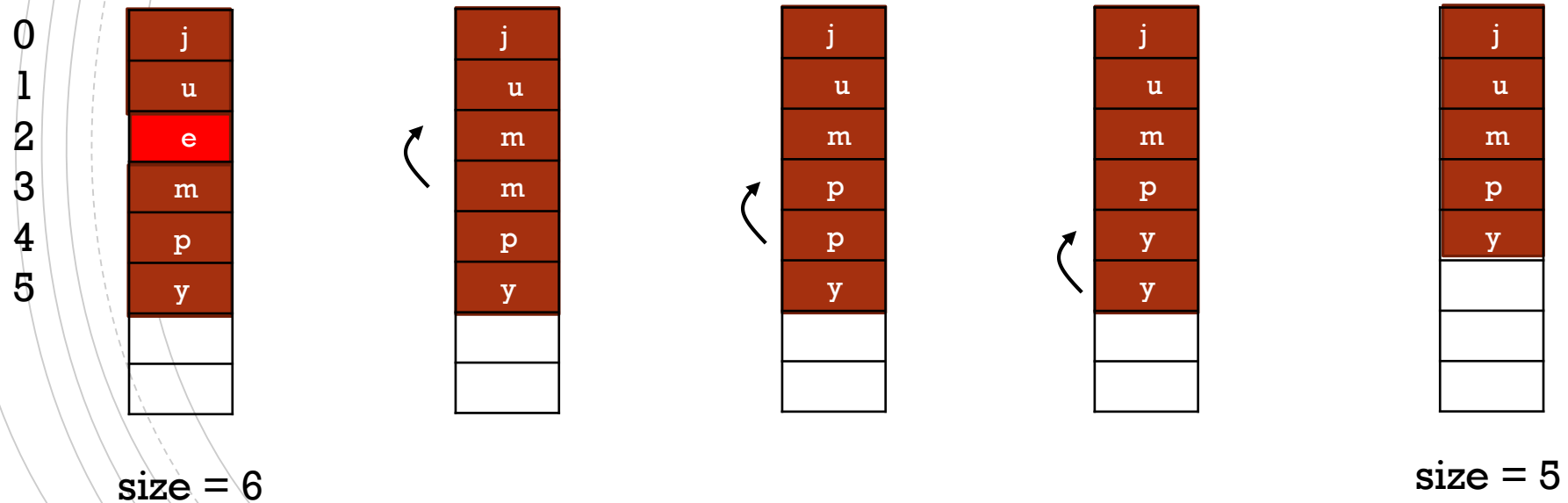
- We can just change the element at position  $i$  to null, can't we?
- We can't, because this brings the consecutive sequence assumption of the list.

$$a_0, a_1, a_2, a_3, \dots, a_{N-1}$$

## REMOVING AN ELEMENT

`remove( i )`

// in the figure below,  $i = 2$



## REMOVING AN ELEMENT

**remove(i)**

```
if ( (i >= 0) and (i < size) ){
```

```
    tmp = a[i]                // put aside and later return it
```

```
    for ( k = i; k < size-1; k++){  
        a[ k ] = a[ k + 1 ]    // shift (copy)  
    }
```

```
    size = size - 1  
    a[ size ] = null          // clean  
    return tmp
```

```
}
```



## NEWS FROM QUIZ 1

- You can now see your answers on MyCourses.
- 200 students received 10/10!  
Average: 8.2/10
- If you missed the quiz, unfortunately you cannot make it up, but the total quiz grade can be replaced by the final exam.

