

COMP 250

Lecture 9

Array lists

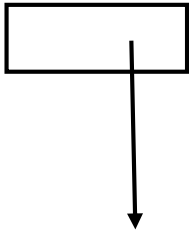
Sept. 26, 2018

Arrays in Java

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

Array whose elements have a *primitive* type


myInts



0	0
1	0
2	0
3	-732
:	:
14	0

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

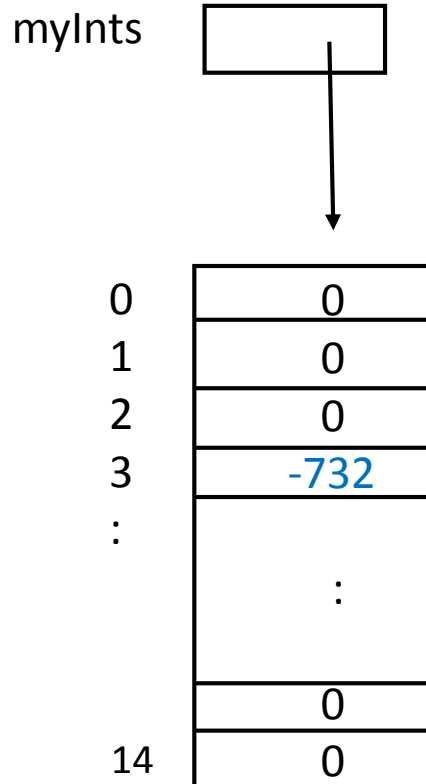
Arrays in Java

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

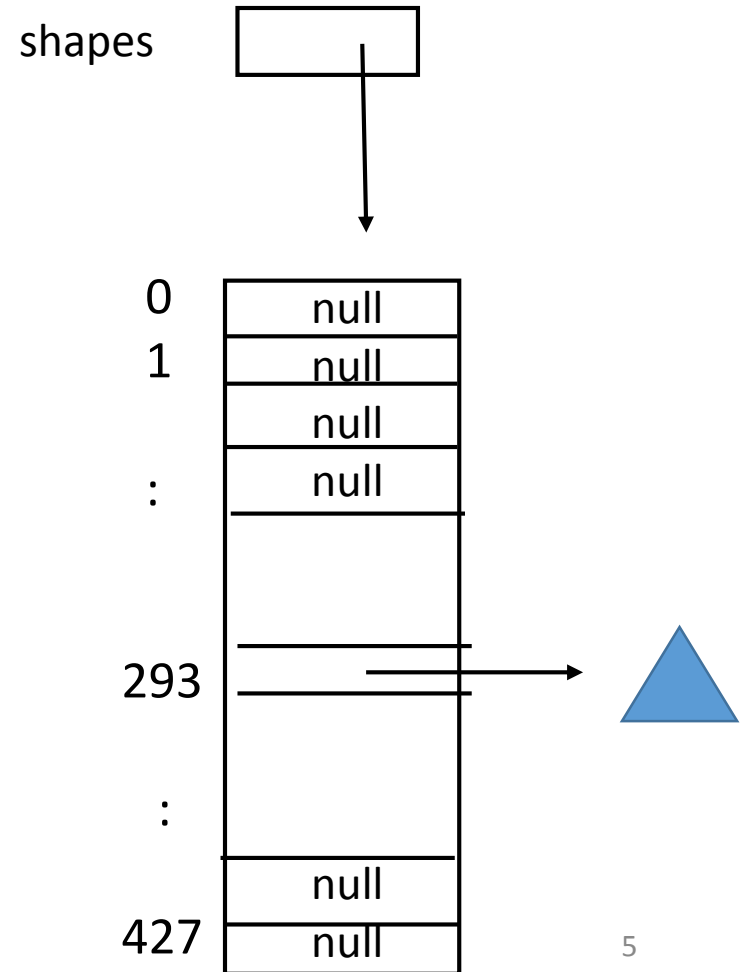
The symbol here corresponds to some arguments that specify a shape.

Array whose elements have a *reference* type

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

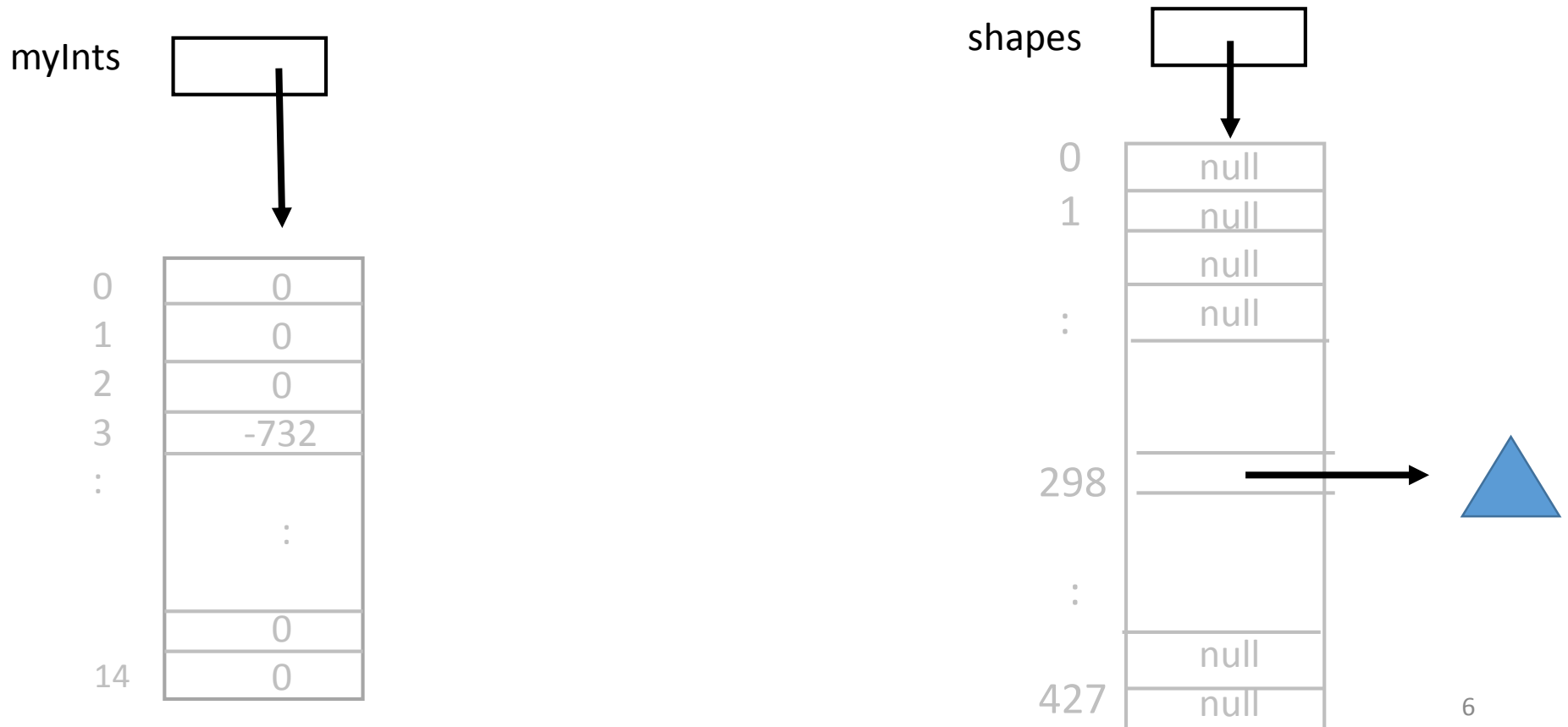


```
Shape[ ] shapes = new Shape[428];  
shapes[293] = new Shape( ▲ );
```



The value of a reference variable is an “*address*” which specifies where an object is in the computer memory (in fact, Java Virtual Machine memory). We often represent a reference with an arrow.

In the C programming language, you have access to that value and manipulate it (COMP 206). In Java, you have access to it but you can’t manipulate it.



Arrays have constant time access

A computer accesses an element in an array in *constant* time

i.e. 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.

Arrays versus 'Array Lists'

Arrays can be used to make lists,
sometimes called 'array lists'.

Java has an ArrayList class.

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, often called the “size” of the list.

What things do we do with a list?

`get(i)` `// Returns the i-th element (but doesn't remove it)`

`set(i,e)` `// Replaces the i-th element with e`

`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)`

`clear()` `// Empties the list.`

`isEmpty()` `// Returns true if empty, false if not empty.`

`size()` `// Returns number of elements in the list`

Lists

- array list (today)

- singly linked list

- doubly linked list

:



next two lectures








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			
8	null		
9	null		

size = 7

length = 10

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

`get(i)`

`set(i,e)`

`add(i,e)`

`remove(i)`

`remove(e)`

`clear()`

`isEmpty()`

`size()`

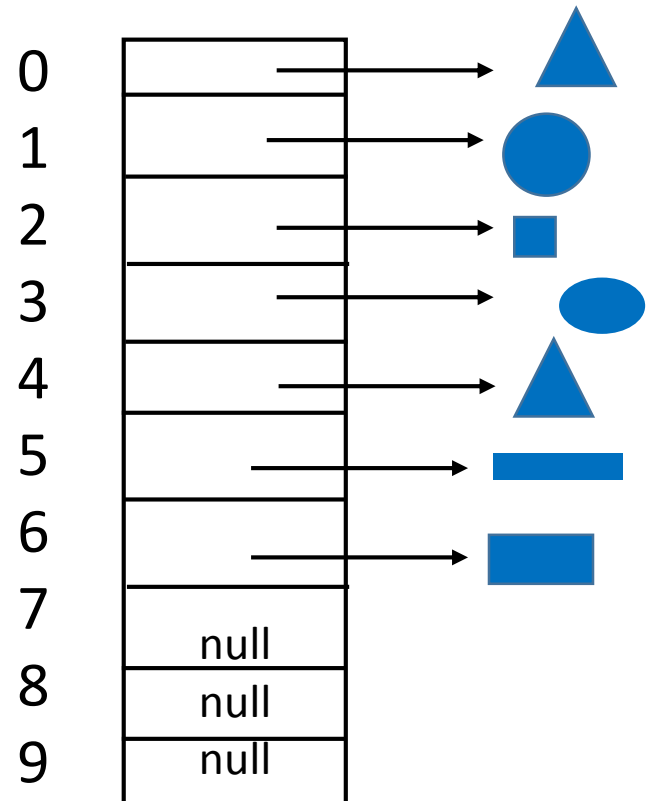
.....

```
get(i) {
```

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

```
        return a[ i ]
```

```
}
```



size = 7

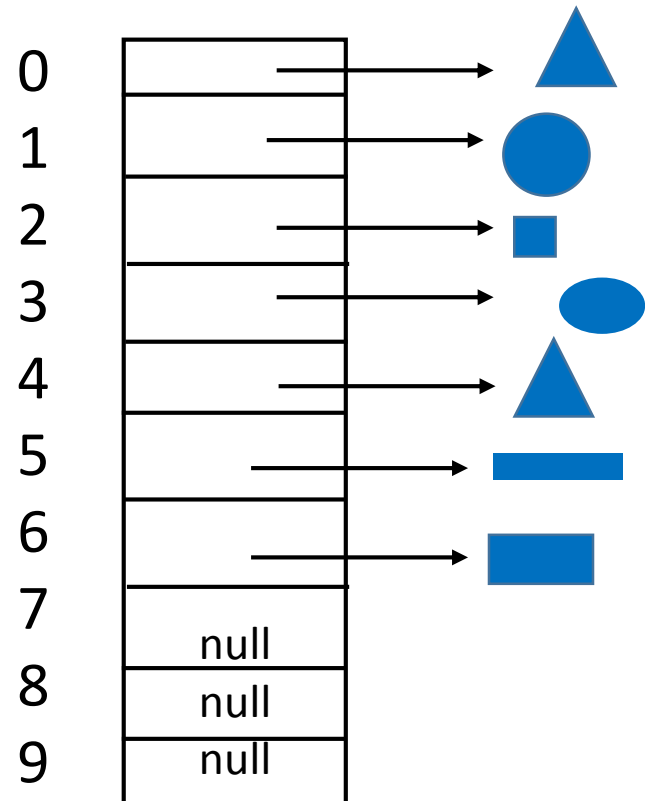
length = 10

```
get(i) {
```

```
    if (i >= 0) & (i < size)
        return a[ i ]
    else
        // index out of bounds
        // exception
```

```
}
```

I will not mention this check
in rest of methods today.
But be aware that it should
be added in proper
implementation.



size = 7

length = 10

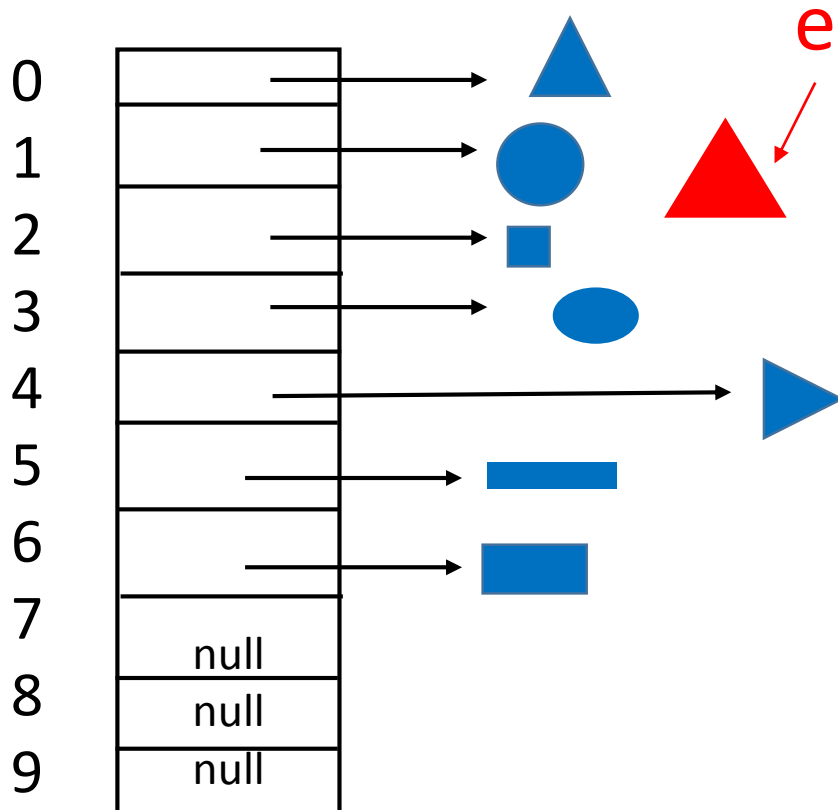
`set(i, e)` // replaces the object at index i

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

a[i] = e

}

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



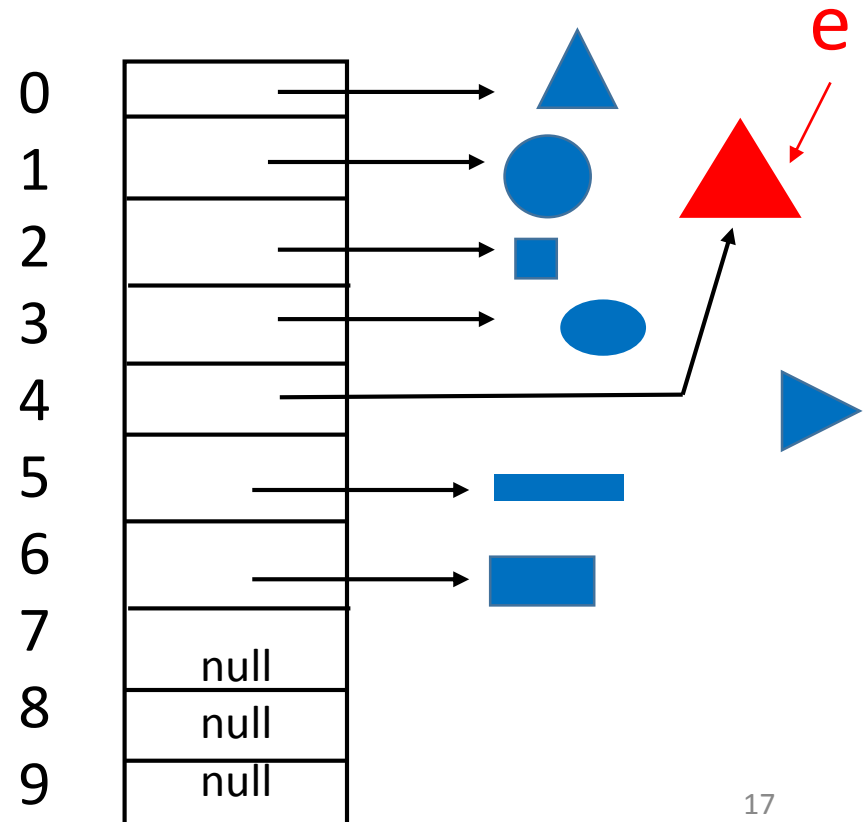
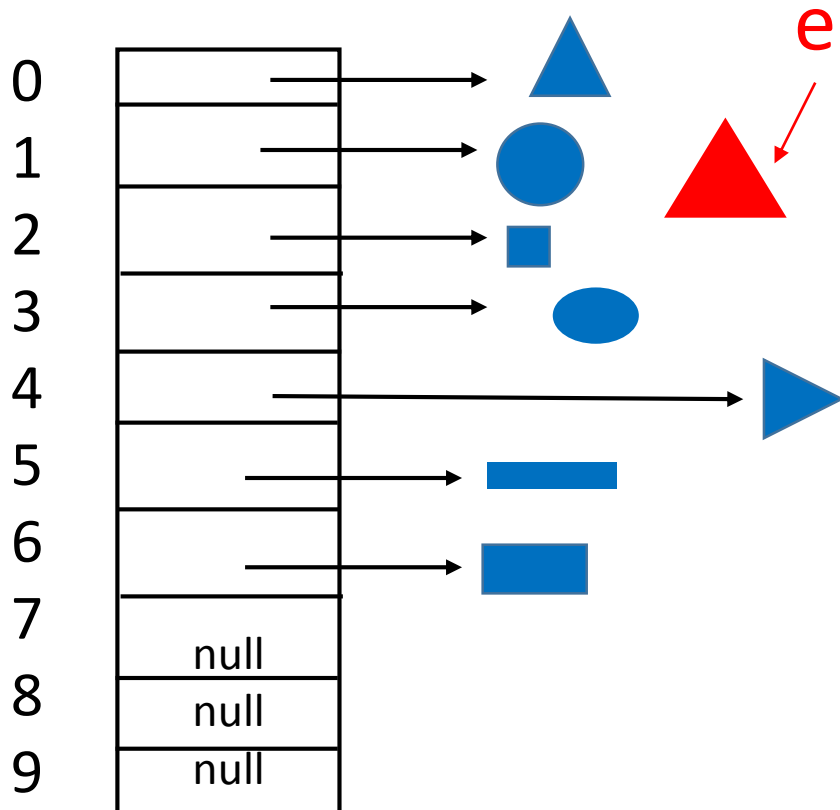
`set(i, e)` { // replaces the object at index i

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

a[i] = e

}

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



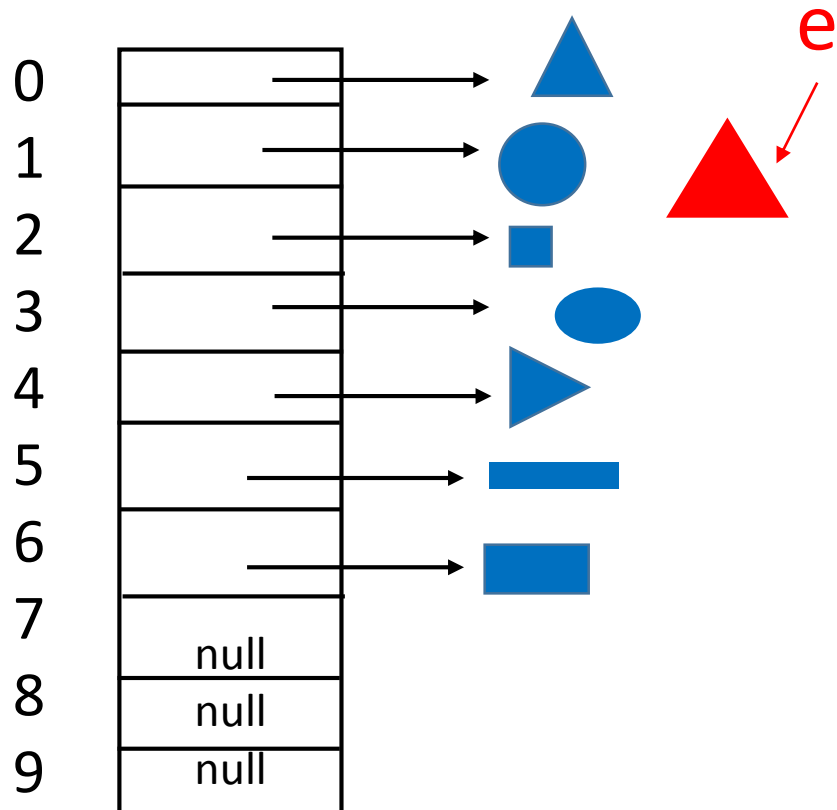
We could also implement it like this:

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

    if (i >= 0) & (i < size) {
        tmp = a[i]
        a[i] = e
    }
    return tmp // Return the element that is replaced.
}
```

add(i, **e**) // insert

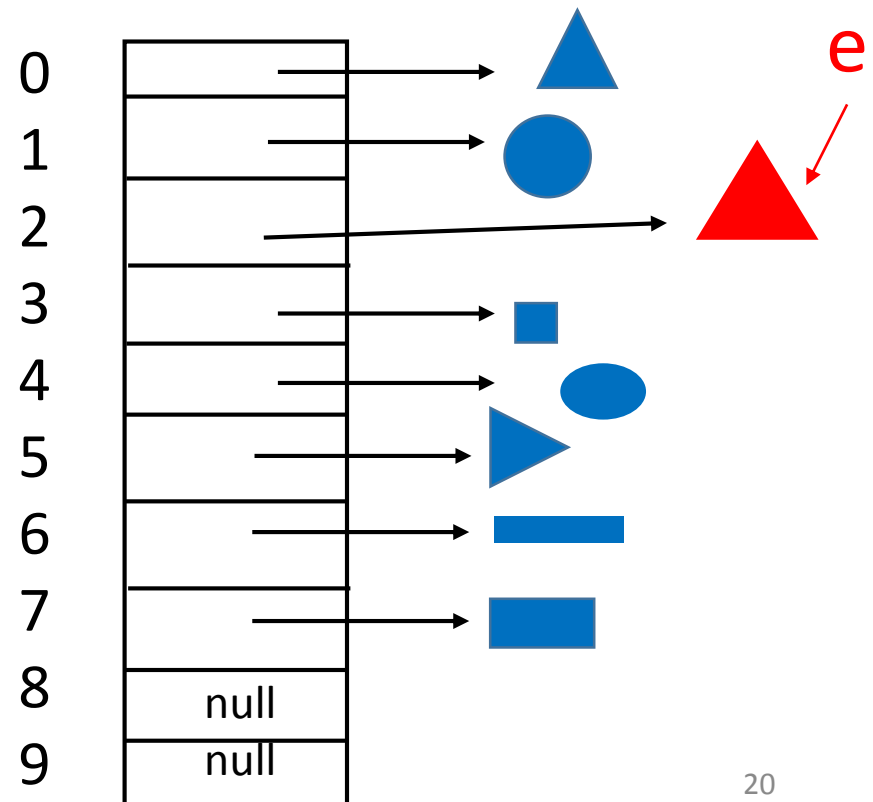
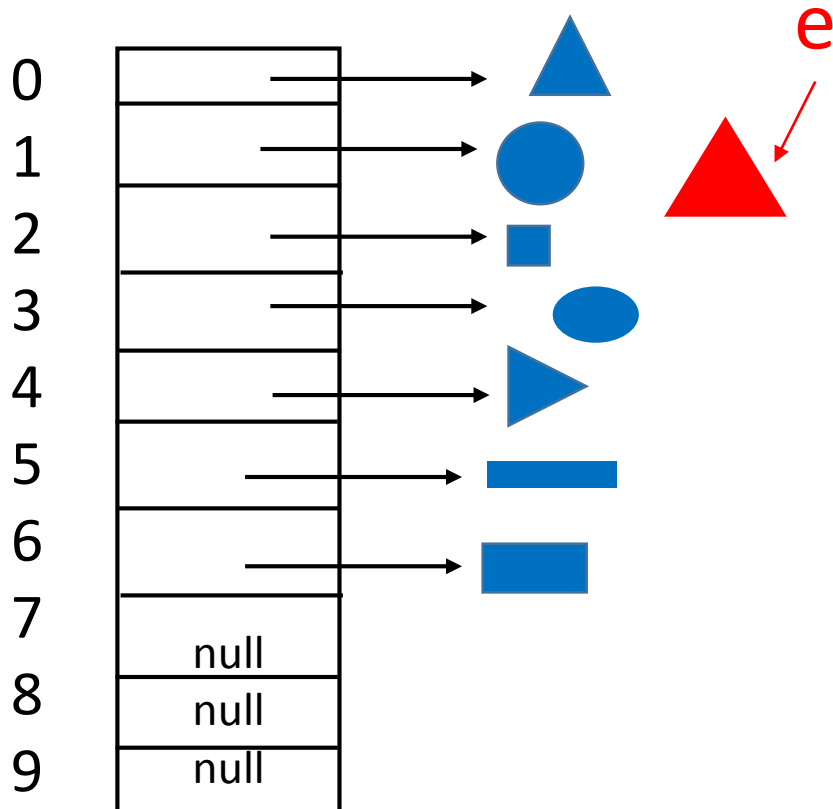
e.g. add(2, **e**)



add(i, **e**) // insert

Make room by shifting, and then change reference.

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

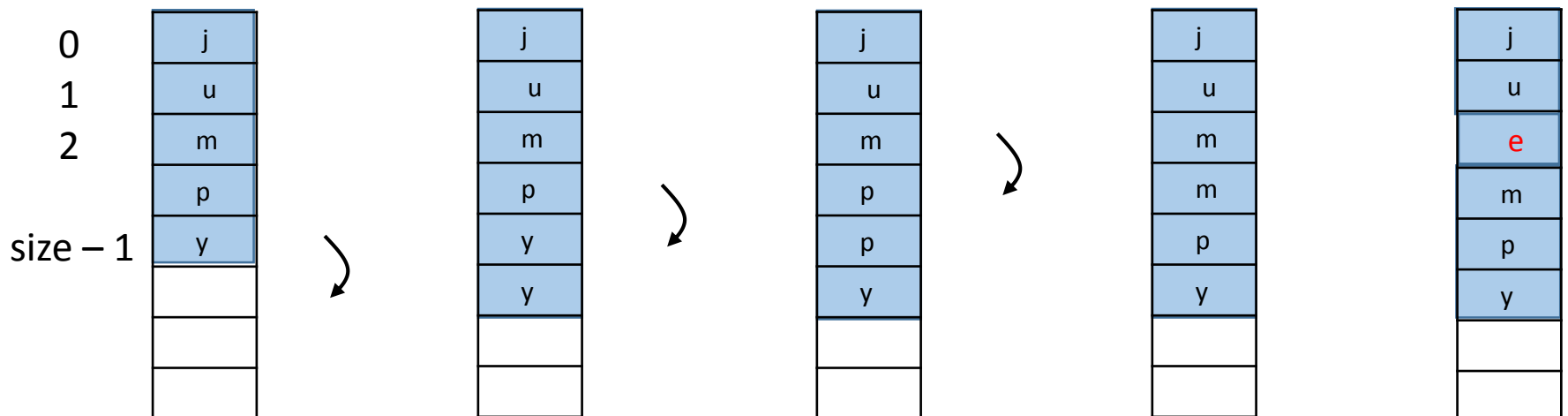


add(i, **e**)

Make room by shifting, and then change reference.

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

Take an example of inserting character '**e**' into array of characters.



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

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

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

```
// else throw index out of bounds
```

```
        for (j = size; j > i; j--)
```

```
            a[j] = a[j-1]
```

```
// sequence of copy forwards
```

```
        a[i] = e
```

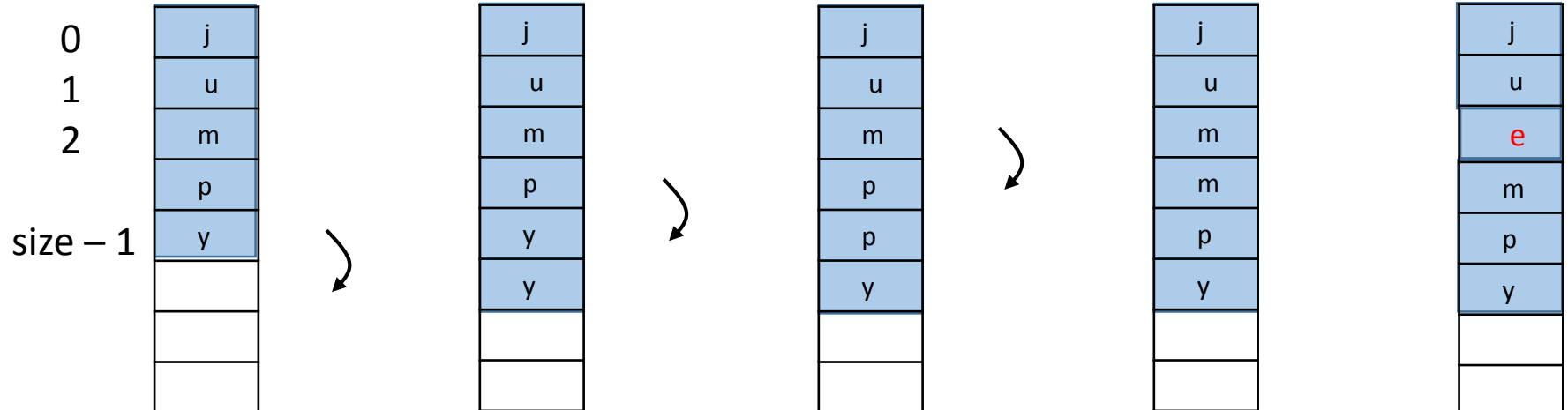
```
// replace value
```

```
        size = size + 1
```

```
// increase number of elements
```

```
    }
```

```
}
```



How to add an element to an array list
when array is full ?

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

```
}
```

How to add an element to an array list
when array is full ?

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

```
    // Create an empty bigger array.
```

```
    // Copy all elements to bigger array.
```

```
    // Add new element to the bigger array.
```

```
}
```



```
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 here the add( i , e ) code from a few slides back
```

```
}
```

Suppose you want to add an element to the list and you don't care where it goes.

Use `add(e)`.

Where is the best place to put `e` and why?

See Exercises.

SLIDE ADDED AFTER LECTURE

Note that you *are* allowed to add(i, e) to slot “i == size”. The set(i, e) method does not allow this. e.g. See Java ArrayList specifications of these methods.

```
add( i, e) {                               // in the figure below, i = 2
    if (a.size == a.length){                // is array full?
        make new bigger array b             // e.g. b.length = 2*a.length
        for ( int i=0; i < size; i++)
            b[i] = a[i]                     // copy elements to b

        a = b
    }

    if (i >= 0) & (i <= size){               // else throw index out of bounds
        for (j = size; j > i; j--)
            a[j] = a[j-1]                   // sequence of copy forwards

        a[i] = e                           // replace value
        size = size + 1                     // increase number of elements
    }
}
```

Java ArrayList class

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

It uses an array as the underlying data structure

When the array is full and a new element is added, it grows the array (by 50%).

You don't use the usual array notation `a[]`. Instead, use `get()` and `set()` and other methods.

Java generic type

An array of what? `ArrayList<T>`

Example:

```
ArrayList< Shape >    shape = new ArrayList< Shape >();
```

```
// initializes the array length (capacity) to 10
```

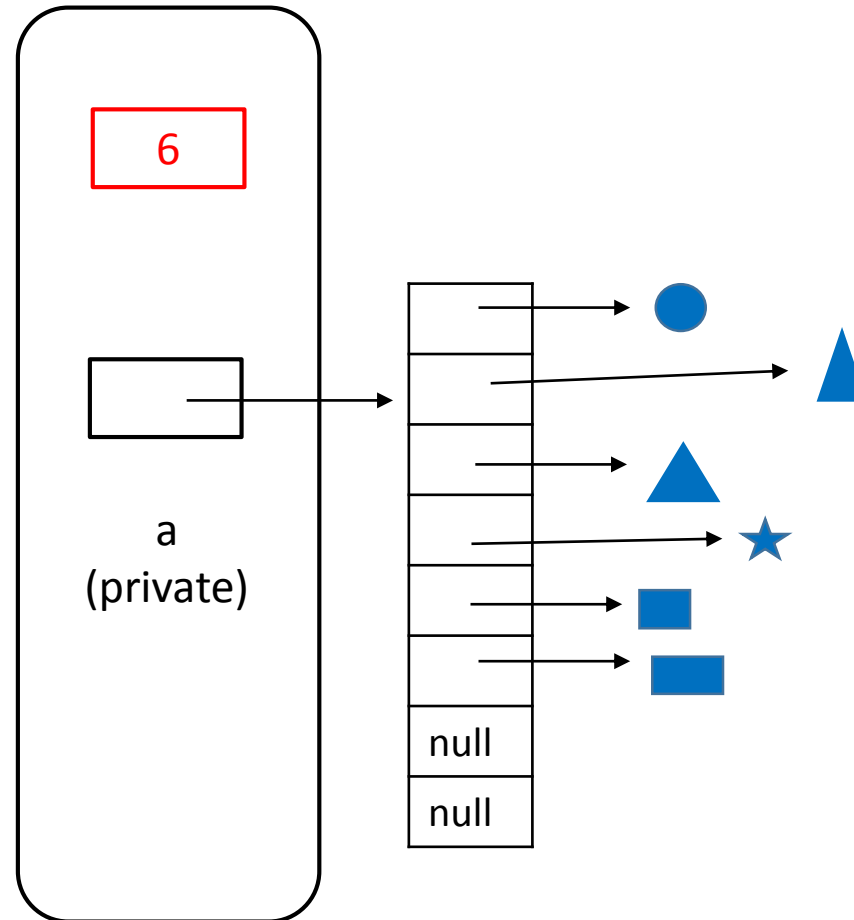
```
ArrayList< Shape >    shape = new ArrayList< Shape >( 23 );
```

```
// initializes the underlying array length to 23
```

Java ArrayList object

Has private field that holds the number of elements in the list (size).

Has a private field that references an array object.



These Shape objects do not belong to the ArrayList object.

Rather they are *referenced* by it.

List Operations (ArrayList too)

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.

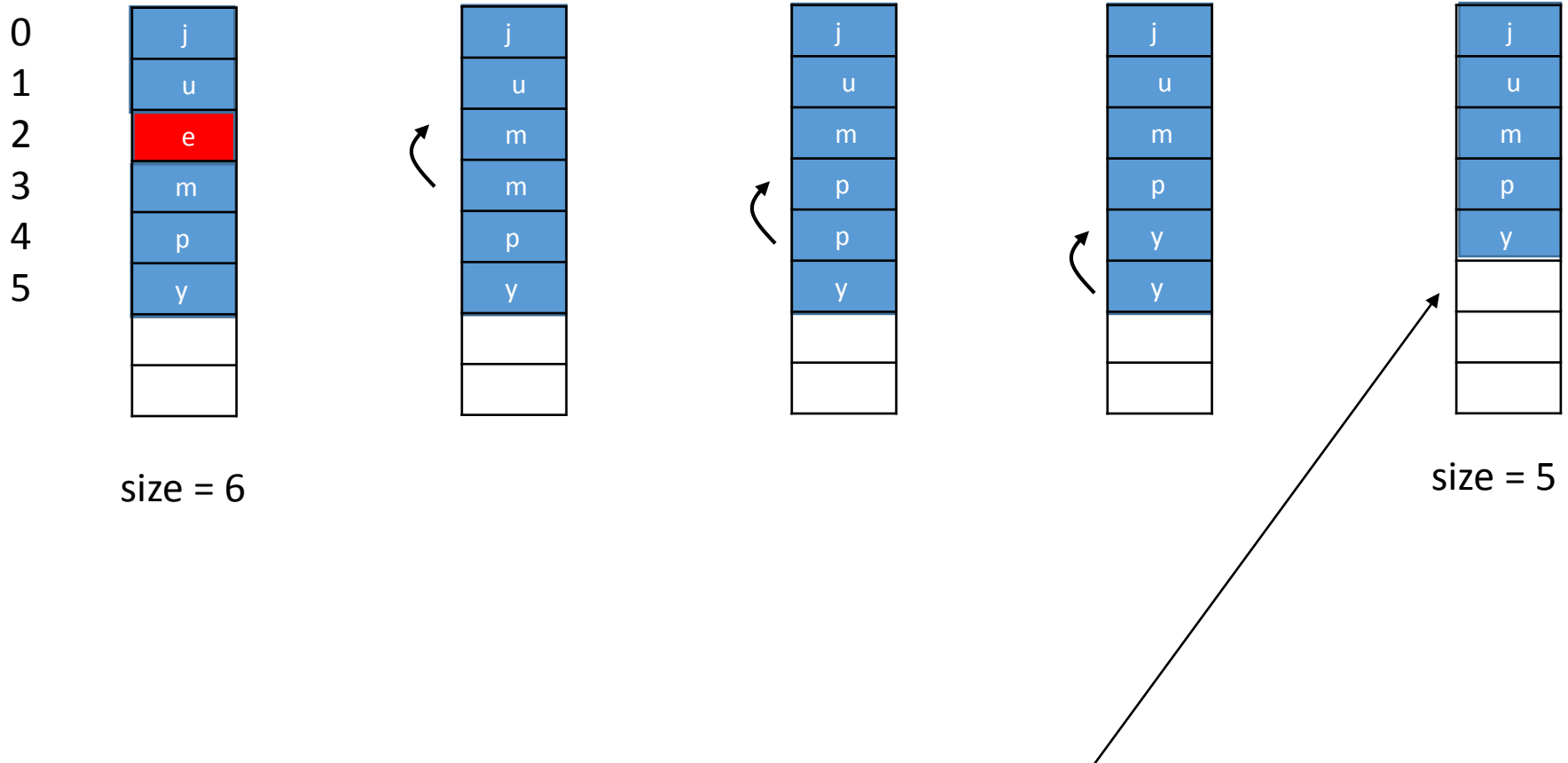
isEmpty() // Returns true if empty, false if not empty.

size() // Returns number of elements in the list

:

remove(i)

// in the figure below, i = 2



The value 'y' will still be there, but it is not accessible. Why not?

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          // not necessary – see previous slide  
    return tmp  
}
```

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)

Time Complexity: big O

Let N be the number of elements in the array list (size).

$\text{add}(i, e)$ requires up to N shifts. We say it is " $O(N)$ ".

$\text{remove}(e)$ requires up to N shifts. We say it is " $O(N)$ ".

$\text{add}(e)$ requires constant time, if array is not full
and requires N copies if array is full.

We say it is " $O(1)$ " or " $O(N)$ ", respectively.

Announcements

- **Tutorials:** only 3 people came yesterday
(hold them 5:30-7 instead? Weekends?)
- **Assignment 1**
 - subscribe to Announcements & Discussion Board
- **Quiz 1** is Friday (see Course Outline for policy)