

## lecture 6

- list ("abstract data type")
- Java ArrayList
- Java LinkedList

What is a List?  
(in an abstract sense)

- sequence of a certain type of thing
- certain operations which affect the sequence in a well defined way

### List methods

- add(element) // to end
- add(i, element)
- get(i)
- remove(i)
- set(i, element)
- clear()
- isEmpty()
- size()

List Implementations  
in Java

- ArrayList
- LinkedList

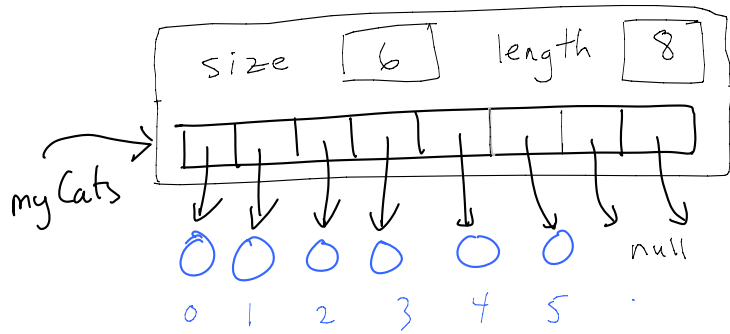
### Java ArrayList

- uses an array as underlying implementation  
BUT you don't use usual array notation  $a[i]$
- instead use methods such as  $get(i)$ ,  $set(i)$ , ...

```
ArrayList<Cat> myCats  
= new ArrayList<Cat>();
```

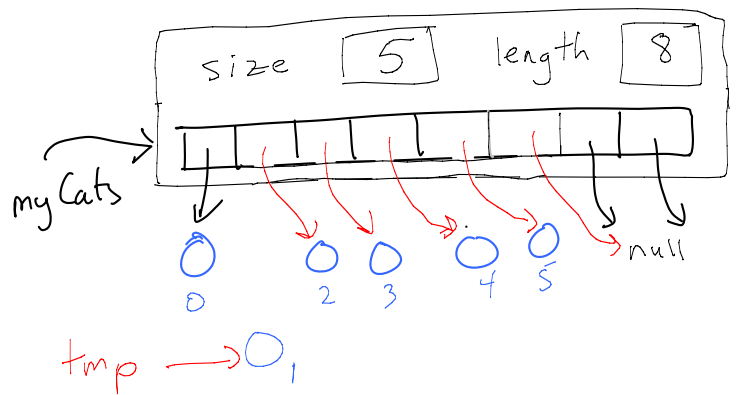
```
myCats.add(new Cat());
```

```
⋮  
⋮  
⋮
```



(These are six different cats)

Cat tmp = myCats.remove(1)



## Array List

- add(i, element)
  - remove(i)
  - clear()
  - add(element) // to end
  - get(i)
  - set(i, element)
  - isEmpty()
  - size()
- } n (size)  
} 1

Careful: arrays

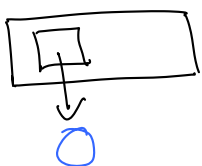
have fixed size.

What if we want to add to a full array?

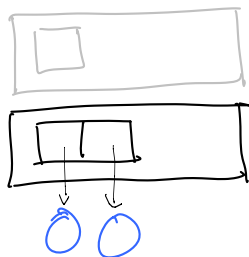
IDEA: start w/ array of length 1, and double length whenever it fills up.

To add a second cat, we need a bigger array

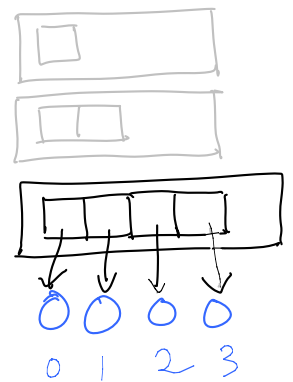
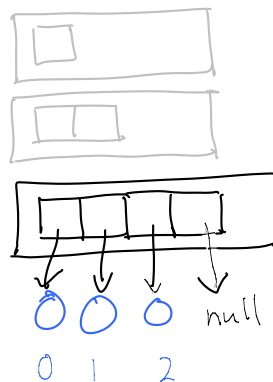
BEFORE



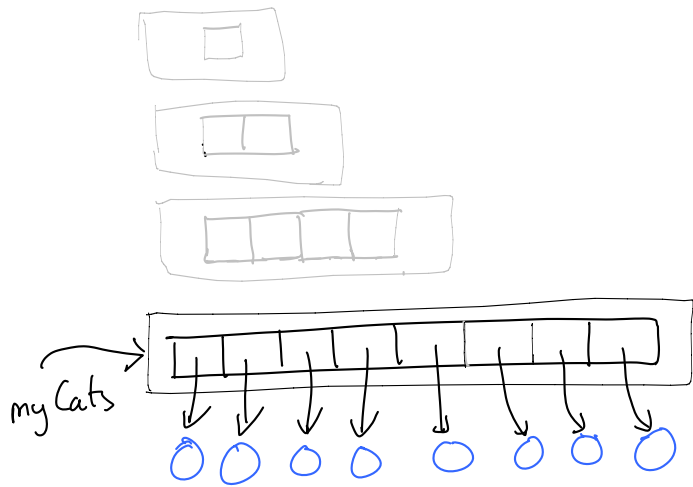
AFTER



Adding another cat, and then another ....



... and another four more ...



How many references must be set to build an ArrayList of size  $n = 8$ ?

- first time a reference is set to each cat (8)
- recopying references when doubling the array size (1 + 2 + 4)

How many references must be set to build an ArrayList of size  $n = 2^k$ ?

- first time a reference is set to each cat ( $n$ )
- recopying references when doubling the array size (1 + 2 + 4 + ... +  $2^{k-1} = n - 1$ )

$$1 + 2 + 4 + \dots + 2^{k-1} = n - 1$$

Why?

$$1 + x + x^2 + \dots + x^{k-1} = \frac{x^k - 1}{x - 1} \quad \text{where } x = 2$$

### ArrayList (Summary)

- get and set are fast
- $\text{add}(i, e)$  and  $\text{remove}(i)$  can take  $n$  steps (worst case)
- $\text{add}(e)$   $n$  times takes  $\sim 2n$  steps

### ASIDE: method "overloading"

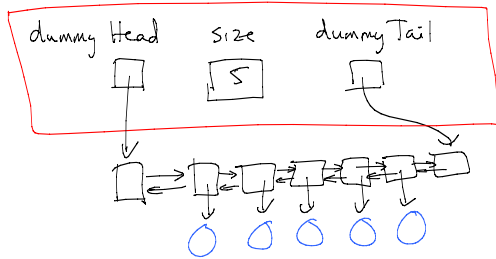
$\left. \begin{array}{l} \text{add}(E) \\ \text{add}(int, E) \end{array} \right\} \text{very different methods!}$

$\left. \begin{array}{l} \text{remove}(E) \\ \text{remove}(int) \end{array} \right\} \text{very different methods!}$

Constructor  $\left\{ \begin{array}{l} \text{Employee}(String) \\ \text{Employee}(String, String) \\ \text{Employee!}(String, int) \end{array} \right.$

# Java Linked List

- implemented with doubly linked list
- Node class is private



```

LinkedList<Cat> myCats
    = new LinkedList<Cat>();

myCats.add(new Cat());

⋮
    
```

Why  $n/2$ ?

Because you can start at either end.

WORST CASE

	<u>LinkedList</u>	<u>ArrayList</u>
- add(i, element)	$n/2$	$n$
- remove(i)	$n/2$	$n$
- clear()	1	1
- add(element) // to end	1	1
- get(i)	$n/2$	1
- set(i)	$n/2$	1
- isEmpty()	1	1
- size()	1	1

```

for (i = 0; i < n; i++) {
    print(get(i))
}
    
```

How many steps?

$$1 + 2 + 3 + 4 + \dots + n$$

$$= \frac{n(n+1)}{2}$$

**Eek!**  
That is slow.

## Abstract Data Type (ADT)

- set of things
  - set of operations that can be performed on them
  - implementation details are NOT given
- e.g. list, stack, queue, ...

## Interface (Java)

- set of methods and arguments (but no method bodies!)
- a class is said to **implement** an interface

e.g. `ArrayList` } implement `List`  
`LinkedList` }