COMP 250
INTRODUCTION TO COMPUTER SCIENCE

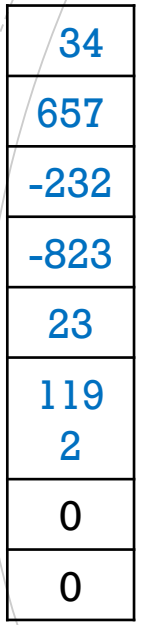
Lecture 13 – Lists I

Roman Sarrazin-Gendron, Winter 2020

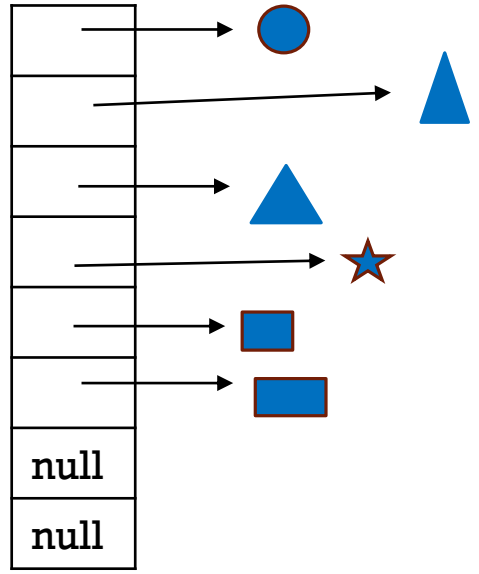
Slides very much based on Michael Langer and Giulia Alberini

RECALL LAST LECTURE: JAVA ARRAY

array of **int**



array
of **Shape** objects



I have drawn each of these as array lists.

JAVA ARRAYLIST CLASS

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

- It uses an array as the underlying data structure
- It grows the array (by 50%, not 100%) when the array is full and a new element is added.
- You don't use the usual array notation a[ ]. Instead, use list-style **get()** and **set()** and other methods.

JAVA GENERIC TYPE

An array of what? ArrayList<T>

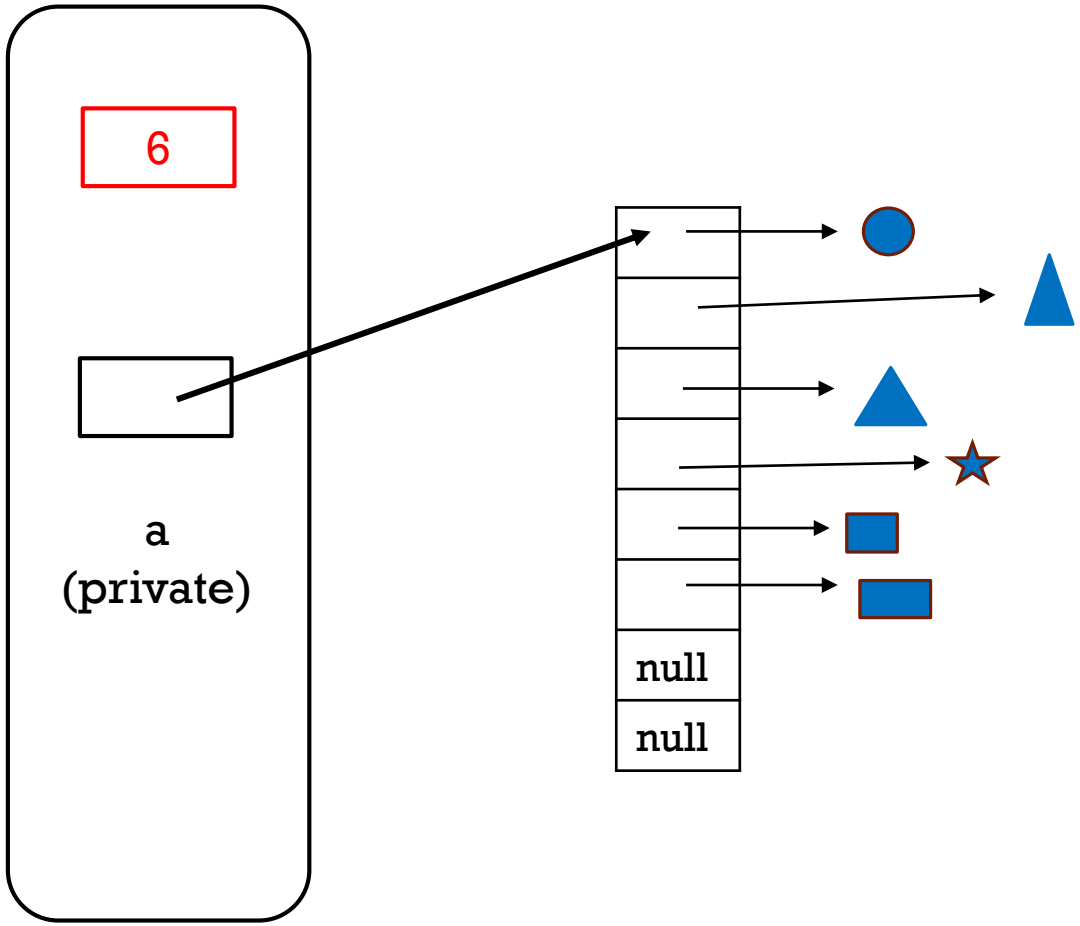
Example:

```
ArrayList< Shape >    shape  =  new ArrayList<  Shape >();
        //  initializes the array length (capacity) to 10
ArrayList<  Shape >    shape  =  new ArrayList<  Shape >( 23 );
        //  initializes the 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.



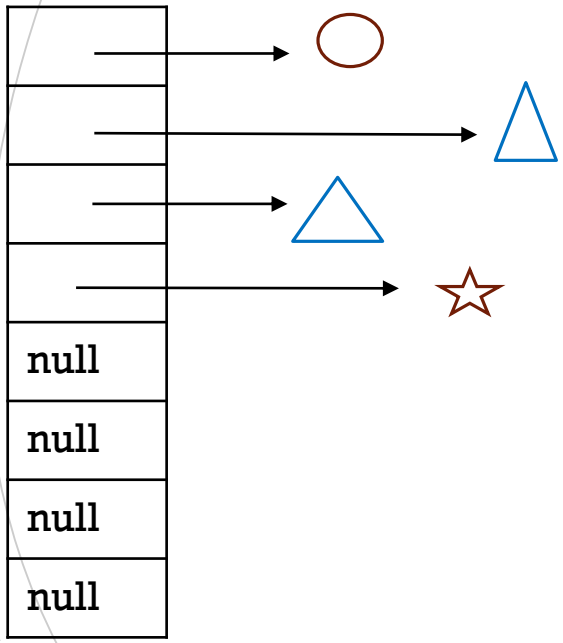
REMINDER : LAST LECTURE

- Monday : array lists
  - Resizable array "disguised" as a list

- What's the downside of array lists?
- Can we just stop using arrays altogether?
  - Today : Linked List

ARRAY LIST VS LINKED LIST

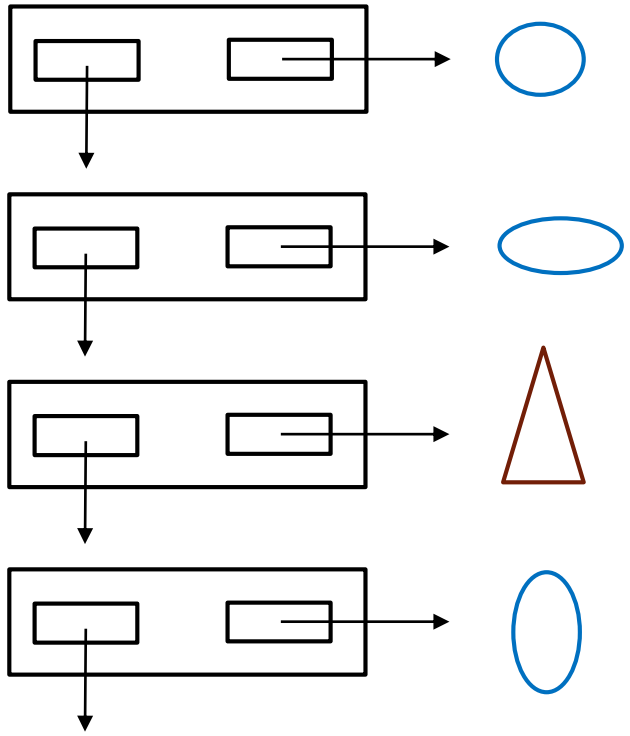
array list



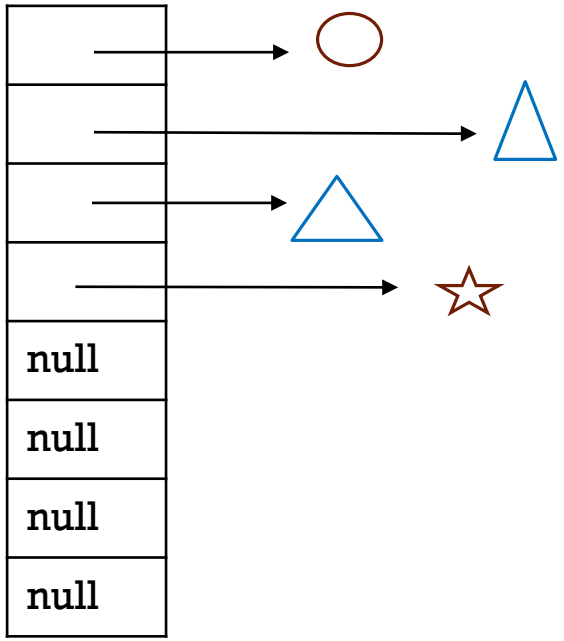
size = 4

linked list

"nodes"

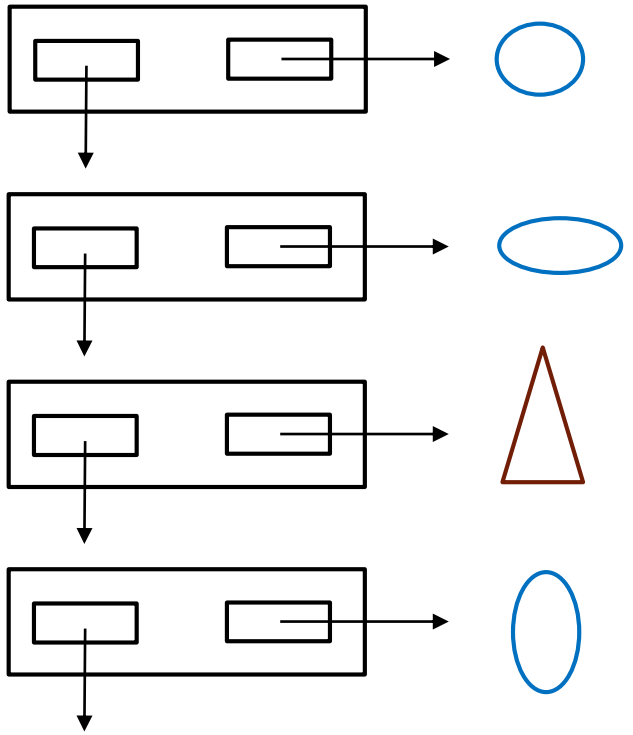


array list



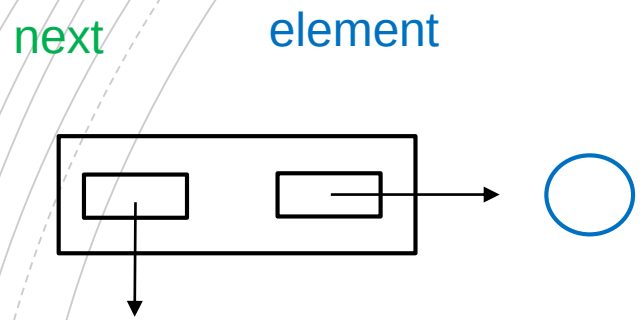
Array slots are in consecutive locations (addresses) in memory, but objects can be anywhere.

linked list



Linked list "nodes" and objects can be anywhere in memory.

SINGLY LINKED LIST NODE ("S" FOR SINGLY)

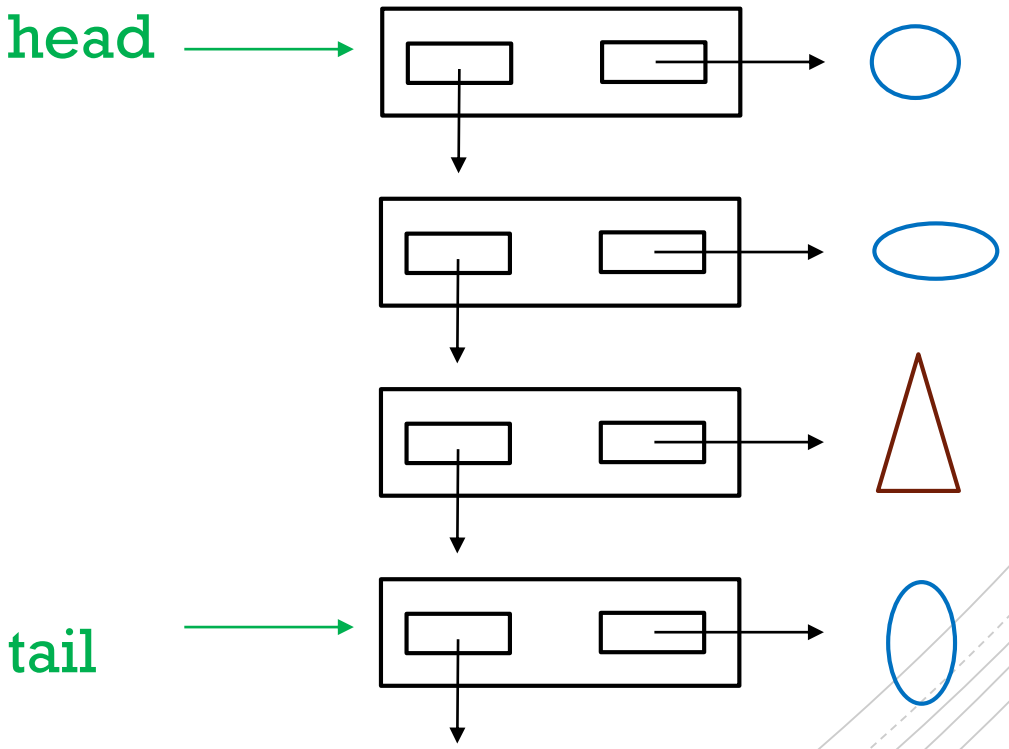



```
class   SNode<E> {

        SNode<E>   next;
        E                element;
}
```

e.g. E might be Shape

A SEQUENCE OF NODES

A linked list consists of a sequence of nodes, along with a reference to the first (head) and last (tail) node.



IN JAVA

- Linked lists are deceptively simple!
- Most elements not directly accessible from the fields of the SLinkedList.
- To access the second element of SLinkedList<E> L:
  - **L.head.next**
- Why is the node class **private**?

```
class   SLinkedList<E> {

        SNode<E>   head;
        SNode<E>   tail;
        int        size;

         private   class   SNode<E> {

             SNode<E>   next;
             E          element;
         }
}
```

SOME LIST OPERATIONS

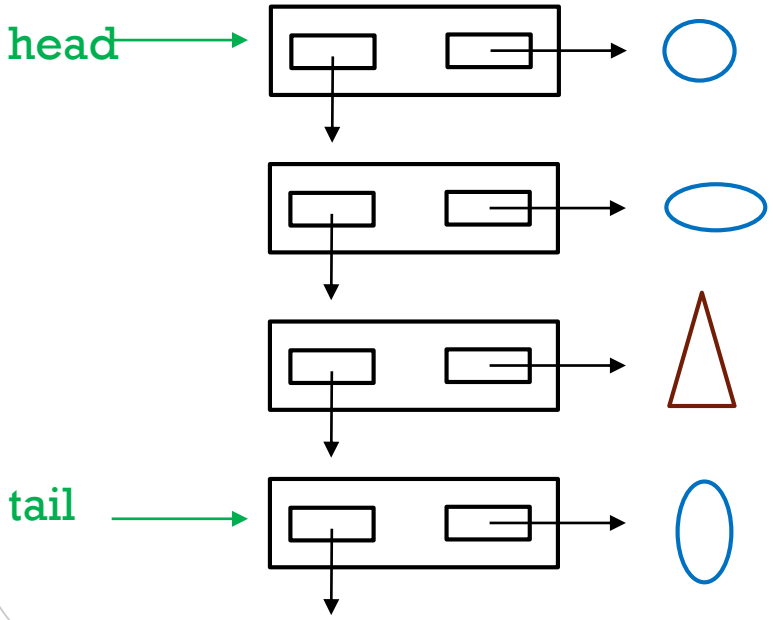
- addFirst (e)
- removeFirst( )
- addLast (e)
- removeLast( )

- ……. many other list operations

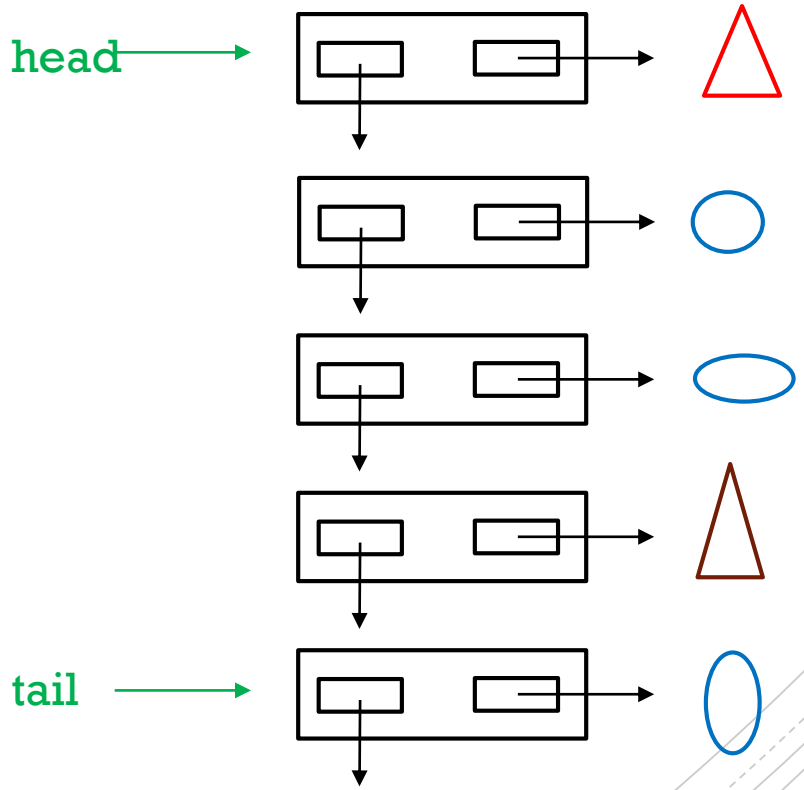
ADD TO FIRST POSITION

addFirst ()

BEFORE



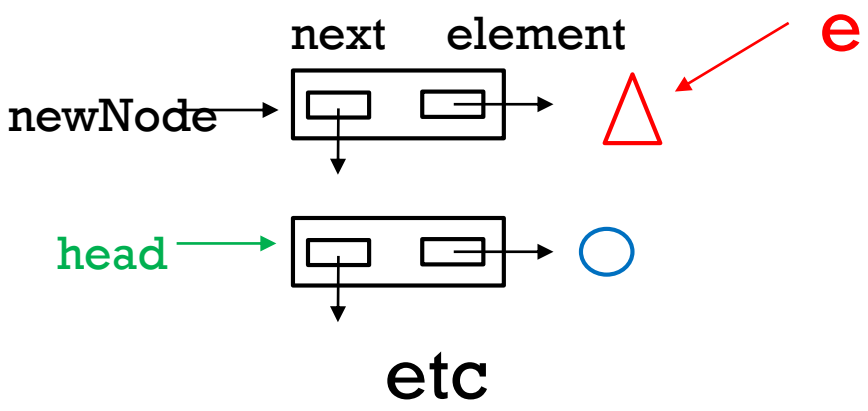
AFTER



PSEUDOCODE FOR ADDFIRST

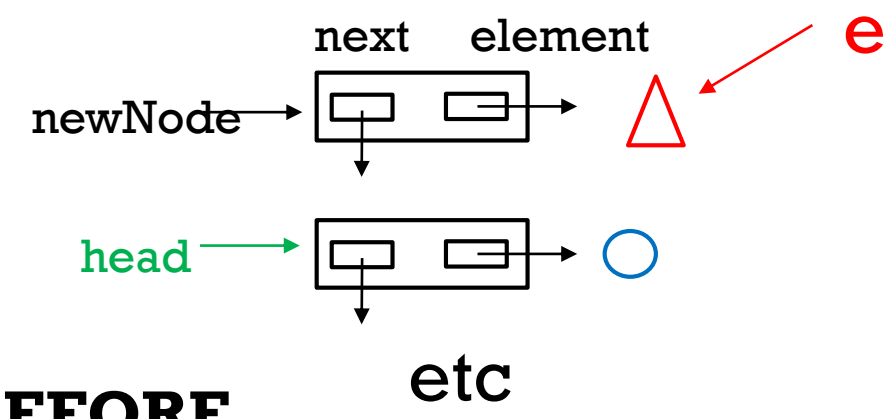
addFirst (e)

```
construct newNode
newNode.element = e
newNode.next    = head
```



addFirst (e) pseudocode

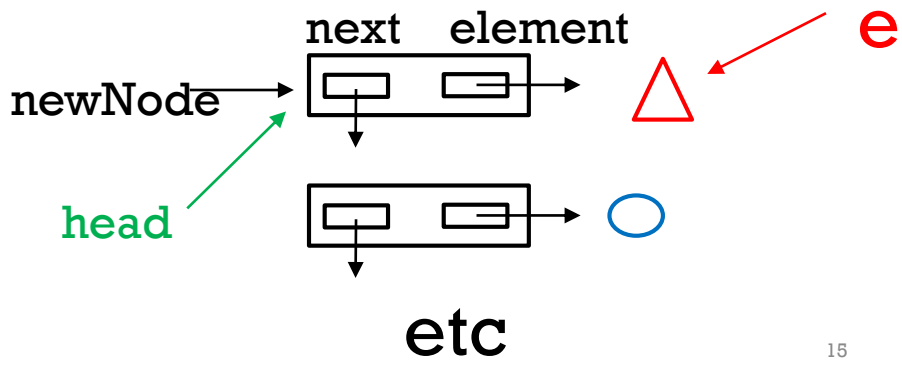
```
construct newNode
newNode.element = e
newNode.next    = head

//  edge case if list is empty
if head == null
    tail = newNode
```



BEFORE

AFTER

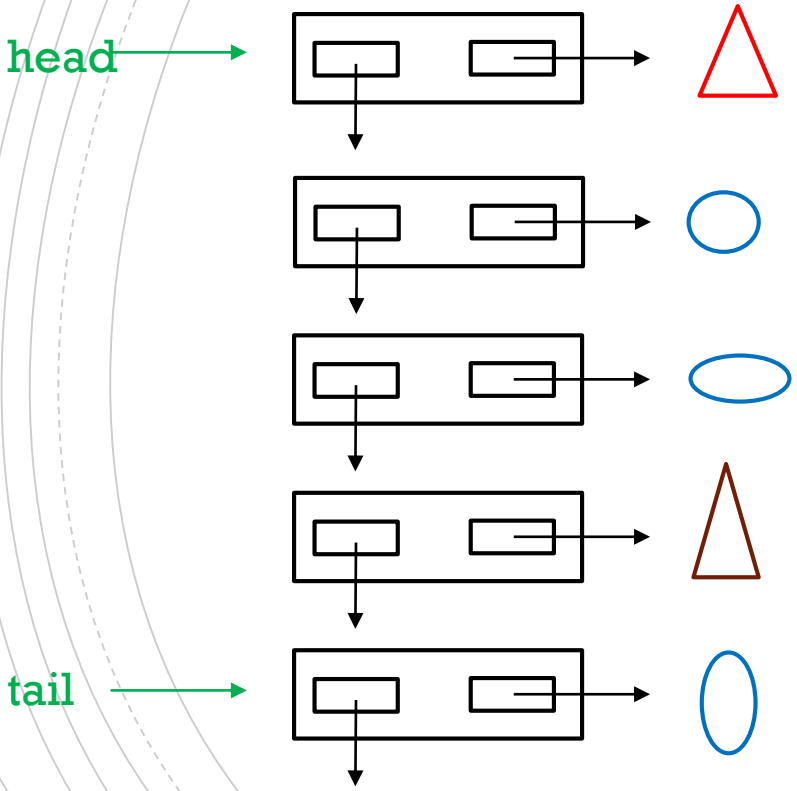
```
head =  newNode
size = size+1
```



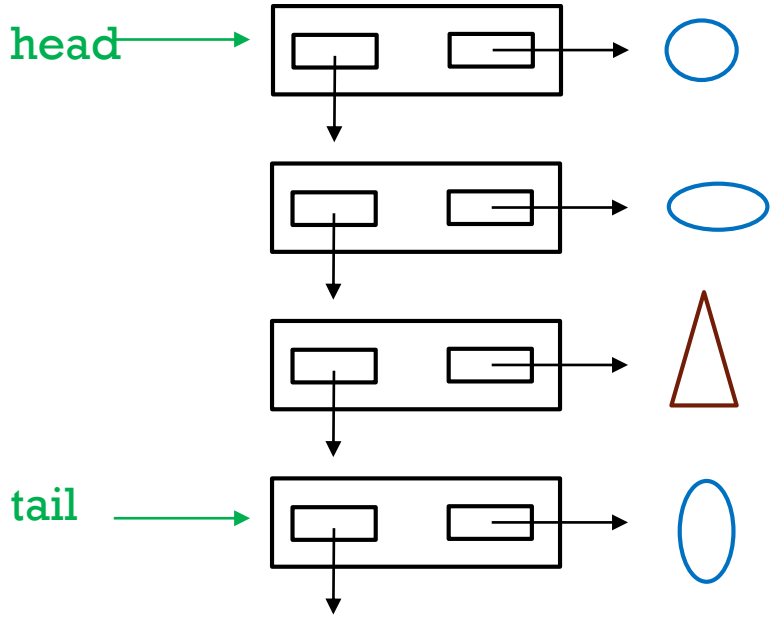
REMOVING THE FIRST ELEMENT

removeFirst ( )

BEFORE

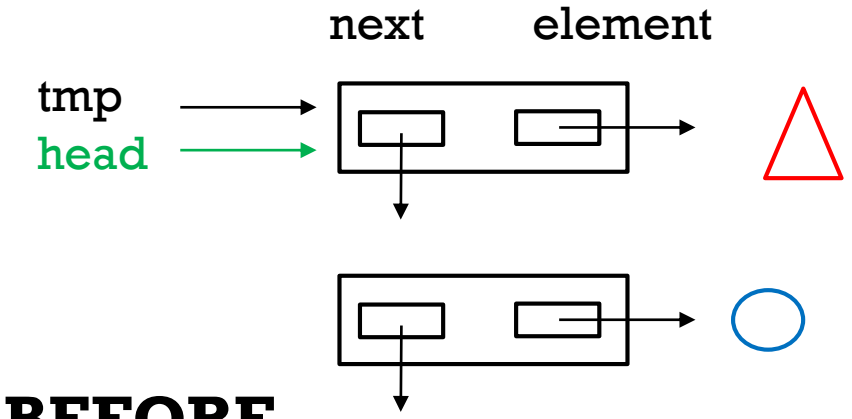


AFTER



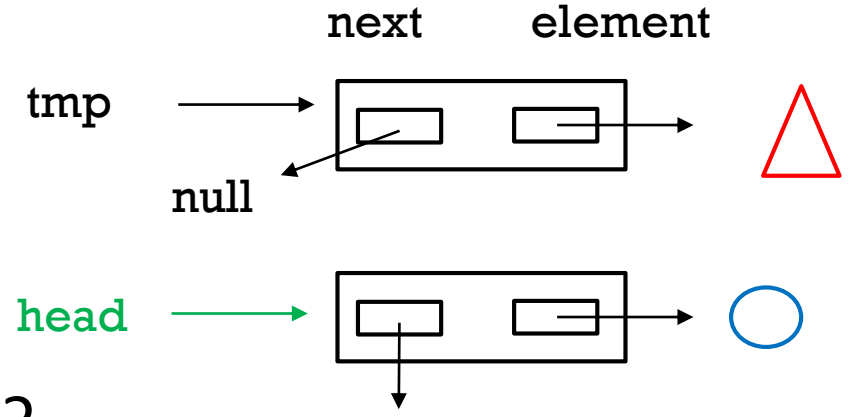
REMOVEFIRST PSEUDOCODE

```
tmp  =  head
head    =  head.next
tmp.next =  null
size = size – 1
```



BEFORE

AFTER

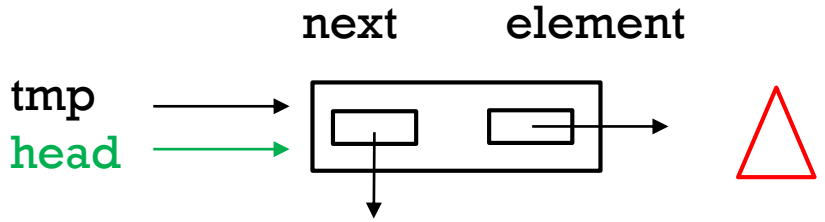


Why are we assigning the initial head to tmp?

Also, what happens if the list is empty?

removeFirst() edge cases (size is 0 or 1)

tmp = head

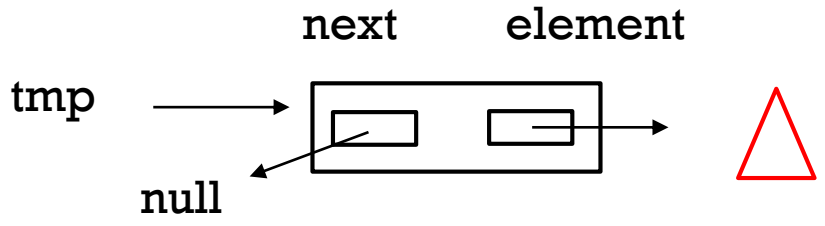


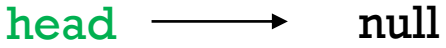
if (size == 0)
  throw exception
head = head.next
tmp.next = null
size = size – 1

if (size == 0) // size was 1
  tail = null

BEFORE

AFTER





COMPLEXITY

- Q: Are addFirst() and removeFirst() more **efficient** with an **array list** or with **a linked list** ?

- **What does it mean to be efficient?**
  - **Time complexity**: amount of time or, by extension, number of steps required to complete the execution of an algorithm.

TIME COMPLEXITY IN THE WORST CASE

	array list	linked list
addFirst	increases with size	constant
removeFirst	increases with size	constant

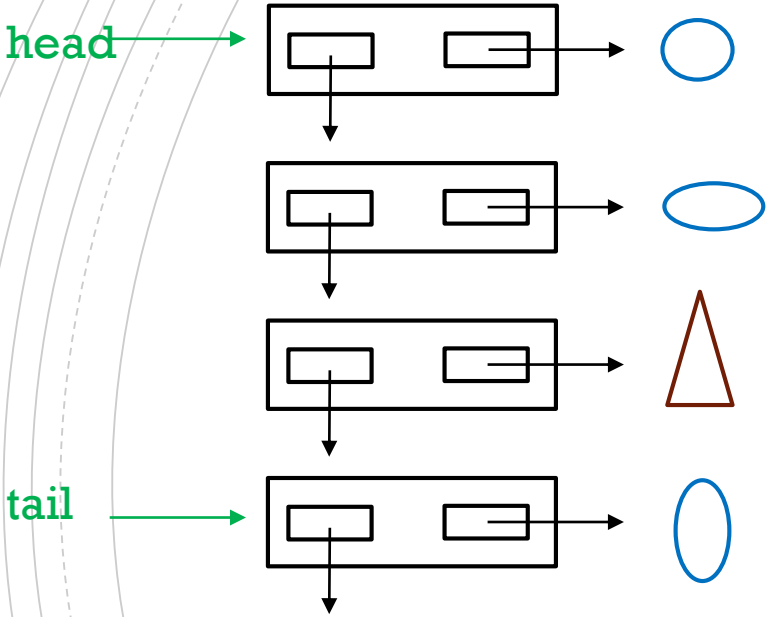
WHAT DETERMINES THE TIME COMPLEXITY?

	array list	linked list
addFirst	size	constant
removeFirst	size	constant
addLast	constant*	?
removeLast	constant	?

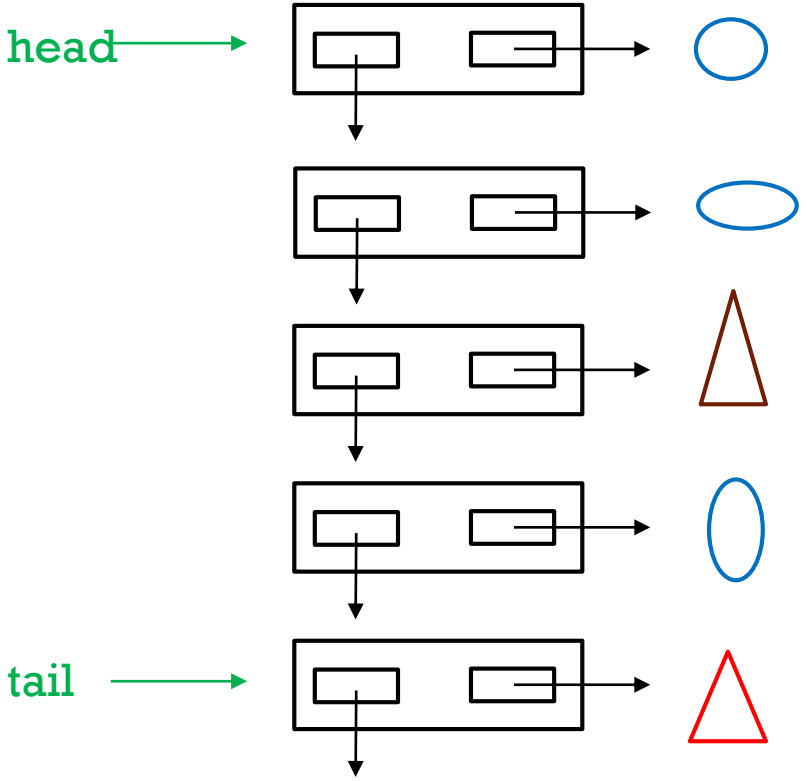
*if array is not full

APPENDING TO A LIST

BEFORE



AFTER

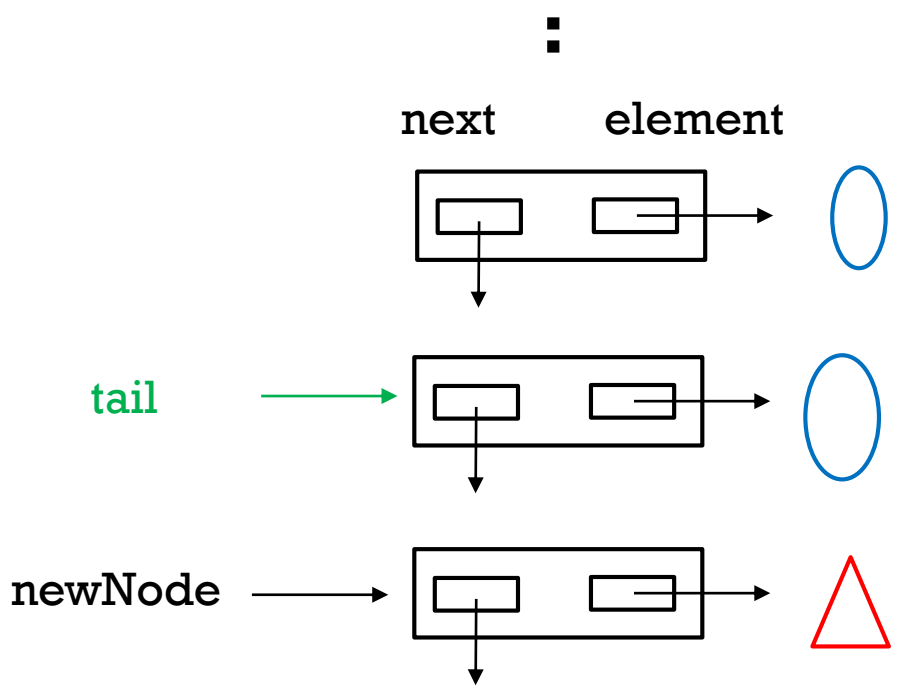


APPENDING TO A LIST

addLast (△)

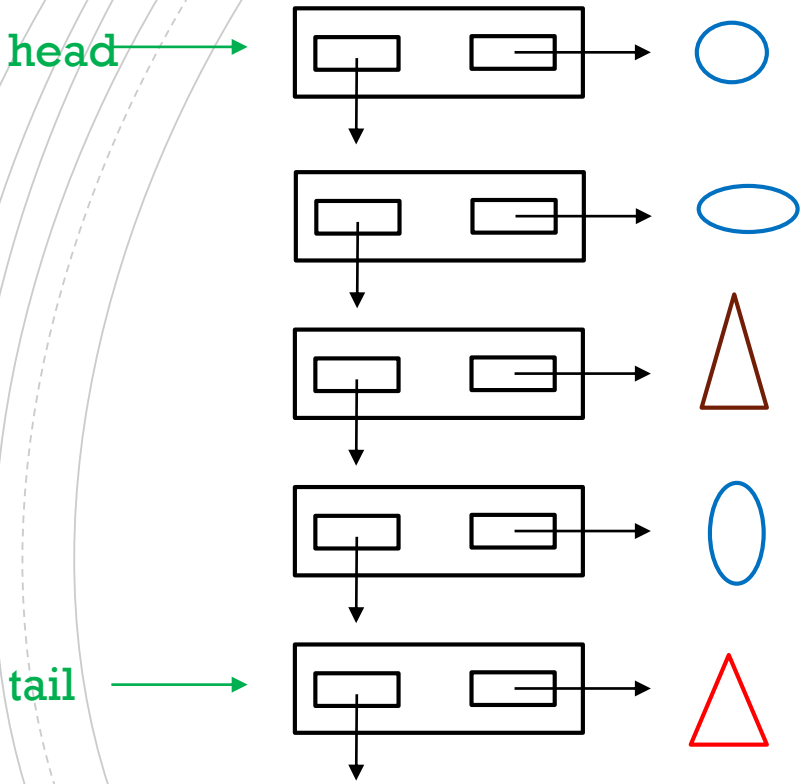
newNode = construct a new node
newNode.element = the new list element
tail.next = newNode

tail = tail.next
size = size+1

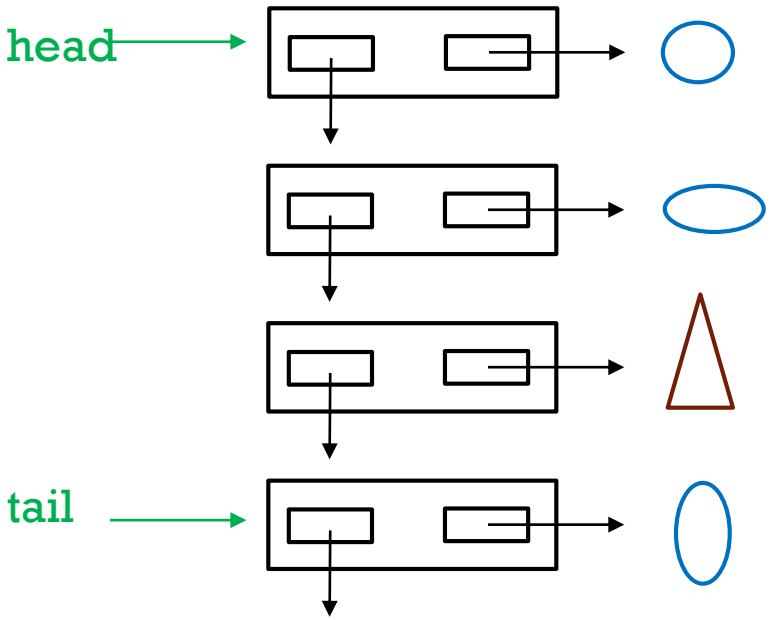


REMOVING THE LAST ELEMENT

BEFORE



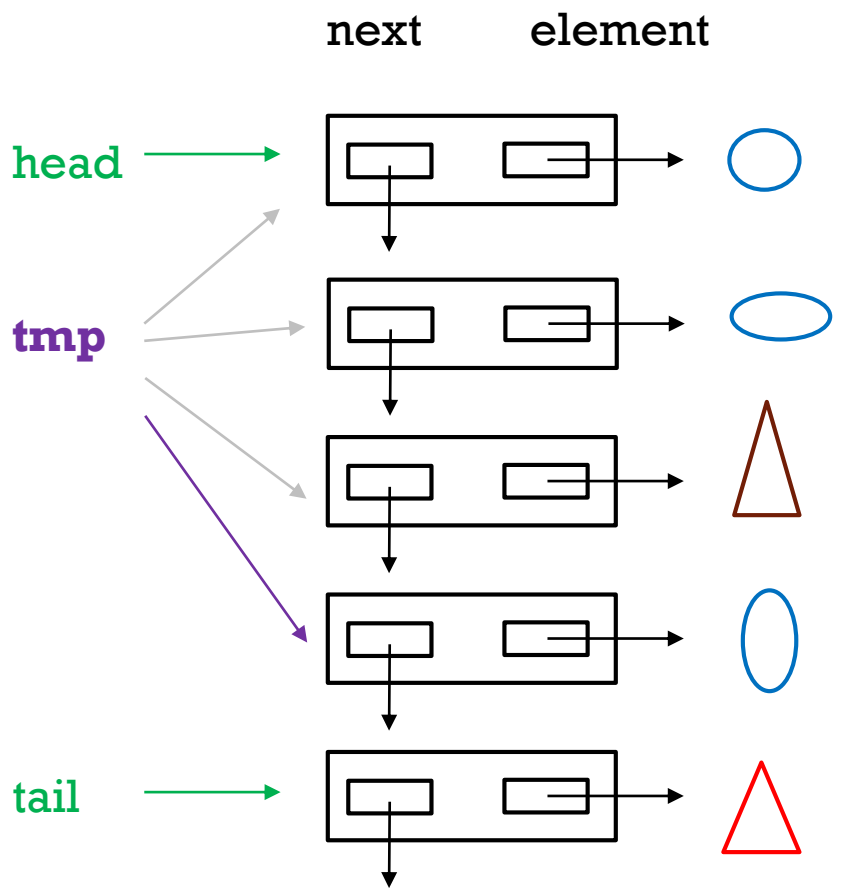
AFTER



Problem: we have no *direct* way to access the node before tail.

GETTING TO THE SECOND TO LAST ELEMENT

```
if  (head == tail){
    head = null
    tail    = null
}
else {

   tmp  = head
   while (tmp.next  != tail)
         tmp = tmp.next

   tail = tmp
   tail.next = null
}
size = size – 1
```



WHAT DETERMINES THE TIME COMPLEXITY?

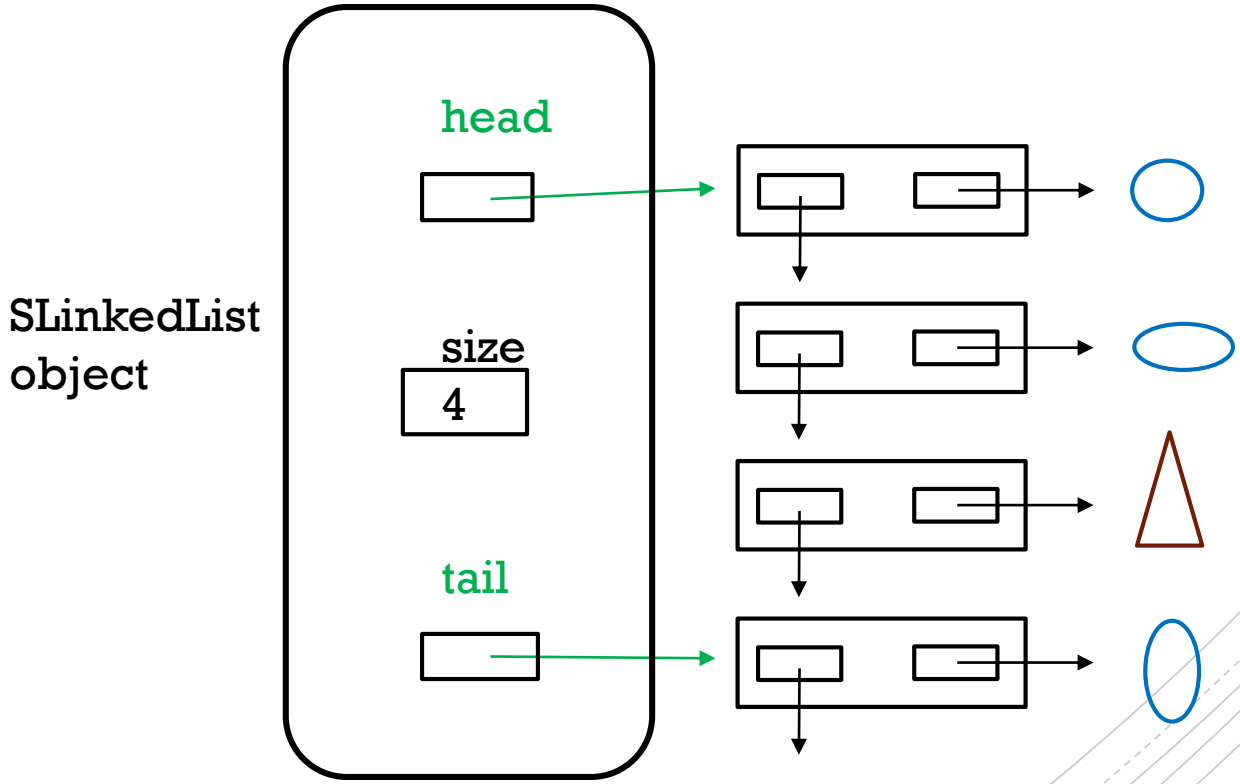
	array list	linked list
addFirst	size	constant
removeFirst	size	constant
addLast	constant*	constant
removeLast	constant	size

*If not full, otherwise size

Getting to **tail** is instant. Getting to **tail-1** depends on size

CONCLUSION : SINGLY LINKED LIST CLASS

```
class   SLinkedList<E> {

        SNode<E>   head;
        SNode<E>   tail;
        int              size;

         :     //   various methods

         private   class   SNode<E> {

                 SNode<E>   next;
                 E                element;
                  :
         }
}
```



CONCLUSION : SINGLY LINKED LIST

- Singly linked lists are good at everything except removing nodes near the end of the list

- Other downside: more expensive in **space** than ArrayLists
- Say we want to store a list of 4 Shape objects. Let's count the number of objects we have in total.
- Array List: the 4 elements of the list, the ArrayList object, and the underlying array object.
- Linked List: the LinkedList object, the 4 elements, and the 4 nodes.

- So in the ideal case, it can still be worth using Array Lists compared to Linked Lists.
- But linked lists are very fast and we often care more about speed than memory.

ANNOUNCEMENTS

- Reminder: the assignment is due Sunday(!), 11:59 PM.
- Submitting one minute late is like submitting 23 hours late.
- We will give you feedback on your submission, via mycourses, and via email.
- **It is your responsibility to monitor your mcgill email around assignment deadlines.**

- Take advantage of office hours this week!