Last time's big ideas

1. When we want an array of objects, we store their *references* in the array

2. It is important to distinguish between the specification and implementation of a class

3. **public** and **private** control access to fields and methods

minute paper: Pick 2 of the 3 big ideas. In 1-2 sentences tell how the two ideas you picked are related to each other.

Specification of the PatientDatabase class

```
class PatientDatabase {
// Register a new Patient in the database
// return false if out of space
  boolean registerNewPatient(String name) { ... }

// Print all patient names in alphabetical order
  void printNamesAlphabetically() { ... }

  public static void main(String[] args) {
    PatientDatabase db = new PatientDatabase(100);
    db.registerNewPatient("Ron");
    db.registerNewPatient("Hermoine");
    db.registerNewPatient("Snape");
    db.registerNewPatient("Harry");
    db.printNamesAlphabetically();
 }
}
```

What a PatientDatabase needs to be able to do

An example of using a PatientDatabase

```
// Register a new Patient in the database
boolean registerNewPatient(String name) { ... }
```

algorithm: insert new element at the the end, then swap until it is in the right place

Harry	Ron	Snape			

```
registerNewPatient("Hermione")
```

Harry	Ron	Snape	Hermione		

Harry	Ron	Hermione	Snape		

Harry	Hermione	Ron	Snape		

```java
// Register a new Patient in the database (if we have space)
boolean registerNewPatient(String name) {
  if (numPatients == patients.length) return false;

  // since they haven't been measured we will give height=0
  Patient newp = new Patient(name, 0);

  // start with the new patient at the end of the list
  patients[numPatients] = newp;

  numPatients+=1;

  // keep swapping the patient with the previous patient
  // until it is in alphabetical order
  int i = numPatients-1;
  while (i > 0 &&
         patients[i].name.compareTo(patients[i-1].name) < 0) {

     swapPatients(i, i-1);
     i--;
   }

   return true;
}
```

compareTo returns -1 if first < second, 0 if equal, 1 if first > second
"Alligator".compareTo("Bobcat") → -1
"Bobcat".compareTo("Alligator") → 1

we'll get to `swapPatients()` next

let’s finish the PatientDatabase

major point here:

- **public** and **private** control access to fields and methods

```java
class PatientDatabase {
  private Patient[] patients;


  private void swapPatients(int a, int b) {
    Patient pa = patients[a];
    Patient pb = patients[b];
    patients[a] = pb;
    patients[b] = pa;
  }


  boolean registerNewPatient(String name) {

    while ( ) {

      swapPatients(i, i-1);

    }
  }
}
```

```java
class PatientDatabase {
  private Patient[] patients;

  private void swapPatients(int a, int b) {
    Patient pa = patients[a];
    Patient pb = patients[b];
    patients[a] = pb;
    patients[b] = pa;
  }


  boolean registerNewPatient(String name) {

    while ( ) {

      swapPatients(i, i-1);

    }
  }
}
```

`swapPatients()` wouldn't make sense to outsiders!

- it is just an *implementation detail* used by `registerNewPatient()`
- `PatientDatabase` could have been implemented with something other than an array sorted by names

Peer instruction

https://b.socrative.com/login/student/
CS2230X – 1000-2999
CS2230Y – 3000-5999

Making `patients` and `swapPatients` private is *most* an example of which object-oriented design principle?

A) Abstraction
B) Encapsulation
C) Modularity

(page 61 of GTG)

```
class PatientDatabase {
  private Patient[] patients;


  private void swapPatients(int a, int b) {

  }


  boolean registerNewPatient(String name) {

    while ( ) {

      swapPatients(i, i-1);

    }
  }
}
```

Specification of the PatientDatabase class

```
class PatientDatabase {
// Register a new Patient in the database
// return false if out of space
  boolean registerNewPatient(String name) { ... }

// Print all patient names in alphabetical order
  void printNamesAlphabetically() { ... }

  public static void main(String[] args) {
    PatientDatabase db = new PatientDatabase(100);
    db.registerNewPatient("Ron");
    db.registerNewPatient("Hermoine");
    db.registerNewPatient("Snape");
    db.registerNewPatient("Harry");
    db.printNamesAlphabetically();
 }
}
```

What a PatientDatabase needs to be able to do

An example of using a PatientDatabase

Implementation of the other method

```
// Print all patient names in alphabetical order
void printNamesAlphabetically() {
  for (int i=0; i<numPatients; i++) {
    System.out.println(patients[i].name);
  }
}
```

(see the whole PatientDatabase class in PatientDatabase.java, and run the program for yourself)

https://b.socrative.com/login/student/
room CS2230X – ID 1000-2999
room CS2230Y – ID 3000-5999

Peer instruction

Why is it important to distinguish between the *specification* and the *implementation* of a class? (short answer)

CS 2230
CS II: Data structures

Meeting 5: more references(pointers), linked lists

Brandon Myers

University of Iowa

Today’s big ideas 1

- practice drawing boxes-and-arrows to describe reference manipulations
- we can build a *list* out of ”ListNodes” linked together by references (a **linked list**)
- references that don’t point to anything store the value **null**

https://b.socrative.com/login/student/
CS2230X – 1000-2999
CS2230Y – 3000-5999

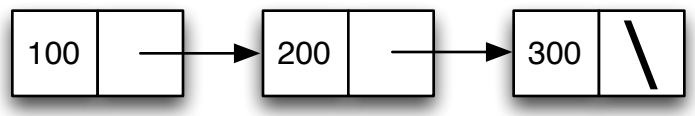
What are some cons of Arrays?

(i.e., downsides, as in pros and cons)

New data structure: *linked list*

[100,200,300]

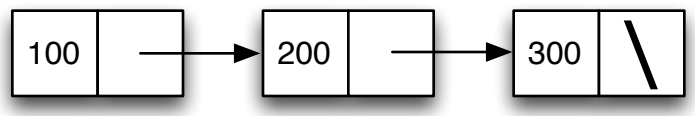
Let's implement the python list like this:



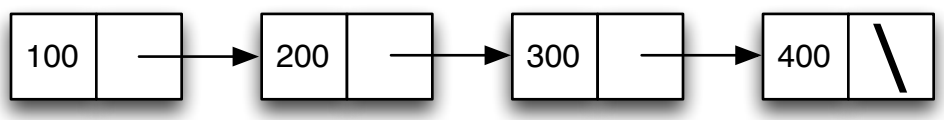
New data structure: *linked list*

[100,200,300]

Let's implement the python list like this:



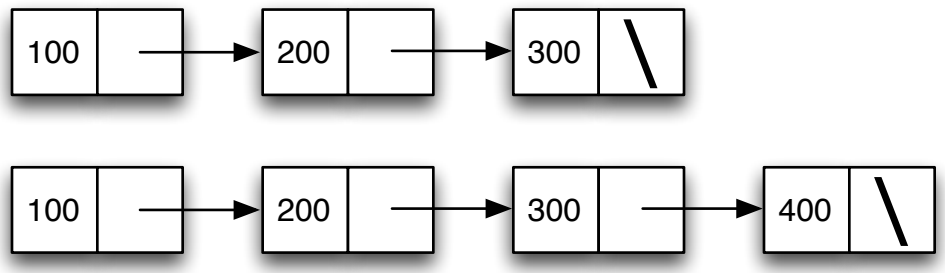

```
hundreds = [100,200,300]
hundreds += [400]
```



solves the out-of-space problem that arrays have

just add a new node to the end

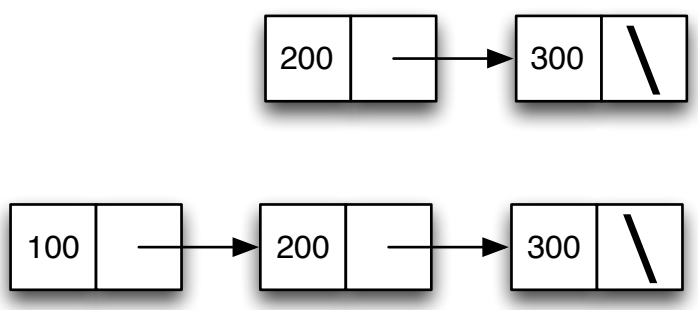
```
hundreds = [100,200,300]
hundreds += [400]
```



solves the out-of-space problem that arrays have

just add a new node to the end

```
hundreds = [200,300]
alphabet.insert(0, 100)
```



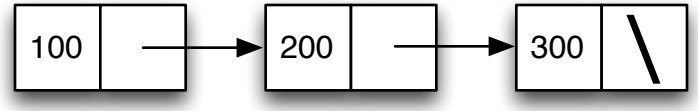
solves the insert at the front problem that arrays have

just add a new node to the front

Peer instruction

https://b.socrative.com/login/student/
CS2230X – 1000-2999
CS2230Y – 3000-5999

example linked list



We are going to make a linked list from *ListNode* objects.


```
1
2   public class ListNode {
3       private int          data;
4       private [redacted]   next;
5   }
```

The type for *data* is *int*. What should be the type for *next*?

a) int[]
b) int
c) double
d) ListNode[]
e) ListNode

Let’s play with references

convention is to denote null as a slash \

```
ListNode p1 = new ListNode(100);
```

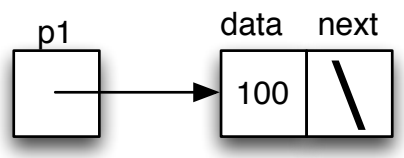
```
ListNode p2 = new ListNode(200);
```

```
p2 = p1;
```

```
p2.next = new ListNode(300);
```

Let’s play with references

convention is to denote null as a slash \




```
ListNode p1 = new ListNode(100);
```

```
ListNode p2 = new ListNode(200);
```

```
p2 = p1;
```

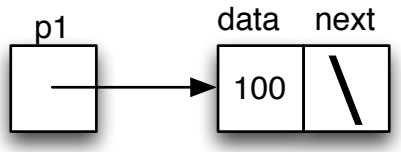
```
p2.next = new ListNode(300);
```

Draw the boxes-and-arrows diagram after each line of code finishes.

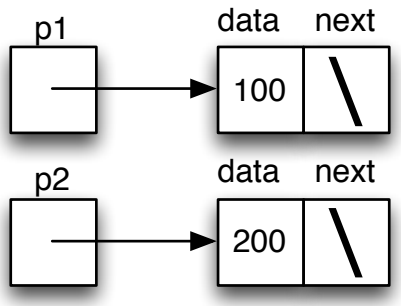
Let’s play with references

convention is to denote null as a slash \

```
ListNode p1 = new ListNode(100);
```




```
ListNode p2 = new ListNode(200);
```



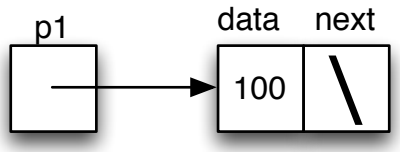

```
p2 = p1;
```

```
p2.next = new ListNode(300);
```

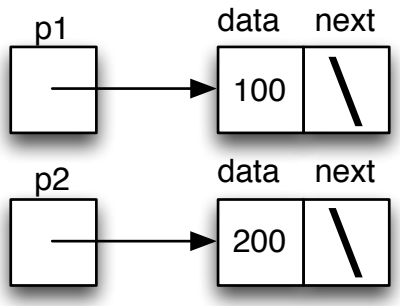
Let’s play with references

convention is to denote null as a slash \

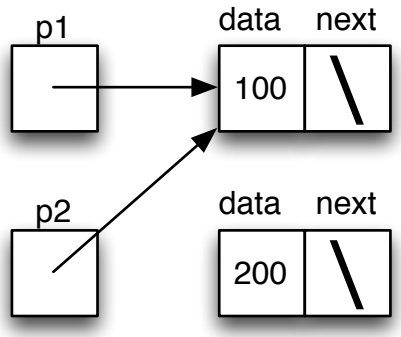
```
ListNode p1 = new ListNode(100);
```




```
ListNode p2 = new ListNode(200);
```




```
p2 = p1;
```

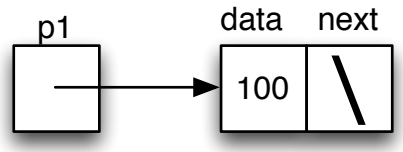



```
p2.next = new ListNode(300);
```

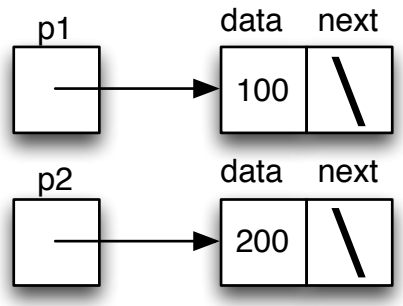
Let's play with references

convention is to denote null as a slash \

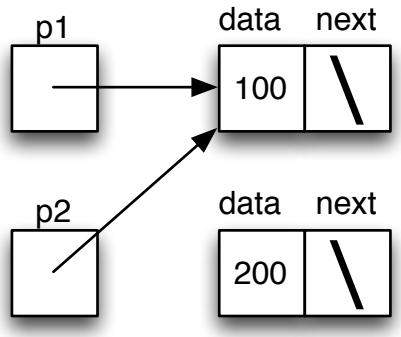
```
ListNode p1 = new ListNode(100);
```



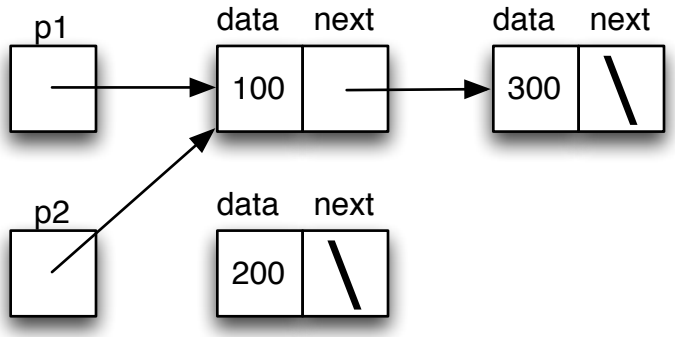

```
ListNode p2 = new ListNode(200);
```




```
p2 = p1;
```

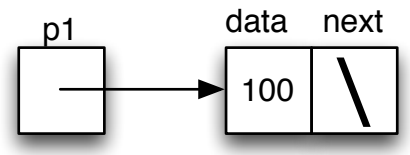



```
p2.next = new ListNode(300);
```



More references practice

```
ListNode p1 = new ListNode(100);
```




```
p1.next = new ListNode(40);
```

```
p1.next = p1.next.next;
```

Draw the boxes-and-arrows diagram after each line of code finishes.

https://b.socrative.com/login/student/
CS2230X – 1000-2999
CS2230Y – 3000-5999

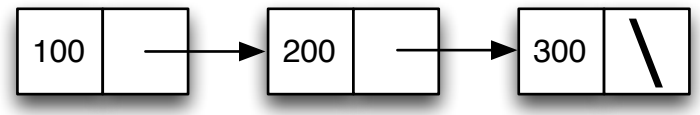
Peer instruction

How do you know if a ListNode is the last one in the list?

a) it has no `next` field
b) its `next` field points to itself
c) its `next` field points to the beginning of the list
d) its `next` field is null
e) its `next` field and `data` field are equal

The *append* method

example linked list




```
1
2   public class ListNode {
        private int         data;
4       private ListNode    next;
5
6       public ListNode(int d) {
7           data = d;
8           next = null;
9       }
10
11      /*
12      Add the new integer to the end of the list
13      */
14      public void append(int d) {
15          if (next == null) {
16              next = new ListNode(d);
17          } else {
18              next.append(d);
19          }
20      }
```

check if this is the last ListNode

create a new ListNode to hold the integer

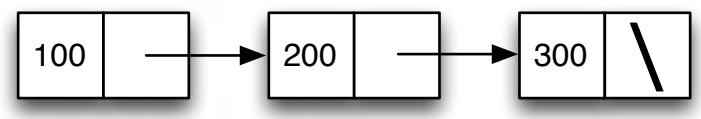
if there is another ListNode following this one, then append to that one

The *append* method

https://b.socrative.com/login/student/
CS2230X – 1000-2999
CS2230Y – 3000-5999

```
1
2   public class ListNode {
        private int         data;
4       private ListNode    next;
5
11      /*
12      Add the new integer to the end of the list
13      */
14      public void append(int d) {
15          if (next == null) {
16              next = new ListNode(d);
17          } else {
18              next.append(d);
19          }
20      }
```

example linked list

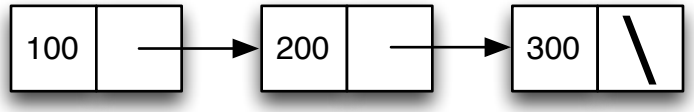


How does the append method traverse (i.e. walk node to node) the linked list?

a) line 18: the Java keyword “next” takes us to the following node in a linked list
b) line 18: by calling append again, it will affect a different ListNode than before
c) line 18: calling append on a different value of d
d) line 18: next looks at the reference to the following ListNode, the dot follows the reference to the actual ListNode object, then we call append on it
e) line 16: assigning next to a new ListNode brings us to the following ListNode

Iterative (for-loop) implementation of append

example linked list

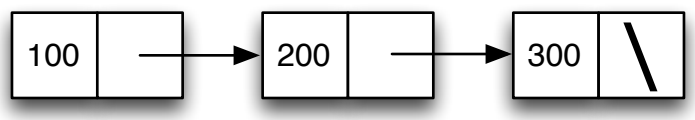



```
// where is the mistake?
public void append(int d) {
    ListNode current = this;
    while (current != null) {
        current = current.next;
    }
    current.next = new ListNode(d);
}
```

Method to get length of the list

```
30        /*
31        Return the number of nodes in this list
32        */
33        public int length() {
```

What should be the algorithm for our implementation of length()?



length() → 3

https://b.socrative.com/login/student/
CS2230X – 1000-2999
CS2230Y – 3000-5999

If it takes 1ms to find the length of a list length 10, how long for a list of size 10,000?

```
/*
Return the number of nodes in this list
*/
public int length() {
    if (next==null) { return 1; }
    else return 1 + next.length();
}
```

a) 1ms
b) 1,000ms
c) 2,000ms
d) 10,000ms
e) 20,000ms

Today’s big ideas 1

- practice drawing boxes-and-arrows to describe reference manipulations
- we can build a *list* out of ”ListNodes” linked together by references (a **linked list**)
- references that don’t point to anything store the value **null**

What to do now

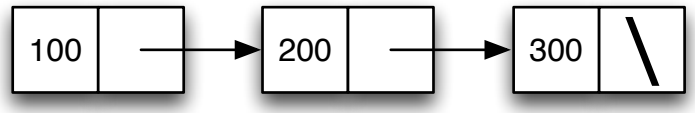
- Finish HW1
- Quiz 2 upcoming
- Pre-lab 2 posted today
- This week debug your brain is today 3pm

- Collect your Change of Registration form Thursday in class

Today’s big ideas 2

- encapsulate ListNodes inside of a LinkedList class so we can try different implementations of a linked list

Some problems with ListNode



- We have to go through the whole list to **append** a new element

```
public void append(int d) {
    if (next == null) {
        next = new ListNode(d);
    } else {
        next.append(d);
    }
}
```

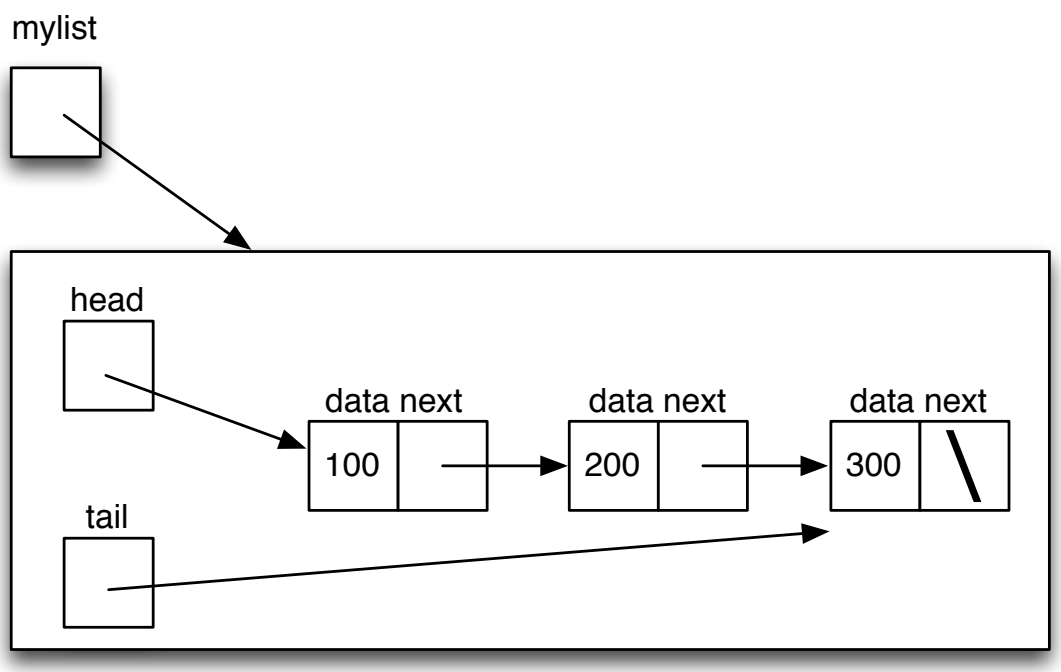
- We have to go through the whole list to get the **length**

```
/*
Return the number of nodes in this list
*/
public int length() {
    if (next==null) { return 1; }
    else return 1 + next.length();
}
```

A new class, LinkedList

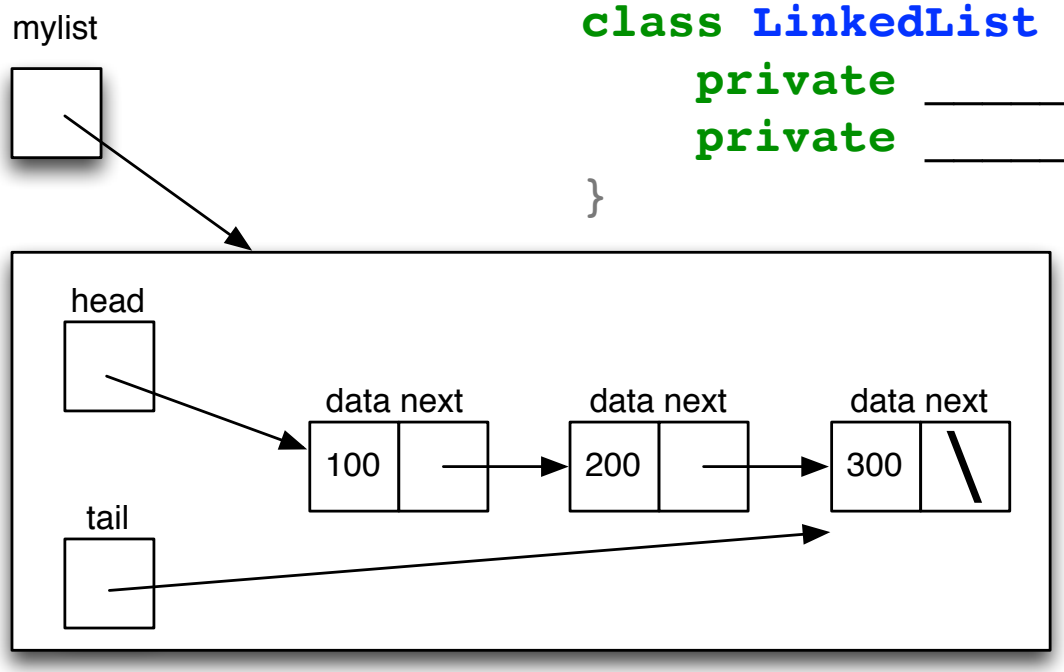
LinkedList uses the `ListNode` class in its *implementation*

Inside LinkedList, we can privately keep a reference to the front (head) **and** the back (tail)




```
LinkedList mylist = new LinkedList();
mylist.append(100); mylist.append(200); mylist.append(300);
```

What should be the type for head and tail?

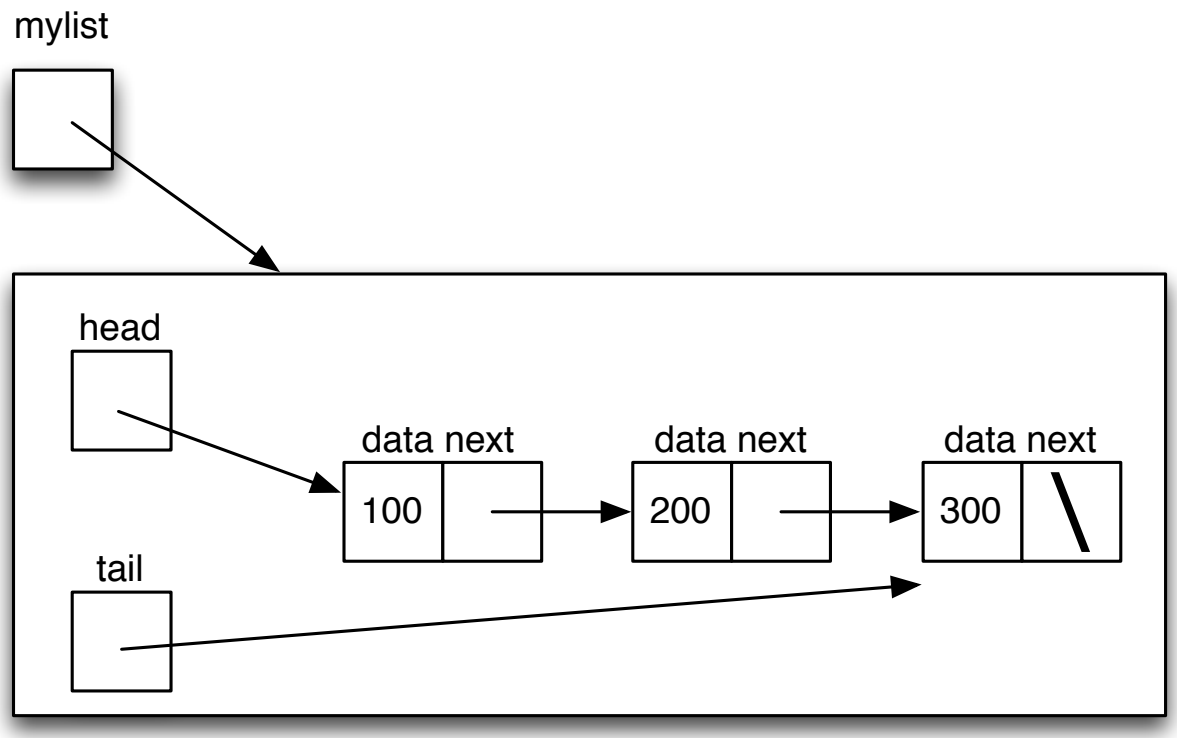



```
class LinkedList {
    private _________ head;
    private _________ tail;
}
```

A) int
B) int[]
C) ListNode
D) ListNode[]
E) LinkedList

https://b.socrative.com/login/student/
CS2230X – 1000-2999
CS2230Y – 3000-5999

New algorithm for append() ?



example usage

1 sentence answer

```
LinkedList mylist = new LinkedList();
...

mylist.append(400);
```

```java
public void append(int d) {
        ListNode n = new ListNode(d);
        tail.next = n;
        tail = n;
    }
```

empty list case

UH OH...

empty list case

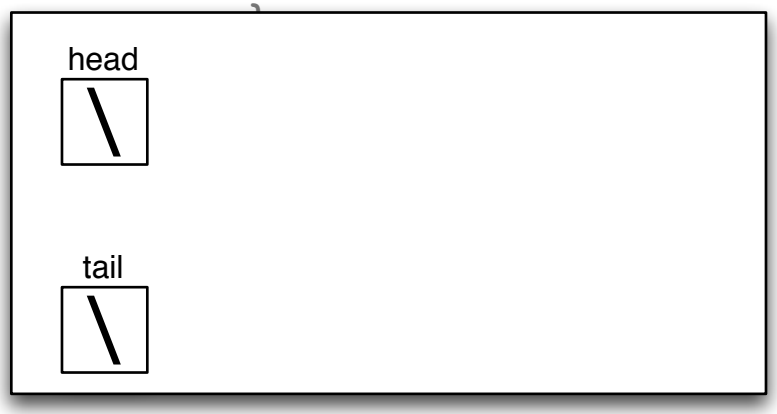
non empty list case

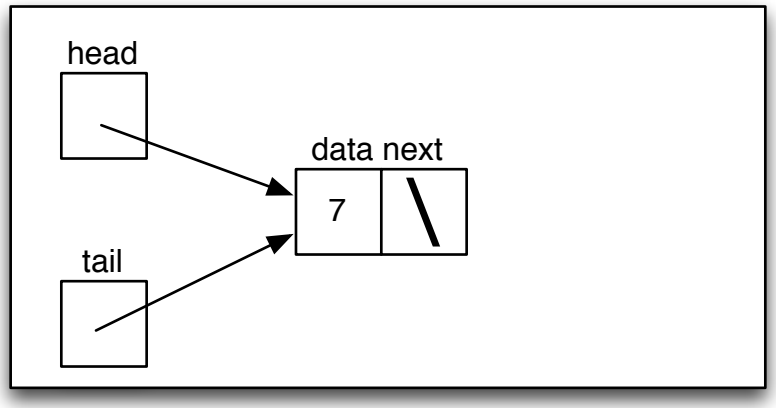
```
public void append(int d) {
  ListNode n = new ListNode(d);
  if (tail == null) {
    // list is empty
    head = n;
    tail = n;
  } else {
    tail.next = n;
    tail = n;
  }
```

BEFORE

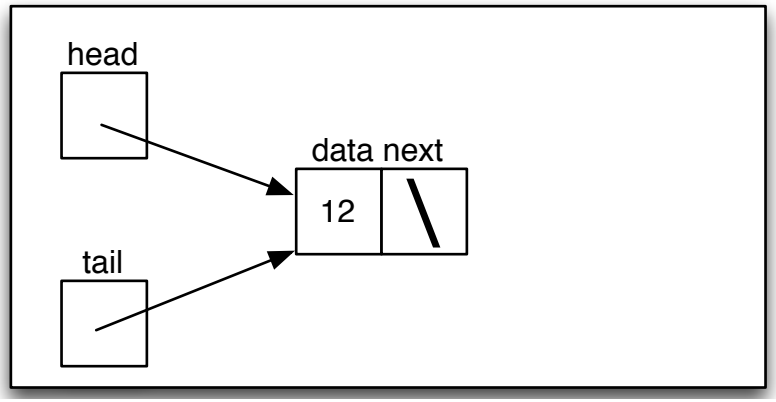
AFTER

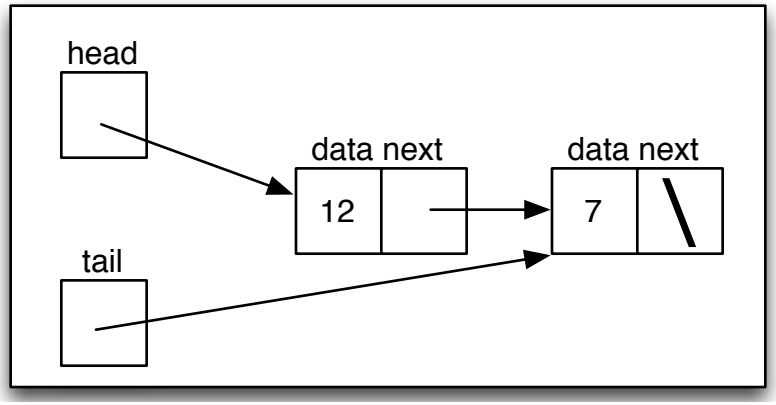
empty list case





non empty list case





Today’s big ideas 2

- encapsulate ListNodes inside of a LinkedList class so we can try different implementations of a linked list