Nodes

Poll

Q: How can we indicate the absence of an Object in Java code?

Poll

Q: How can we indicate the absence of an Object in Java code?

A: Use a NULL reference

Poll

Q: Given the following code, what is printed? (Recall the simple `Point.java` class, with `x` and `y` instance variables & getters/setters)

```
public class UsePoint {
  public static void main(String[] args) {
    Point p = new Point(3, 2);
    Point q = p;
    p.setX(64);
    System.out.println(q.getX());
  }
}
```

Poll

Q: Given the following code, what is printed? (Recall the simple `Point.java` class, with `x` and `y` instance variables & getters/setters)

A: **64**

```
public class UsePoint {
  public static void main(String[] args) {
    Point p = new Point(3, 2);
    Point q = p;
    p.setX(64);
    System.out.println(q.getX());
  }
}
```

Previously: Arrays

Previously, if we ever wanted to store a sequence of data, we used arrays (or ArrayLists, which are just arrays that Java manages in fancy ways for you.)

- Arrays store data in contiguous memory (each element is next to each other in memory)
- We could access a specific position with an index

Example array declaration: `int[] values = {2814, 2048, 867, 5309};`

0	1	2	3
2814	2048	867	5309

Array Flaws

What if we wanted to add a new value to the beginning of an array?

- All we have is a variable `values` that stores a reference to an array object in memory
- That array object is a contiguous portion of "heap" space that takes up a fixed amount of space

0	1	2	3
2814	2048	867	5309

Array Flaws

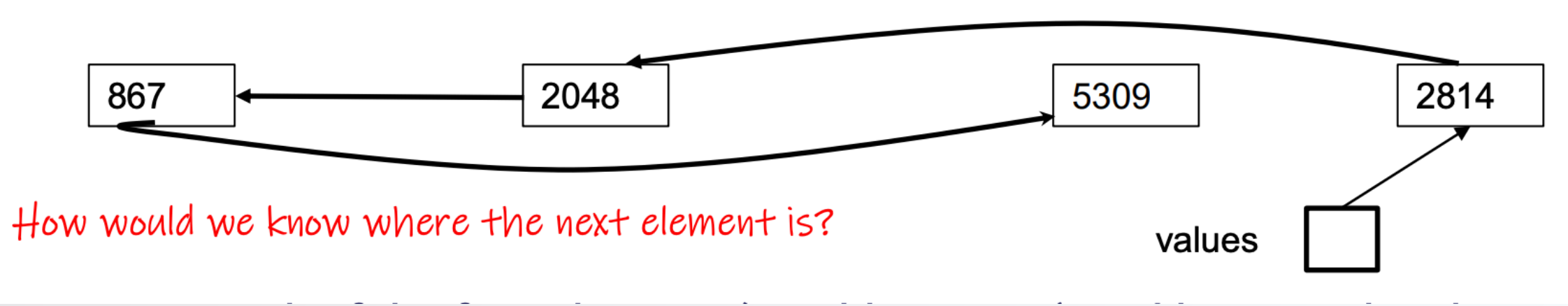
What if we wanted to add a new value to the beginning of an array?

Need to:

- create a new array object (using `new`) with a bigger size
- copy over old values to positions `1`-`4`
- and then insert new value at position `0`

Idea: Non-contiguous Storage

What if we tried to store data in memory that is noncontiguous, where each element is spread apart from one another?



Want to:

- Keep track of the first element (just like arrays)
- have each value store a reference to the "next" value

Introduction: Nodes

Node: a class containing one or more data fields that store data, and a *reference* to another linked node

- connect these objects together to form a structure of linked nodes

Linked nodes are the building blocks of programs (data structures) that store a large amount of data without using an array

- Allow us to more easily modify a collection of data
- Don’t have to worry about knowing the length before hand

Node Class

A node stores a data point and a reference to another node.

```
public class Node {
  public Node next; // Point to next node
  public int data;  // Value (int) for this node
  public Node(int data, Node next) {
    this.data = data;
    this.next = next;
  }
  // data fields are public, no need for getters and setters
}
```

Another Node Class

A node stores a data point and a reference to another node.

```
public class Node {
  public Node next;     // Point to next node
  public Computer data; // Value (Computer) for this node
  public Node(Computer data, Node next) {
    this.data = data;
    this.next = next;
  }
}
```

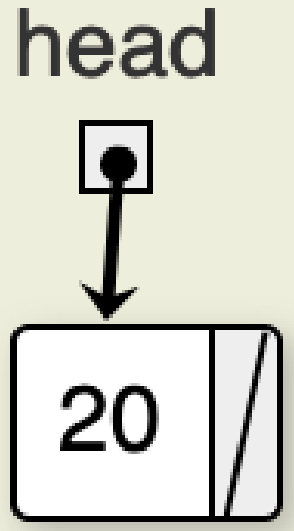
Chain of Nodes: Starting the Chain

Let's build a chain of nodes! Each node stores an integer value and a reference to another Node.

```
Node head = new Node(20, null);
```

The `next` (following) node of head's pointee is a **null reference**

The `data` stored in `head`'s pointee is **20**

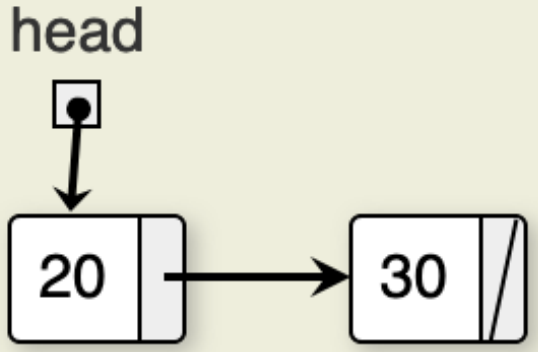


Chain of Nodes: Adding a Follower

To add a new node at the end of the chain, we must construct a new `Node` and point the `head` node to it.

```
head.next = new Node(30, null);
```

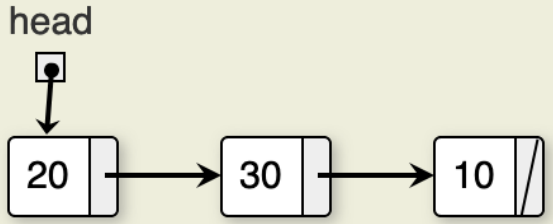
The `next` (follower) node of `head`'s pointee is a new `Node` storing `30` as its `data`



Chain of Nodes: Adding Another Follower

Chain of nodes

```
head.next.next = new Node(10, null);
```

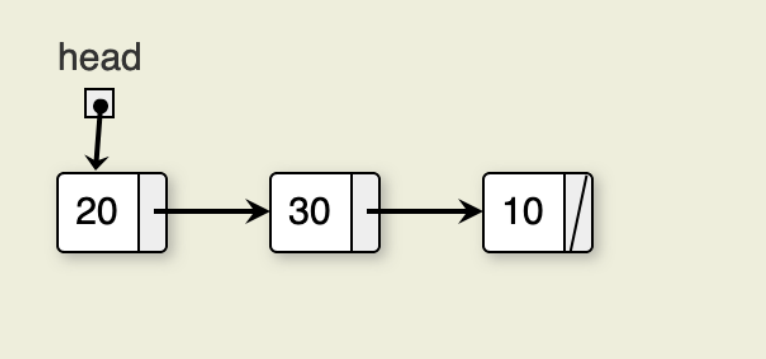


The `next` (follower) node of `head`'s follower is a new `Node` storing `10` as its `data`.

Chain of Nodes: All at Once

Putting everything together:

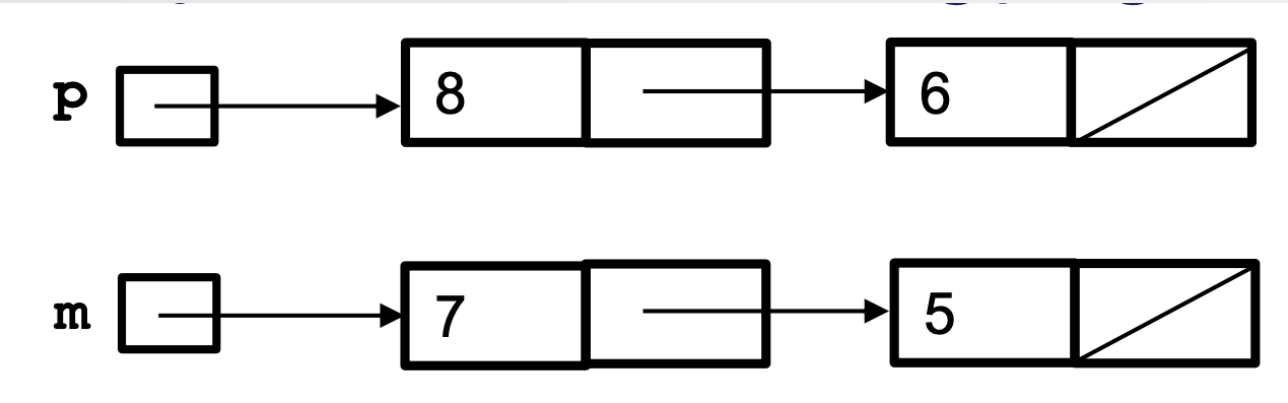
```
Node head = new Node(20, null);
head.next = new Node(30, null);
head.next.next = new Node(10, null);
```



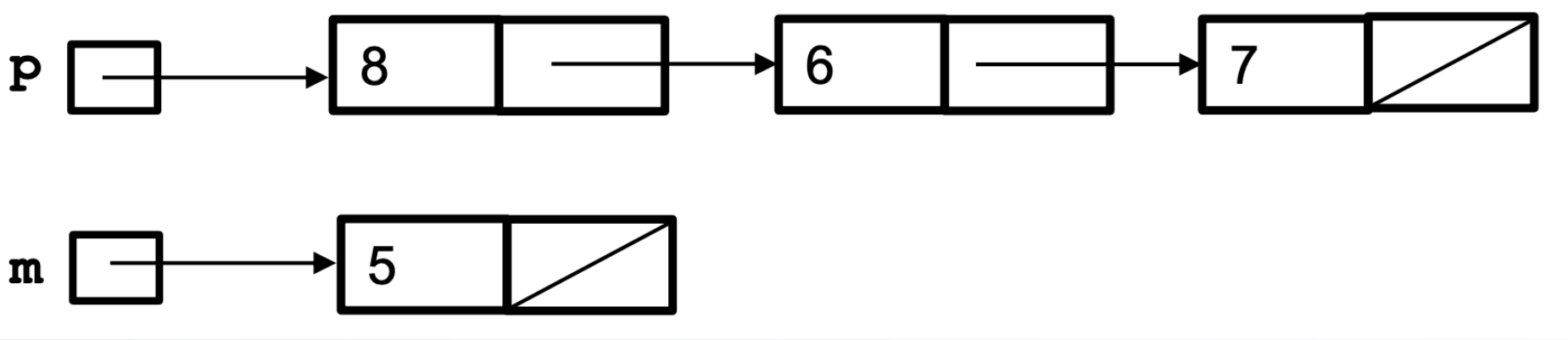
The last node has `null` as its `next` to mark that there are no more nodes.

Practice Working with Nodes

How can we get from this program state:



To this program state? (Without modifying node data directly…)



Chain of nodes: iteration

To iterate through a **every node** in a chain of nodes:

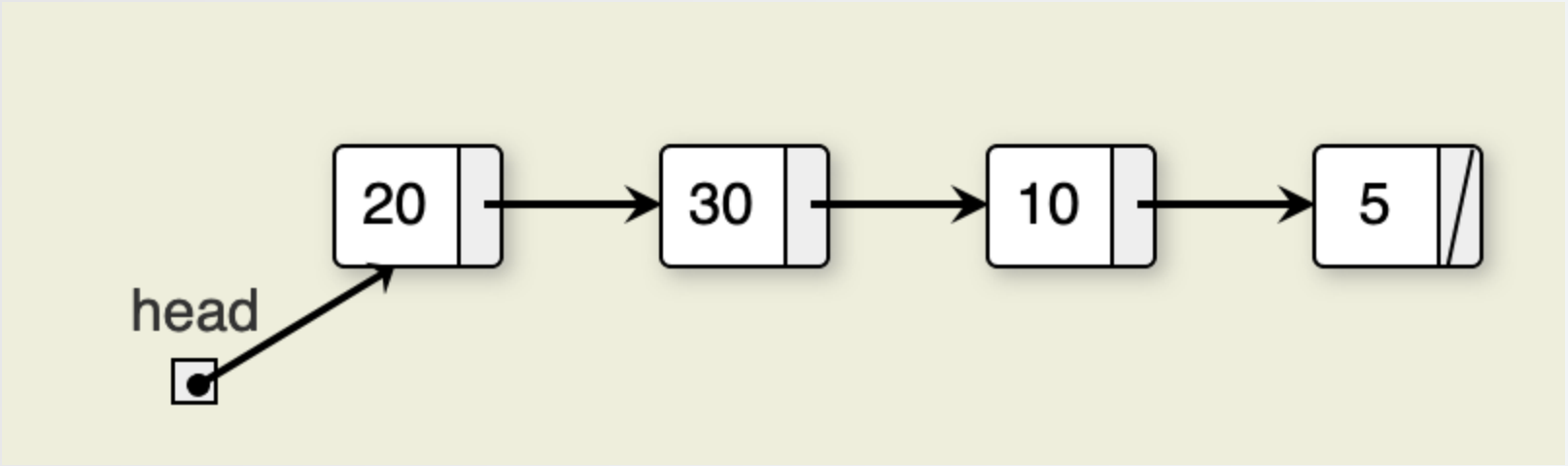
- We don’t need to know how many nodes are in the chain
- Just know that the last `node`'s `next` field is a `null` reference

Steps:

- Create a temporary node **variable** that starts by pointing to the head of the chain (aliasing)
- Iterate/loop by following the next references with each iteration, update the pointee of the temporary node
- Stop when the temporary node points to a null reference

Chain of nodes: iteration

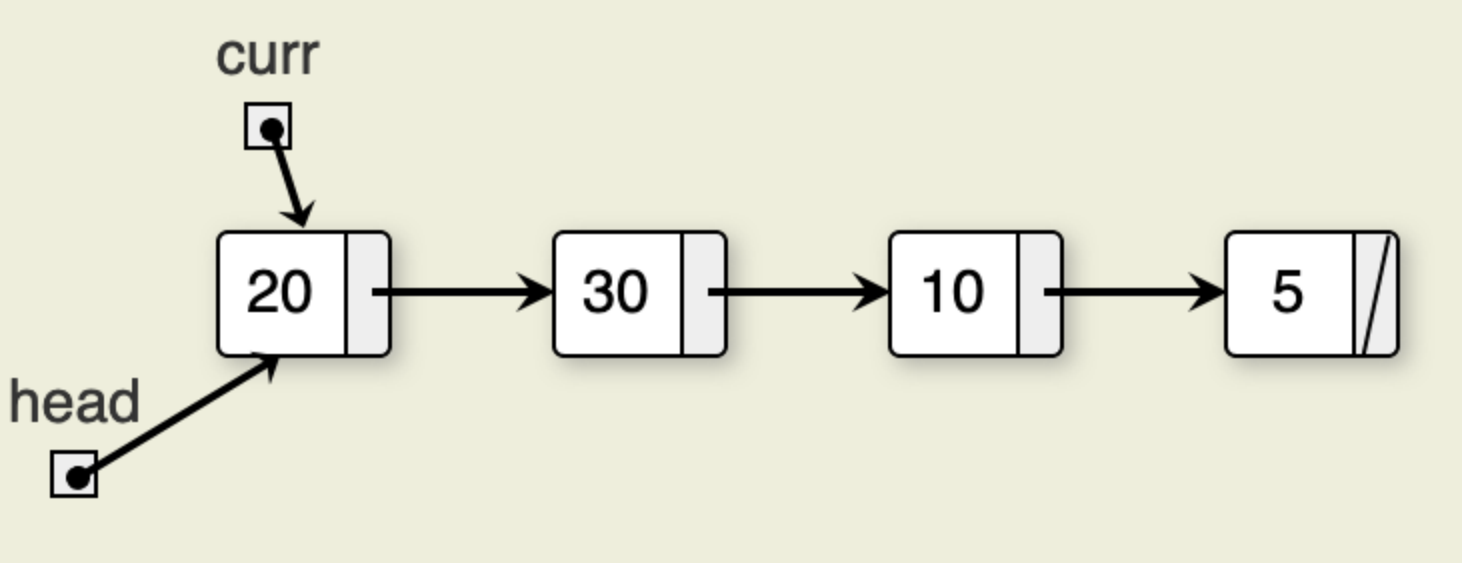
Given the following chain...



Chain of nodes: iteration

Create a temporary node that points to the head of the chain

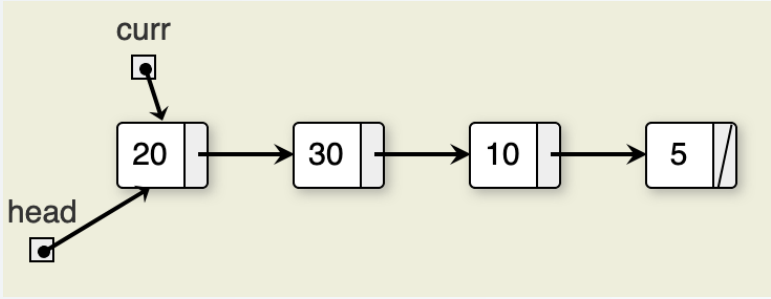
```
Node curr = head; //curr and head are aliases for each other
```



Chain of nodes: iteration

Start the loop & stop when `curr` points to the last node in the chain

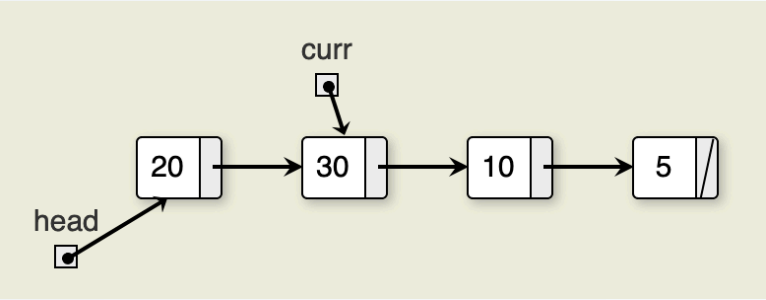
```
Node curr = head; //curr and head are aliases for each other
while (curr != null) { // while the pointee of curr is not null,
   curr = curr.next;   // advance curr to point to the next node.
}
```



Chain of nodes: iteration

`curr` now points to the node storing 30

```
Node curr = head; //curr and head are aliases for each other
while (curr != null) { // while the pointee of curr is not null,
   curr = curr.next;   // advance curr to point to the next node.
}
```

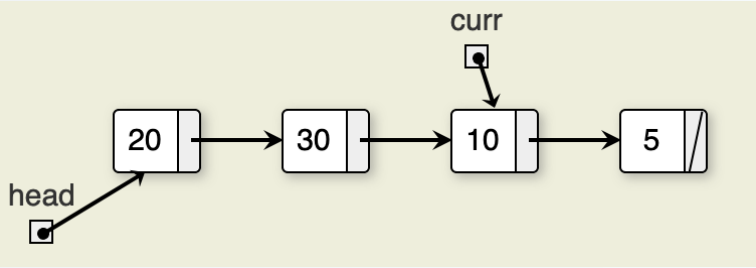


Note that `head` did not change the reference it stores.

Chain of nodes: iteration

`curr` now points to the node storing 10

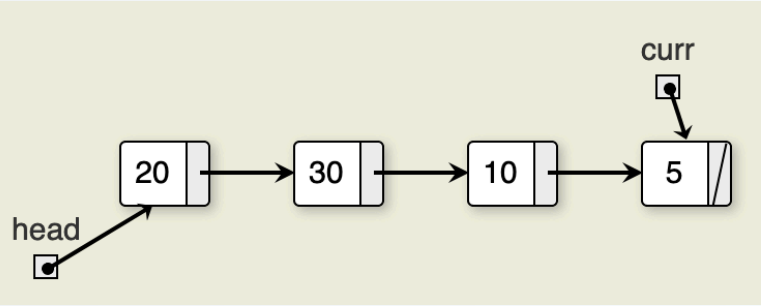
```
Node curr = head; //curr and head are aliases for each other
while (curr != null) { // while the pointee of curr is not null,
   curr = curr.next;   // advance curr to point to the next node.
}
```



Chain of nodes: iteration

`curr` now points to the node storing 5

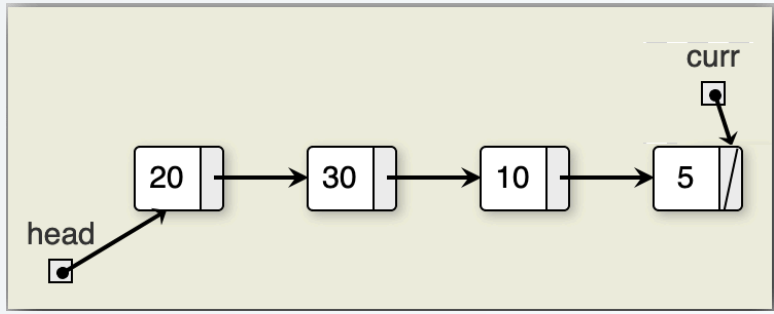
```
Node curr = head; //curr and head are aliases for each other
while (curr != null) { // while the pointee of curr is not null,
   curr = curr.next;   // advance curr to point to the next node.
}
```



Chain of nodes: iteration

`curr` now contains a `null` reference

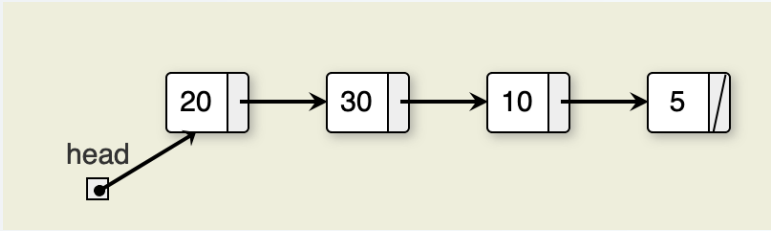
```
Node curr = head; //curr and head are aliases for each other
while (curr != null) { // while the pointee of curr is not null,
   curr = curr.next;   // advance curr to point to the next node.
}
```



Chain of nodes: iteration

Putting everything together:

The following code will print all the values stored in our chain



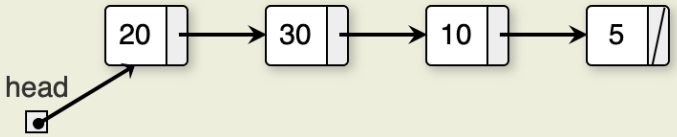

```
Node curr = head; //curr and head are aliases for each other
while (curr != null) {
   System.out.println(curr.data);
   curr = curr.next;
}
```

Will print: `20 30 10 5`

Chain of nodes: iteration (for loop)

As a for loop instead:

```
for (Node curr = head; curr != null; curr = curr.next) {
   System.out.println(curr.data);
}
```



Will print: `20 30 10 5`

Iterating to a Point

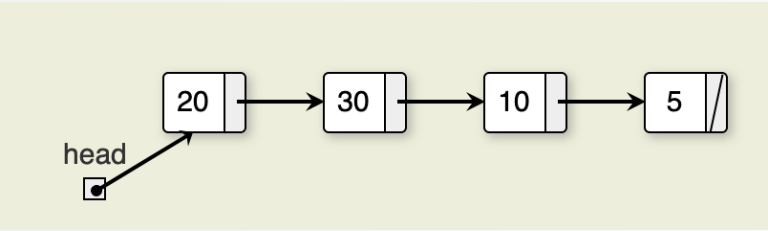
What if we wanted to print out the values of every node in the chain *except for the last one?* What would we change?

```
Node curr = head; //curr and head are aliases for each other
while (curr != null) {
    System.out.println(curr.data);
    curr = curr.next;
}
```

Iterating to a Point

If we try this:

```
Node curr = head; //curr and head are aliases for each other
while (curr.next != null) {
   System.out.println(curr.data);
   curr = curr.next;
}
System.out.println("Last: " + curr.data);
```



1. What is the last value that gets printed?
2. What happens when `head == null`?

Iterating to a Point

If we try this:

```
Node curr = head; //curr and head are aliases for each other
while (curr.next != null) {
   System.out.println(curr.data);
   curr = curr.next;
}
System.out.println("Last: " + curr.data);
```

1. `Last: 5`
2. Program crashes with `NullPointerException` because `curr.next != null` dereferences `null`.

Takeaways

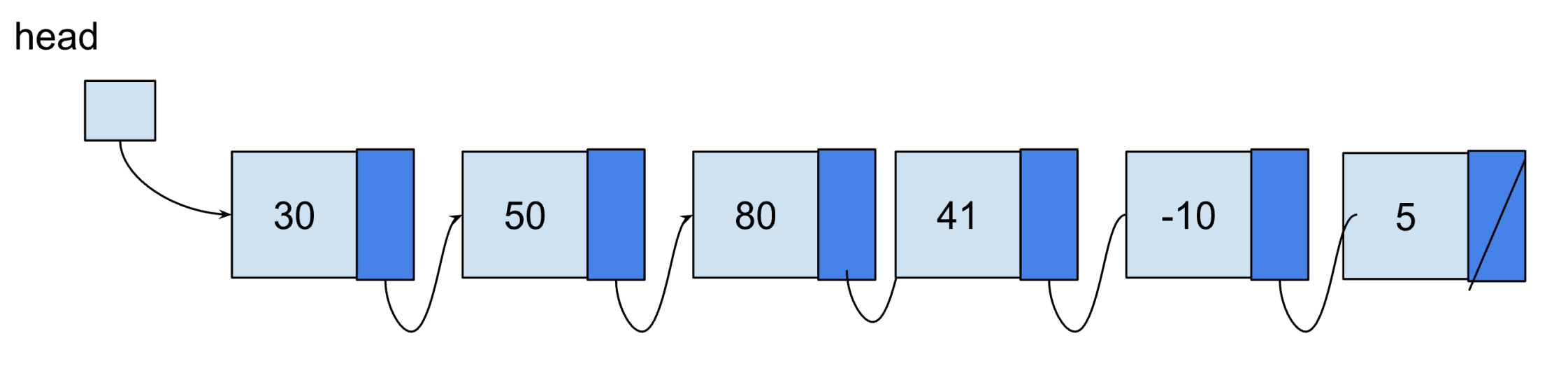
Practice safe iteration: make sure that you can guarantee that your node variables store non-null references when you try to dereference them.

Traversing to a particular node: you can write your while loop condition so that it stops at a non-null node. A reference to that node will be stored inside of your loop control variable afterwards.

Challenge!

Given a `Node head`, return a reference to the node in the chain that is first node with negative data. If no such node exists, or if the input is `null`, return `null`.

```
public static Node findFirstNegative(Node head) {...}
```


head
30
50
80
41
-10
5

```
public static Node findFirstNegative(Node head) {
    Node curr = head;
    while (curr != null && curr.data > 0) {
        curr = curr.next;
    }
    return curr;
}
```