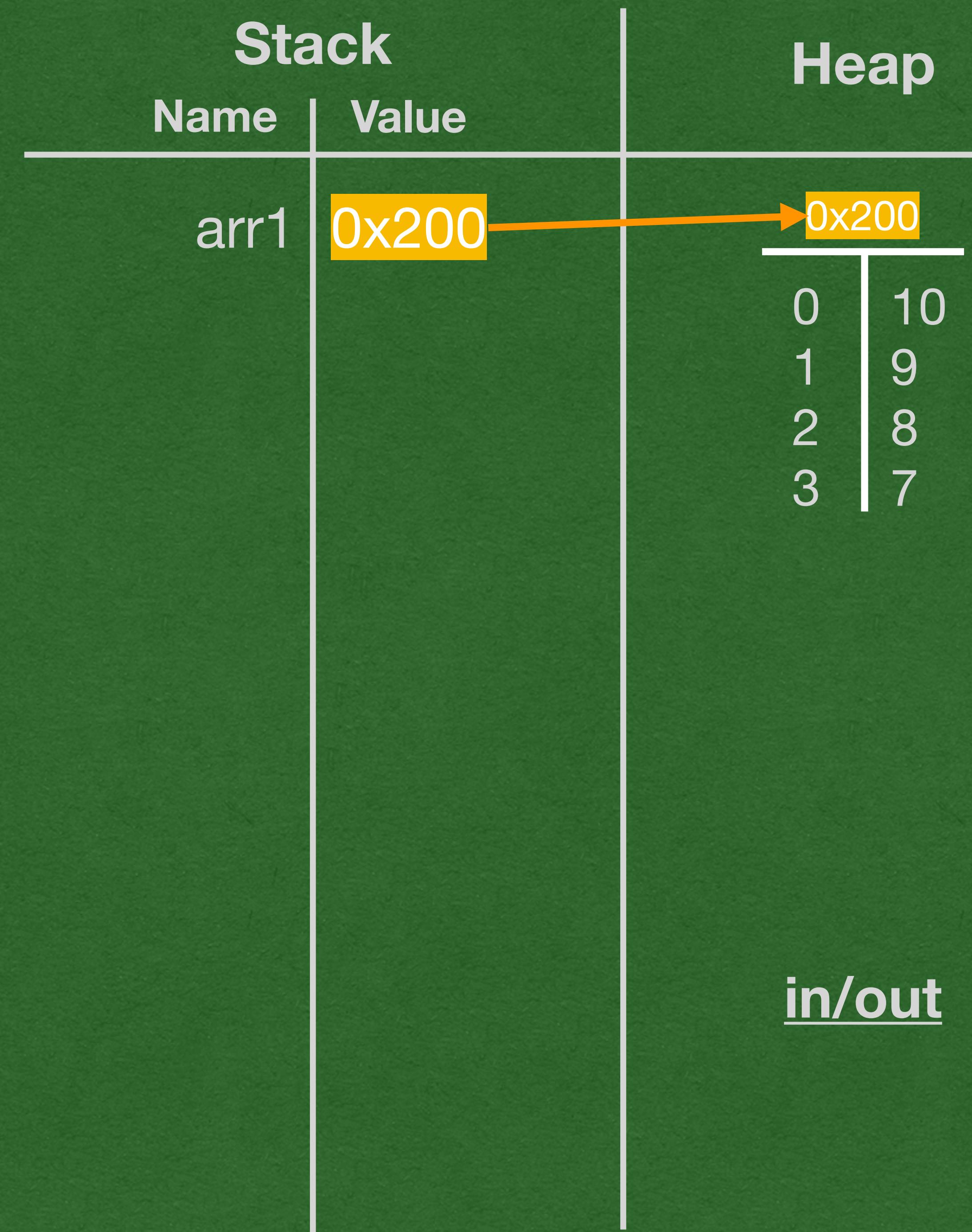


Linked List

ArrayList [And Array]

- Sequential
 - One continuous block of memory
 - Random access based on memory address
 - $\text{address} = \text{first_address} + (\text{element_size} * \text{index})$
- Fixed Size
 - Since memory adjacent to the block may be used
 - Efficient when you know how many elements you'll need to store

- We show an ArrayList on the heap in columns
- Values are all located in one continuous block of memory
- This is actually how ArrayLists [and Arrays] are stored



- This ArrayList stores 32-bit ints (4 bytes) and the ArrayList is stored at memory address 0x200
 - Find the element of each value using
 - $address = 0x200 + (4 * index)$
 - Easy to find any value, given it's index

Stack		Heap		
Name	Value			
arr1	0x200			
		0x200	0	10
		0x204	1	9
		0x208	2	8
		0x212	3	7

in/out

- This is called random access
- Memory is like a giant array
 - We call it RAM (Random Access Memory)

Stack		Heap	
Name	Value		
arr1	0x200	0x200	0x200
		0	10
		1	9
		2	8
		3	7

in/out

Linked List

- Sequential
- Spread across memory
- Each element knows the memory address of the next element
 - Follow the addresses to find each element
- Variable Size
- Store new element anywhere in memory

Linked List

- Each value in a list is stored in a separate object on the heap
- Also stores a reference to the next element
- A reference to the list is only a reference to the first value
- Last link stores null
 - We say the list is "null terminated"
 - When we read a value of null we know we've reached the end of the list

Linked List

```
package week4;

public class LinkedListNodeInt {
    private int value;
    private LinkedListNodeInt next;

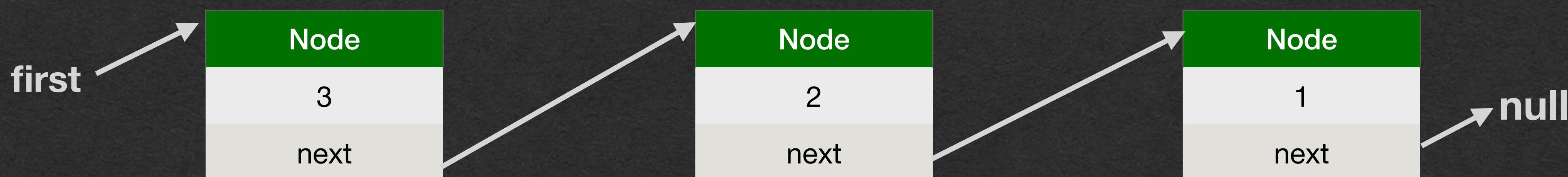
    public LinkedListNodeInt(int value, LinkedListNodeInt next) {
        this.value = value;
        this.next = next;
    }

    public static void main(String[] args) {
        LinkedListNodeInt first = new LinkedListNodeInt(1, null);
        first = new LinkedListNodeInt(2, first);
        first = new LinkedListNodeInt(3, first);
    }
}
```

- We create our own linked list node class
- A node represents one "link" in the list
- The list itself is a reference to the first/head node

Structure

- Each node stores one value of the list
- Each node refers to the next node
- A variable "storing" a list stores a reference to the first node of the list



Memory Diagram

- `LinkedListNode<Int> -> LLNode`
- To save space on the slide

```
public class LLNode {
    private int value;
    private LLNode next;

    public LLNode(int value, LLNode next) {
        this.value = value;
        this.next = next;
    }

    ➔ public static void main(String[] args) {
        LLNode first = new LLNode(1, null);
        first = new LLNode(2, first);
        first = new LLNode(3, first);
    }
}
```

Name	Stack	Value	Heap
			<u>in/out</u>

- Create a LLNode object
- next is equal to null
 - The lack of a reference

```

public class LLNode {
    private int value;
    private LLNode next;

    public LLNode(int value, LLNode next) {
        this.value = value;
        this.next = next;
    }

    public static void main(String[] args) {
        LLNode first = new LLNode(1, null);
        first = new LLNode(2, first);
        first = new LLNode(3, first);
    }
}

```

Stack		Heap
Name	Value	
LLNode	0x350	LLNode value 1 next null 0x350
first		
this	1	
value	null	
next		

in/out

- Call the constructor again
- Pass myList (0x350) as next

Stack		Heap
Name	Value	
first	0x350	LLNode
this	0x350	value 1
value	1	next null
next	null	0x350
this	0x200	LLNode
value	2	value 2
next	0x350	next 0x350

```

public class LLNode {
    private int value;
    private LLNode next;

    public LLNode(int value, LLNode next) {
        this.value = value;
        this.next = next;
    }

    public static void main(String[] args) {
        LLNode first = new LLNode(1, null);
        first = new LLNode(2, first);
        first = new LLNode(3, first);
    }
}

```

in/out

- Reassign first to the reference returned by the constructor
- first now stores 0x200 which has a next of 0x350

```
public class LLNode {
    private int value;
    private LLNode next;

    public LLNode(int value, LLNode next) {
        this.value = value;
        this.next = next;
    }

    public static void main(String[] args) {
        LLNode first = new LLNode(1, null);
        first = new LLNode(2, first);
        first = new LLNode(3, first);
    }
}
```



Stack		Heap
Name	Value	
first	0x350 0x200	LLNode
this	0x350	LLNode
value	1	value 1
next	null	next null
this	0x200	LLNode
value	2	value 2
next	0x350	next 0x350

in/out

- Repeat the process for the node with value 3
- We now have a linked list with 3 elements

```
public class LLNode {
    private int value;
    private LLNode next;

    public LLNode(int value, LLNode next) {
        this.value = value;
        this.next = next;
    }

    public static void main(String[] args) {
        LLNode first = new LLNode(1, null);
        first = new LLNode(2, first);
        first = new LLNode(3, first);
    }
}
```

Stack		Heap
Name	Value	
first	0x350 0x200	
this	0x480	
value	0x350	
next	1	
this	null	
value	0x200	
next	2	
this	0x350	
value	0x480	
next	3	
	0x200	

LLNode
value 1
next null

LLNode
value 2
next 0x350

LLNode
value 3
next 0x200

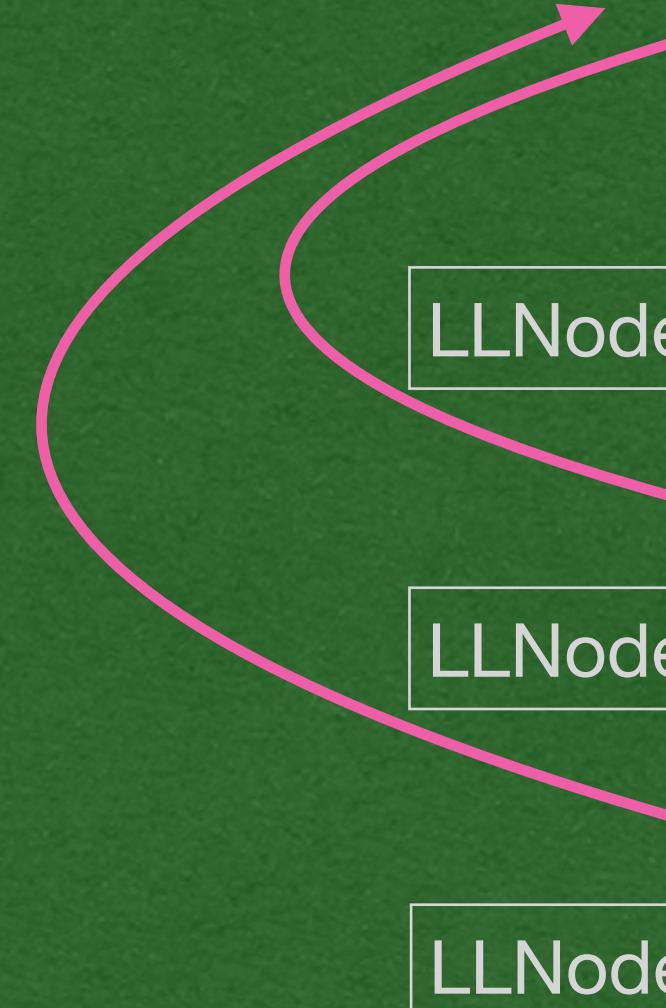
in/out

- Our variable "first" only stores a reference to the first node of the list
- We call the first node the head of the list

```
public class LLNode {
    private int value;
    private LLNode next;

    public LLNode(int value, LLNode next) {
        this.value = value;
        this.next = next;
    }

    public static void main(String[] args) {
        LLNode first = new LLNode(1, null);
        first = new LLNode(2, first);
        first = new LLNode(3, first);
    }
}
```



The diagram illustrates the state of memory. On the left, three `LLNode` objects are shown on the heap, each with a `value` field (1, 2, 3) and a `next` field pointing to the next node. On the right, the stack shows the execution environment. The `first` variable at the top of the stack initially points to the first node (value 1). As the code executes, it creates new nodes and assigns them to `first`, which then points to the new head of the list. A green arrow points from the closing brace of the main method to the stack diagram.

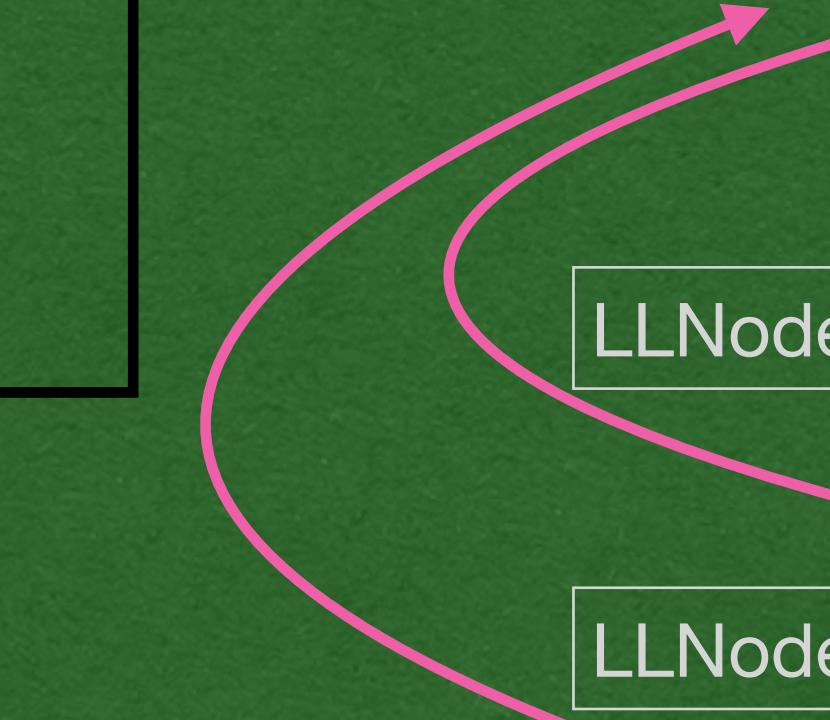
Stack		Heap
Name	Value	
first	0x350 0x200	
this	0x350	<code>LLNode</code>
value	1	value 1
next	null	next null
this	0x200	<code>LLNode</code>
value	2	value 2
next	0x350	next 0x350
this	0x480	<code>LLNode</code>
value	3	value 3
next	0x200	next 0x200

- Each node stores one value of the list and a reference to the next node
- Each node can be anywhere on the heap

```
public class LLNode {
    private int value;
    private LLNode next;

    public LLNode(int value, LLNode next) {
        this.value = value;
        this.next = next;
    }

    public static void main(String[] args) {
        LLNode first = new LLNode(1, null);
        first = new LLNode(2, first);
        first = new LLNode(3, first);
    }
}
```



Stack		Heap	
Name	Value		
first	0x350 0x200		
this	0x480		
value	0x350		
next	1		
this	0x200		
value	0x350		
next	2		
this	0x480		
value	0x350		
next	3		
	0x200		

LLNode	
value	1
next	null

0x350

LLNode	
value	2
next	0x350

0x200

LLNode	
value	3
next	0x200

0x480

in/out

toString

```
public class LLNode {  
    private int value;  
    private LLNode next;  
  
    public LLNode(int val, LLNode next) {  
        this.value = val;  
        this.next = next;  
    }  
  
    public String toString() {  
        String out = "";  
        out += this.value;  
        if (this.next != null) {  
            out += " " + this.next.toString();  
        }  
        return out;  
    }  
  
    public static void main(String[] args) {  
        LLNode first = new LLNode(1, null);  
        first = new LLNode(2, first);  
        first = new LLNode(3, first);  
        String value = first.toString();  
        System.out.println(value);  
    }  
}
```

- Let's add a `toString` method to our Linked List
- This will return the values separated by spaces
- .. aaand it uses recursion!
 - Remember recursion?
 - We're using it!

2 Memory 2 Diagram

Stack

Name	Value
main	
... first	0x002 0x003 0x004
... out	"3 2 1"
LinkedListNodeInt	
... this	0x002
... val	1
... next	null
LinkedListNodeInt	
... this	0x003
... val	2
... next	0x002
LinkedListNodeInt	
... this	0x004
... val	3
... next	0x003
toString	
... this	0x004
... out	"3 2 1"
toString	
... this	0x003
... out	"2 1"
toString	
... this	0x002
... out	"1"

Heap

Name	Value
LinkedListNodeInt	LinkedListNodeInt
... value	1
... next	null
0x002	LinkedListNodeInt
... value	2
... next	0x002
0x003	LinkedListNodeInt
... value	3
... next	0x003
0x004	Create Heap Object

IO

3 2 1

```

1 package week5;
2
3 public class LinkedListNodeInt {
4     private int value;
5     private LinkedListNodeInt next;
6
7     public LinkedListNodeInt(int val, LinkedListNodeInt next) {
8         this.value = val;
9         this.next = next;
10    }
11
12    public String toString() {
13        String out = "";
14        out += this.value;
15        if (this.next != null) {
16            out += " " + this.next.toString();
17        }
18        return out;
19    }
20
21    public static void main(String[] args) {
22        LinkedListNodeInt first = new LinkedListNodeInt(1, null);
23        first = new LinkedListNodeInt(2, first);
24        first = new LinkedListNodeInt(3, first);
25        String out = first.toString();
26        System.out.println(out);
27    }
28}

```

- We could write
- `System.out.println(first)`
- We're explicitly calling `toString` to be clear of our intentions

```

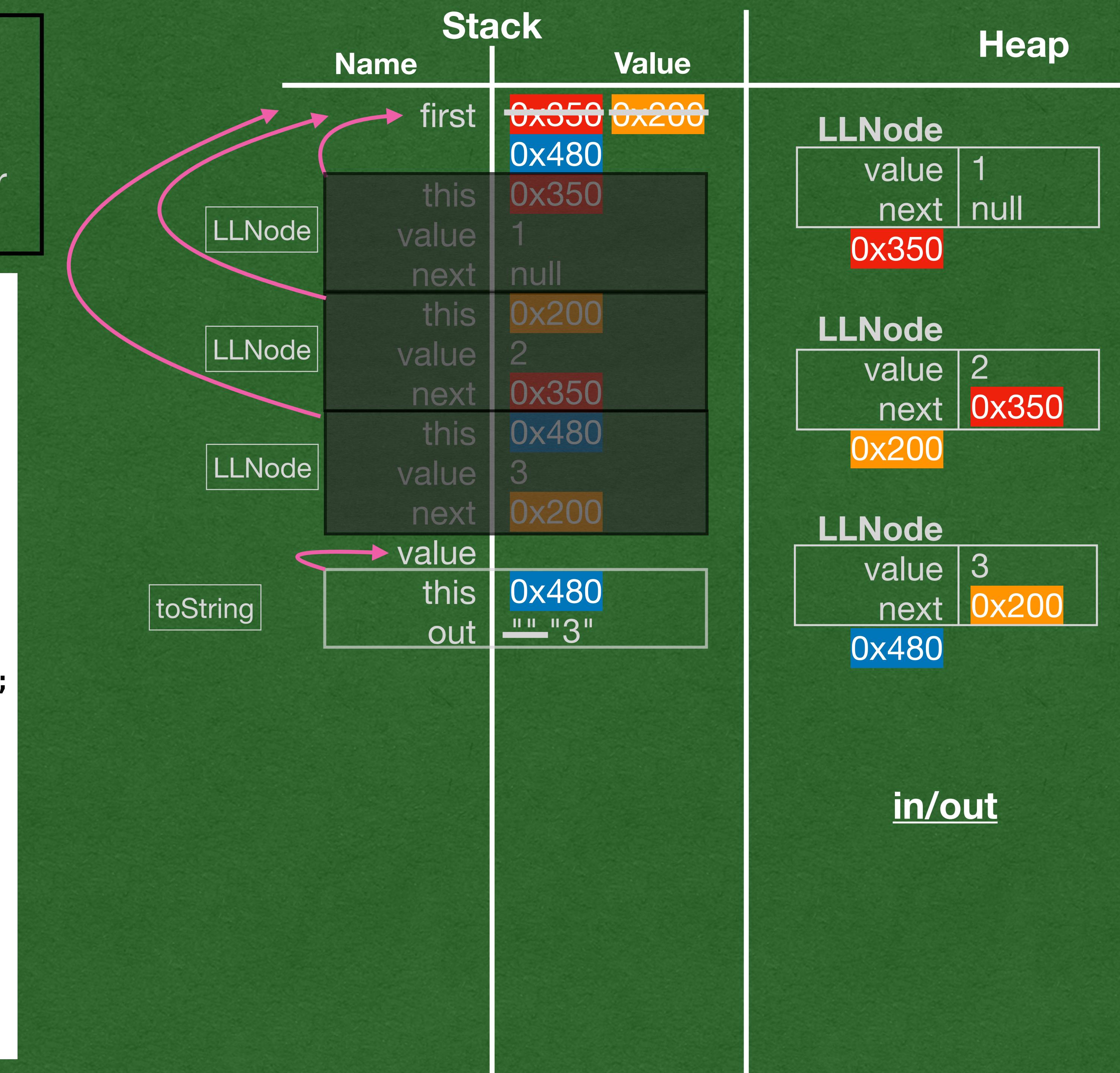
public class LLNode {
    private int value;
    private LLNode next;

    public LLNode(int value, LLNode next) {
        this.value = value;
        this.next = next;
    }

    public String toString() {
        String out = "";
        out += this.value;
        if (this.next != null) {
            out += " " + this.next.toString();
        }
        return out;
    }

    public static void main(String[] args) {
        LLNode first = new LLNode(1, null);
        first = new LLNode(2, first);
        first = new LLNode(3, first);
        → String value = first.toString();
        System.out.println(value);
    }
}

```



- If next is not null, we are not at the end of the list
 - There's more work to be done
 - Make a recursive call

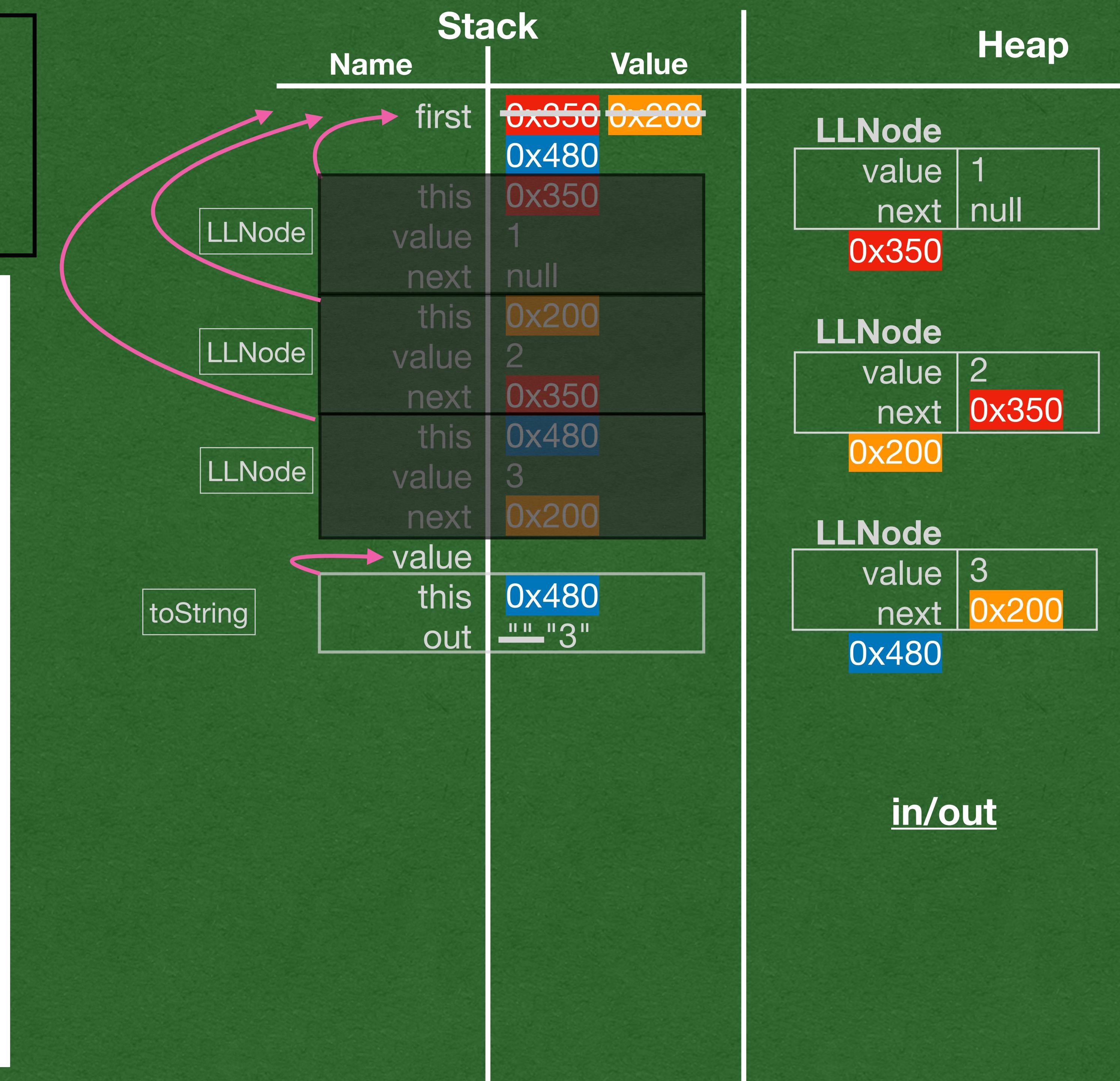
```

public class LLNode {
    private int value;
    private LLNode next;

    public LLNode(int value, LLNode next) {
        this.value = value;
        this.next = next;
    }

    public String toString() {
        String out = "";
        out += this.value;
        if (this.next != null) {
            ➔ out += " " + this.next.toString();
        }
        return out;
    }

    public static void main(String[] args) {
        LLNode first = new LLNode(1, null);
        first = new LLNode(2, first);
        first = new LLNode(3, first);
        ➔ String value = first.toString();
        System.out.println(value);
    }
}
  
```



- The recursive call is made on the next node
- The first stack frame waits for the return value of the recursive call

```

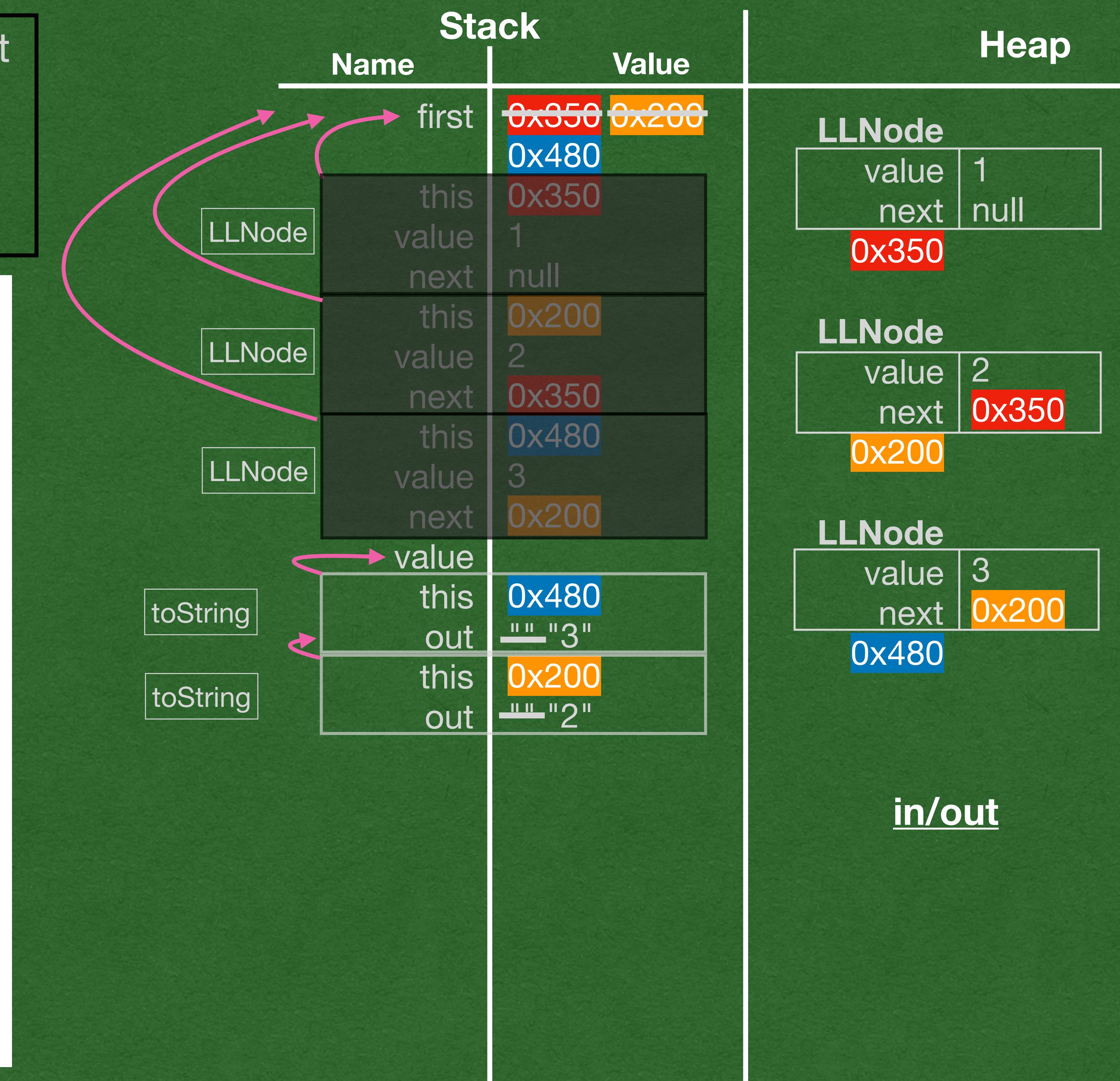
public class LLNode {
    private int value;
    private LLNode next;

    public LLNode(int value, LLNode next) {
        this.value = value;
        this.next = next;
    }

    public String toString() {
        String out = "";
        out += this.value;
        if (this.next != null) {
            out += " " + this.next.toString();
        }
        return out;
    }

    public static void main(String[] args) {
        LLNode first = new LLNode(1, null);
        first = new LLNode(2, first);
        first = new LLNode(3, first);
        → String value = first.toString();
        System.out.println(value);
    }
}

```



- Make another recursive call
- In this stack frame, the condition is false

```

public class LLNode {
    private int value;
    private LLNode next;

    public LLNode(int value, LLNode next) {
        this.value = value;
        this.next = next;
    }

    public String toString() {
        String out = "";
        out += this.value;
        if (this.next != null) {
            out += " " + this.next.toString();
        }
        return out;
    }

    public static void main(String[] args) {
        LLNode first = new LLNode(1, null);
        first = new LLNode(2, first);
        first = new LLNode(3, first);
        → String value = first.toString();
        System.out.println(value);
    }
}

```

Stack		Heap
Name	Value	
first	0x350 0x200	
this	0x350	
value	1	
next	null	
this	0x200	
value	2	
next	0x350	
this	0x480	
value	3	
next	0x200	
value	0x480	
this	0x200	
out	"" "3"	
this	0x200	
out	"" "2"	
this	0x350	
out	"" "1"	
toString		
toString		
toString		

in/out

- This frame returns "1" to the previous stack frame

```

public class LLNode {
    private int value;
    private LLNode next;

    public LLNode(int value, LLNode next) {
        this.value = value;
        this.next = next;
    }

    public String toString() {
        String out = "";
        out += this.value;
        if (this.next != null) {
            out += " " + this.next.toString();
        }
        return out;
    }

    public static void main(String[] args) {
        LLNode first = new LLNode(1, null);
        first = new LLNode(2, first);
        first = new LLNode(3, first);
        → String value = first.toString();
        System.out.println(value);
    }
}

```

Stack		Heap	
Name	Value		
first	0x350 0x200		
this	0x350		
value	1		
next	null		
this	0x200		
value	2		
next	0x350		
this	0x480		
value	3		
next	0x200		
value	0x480		
this	0x200		
out	"" "3"		
value	0x200		
this	0x480		
out	"" "2"		
value	0x350		
this	0x480		
out	"" "1"		
		in/out	

- The previous stack frame (With `this == 0x200`) is back on top of the stack
- It takes the return value of "1" and continues running code

```

public class LLNode {
    private int value;
    private LLNode next;

    public LLNode(int value, LLNode next) {
        this.value = value;
        this.next = next;
    }

    public String toString() {
        String out = "";
        out += this.value;
        if (this.next != null) {
            out += " " + this.next.toString();
        }
        return out;
    }

    public static void main(String[] args) {
        LLNode first = new LLNode(1, null);
        first = new LLNode(2, first);
        first = new LLNode(3, first);
        → String value = first.toString();
        System.out.println(value);
    }
}

```



- Return "2 1" to the first recursive stack frame

```

public class LLNode {
    private int value;
    private LLNode next;

    public LLNode(int value, LLNode next) {
        this.value = value;
        this.next = next;
    }

    public String toString() {
        String out = "";
        out += this.value;
        if (this.next != null) {
            → out += " " + this.next.toString();
        }
        → return out;
    }

    public static void main(String[] args) {
        LLNode first = new LLNode(1, null);
        first = new LLNode(2, first);
        first = new LLNode(3, first);
        → String value = first.toString();
        System.out.println(value);
    }
}

```



- The frame with `this == 0x480` is back on top of the stack
- Concatenate the returned value to `out`

```

public class LLNode {
    private int value;
    private LLNode next;

    public LLNode(int value, LLNode next) {
        this.value = value;
        this.next = next;
    }

    public String toString() {
        String out = "";
        out += this.value;
        if (this.next != null) {
            → out += " " + this.next.toString();
        }
        return out;
    }

    public static void main(String[] args) {
        LLNode first = new LLNode(1, null);
        first = new LLNode(2, first);
        first = new LLNode(3, first);
        → String value = first.toString();
        System.out.println(value);
    }
}

```



- Return "3 2 1" to the main method

```

public class LLNode {
    private int value;
    private LLNode next;

    public LLNode(int value, LLNode next) {
        this.value = value;
        this.next = next;
    }

    public String toString() {
        String out = "";
        out += this.value;
        if (this.next != null) {
            out += " " + this.next.toString();
        }
        ➔ return out;
    }

    public static void main(String[] args) {
        LLNode first = new LLNode(1, null);
        first = new LLNode(2, first);
        first = new LLNode(3, first);
        ➔ String value = first.toString();
        System.out.println(value);
    }
}

```



- Assign "3 2 1" to value in the main stack frame
- We only called `toString` on the head of the list, but got all the values of the list

```

public class LLNode {
    private int value;
    private LLNode next;

    public LLNode(int value, LLNode next) {
        this.value = value;
        this.next = next;
    }

    public String toString() {
        String out = "";
        out += this.value;
        if (this.next != null) {
            out += " " + this.next.toString();
        }
        return out;
    }

    public static void main(String[] args) {
        LLNode first = new LLNode(1, null);
        first = new LLNode(2, first);
        first = new LLNode(3, first);
        ➔ String value = first.toString();
        System.out.println(value);
    }
}

```



- Print to the screen and program ends

```

public class LLNode {
    private int value;
    private LLNode next;

    public LLNode(int value, LLNode next) {
        this.value = value;
        this.next = next;
    }

    public String toString() {
        String out = "";
        out += this.value;
        if (this.next != null) {
            out += " " + this.next.toString();
        }
        return out;
    }

    public static void main(String[] args) {
        LLNode first = new LLNode(1, null);
        first = new LLNode(2, first);
        first = new LLNode(3, first);
        String value = first.toString();
        System.out.println(value);
    }
}
  
```



Stack

Name	Value
main	
... first	0x002 0x003 0x004
... out	"3 2 1"
LinkedListNodeInt	
... this	0x002
... val	1
... next	null
LinkedListNodeInt	
... this	0x003
... val	2
... next	0x002
LinkedListNodeInt	
... this	0x004
... val	3
... next	0x003
toString	
... this	0x004
... out	"3 2 1"
toString	
... this	0x003
... out	"2 1"
toString	
... this	0x002
... out	"1"

Heap

Name	Value
LinkedListNodeInt	LinkedListNodeInt
... value	1
... next	null
0x002	LinkedListNodeInt
... value	2
... next	0x002
0x003	LinkedListNodeInt
... value	3
... next	0x003
0x004	Create Heap Object

IO

3 2 1

```

1 package week5;
2
3 public class LinkedListNodeInt {
4     private int value;
5     private LinkedListNodeInt next;
6
7     public LinkedListNodeInt(int val, LinkedListNodeInt next) {
8         this.value = val;
9         this.next = next;
10    }
11
12    public String toString() {
13        String out = "";
14        out += this.value;
15        if (this.next != null) {
16            out += " " + this.next.toString();
17        }
18        return out;
19    }
20
21    public static void main(String[] args) {
22        LinkedListNodeInt first = new LinkedListNodeInt(1, null);
23        first = new LinkedListNodeInt(2, first);
24        first = new LinkedListNodeInt(3, first);
25        String out = first.toString();
26        System.out.println(out);
27    }
28}

```