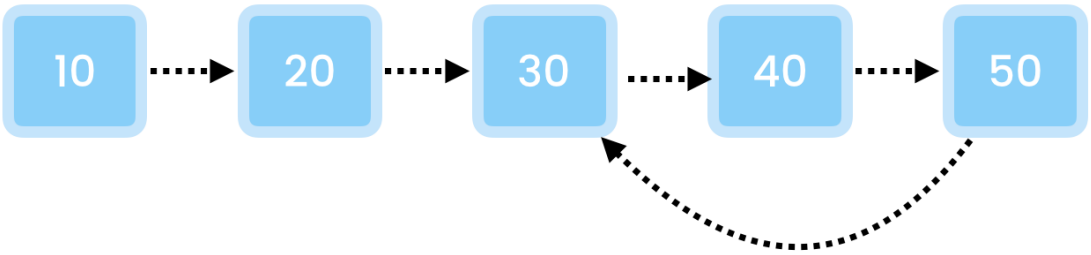
Linked Lists

Exercises

1- Find the middle of a linked list in one pass. If the list has an even number of nodes, there would be two middle nodes. (Note: Assume that you don't know the size of the list ahead of time.)

Solution: LinkedList.printMiddle() - You can read my analysis of the solution in the next page.

2- Check to see if a linked list has a loop.



Hint: use two pointers (slow and fast) to traverse the list. Move the slow pointer one step forward and the fast pointer two steps forward. If there's a loop, at some point, the fast pointer will meet the slow pointer and overtake it. Draw this on a paper and see it for yourself. This algorithm is called *Floyd's Cycle-finding Algorithm*.

Solution: LinkedList.hasLoop() - You can read my analysis later in this document.

Solutions

1- Find the middle of a linked list in one pass.

As always, we start by simplifying and narrowing the problem. Let's imagine the list has an odd number of nodes.

Since we have to find the middle node in one pass, we need two pointers. The tricky part here is that we should figure out how many nodes should the first and second pointers be apart. Let's throw a few numbers to find the relationship between these pointers.

Number of Nodes	Middle Node
1	1
3	2
5	3
7	4
9	5
11	6

Do you see a pattern here? In every row, the number of nodes is increasing by two where as the position of the middle node is increasing by one. Agreed?

So, we can define two pointers that reference the first node initially. Then, we use a loop to move these pointers forward. In every iteration, we move the first pointer one step and the second pointer two steps forward. The moment the second pointer hits the tail node, the first node is pointing the middle node.

Now, let's expand our problem. What if the list has an even number of nodes?

Number of Nodes	Middle Node
2	1,2
4	2,3
6	3,4
8	4,5
10	5,6

We see the same pattern. In every step, the number of nodes increases by two whereas the position of the middle node is increasing by one. The only difference is that here we need to return two middle nodes. That's easy. Once we find the left middle node, we'll also return the node next to it.

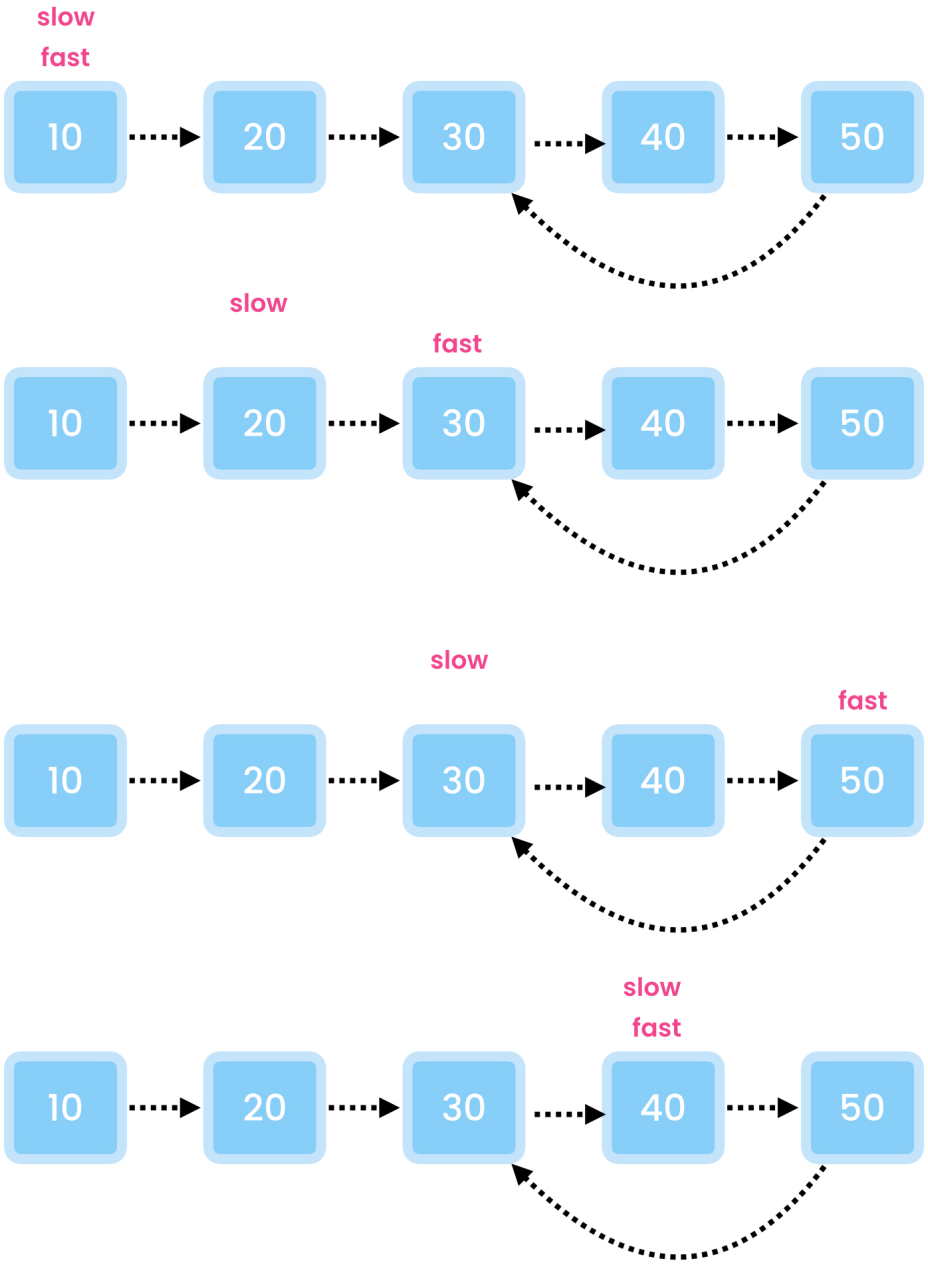
How do we know if the list has an even or odd number of items? We can declare a count variable and increment it in each step. But we don't really need this. If the list has an even number of nodes, at the end of the last iteration, the second pointer will reference the tail node; otherwise, it'll be null. (Try a few examples and you'll see this yourself.)

Here's how we can implement this algorithm:

```
1:    var a = first;
2:    var b = first;
3:    while (b != last && b.next != last) {
4:       b = b.next.next;
5:       a = a.next;
6:   }
7:
8:   if (b == last)
9:      System.out.println(a.value);
10:   else
11:      System.out.println(a.value + ", " + a.next.value);
```

This algorithm works for any lists with at least one element. For an empty list, we need to throw an **IllegalStateException**. You can see the complete solution in **LinkedList.printMiddle().**

2- Check to see if a linked list has a loop.



You can find the implementation of this algorithm in **LinkedList.hasLoop().** To test this:

```
var list = LinkedList.createWithLoop();
System.out.println(list.hasLoop());
```