ICS 240
Introduction to Data Structures

Jessica Maistrovich

Metropolitan State University

New Data Structure!

Why Do We Need Another Data Structure

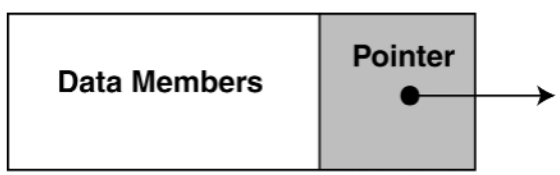
- Recall that the array has a fixed size, determined when it is created.
  - If you specify a size that is **too big**, then you are wasting memory
  - If you specify a size that is **too small**, then you may have to re-copy the elements to another bigger array when the original array gets full. This may slow your program
  - What if we had a data structure that allowed the size to change dynamically? (In other words on an as needed basis during runtime.)
- Also recall the amount of effort required to add or delete from a sorted array.
  - What if we had a data structure that allowed us to insert/delete an element with much less work?

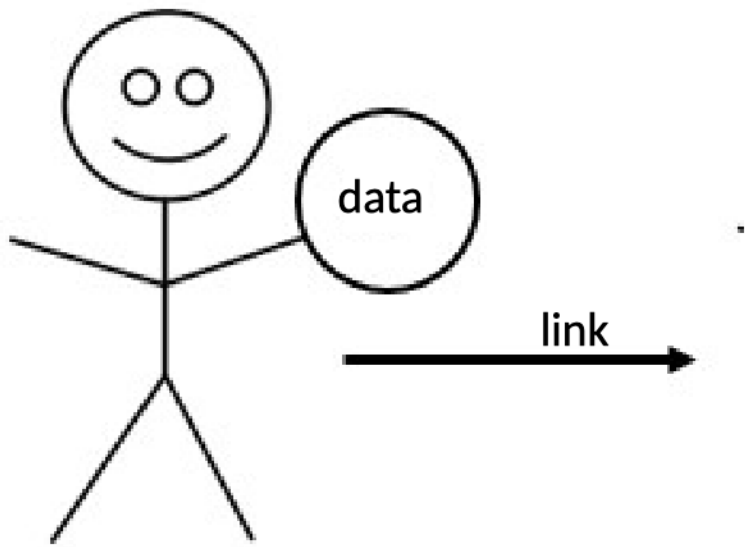
Inspiration for a Linked List Data Structure

Node class

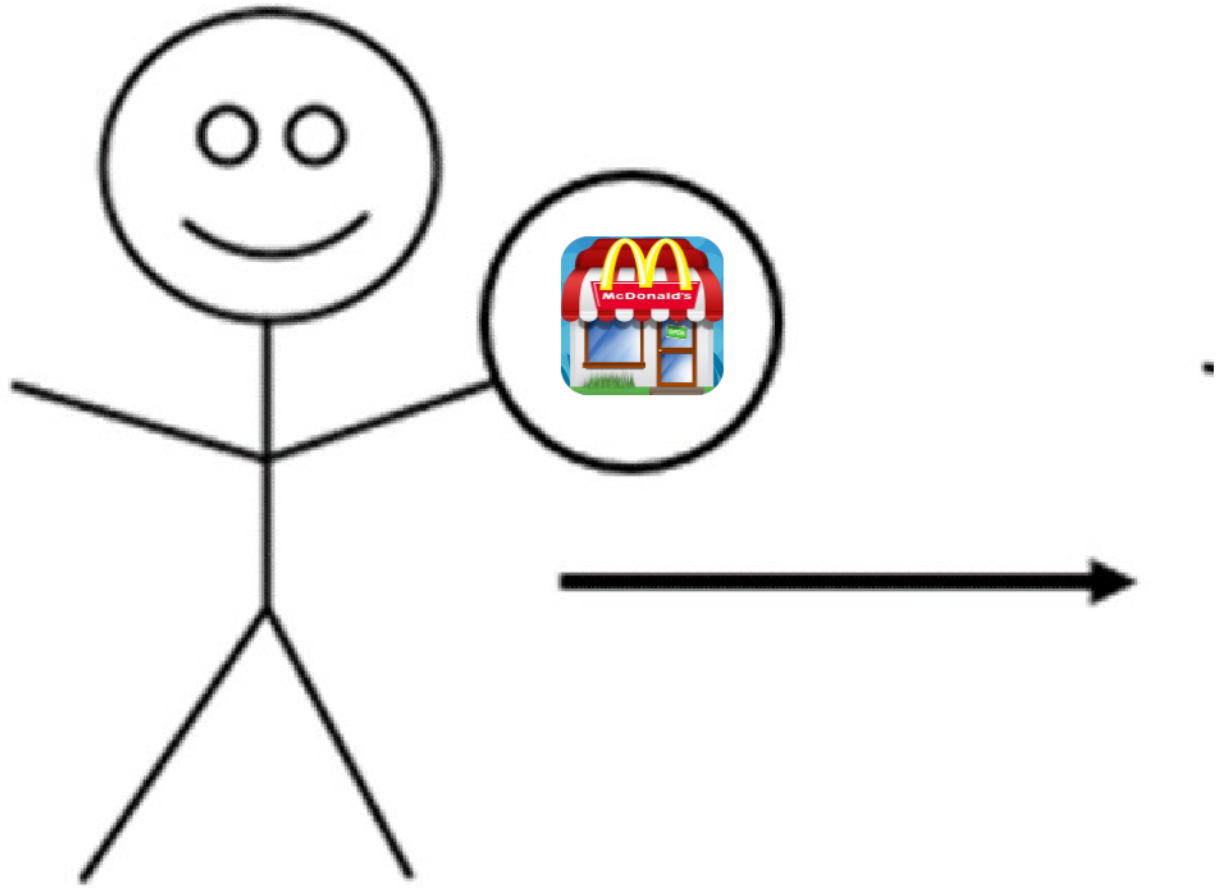
A Linked List Consists of nodes

- Each `node` in a linked list contains two instance variables:
  - **`data`**:
    - Data element(s) stored in this node
  - **`link`**:
    - a pointer (or a `reference`), that `links` th node to the next node in the list



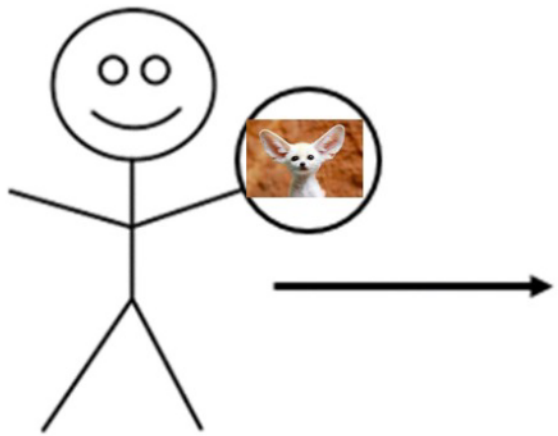


Let’s make the node specific to our Class



Try it!

- Open the Animal Shelter project in eclipse. Create a node class.
  - What will it be holding?
  - What will you call it?



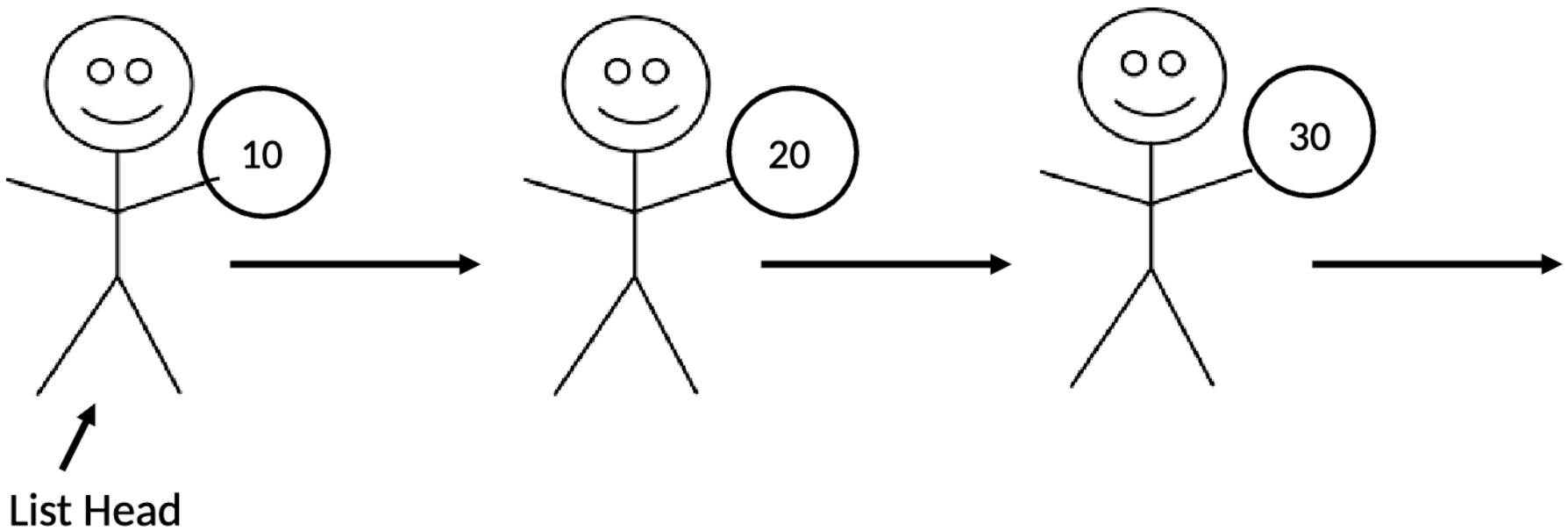
- Add a constructor, getters, and setters.

- Why doesn’t our node class have a toString or an equals?

Linked List Implementation

What is a Linked List?

- A linked list is a series of connected *nodes*
- A linked list can grow or shrink in size as nodes are added or deleted
- A linked list is called "linked" because each node in the list has a *reference* that *links* it to the next node in the list.



`LinkedList` Implementation

- The ADT should be the same whether it is implemented using an **array** or using a **linked list**
  - One big difference: When an ADT is implemented using a linked list we do not have to worry about `capacity.` So capacity handling operations are not needed
- The linked list collection is implemented using two instance variables as follows:

```
public class McDonaldsLinkedList{
        private McDonaldsNode head;
        private int manyNodes;
}
```

contrast to

```
public class McDonaldsArray{
      private McDonalds[] someName;
      private int manyItems;
}
```

`Create` a New Linked List

- The constructor for a linked list simply sets the head variable to `null`
- The constructor does NOT instantiate a new `Node` … because a list is created to be empty

```
public LinkedList() {
      head = null;
      manyNodes = 0;
}
```

Try it!

- Inside the animal shelter project, create a class called AnimalShelterLinked.

- Add appropriate instance variables.
  - head
  - manyNodes

- Create a constructor. What value will be stored in the instance variables?

Collection class methods

What do we want to be able to do with a collection of objects?

Common collection class method themes:

- Add
- Remove
- Search
- Count
- Get (from a specific position)
- Size

- Iterate?
- Combine?

Unordered Add

- Similar to adding to the next open position in the array
  - arrayName[numItems++] = thingToAdd;
  - Essentially adding to “end” of collection

- For a linked list – think about what you have access to (head)
  - head = new DataTypeNode(thingToAdd, head)
  - Essentially adding to “beginning” of collection

Add at the Front of the Linked List

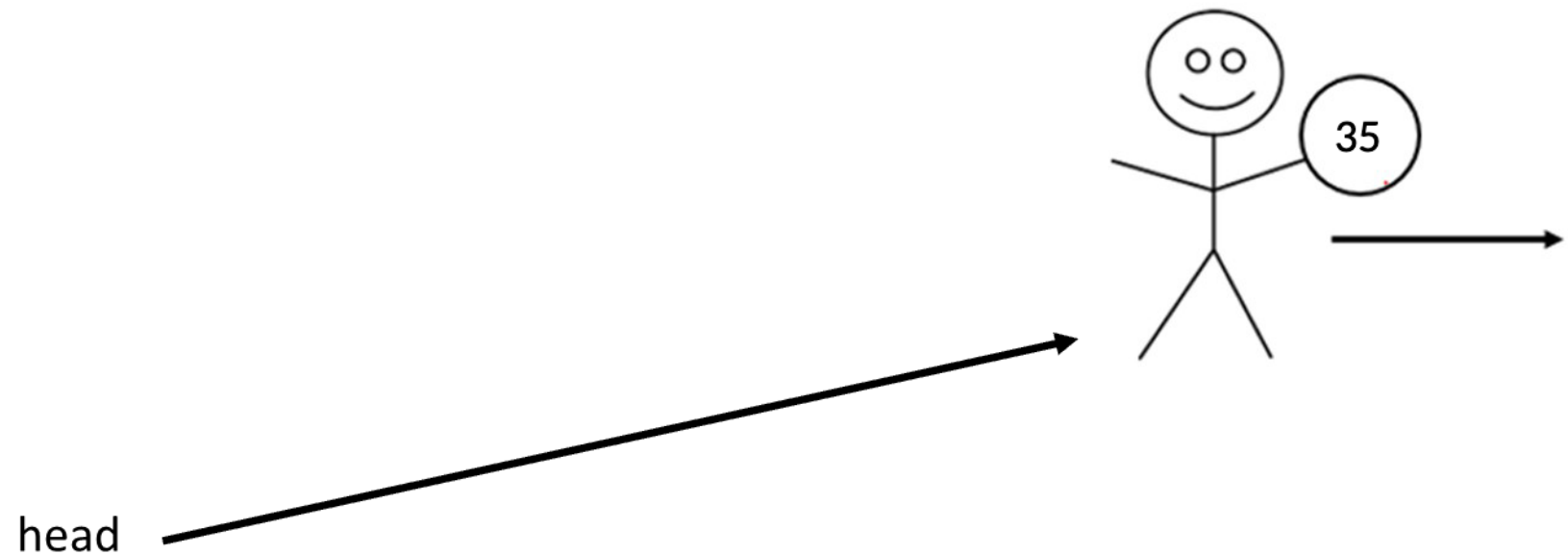
```
IntNode head;
head = new IntNode(35, head);
head = new IntNode (15, head);
head = new IntNode (10, head);
```

Add at the Front of the Linked List

```
IntNode head;
head = new IntNode(35, head);
head = new IntNode (15, head);
head = new IntNode (10, head);
```

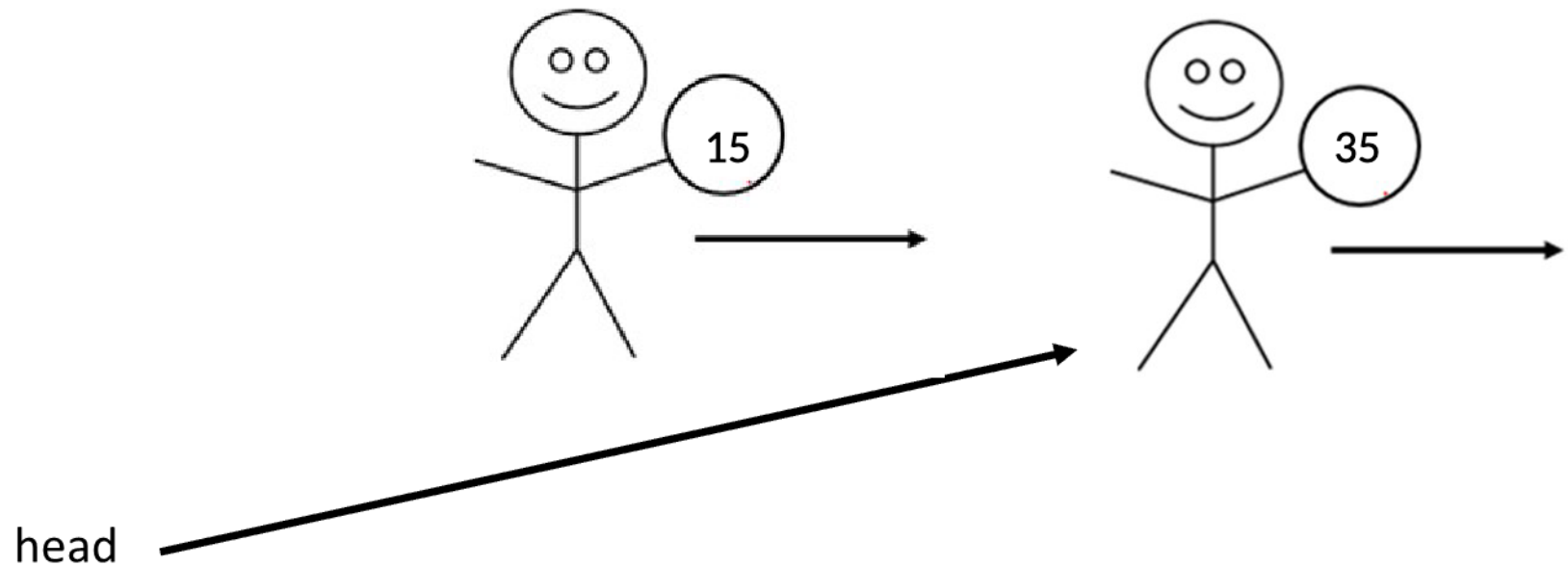
Add at the Front of the Linked List

```
IntNode head;
head = new IntNode(35, head);
head = new IntNode (15, head);
head = new IntNode (10, head);
```



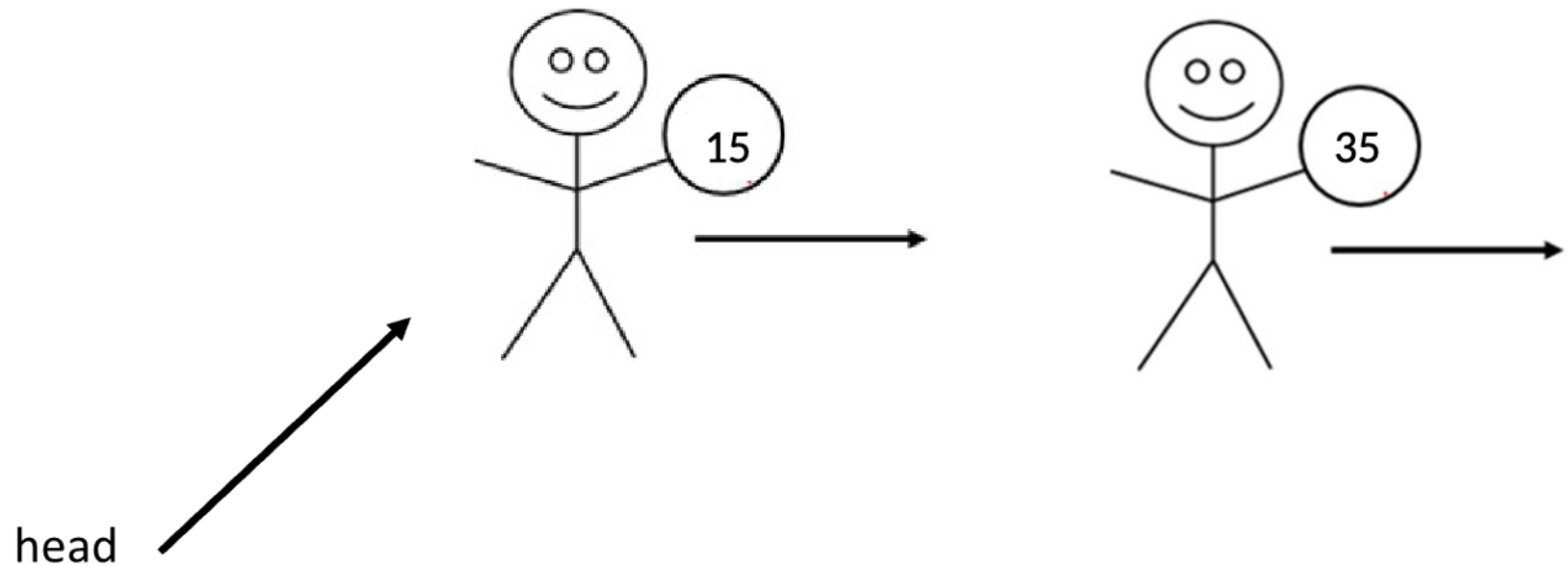
Add at the Front of the Linked List

```
IntNode head;
head = new IntNode(35, head);
head = new IntNode (15, head);
head = new IntNode (10, head);
```



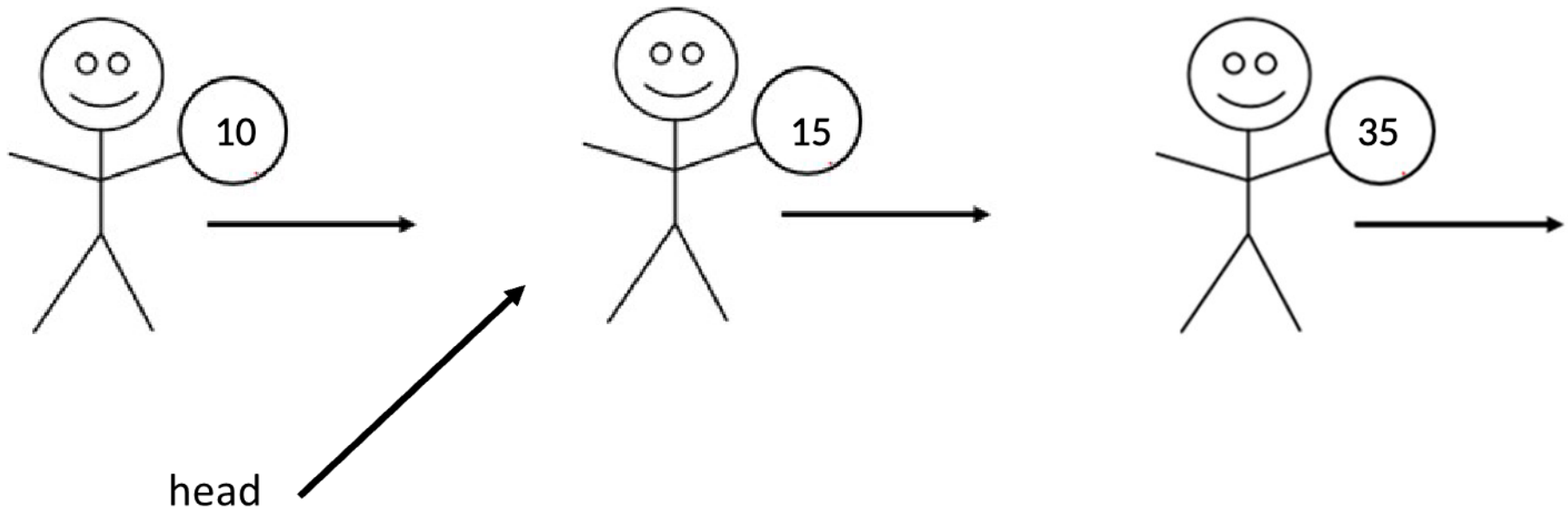
Add at the Front of the Linked List

```
IntNode head;
head = new IntNode(35, head);
head = new IntNode (15, head);
head = new IntNode (10, head);
```



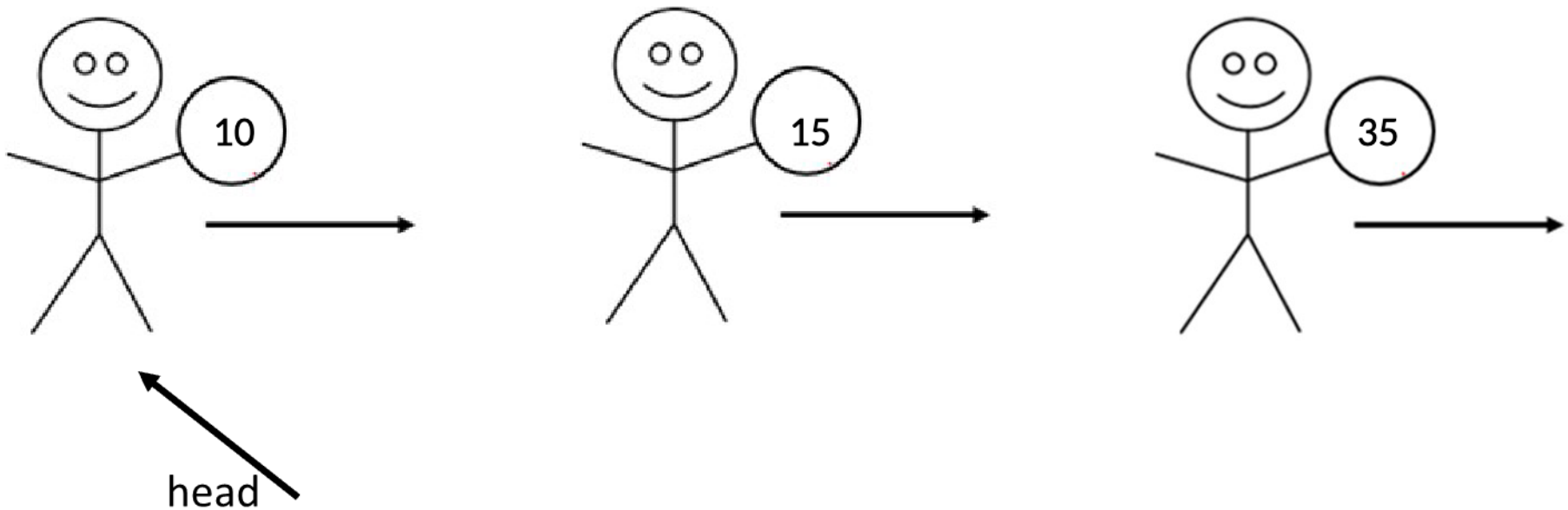
Add at the Front of the Linked List

```
IntNode head;
head = new IntNode(35, head);
head = new IntNode (15, head);
head = new IntNode (10, head);
```



Add at the Front of the Linked List

```
IntNode head;
head = new IntNode(35, head);
head = new IntNode (15, head);
head = new IntNode (10, head);
```



Try it!

- Create an add method in the AnimalShelterLinked class. It will accept an Animal and add it to the linked list.

- In the driver class (you can use the same driver as before), create a linked list animal shelter. Add 3 or 4 animals to the shelter.

- We will try printing it – but first we need one more concept

Traversing a linked list

How do we move through the data structure?

How to Traverse a Linked List

- There is a pattern that can be used whenever you need to step through all the nodes of a linked list one at a time

- The steps are as follows:
  - Start a cursor to refer to the head of the list

    `Node cursor = head;`
  - To move the cursor to the next node, we use

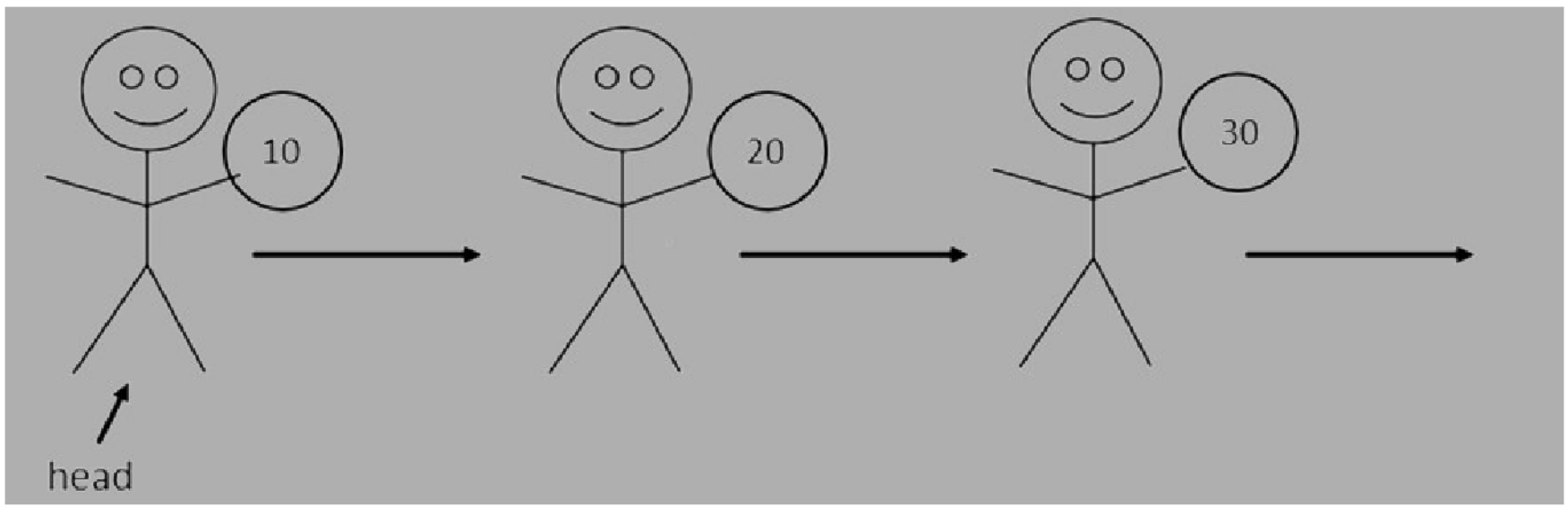
    `cursor = cursor.getLink();`
  - The loop should terminate when `cursor is null` because this means there are no more nodes in the list

```
Node cursor = head;
while (cursor != null){
        //do something with

        //the current node
        cursor =
cursor.getLink();
}
```

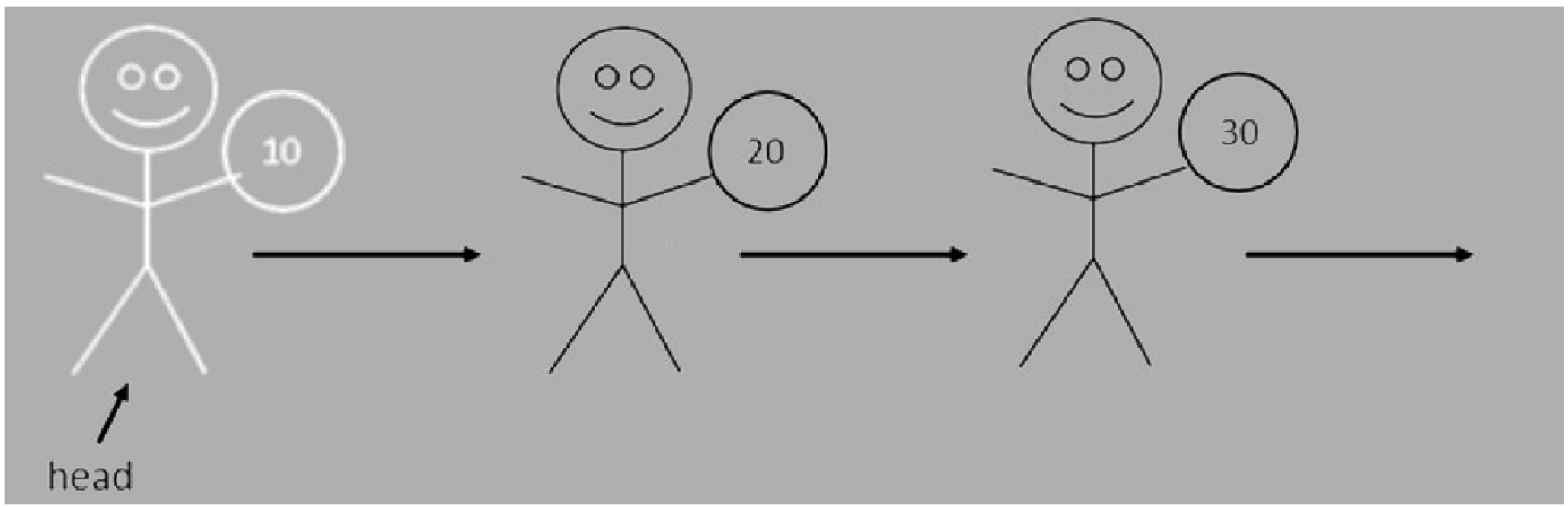
Traverse a Linked List

```
IntNode cursor = head;
while (cursor != null){
          System.out.println(cursor.getData());
          cursor = cursor.getLink();
}
```

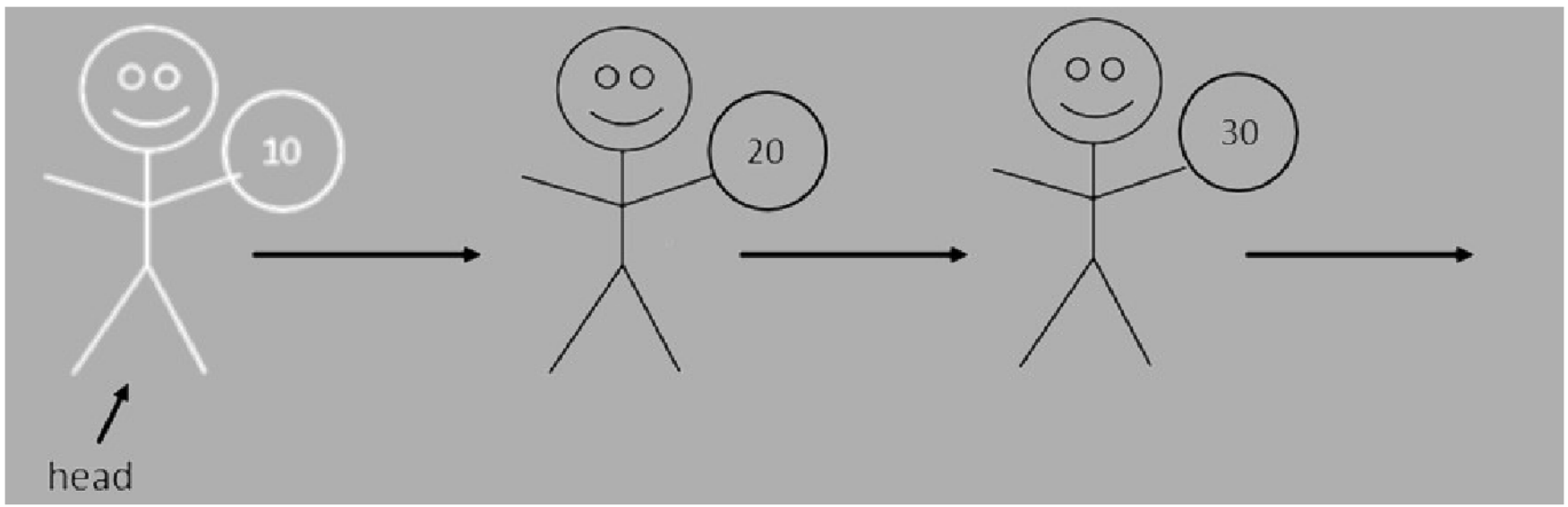
Traverse a Linked List

```
IntNode cursor = head;
while (cursor != null){
          System.out.println(cursor.getData());
          cursor = cursor.getLink();
}
```



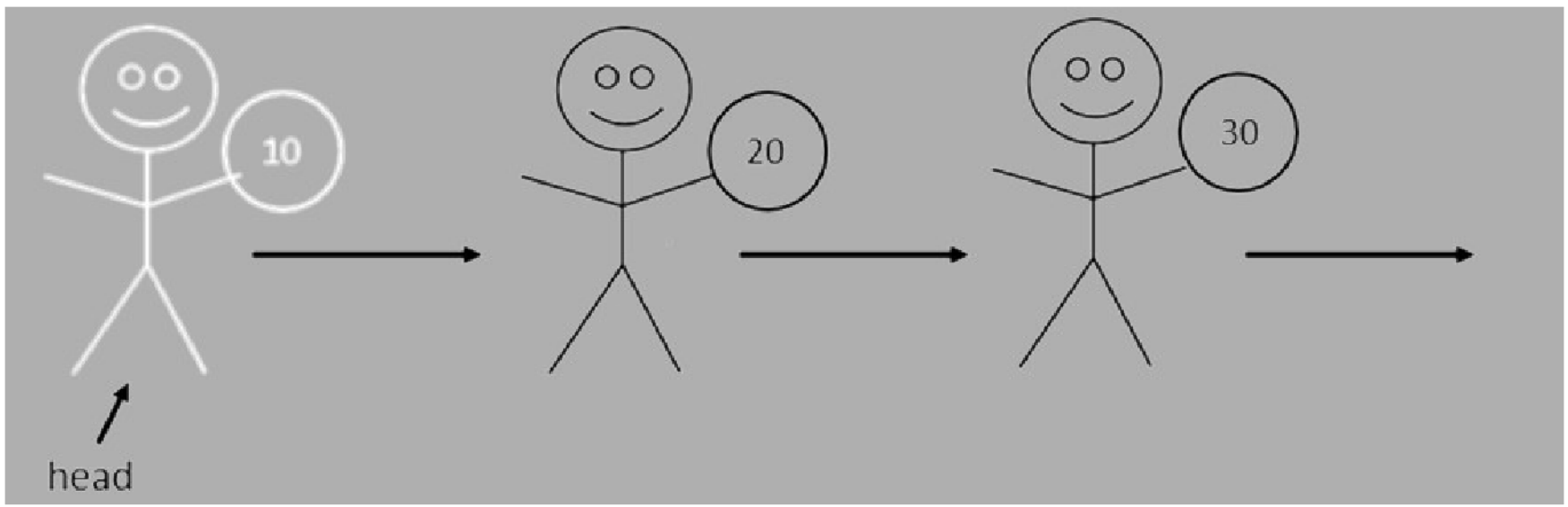
Traverse a Linked List

```
IntNode cursor = head;
while (cursor != null){
          System.out.println(cursor.getData());
          cursor = cursor.getLink();
}
```



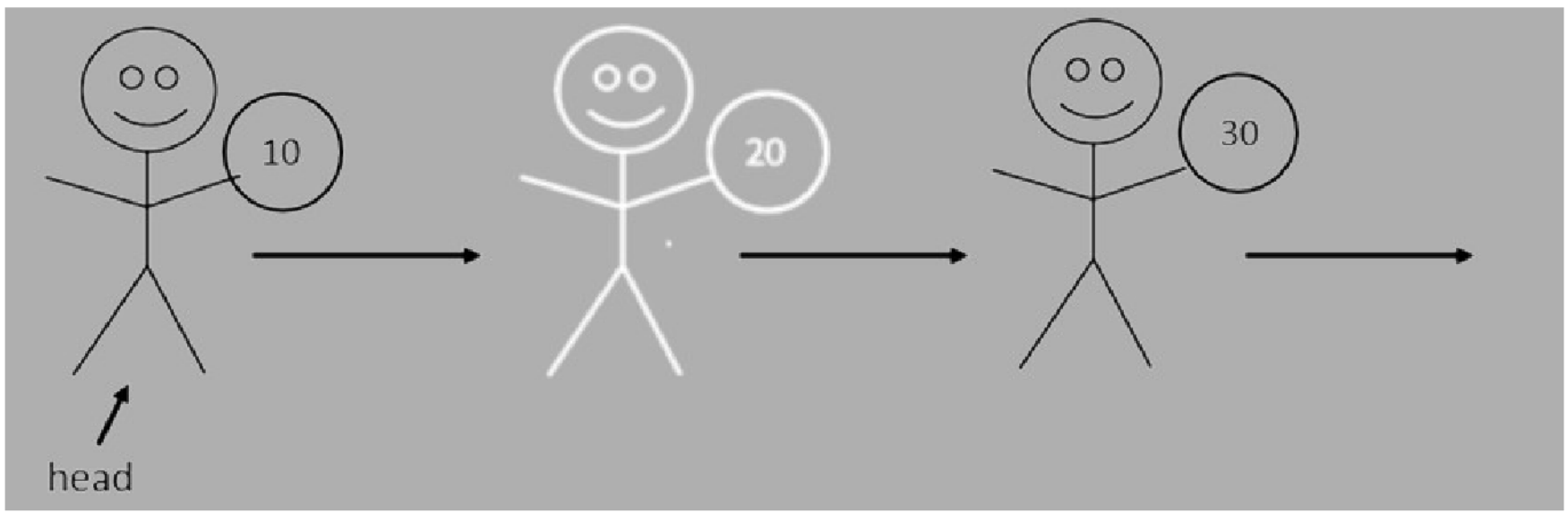
Traverse a Linked List


```
IntNode cursor = head;
while (cursor != null){
        System.out.println(cursor.getData());
        cursor = cursor.getLink();
}
```



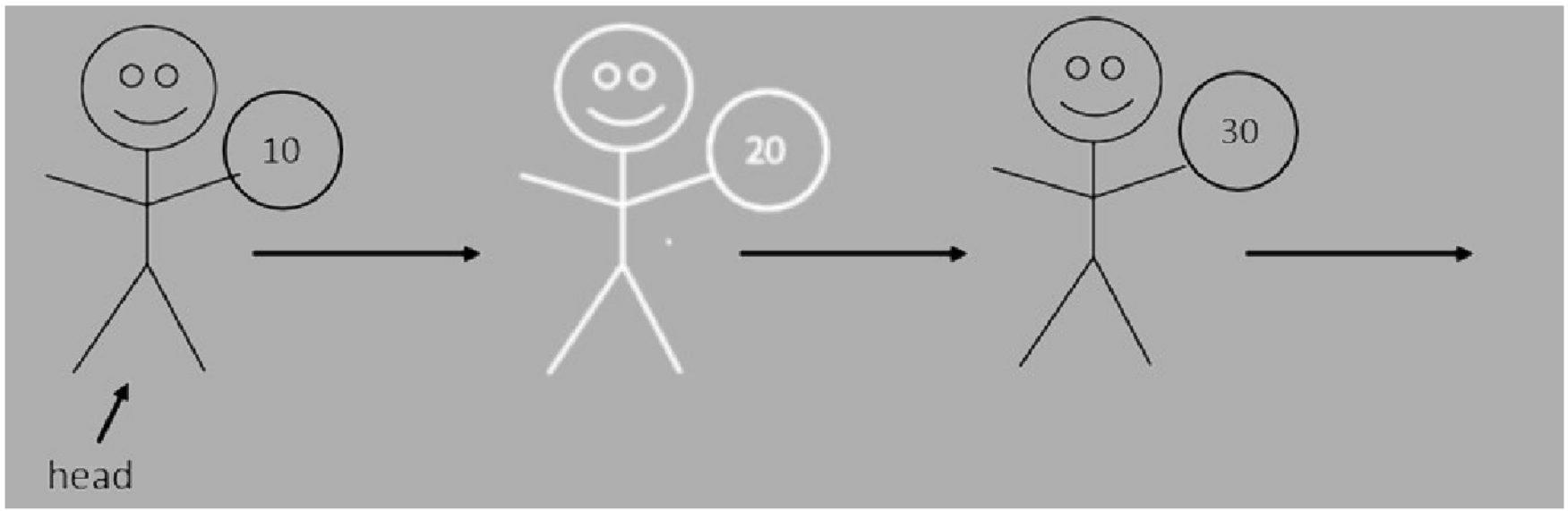
Traverse a Linked List


```
IntNode cursor = head;
while (cursor != null){
          System.out.println(cursor.getData());
          cursor = cursor.getLink();
}
```



Traverse a Linked List

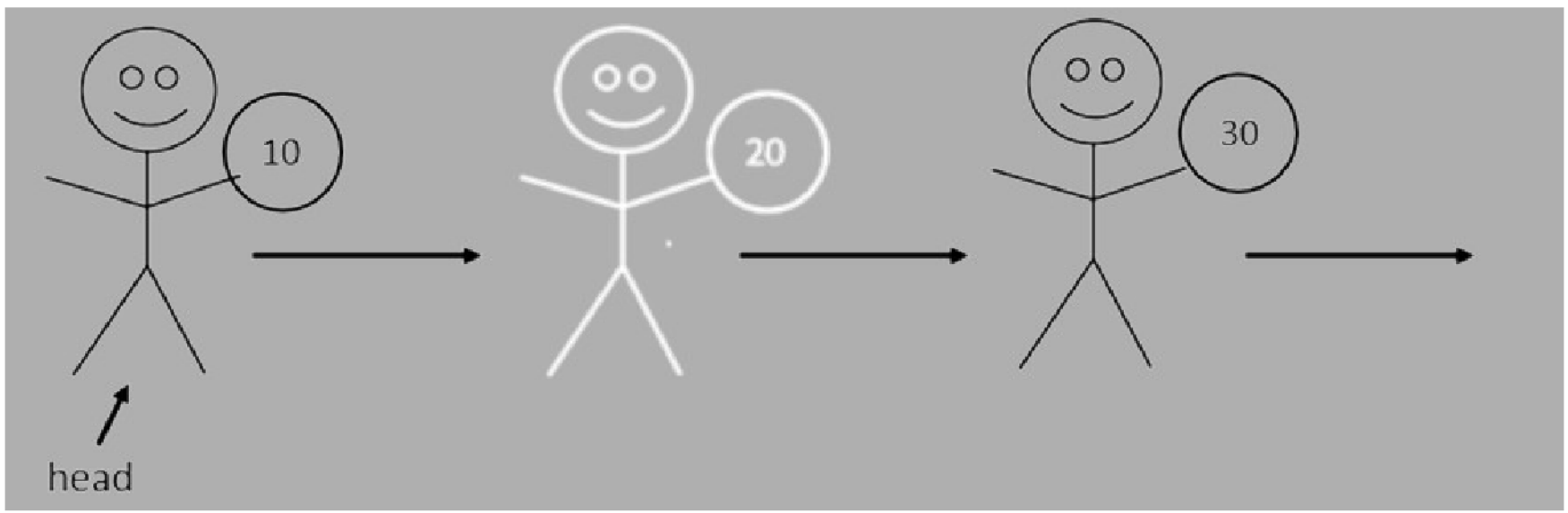

```
IntNode cursor = head;
while (cursor != null){
          System.out.println(cursor.getData());
          cursor = cursor.getLink();
}
```



Traverse a Linked List

```
IntNode cursor = head;
while (cursor != null){
        System.out.println(cursor.getData());
        cursor = cursor.getLink();
}
```

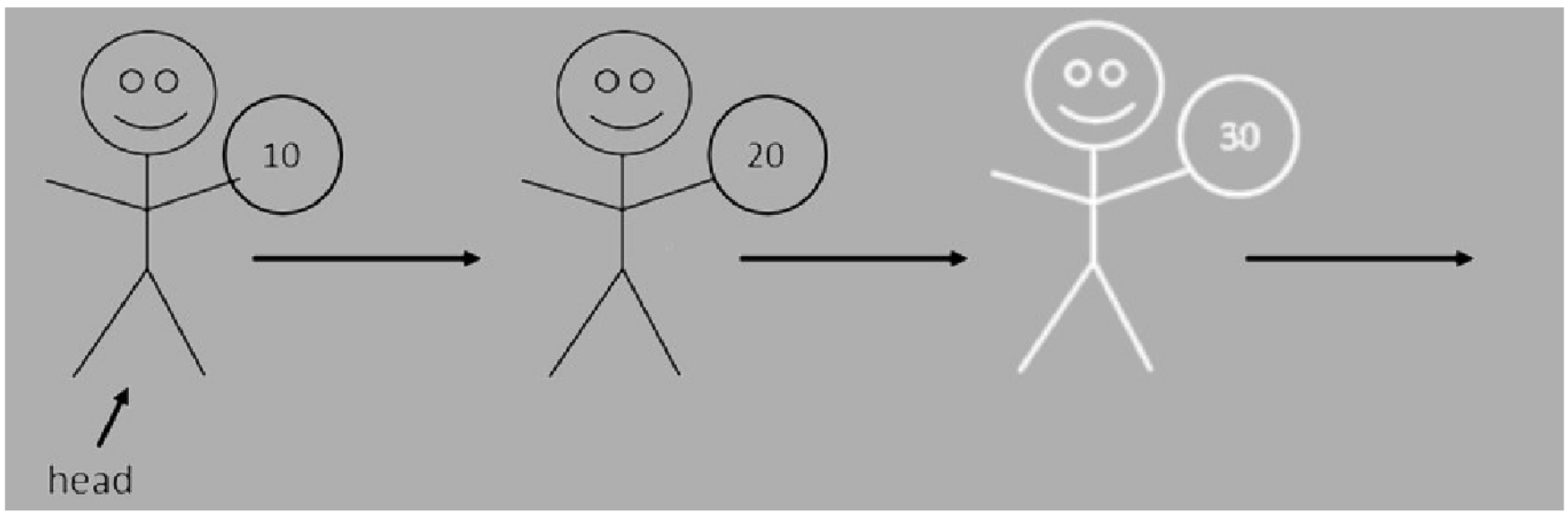
10
20



Traverse a Linked List

```
IntNode cursor = head;
while (cursor != null){
        System.out.println(cursor.getData());
        cursor = cursor.getLink();
}
```

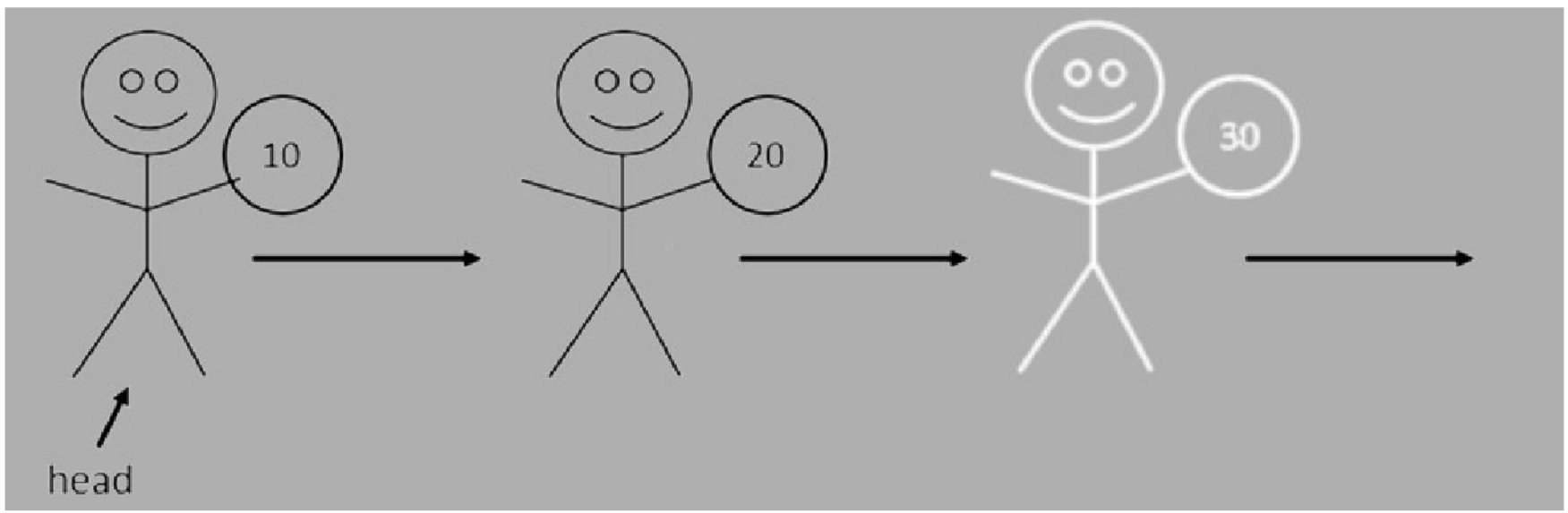
10
20



Traverse a Linked List

```
IntNode cursor = head;
while (cursor != null){
          System.out.println(cursor.getData());
          cursor = cursor.getLink();
}
```

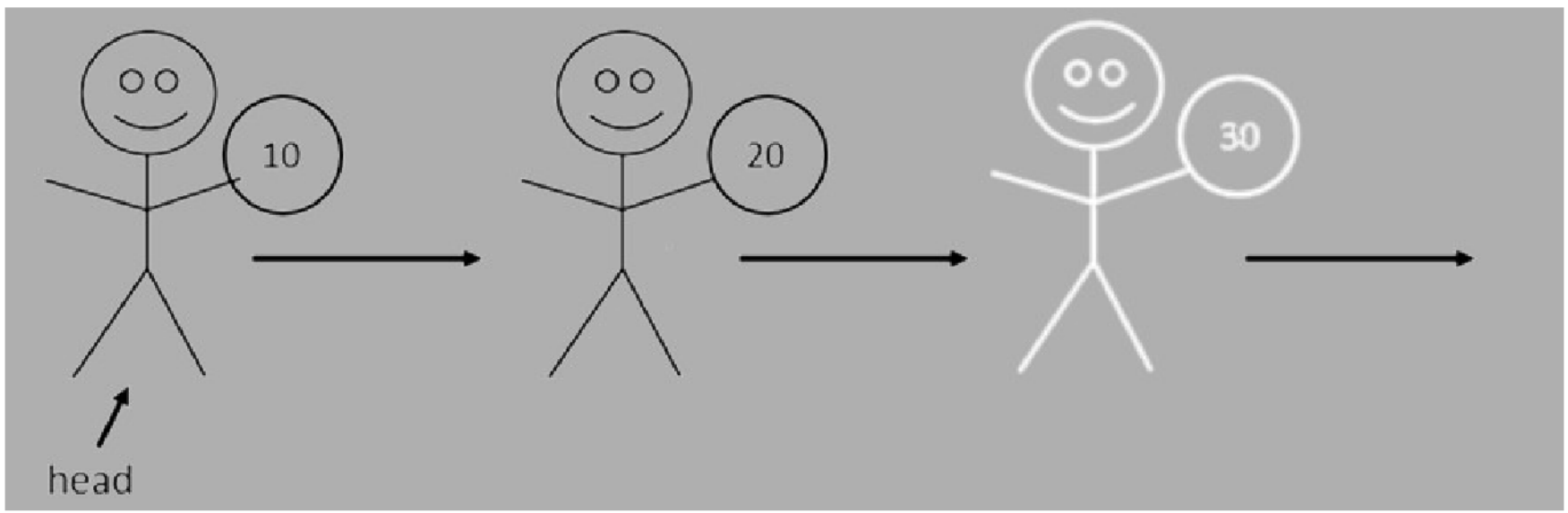
10
20



Traverse a Linked List

```
IntNode cursor = head;
while (cursor != null){
        System.out.println(cursor.getData());
        cursor = cursor.getLink();
}
```

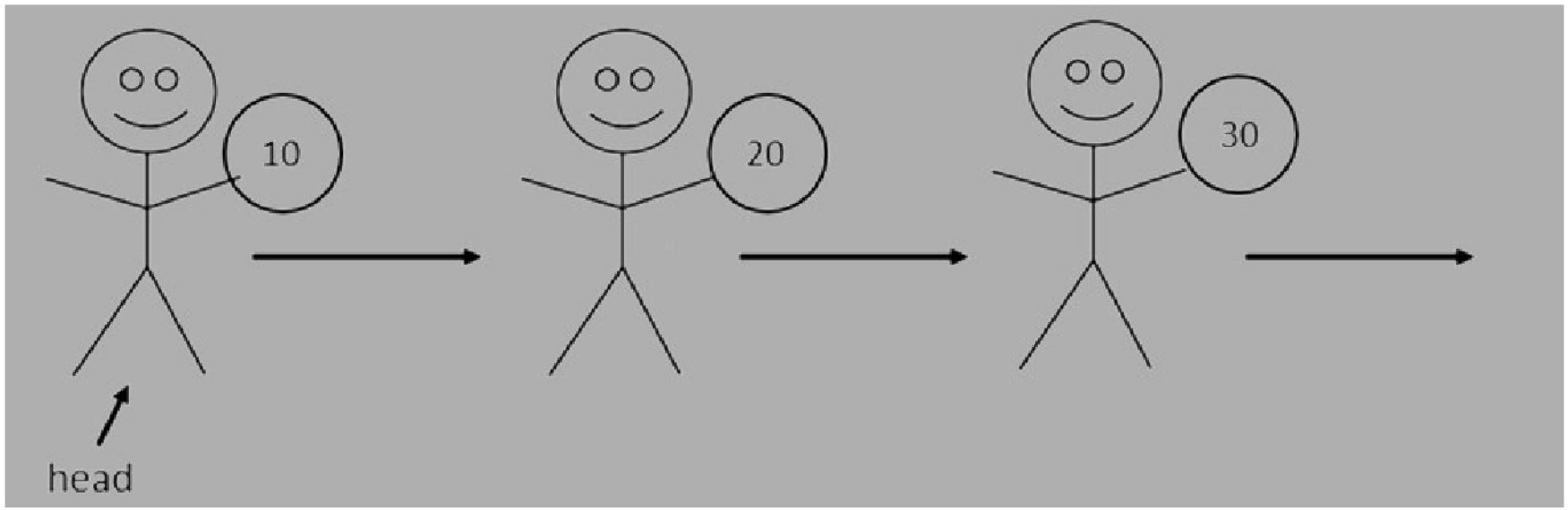
10
20
30

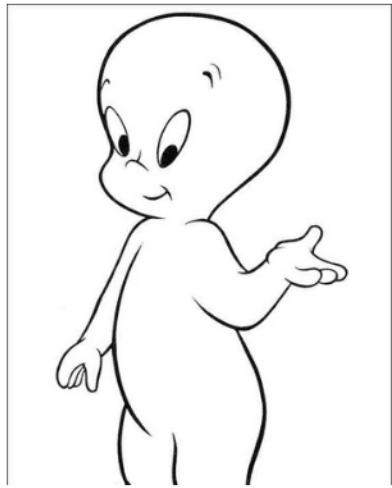


Traverse a Linked List

```
IntNode cursor = head;
while (cursor != null){
        System.out.println(cursor.getData());
        cursor = cursor.getLink();
}
```

10
20
30

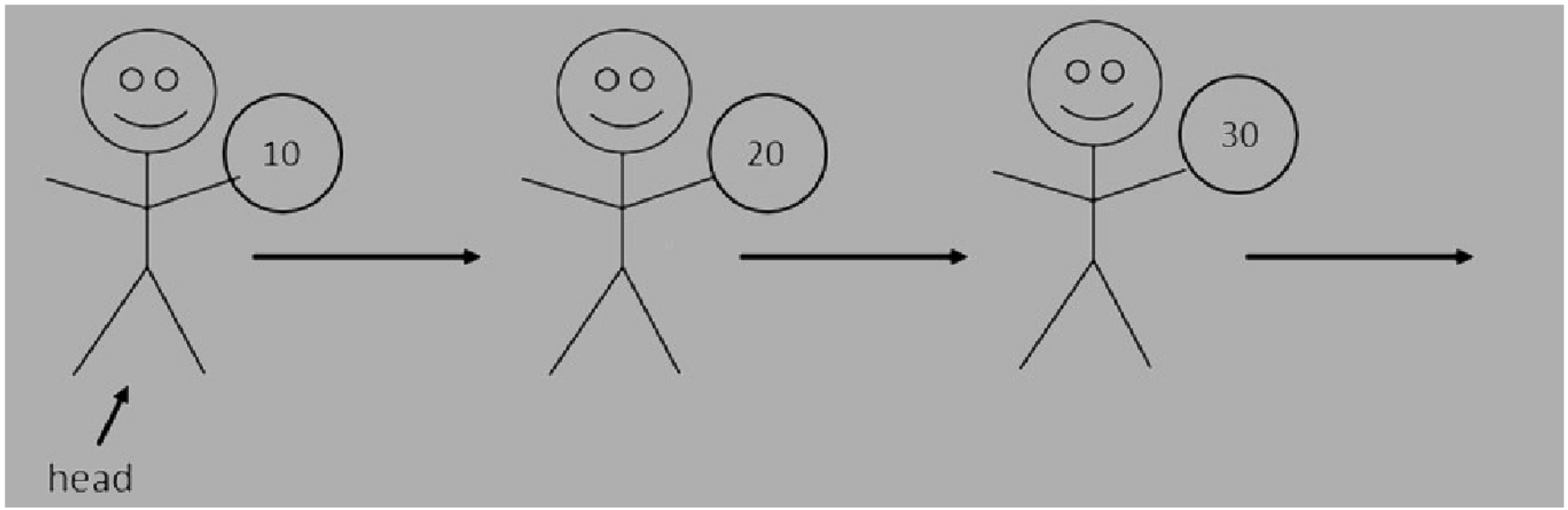




Traverse a Linked List

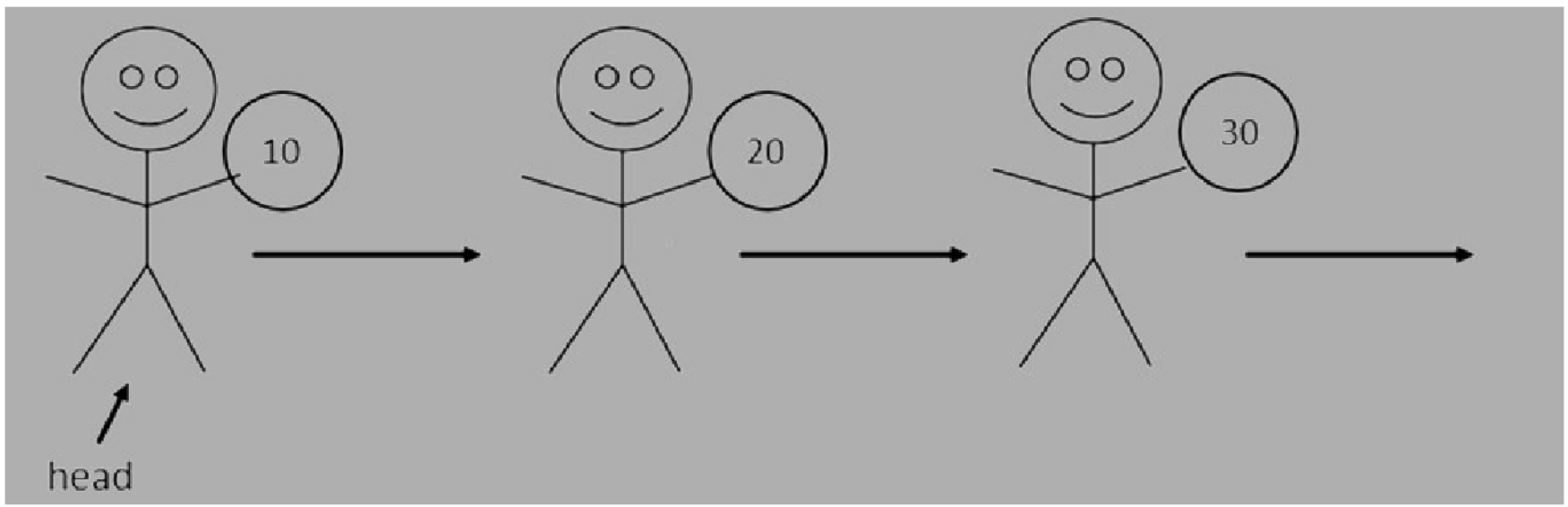
```
IntNode cursor = head;
while (cursor != null){
          System.out.println(cursor.getData());
          cursor = cursor.getLink();
}
```

10
20
30



Traverse a Linked List

```
IntNode cursor = head;
while (cursor != null){
          System.out.println(cursor.getData());
          cursor = cursor.getLink();
}
```



Try it!

- In the AnimalShelterLinked class, create a toString. This should return a string that includes a header and information about all the animals in the shelter.

Add node

Somewhere other than the front

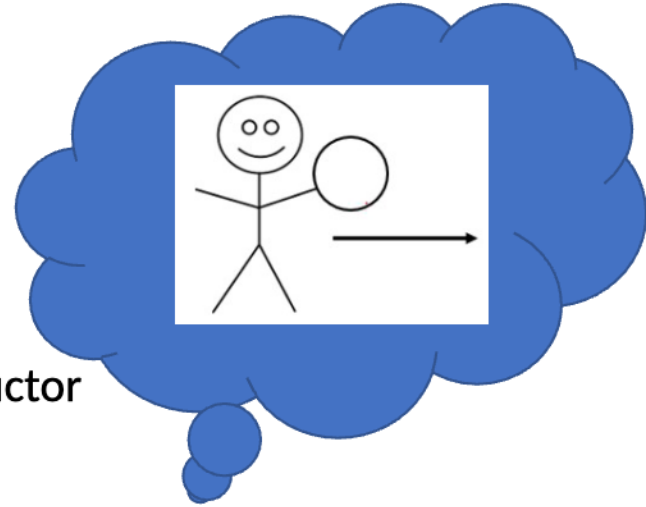
Inserting a Node after another Node

- Note that if we are not inserting at the front of the list, then there must exist a node, let's call it previousNode, that our new node is going to be "behind". In other words, our new node will either be the head of the line or be directly after previousNode.

- Let's consider the process for adding a node after another node.

Insert After a Specific Node

Insert After a Specific Node



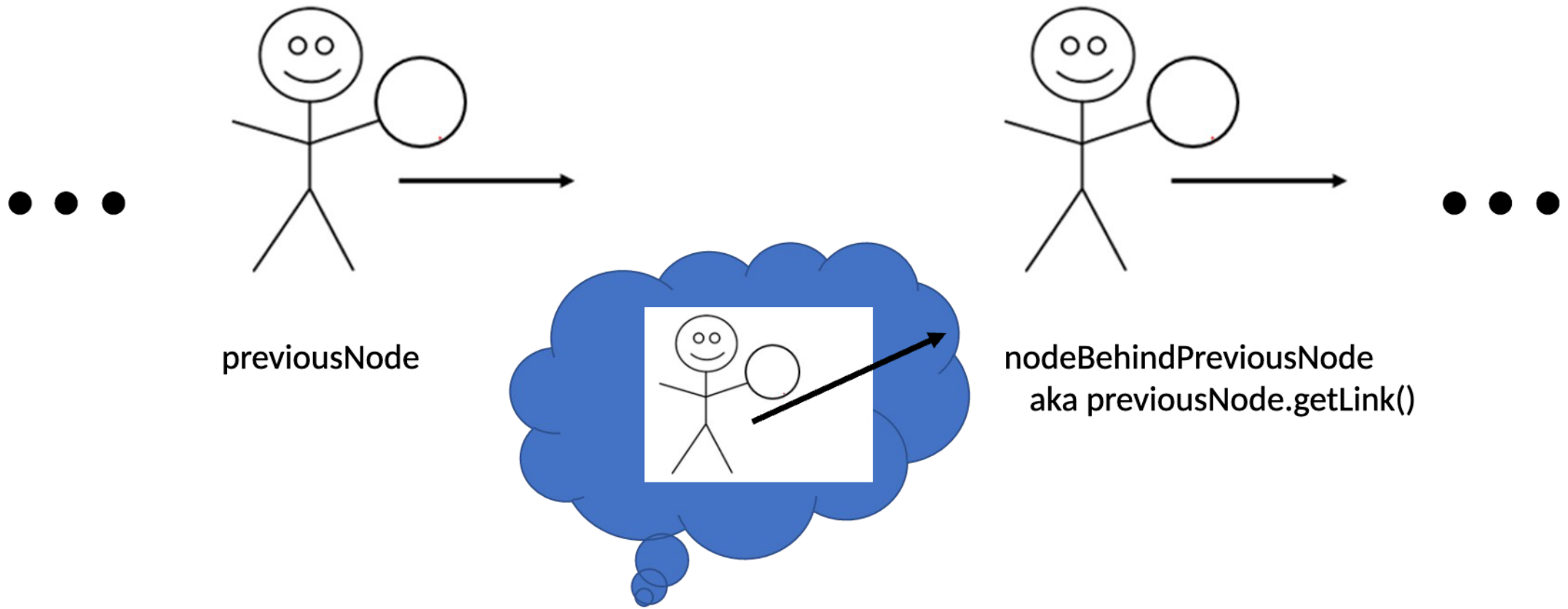
Create a new node to hold the element to insert – recall that we need to send the constructor for the node an element (what it's holding) and a link (who is behind it). The element is whatever we want to insert into the collection. What will be the link?

previousNode

nodeBehindPreviousNode

Insert After a Specific Node

Create a new node to hold the element to insert – recall that we need to send the constructor for the node an element (what it's holding) and a link (who is behind it). The element is whatever we want to insert into the collection. What will be the link?

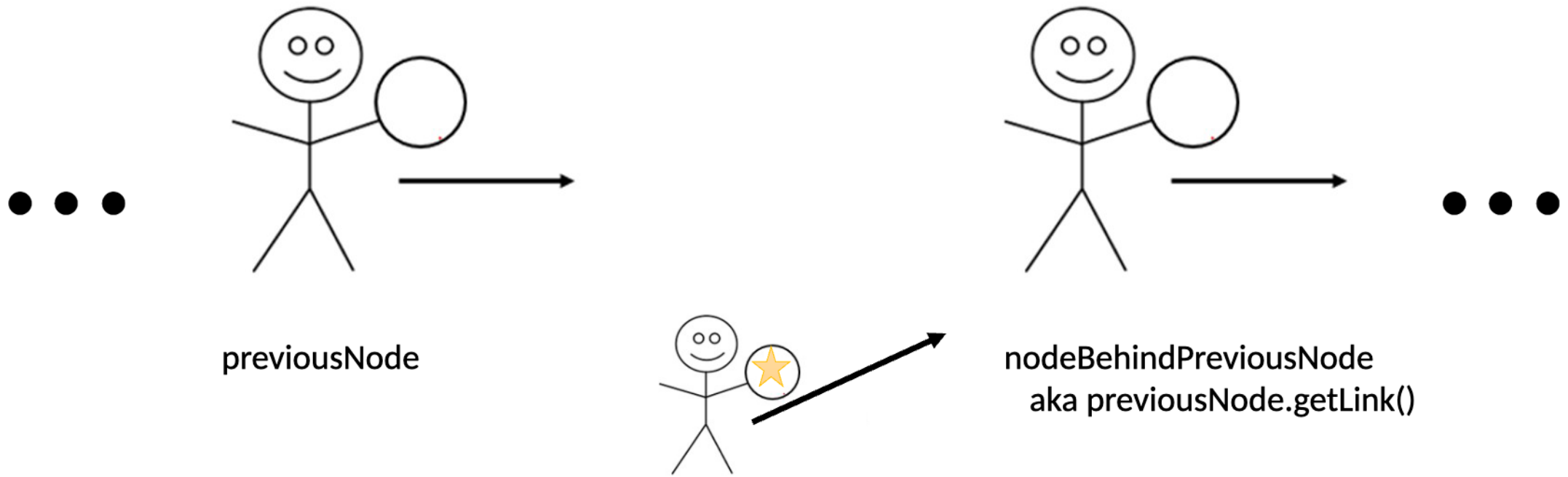


previousNode

nodeBehindPreviousNode
aka previousNode.getLink()

Insert After a Specific Node

Node newNode = new Node(elementToAdd, previousNode.getLink())

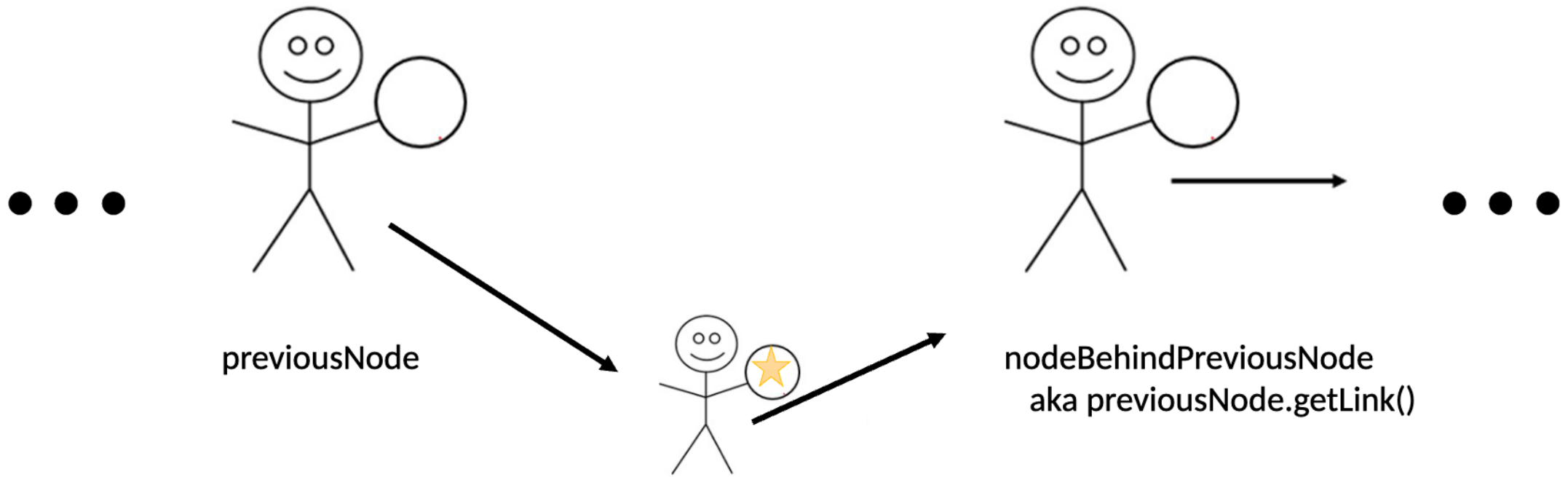


previousNode

nodeBehindPreviousNode
aka previousNode.getLink()

Insert After a Specific Node

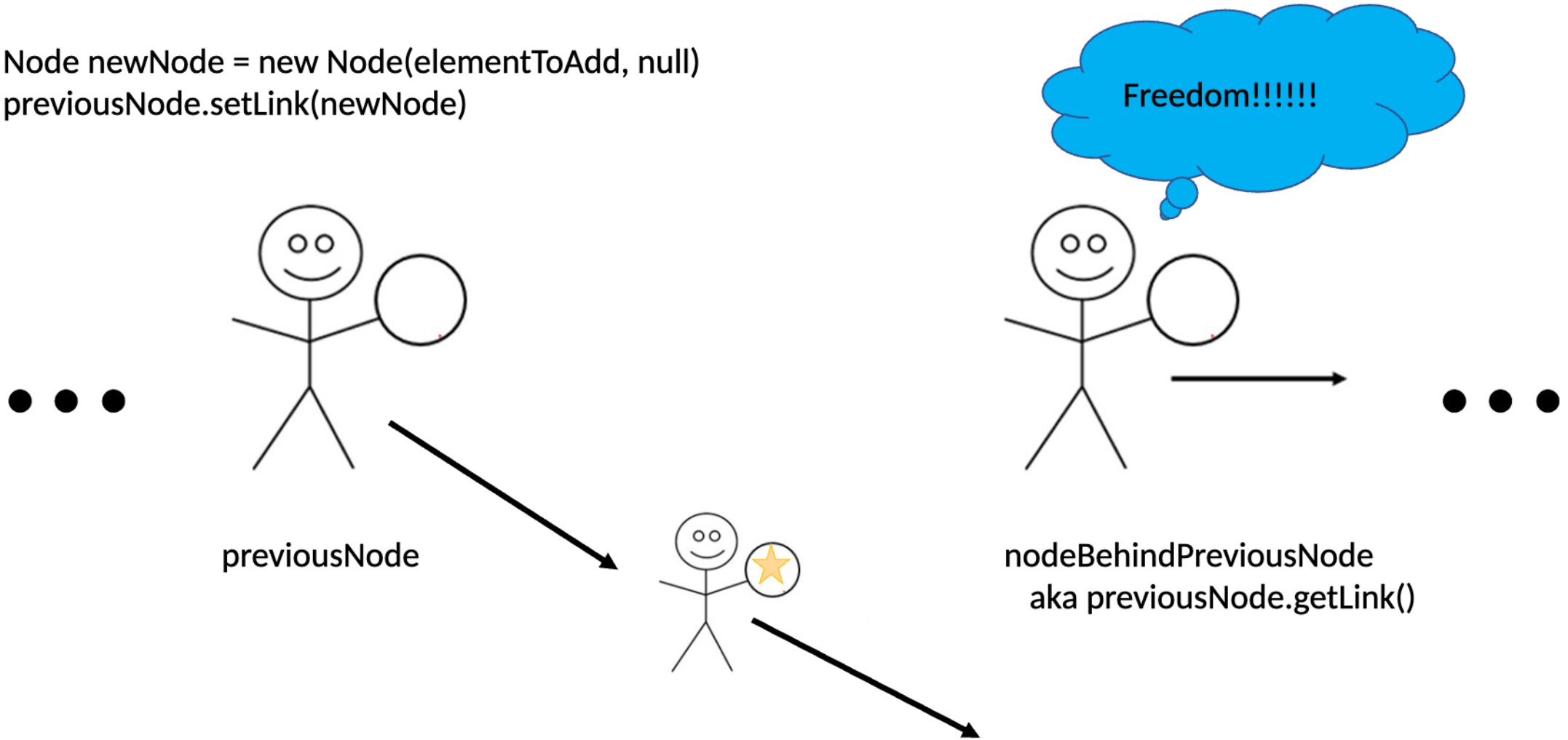
previousNode.setLink(newNode)



Don’t accidently unlink your linked list!!

Insert After a Specific Node

Node newNode = new Node(elementToAdd, null)
previousNode.setLink(newNode)



Now to find previousNode

- Consider where you want to place the new element:
  - A specific position. (8th).
    - What if there are only 5 things in your collection? Either insert in 6th position, or fail the insert.
    - Linked lists typically start counting at 1. (1st position)
    - If I want my item to be 8th in the list, which item would be the previous node? If the head is the 1st node, how many times will the cursor jump to the next node?
  - A specific ordering. (Decreasing, Increasing)
    - When will cursor stop jumping?

Pseudocode for `findPrevious()`

- For each Node in the list
  - If the **next** Node contains the target then
    - Return the current node (this is the predecessor)
  - Else
    - Advance to the next Node

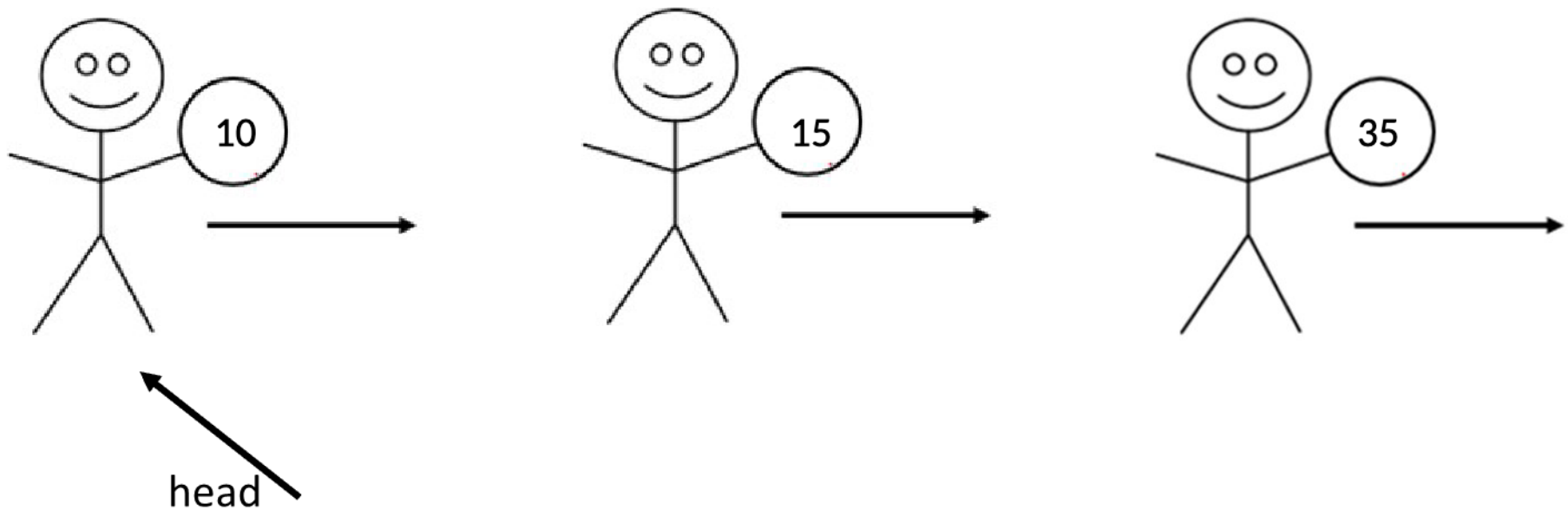
- Can be done in same method as add, or can create helper method(s).
  - Let's create a helper method.
  - Helper methods are private (not intended for other classes to use)

Try it!

Modify the add method in the AnimalShelterLinked class so that the collection stays sorted. Feel free to use or not use a helper method.

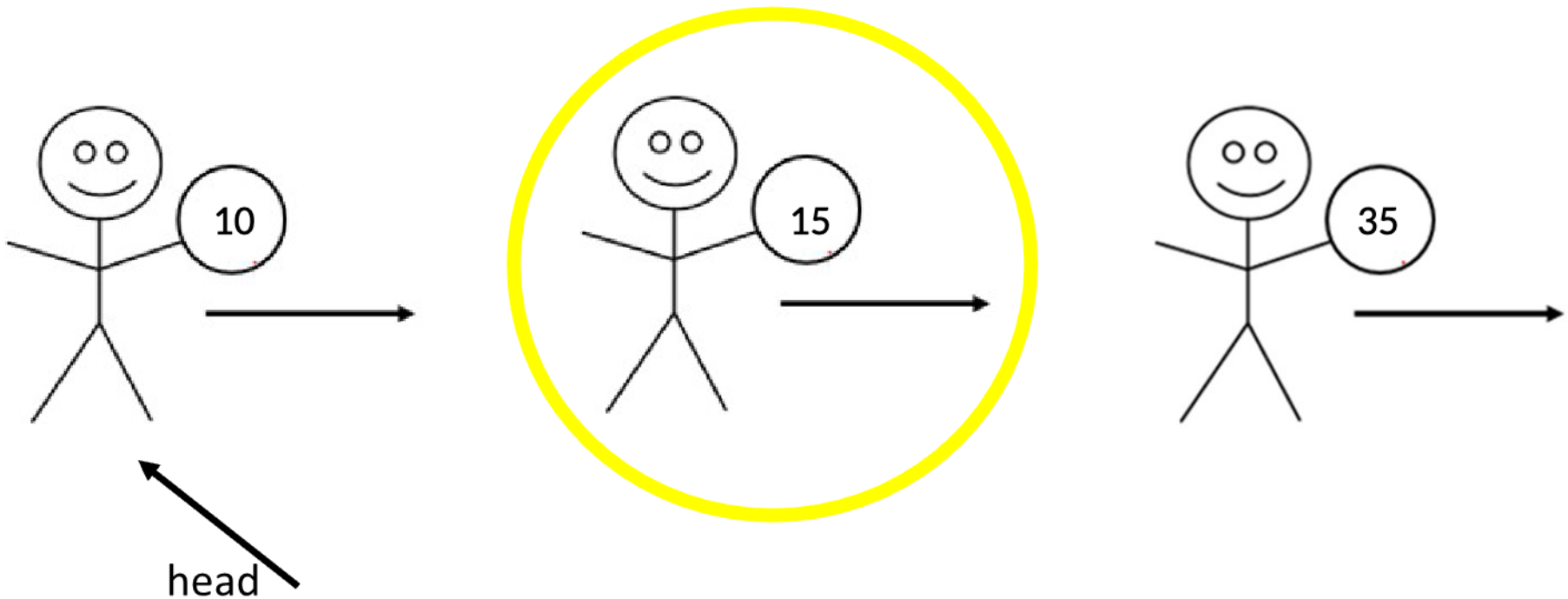
Delete node

Delete from the Front of the Linked List



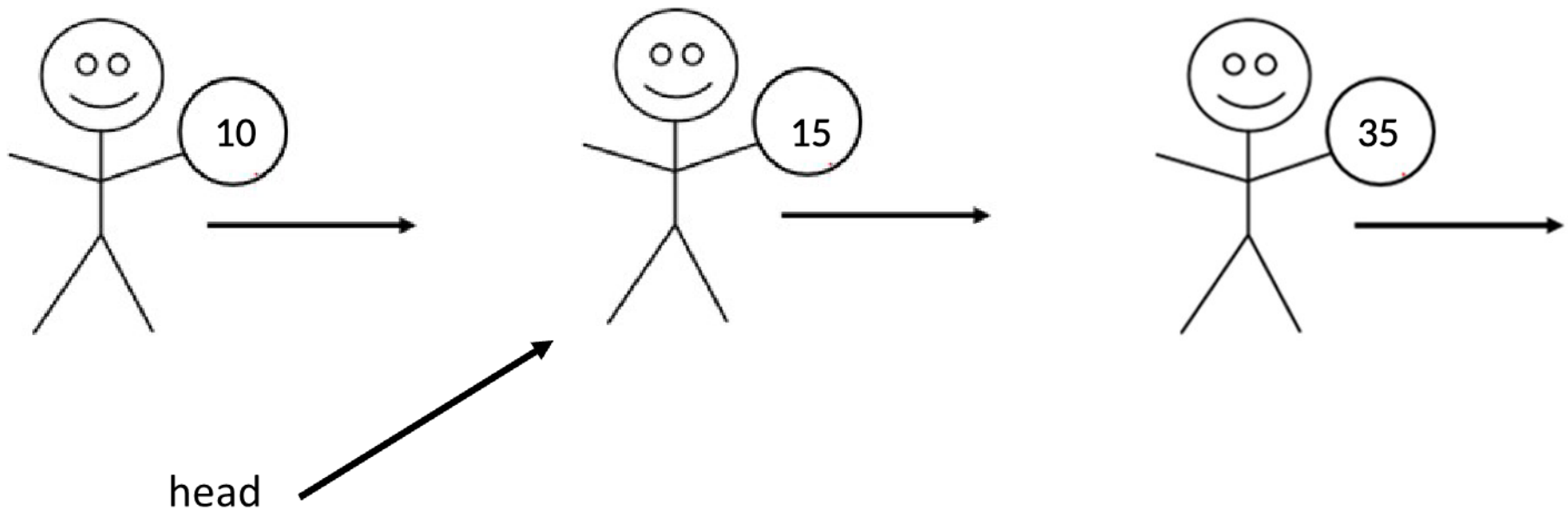
Delete from the Front of the Linked List

head = head.getLink()

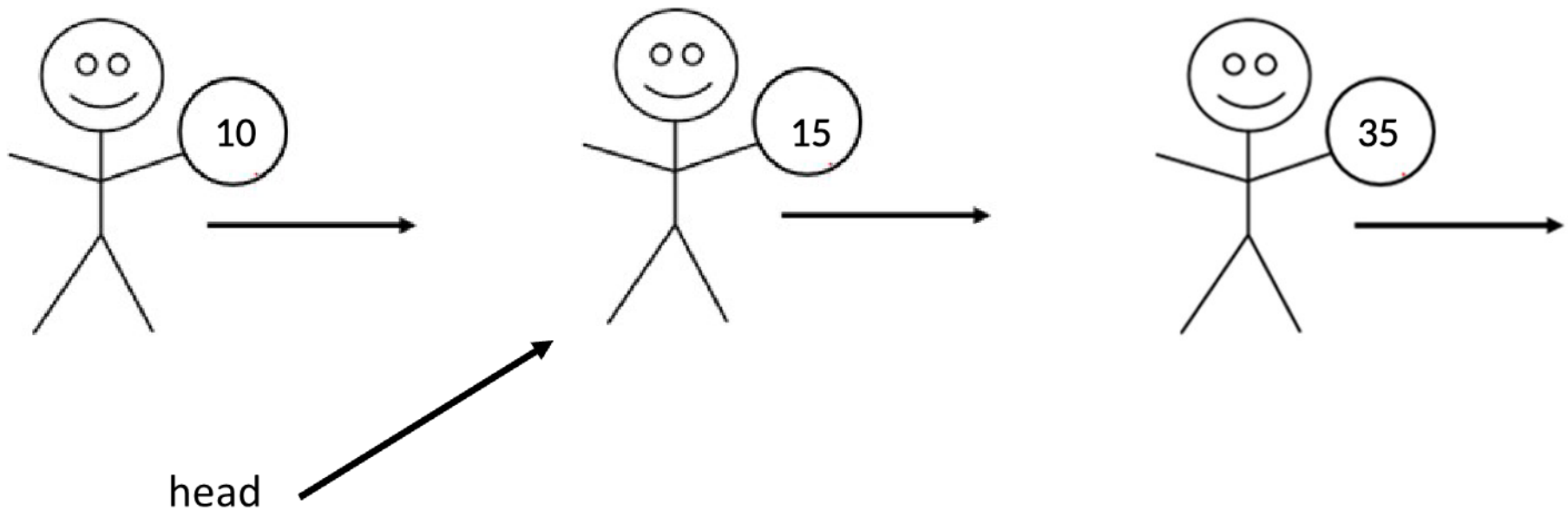


Delete from the Front of the Linked List

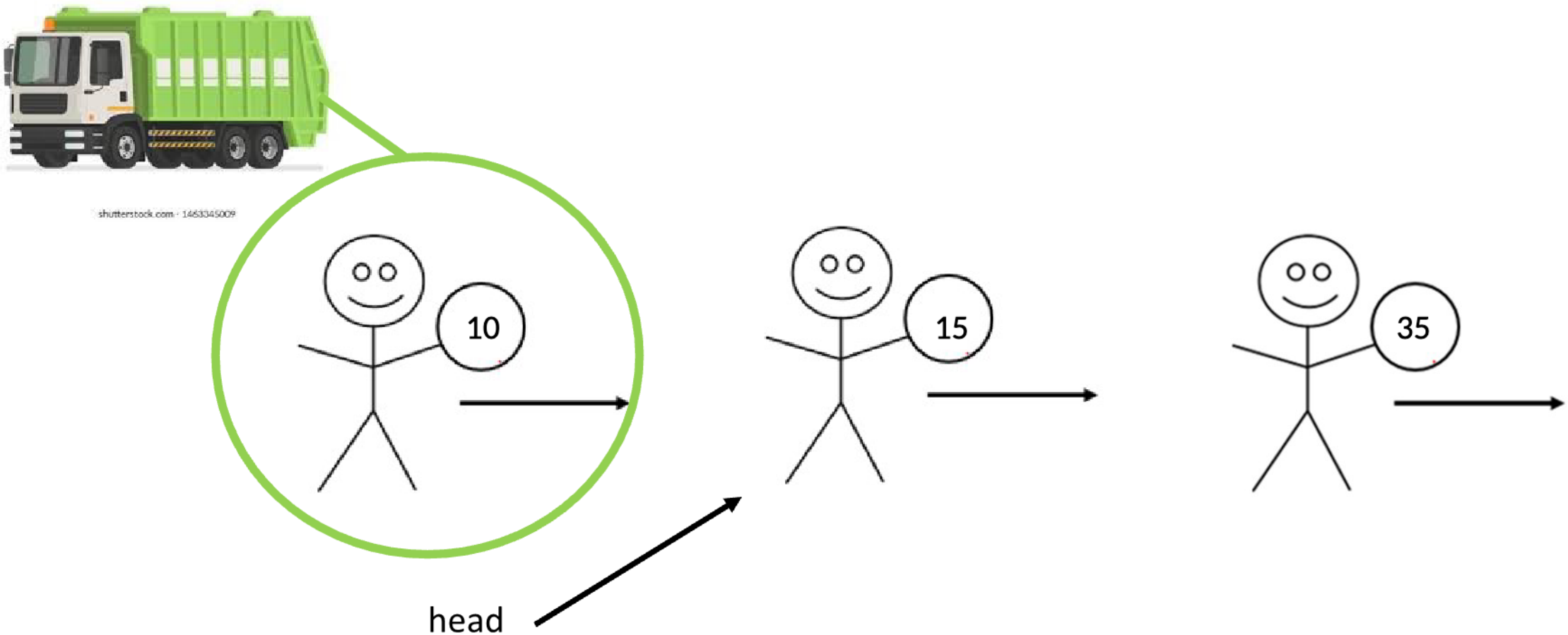
head = head.getLink()



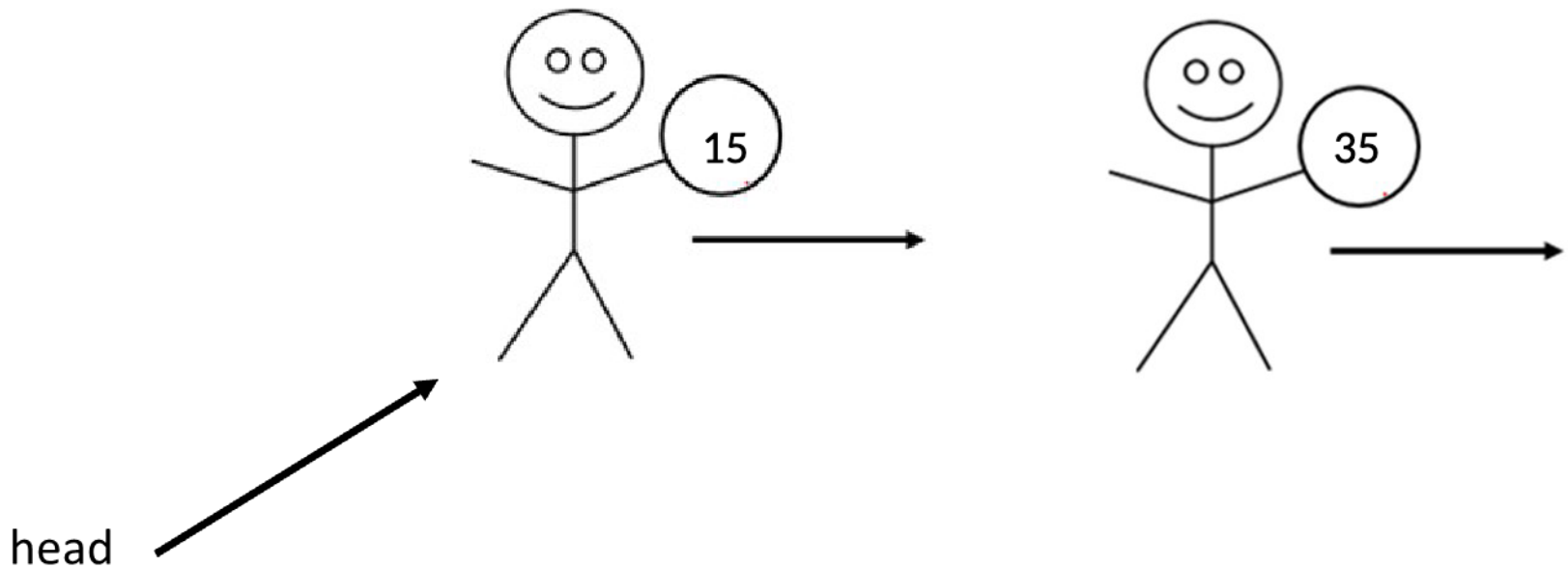
Delete from the Front of the Linked List



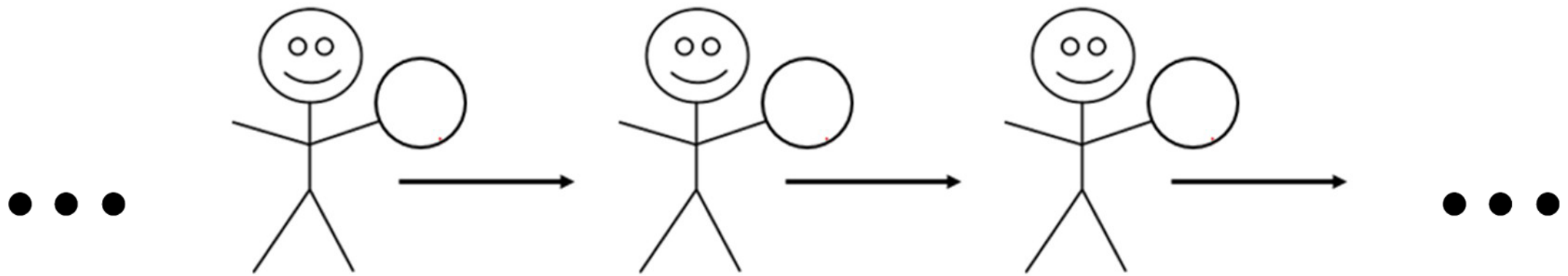
Delete from the Front of the Linked List



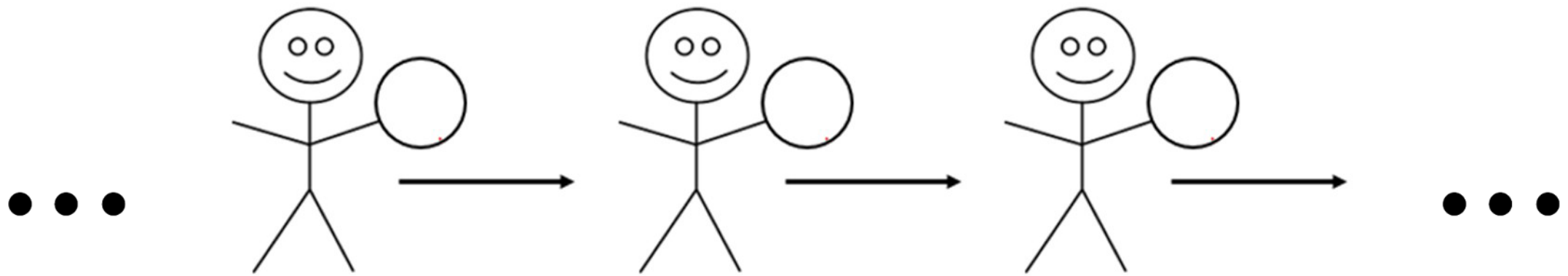
Delete from the Front of the Linked List



Delete After a Specific Node

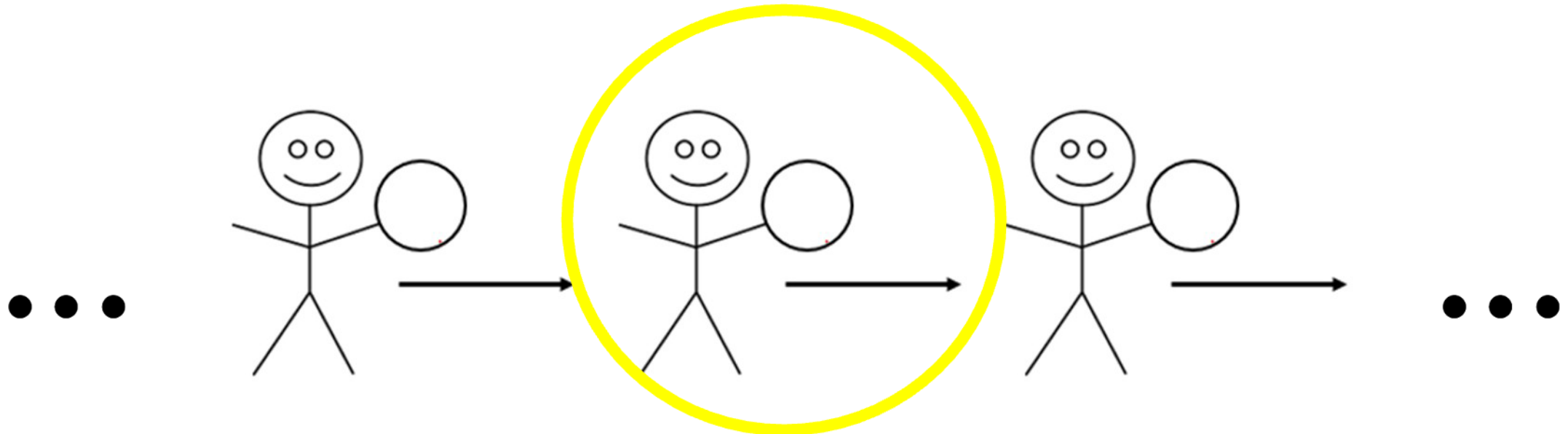


Delete After a Specific Node



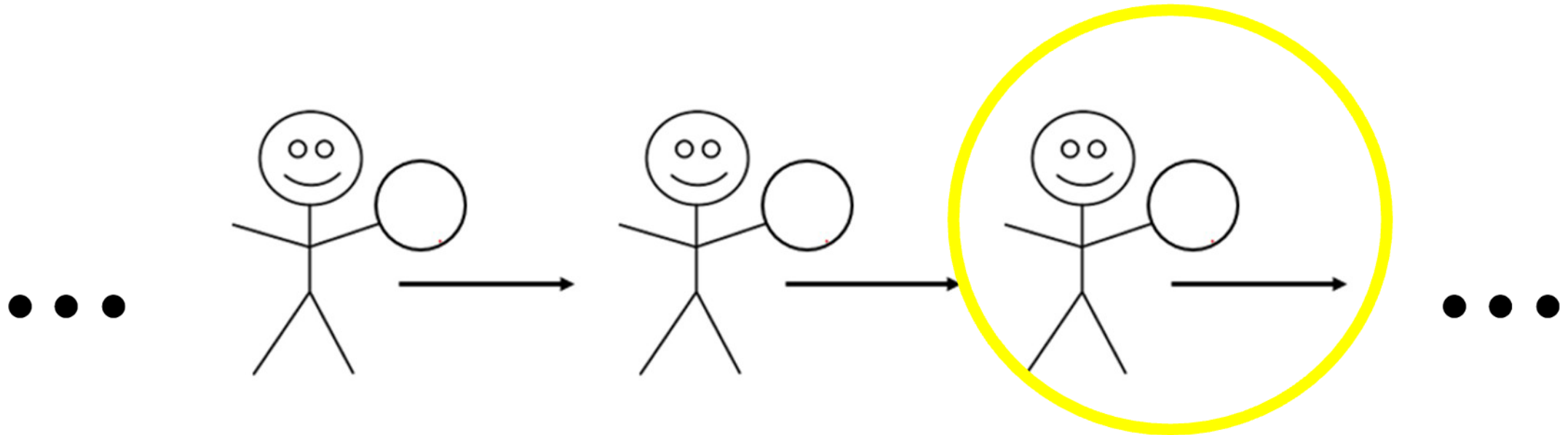
previousNode.setLink(previousNode.getLink().getLink())

Delete After a Specific Node



previousNode.setLink(previousNode.getLink().getLink())

Delete After a Specific Node



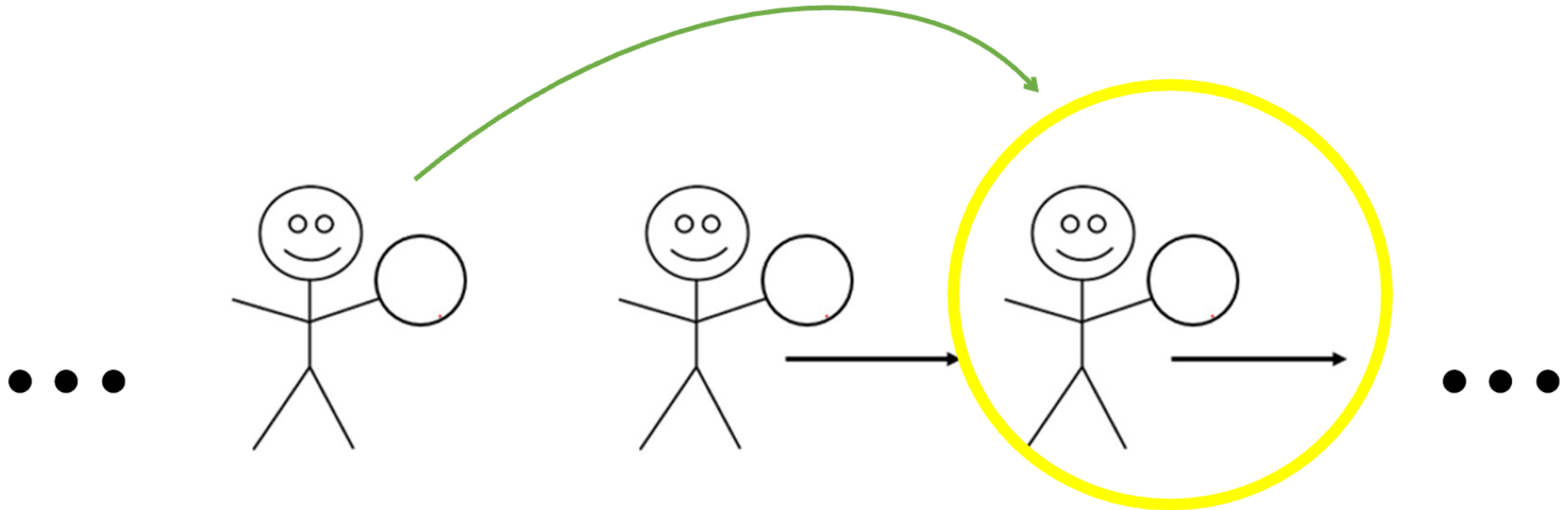
previousNode

nodeBehindPreviousNode
aka previousNode.getLink()

nodeBehindNodeBehindPreviousNode
aka previousNode.getLink().getLink()

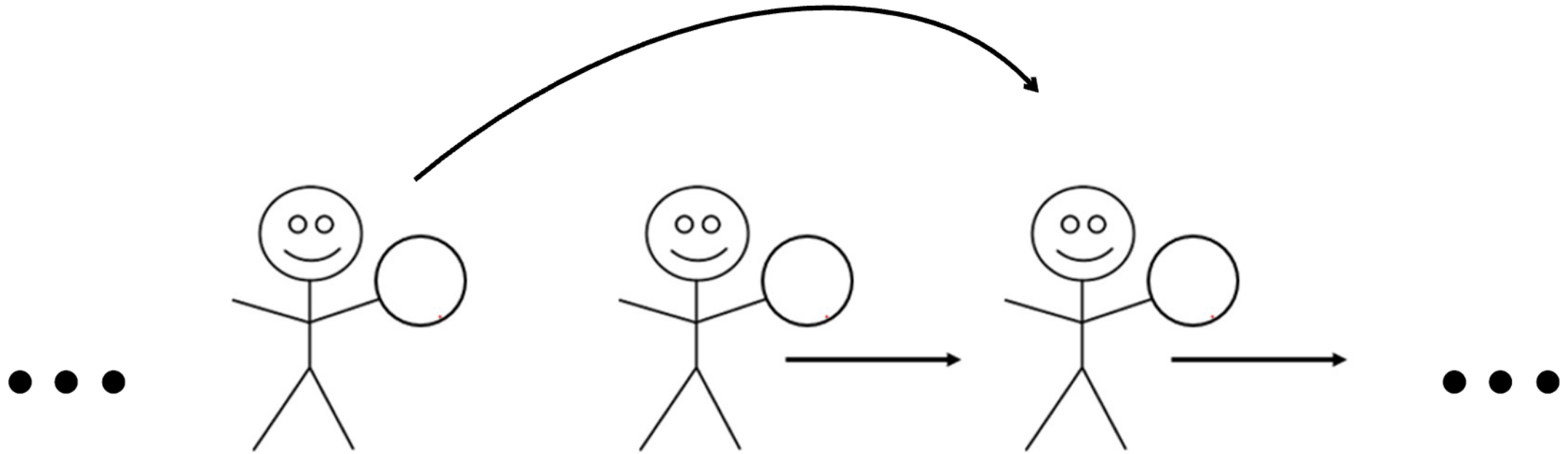
previousNode.setLink(previousNode.getLink().getLink())

Delete After a Specific Node



previousNode.setLink(previousNode.getLink().getLink())

Delete After a Specific Node

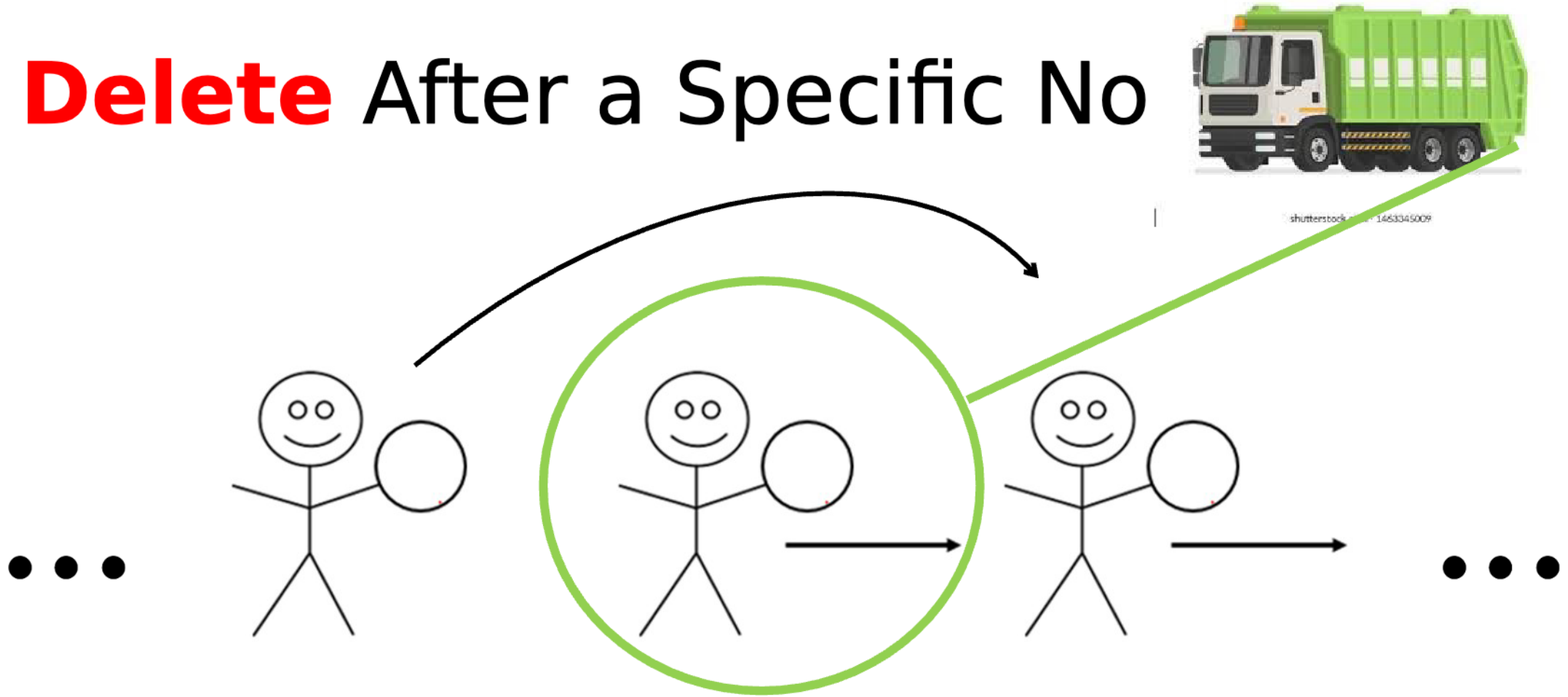


previousNode

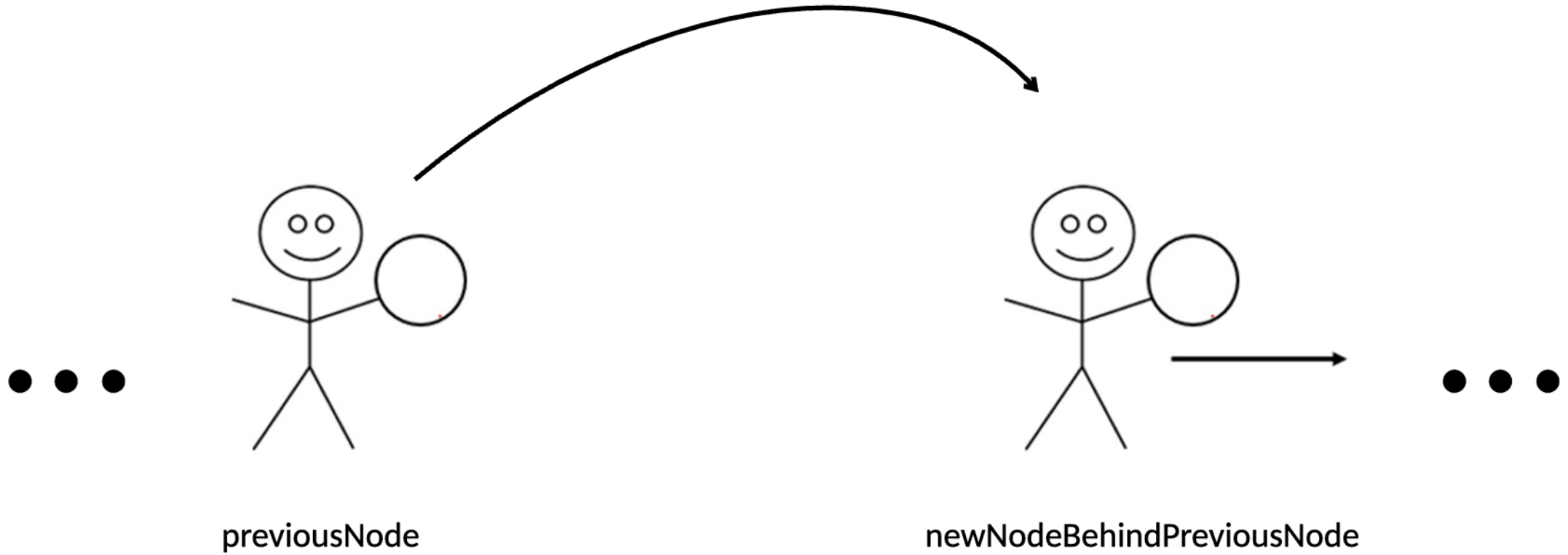
nodeBehindPreviousNode
aka previousNode.getLink()

nodeBehindNodeBehindPreviousNode
aka previousNode.getLink().getLink()

Delete After a Specific No



Delete After a Specific Node



Try it!

- Create a remove method in your animalShelterLinked class.

- Test it in the driver class to make sure that it maintains the order of the collection.

Arrays versus Linked Lists

Deciding which one to use

Guidelines for Choosing Between an Array and a Linked List

Operation	Which data structure to use?
Frequent random access operations	Array
Frequent Insertion and deletion at random locations	Linked list (to avoid moving elements up and down)
Frequent capacity change	Linked list (to avoid the resizing inefficiency)

Complexity of Adding to an Unordered Array-Based List

```
STEPS
0     public void add(int element)   {
1           data[manyItems] = element;
1           manyItems++;
      }
-------
Total:  2 (constant) = O(1)
```

Complexity of Adding to an Unordered Linked List

```
STEPS
0     public void add(int element)  {
1           this.head = new IntNode(element,head);
1           this.manyNodes++;
      }
-------
Total:  2 (constant) = O(1)
```

Complexity of Inserting into an Ordered Array-Based List

```
STEPS
0     public void insert(int num)   {
1           int i=0;
2*N         while (i < manyItems && (data[i] <= num)) {
1*N               i++;
            }

2           for (int move = manyItems; move > i; move-- ) {
0             data[move] = data[move-1];
            }

1           data[i] = num;
1           manyItems++;
      }
-----
Total:  1 + 3*N + 2 + 2 = 3*N + 5 = O(N)
```

Complexity of Inserting into an Ordered Linked List

```
STEPS
0     public void insert (int newValue)   {
1           if (head == null)
1                 head = new IntNode(newValue, head);
1           else if (newValue < head.getData())
1                 head = new IntNode(newValue, head);
            else  {
??                IntNode previousNode = findPrevious(newValue);
1                       previousNode.setLink (new IntNode(newValue,
                              previousNode.getLink()));
                }
      }
-------
Total:            2 OR 2 OR (?? + 1) = ??
```

`findPrevious()` Method for Ordered Lists

```
STEPS
0     public IntNode findPrevious(int newValue)
      {
1           IntNode cur = head;
1*N         while (cur.getLink() != null &&
1*N                cur.getLink().getData() <
                   newValue) {
1*N                cur = cur.getLink();
                 }
1           return cur;
      }
------
Total:  3*N + 2 = O(N)
```

Complexity of Inserting into an Ordered Linked List

```
STEPS
0     public void insert (int newValue)   {
1           if (head == null)
1                 head = new IntNode(newValue, head);
1           else if (newValue < head.getData())
1                 head = new IntNode(newValue, head);
            else  {
3*N+2             IntNode previousNode = findPrevious(newValue);
1                       previousNode.setLink (new IntNode(newValue,
                              previousNode.getLink()));
                }
      }
-------
Total:            2 OR 2 OR (3*N + 2 + 1) = 3*N + 3 = O(N) (worst
case)
```

Complexity of Adding or Inserting into a List

add()/insert()	Unordered List	Ordered List
Array-based list	O(1)	O(N)
Linked list	O(1)	O(N)

Complexity of Removing from a List

remove()	Unordered List	Ordered List
Array-based list	O(N)	O(N)
Linked list	O(N)	O(N)

Before you go

- Linked list animation
  - https://visualgo.net/list

- Null Pointer Exception
  - It's very easy to miscalculate. These are fixed by thinking through the process and figuring out where you are trying to use an object that doesn't exist.