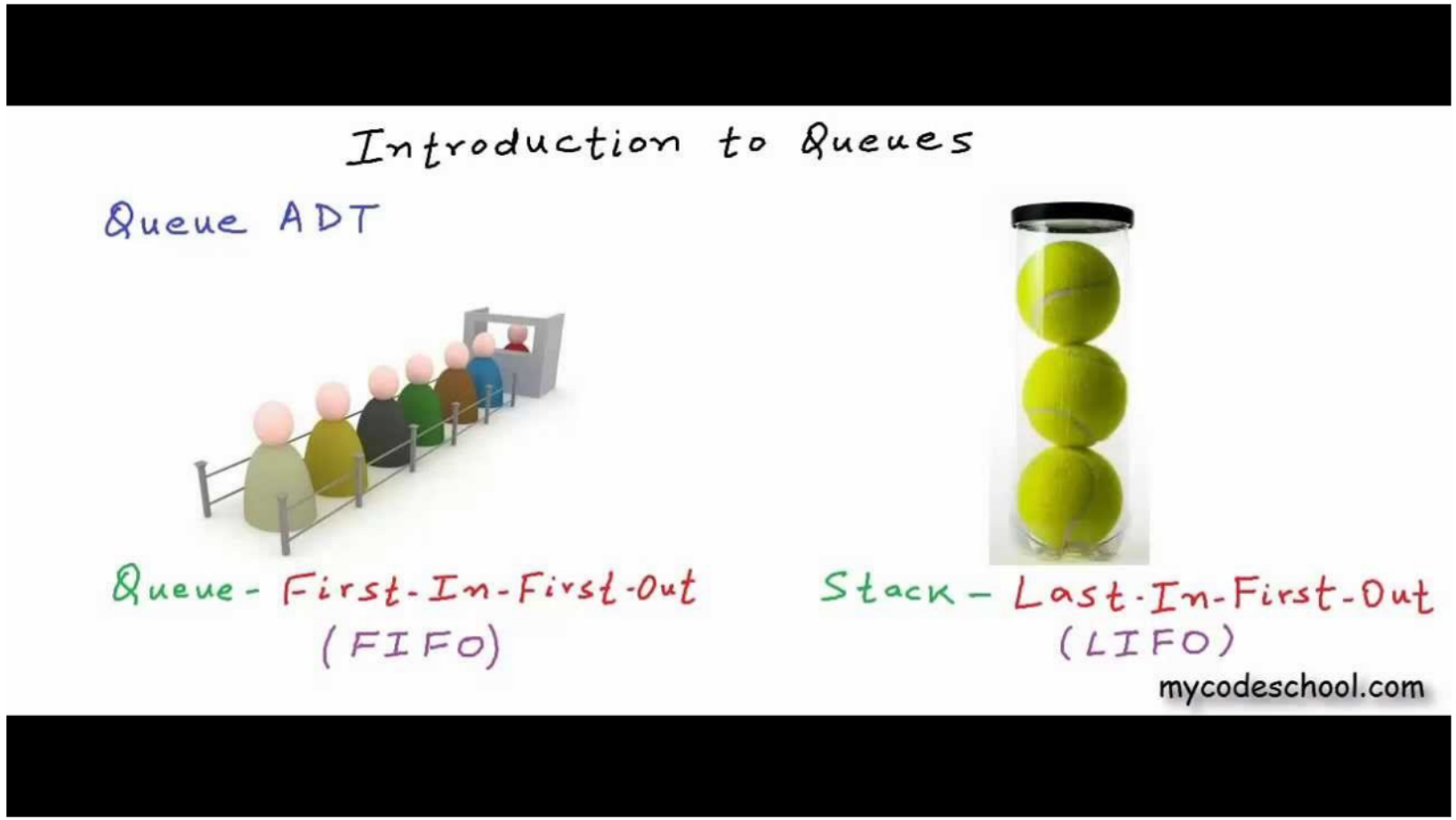
ICS 240
Introduction to Data Structures

Jessica Maistrovich

Metropolitan State University

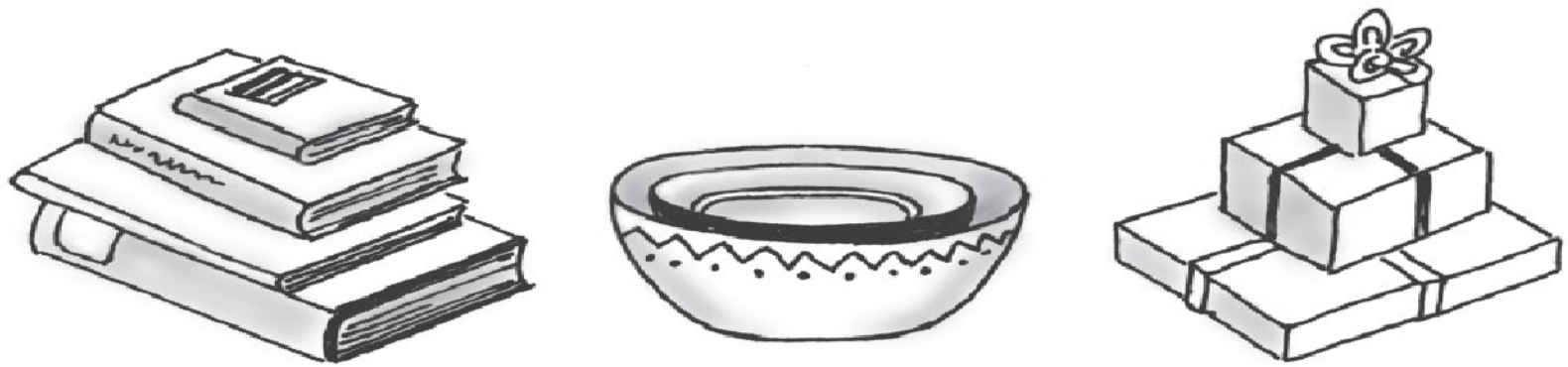
Queues vs. Stacks



Stack

Abstract Data Type

Example Stacks



Introduction

- A stack is a data structure of ordered items such that items can be inserted and removed only at one end (called the top).
- A stack is called: Last-In-First-Out (LIFO) data structure
- You can insert (or push) elements in the stack at any point of time.
- You have access only to the element at the top of the stack (pop)
- A stack can be implemented using either an array or a linked list

`Stack` Methods

Modifier and Type	Method and Description
`boolean`	**`isEmpty()`** Determine whether this stack is empty.
`E`	**`peek()`** Get the top item of this stack, without removing the item.
`E`	**`pop()`** Get the top item, removing it from this stack.
`void`	**`push(E item)`** Push a new item onto this stack.

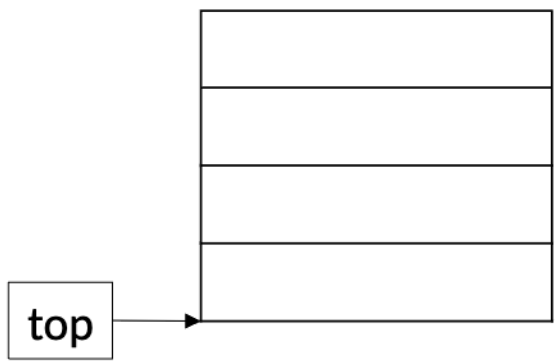
Example of using Java's `Stack`

What is the output of this code?

```
import java.util.Stack;
Stack<Integer> s = new Stack();
s.push(10);
s.push(20);
s.push(30);
s.pop();
s.push(40);
s.pop();
s.pop();
s.push(50);
while (!s.isEmpty()){
      System.out.println(s.pop());
}
```

Example of using Java's `Stack`

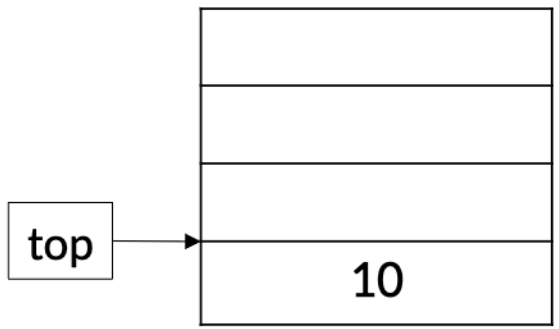
What is the output of this code?




```
import java.util.Stack;
Stack<Integer> s = new Stack();
s.push(10);
s.push(20);
s.push(30);
s.pop();
s.push(40);
s.pop();
s.pop();
s.push(50);
while (!s.isEmpty()){
      System.out.println(s.pop());
}
```

Example of using Java's `Stack`

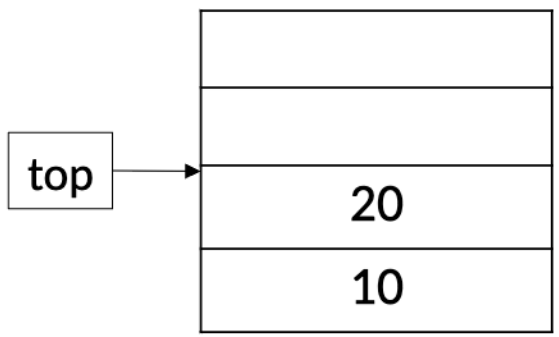
What is the output of this code?




```
import java.util.Stack;
Stack<Integer> s = new Stack();
s.push(10);
s.push(20);
s.push(30);
s.pop();
s.push(40);
s.pop();
s.pop();
s.push(50);
while (!s.isEmpty()){
      System.out.println(s.pop());
}
```

Example of using Java's `Stack`

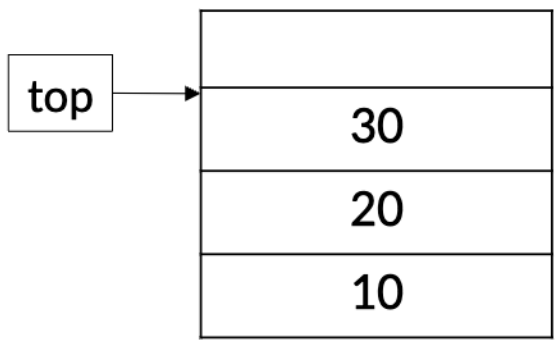
What is the output of this code?




```
import java.util.Stack;
Stack<Integer> s = new Stack();
s.push(10);
s.push(20);
s.push(30);
s.pop();
s.push(40);
s.pop();
s.pop();
s.push(50);
while (!s.isEmpty()){
      System.out.println(s.pop());
}
```

Example of using Java's `Stack`

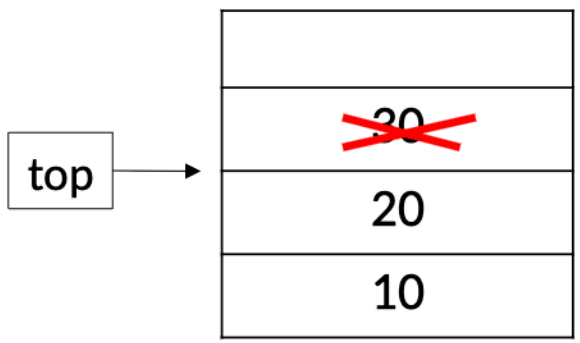
What is the output of this code?




```
import java.util.Stack;
Stack<Integer> s = new Stack();
s.push(10);
s.push(20);
s.push(30);
s.pop();
s.push(40);
s.pop();
s.pop();
s.push(50);
while (!s.isEmpty()){
      System.out.println(s.pop());
}
```

Example of using Java's `Stack`

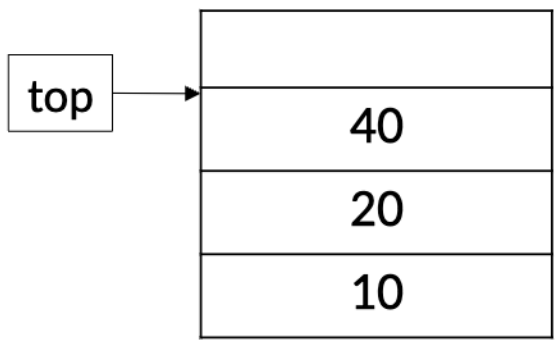
What is the output of this code?




```
import java.util.Stack;
Stack<Integer> s = new Stack();
s.push(10);
s.push(20);
s.push(30);
s.pop();
s.push(40);
s.pop();
s.pop();
s.push(50);
while (!s.isEmpty()){
      System.out.println(s.pop());
}
```

Example of using Java's `Stack`

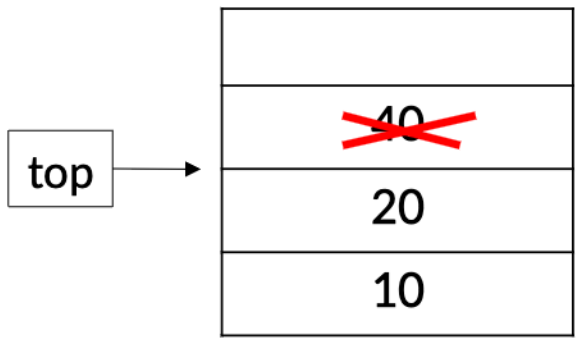
What is the output of this code?




```
import java.util.Stack;
Stack<Integer> s = new Stack();
s.push(10);
s.push(20);
s.push(30);
s.pop();
s.push(40);
s.pop();
s.pop();
s.push(50);
while (!s.isEmpty()){
      System.out.println(s.pop());
}
```

Example of using Java's `Stack`

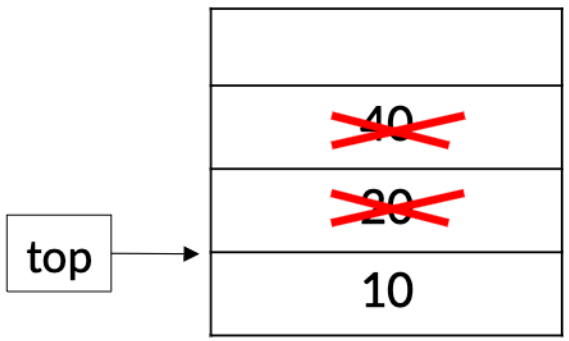
What is the output of this code?




```
import java.util.Stack;
Stack<Integer> s = new Stack();
s.push(10);
s.push(20);
s.push(30);
s.pop();
s.push(40);
s.pop();
s.pop();
s.push(50);
while (!s.isEmpty()){
      System.out.println(s.pop());
}
```

Example of using Java's `Stack`

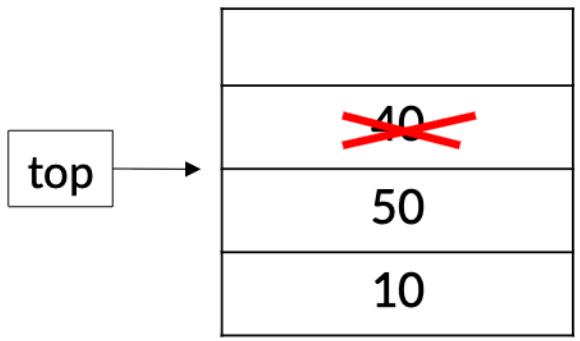
What is the output of this code?




```
import java.util.Stack;
Stack<Integer> s = new Stack();
s.push(10);
s.push(20);
s.push(30);
s.pop();
s.push(40);
s.pop();
s.pop();
s.push(50);
while (!s.isEmpty()){
      System.out.println(s.pop());
}
```

Example of using Java's `Stack`

What is the output of this code?

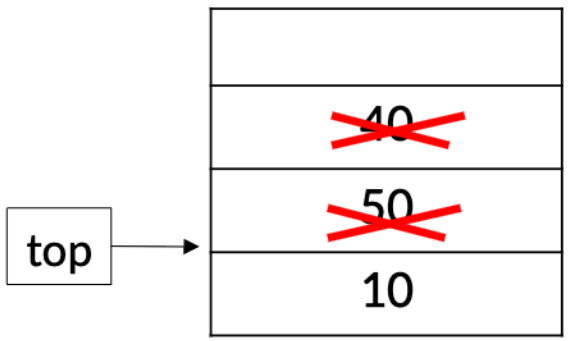



```
import java.util.Stack;
Stack<Integer> s = new Stack();
s.push(10);
s.push(20);
s.push(30);
s.pop();
s.push(40);
s.pop();
s.pop();
s.push(50);
while (!s.isEmpty()){
      System.out.println(s.pop());
}
```

Example of using Java's `Stack`

What is the output of this code?

50



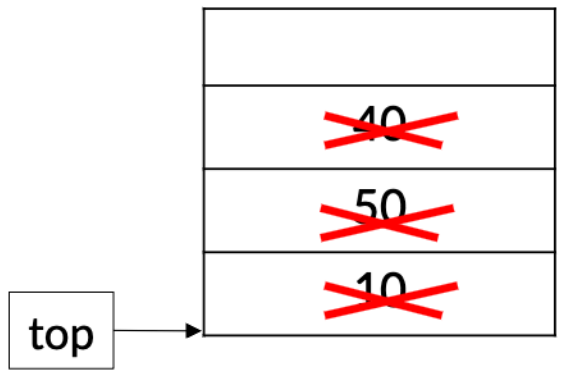

```
import java.util.Stack;
Stack<Integer> s = new Stack();
s.push(10);
s.push(20);
s.push(30);
s.pop();
s.push(40);
s.pop();
s.pop();
s.push(50);
while (!s.isEmpty()){
      System.out.println(s.pop());
}
```

Example of using Java's `Stack`

What is the output of this code?

50

10




```
import java.util.Stack;
Stack<Integer> s = new Stack();
s.push(10);
s.push(20);
s.push(30);
s.pop();
s.push(40);
s.pop();
s.pop();
s.push(50);
while (!s.isEmpty()){
      System.out.println(s.pop());
}
```

Applications that Use a Stack

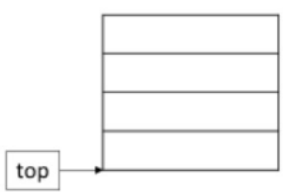
- A web browser keeps a stack of recently visited web pages so that that you can use the back button.
- A text editor uses a stack to handle the undo operation.
- Function calls in any programming languages are handled by a stack

Array Implementation

How would you implement a stack with an array?

Example of using Java's Stack

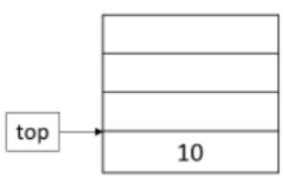
What is the output of this code?




```
import java.util.Stack;
Stack<Integer> s = new Stack();
s.push(10);
s.push(20);
s.push(30);
s.pop();
s.push(40);
s.pop();
s.pop();
s.push(50);
while (!s.isEmpty()){
      System.out.println(s.pop());
}
```

Example of using Java's Stack

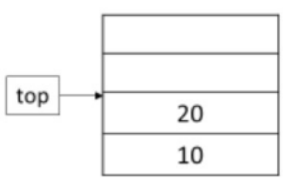
What is the output of this code?




```
import java.util.Stack;
Stack<Integer> s = new Stack();
s.push(10);
s.push(20);
s.push(30);
s.pop();
s.push(40);
s.pop();
s.pop();
s.push(50);
while (!s.isEmpty()){
      System.out.println(s.pop());
}
```

Example of using Java's Stack

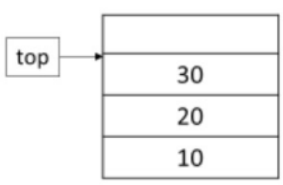
What is the output of this code?




```
import java.util.Stack;
Stack<Integer> s = new Stack();
s.push(10);
s.push(20);
s.push(30);
s.pop();
s.push(40);
s.pop();
s.pop();
s.push(50);
while (!s.isEmpty()){
      System.out.println(s.pop());
}
```

Example of using Java's Stack

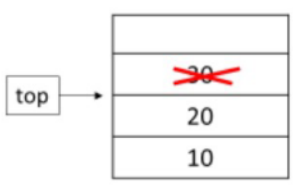
What is the output of this code?




```
import java.util.Stack;
Stack<Integer> s = new Stack();
s.push(10);
s.push(20);
s.push(30);
s.pop();
s.push(40);
s.pop();
s.pop();
s.push(50);
while (!s.isEmpty()){
      System.out.println(s.pop());
}
```

Example of using Java's Stack

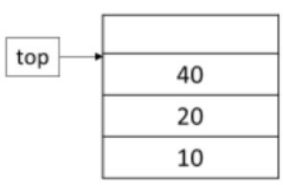
What is the output of this code?




```
import java.util.Stack;
Stack<Integer> s = new Stack();
s.push(10);
s.push(20);
s.push(30);
s.pop();
s.push(40);
s.pop();
s.pop();
s.push(50);
while (!s.isEmpty()){
      System.out.println(s.pop());
}
```

Example of using Java's Stack

What is the output of this code?




```
import java.util.Stack;
Stack<Integer> s = new Stack();
s.push(10);
s.push(20);
s.push(30);
s.pop();
s.push(40);
s.pop();
s.pop();
s.push(50);
while (!s.isEmpty()){
      System.out.println(s.pop());
}
```

Example of using Java's Stack

What is the output of this code?


```
import java.util.Stack;
Stack<Integer> s = new Stack();
s.push(10);
s.push(20);
s.push(30);
s.pop();
s.push(40);
s.pop();
s.pop();
s.push(50);
while (!s.isEmpty()){
      System.out.println(s.pop());
}
```

Example of using Java's Stack

What is the output of this code?


```
import java.util.Stack;
Stack<Integer> s = new Stack();
s.push(10);
s.push(20);
s.push(30);
s.pop();
s.push(40);
s.pop();
s.pop();
s.push(50);
while (!s.isEmpty()){
      System.out.println(s.pop());
}
```

Example of using Java's Stack

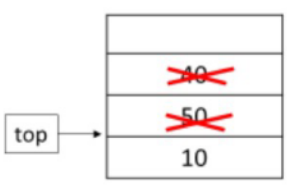
What is the output of this code?


```
import java.util.Stack;
Stack<Integer> s = new Stack();
s.push(10);
s.push(20);
s.push(30);
s.pop();
s.push(40);
s.pop();
s.pop();
s.push(50);
while (!s.isEmpty()){
      System.out.println(s.pop());
}
```

Example of using Java's Stack

What is the output of this code?

50



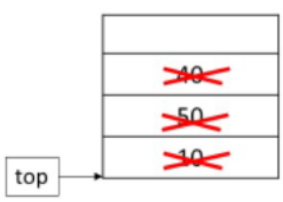

```
import java.util.Stack;
Stack<Integer> s = new Stack();
s.push(10);
s.push(20);
s.push(30);
s.pop();
s.push(40);
s.pop();
s.pop();
s.push(50);
while (!s.isEmpty()){
      System.out.println(s.pop());
}
```

Example of using Java's Stack

What is the output of this code?

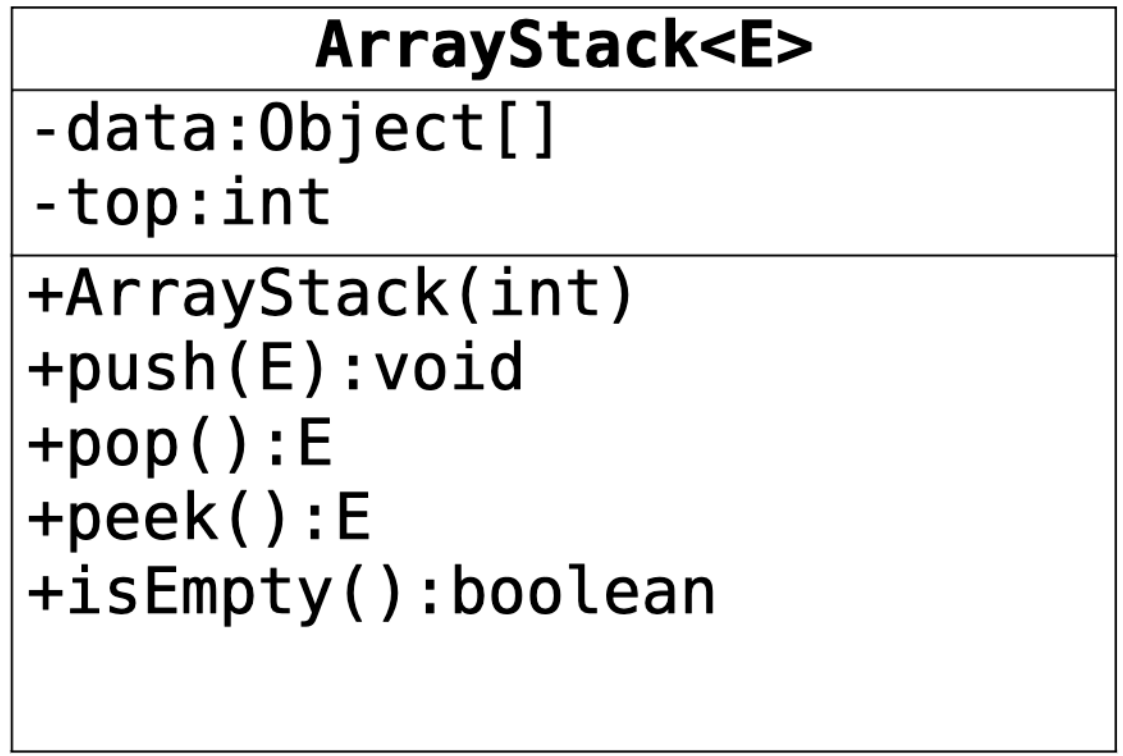
50

10




```
import java.util.Stack;
Stack<Integer> s = new Stack();
s.push(10);
s.push(20);
s.push(30);
s.pop();
s.push(40);
s.pop();
s.pop();
s.push(50);
while (!s.isEmpty()){
      System.out.println(s.pop());
}
```

ArrayStack Class Diagram



ArrayStack Invariant

```
/**
Invariant of the Stack that is implemented using an
Array:

1. The number of items in the stack is stored in the
   instance variable top. 0 <= top <= capacity of
   data array

2. The items in the stack are stored in a partially
filled array called data with the bottom of the stack
at data[0], the next item is at data[1], and so on,
and the top of the stack at data[top-1]
 */
```

Adding or Deleting Elements

Add element to stack (push)

```
data[top++] = elementToAdd;
```

Remove element from stack (pop)

```
return data[--top];
```

Linked List Implementation

How would you implement a stack with a linked list?

Recall: Add at the Front of the Linked List

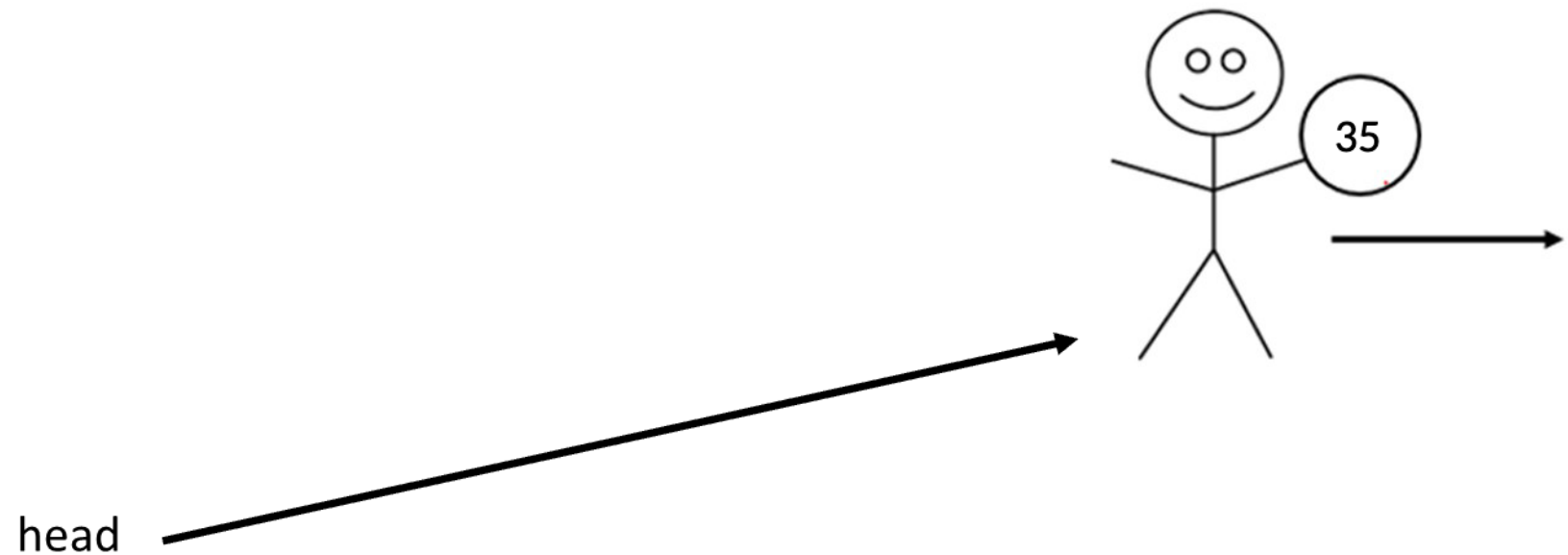
```
IntNode head;
head = new IntNode(35, head);
head = new IntNode (15, head);
head = new IntNode (10, head);
```

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);
```

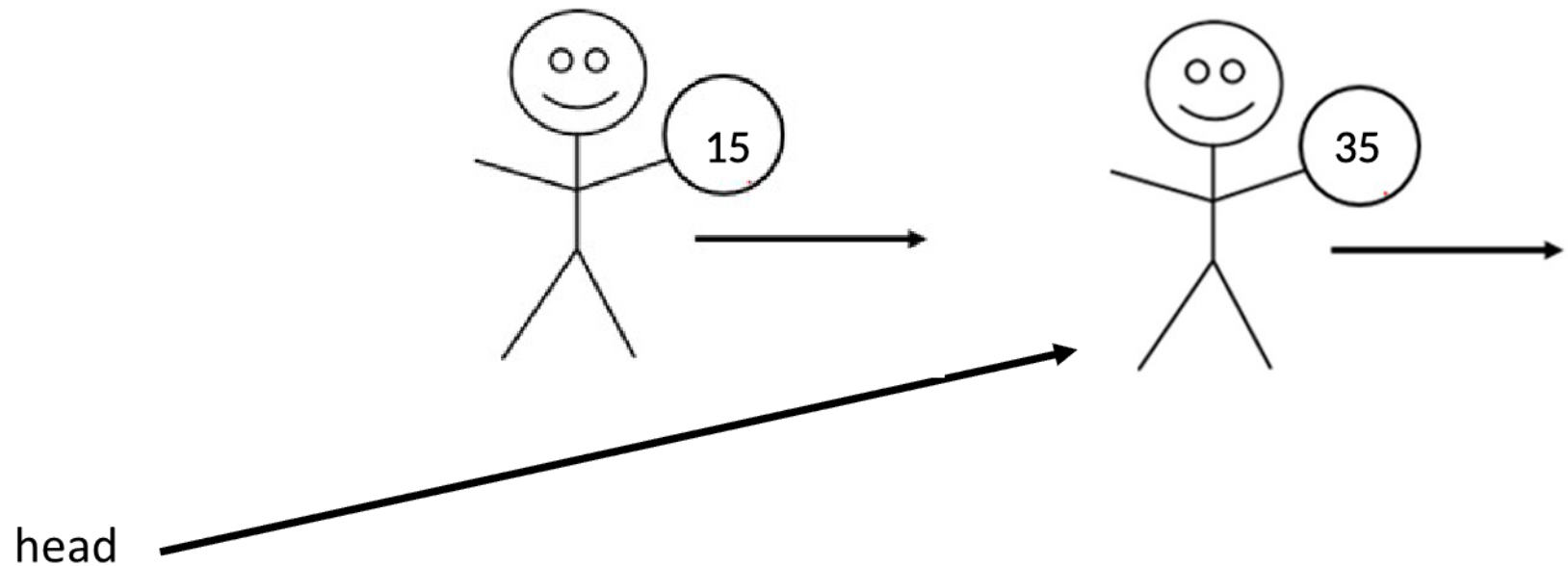
Add at the Front of the Linked List

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



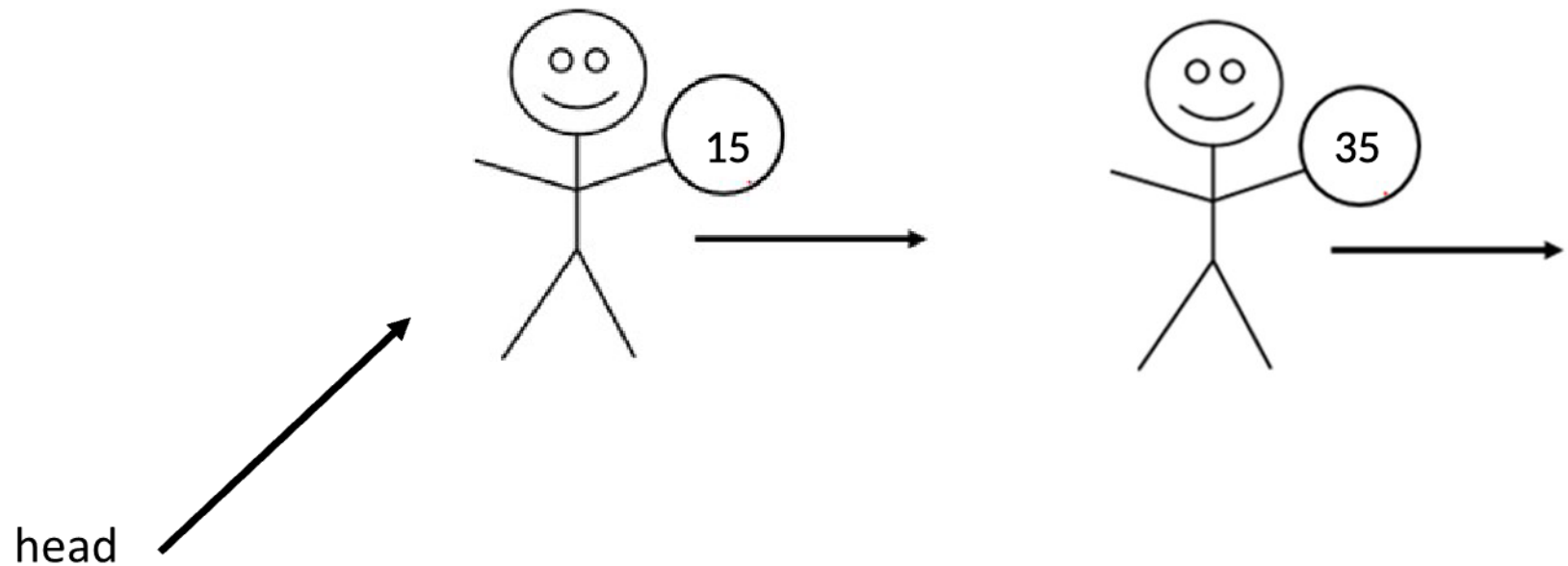
Add at the Front of the Linked List

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



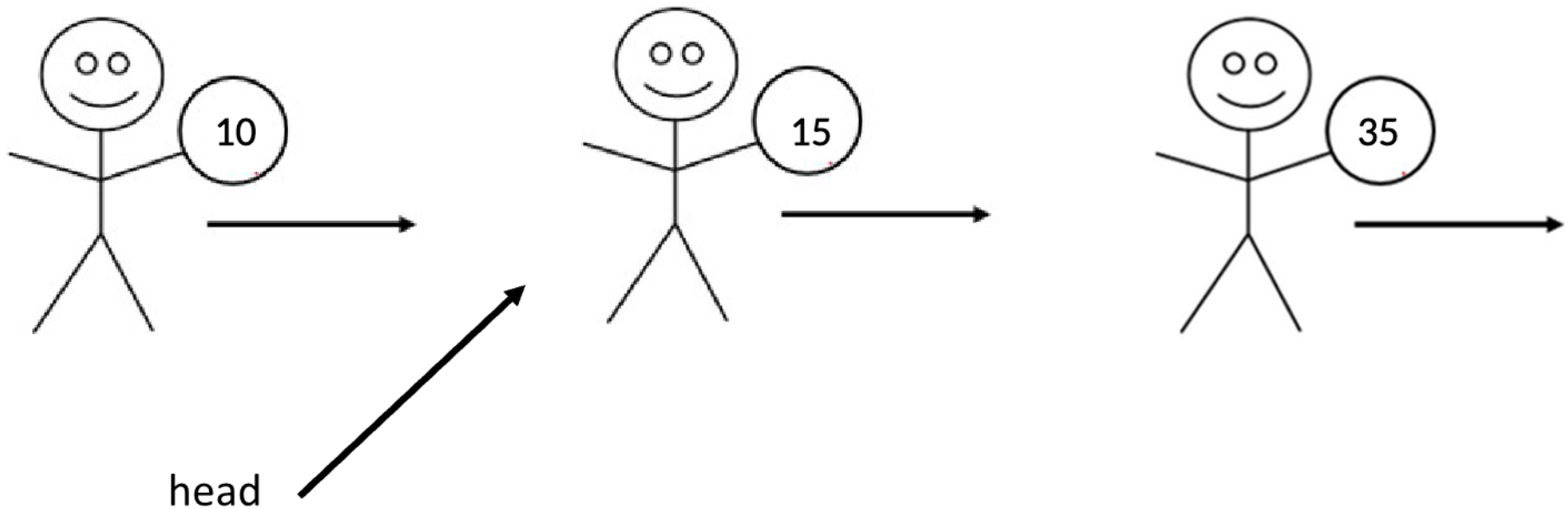
Add at the Front of the Linked List

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



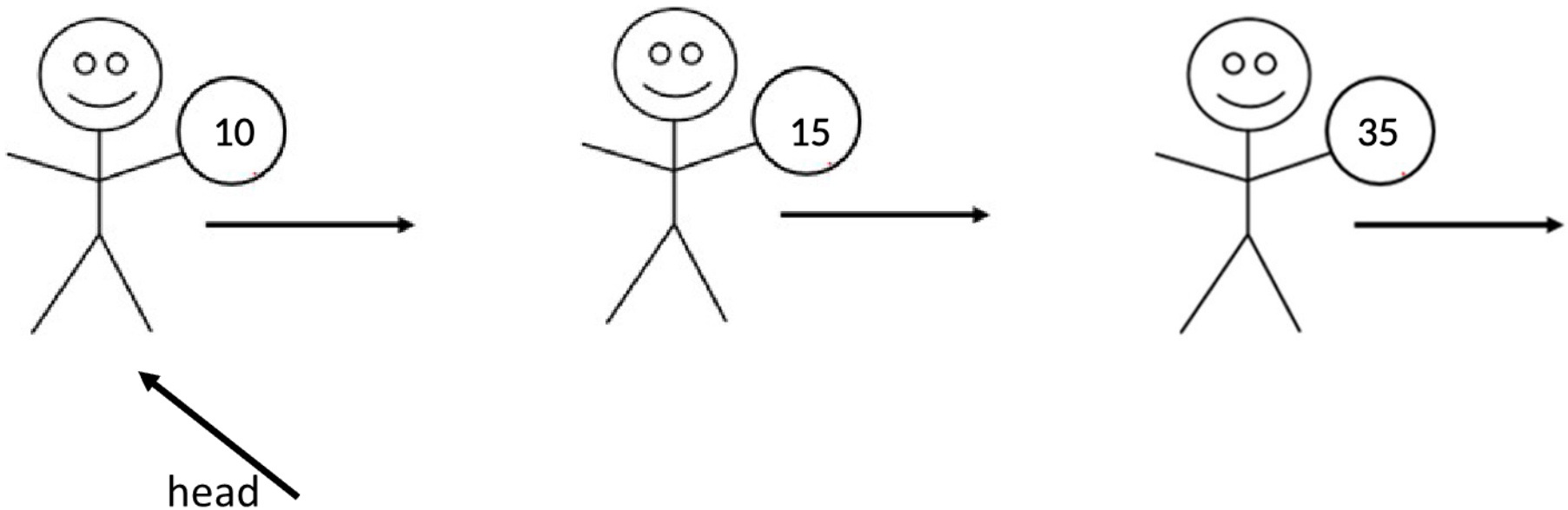
Add at the Front of the Linked List

```
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);
```



Recall: Add at the Front of the Linked List

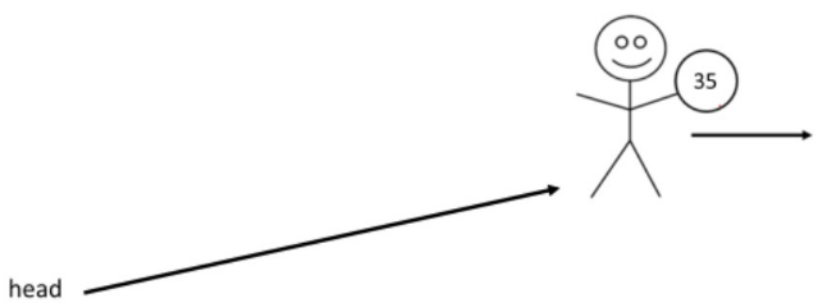
```
IntNode head;
head = new IntNode(35, head);
head = new IntNode (15, head);
head = new IntNode (10, head);
```

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);
```

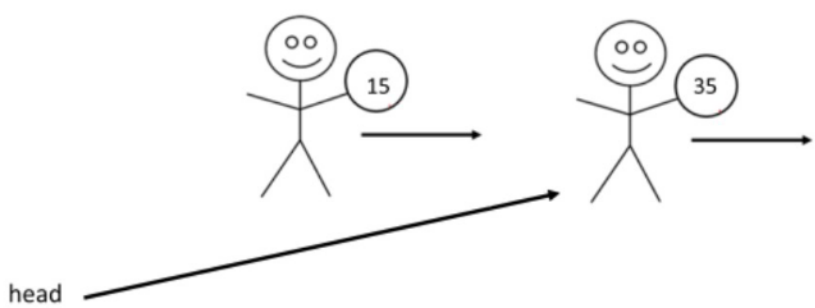
Add at the Front of the Linked List

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



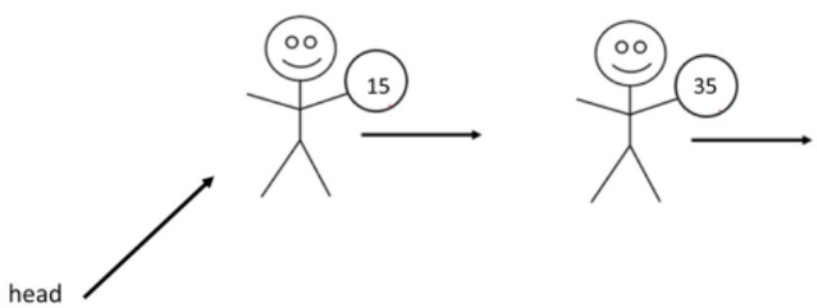
Add at the Front of the Linked List

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



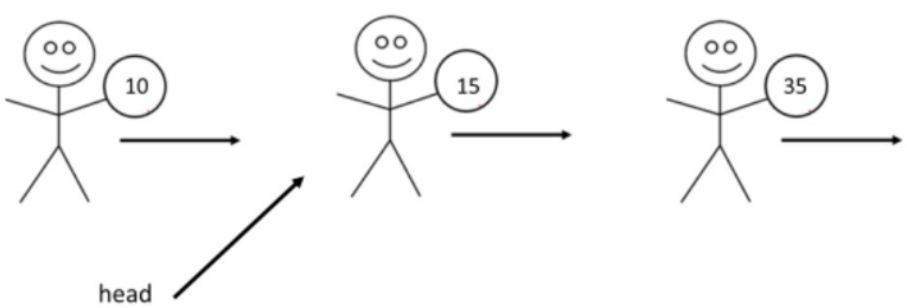
Add at the Front of the Linked List

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



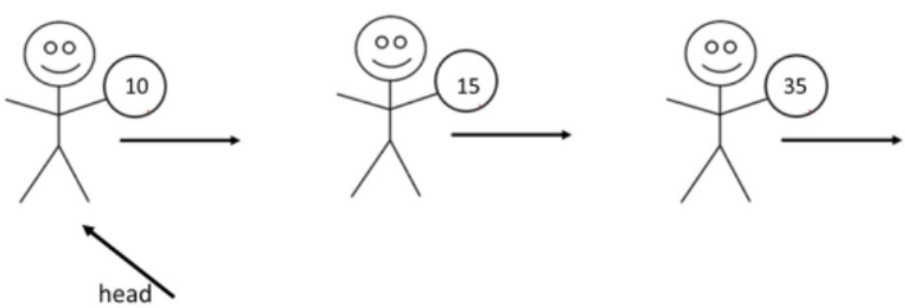
Add at the Front of the Linked List

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

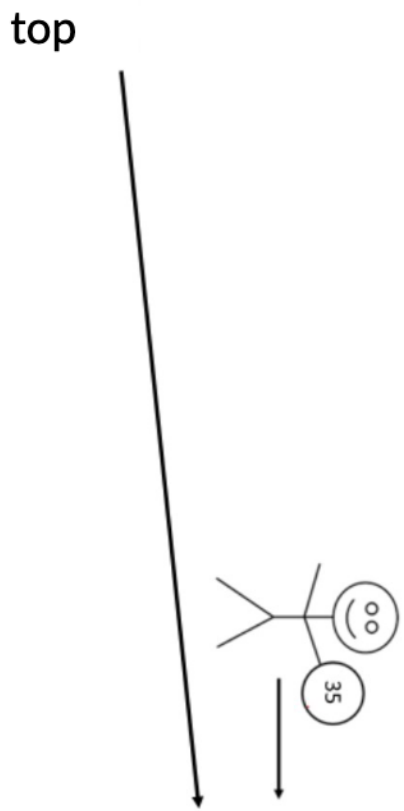


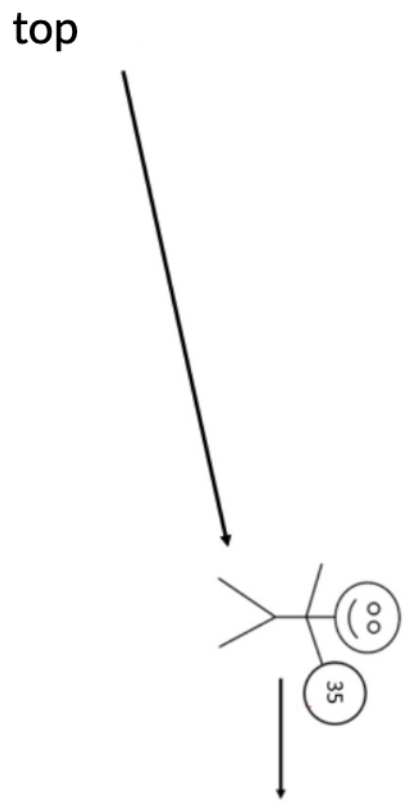
Add at the Front of the Linked List

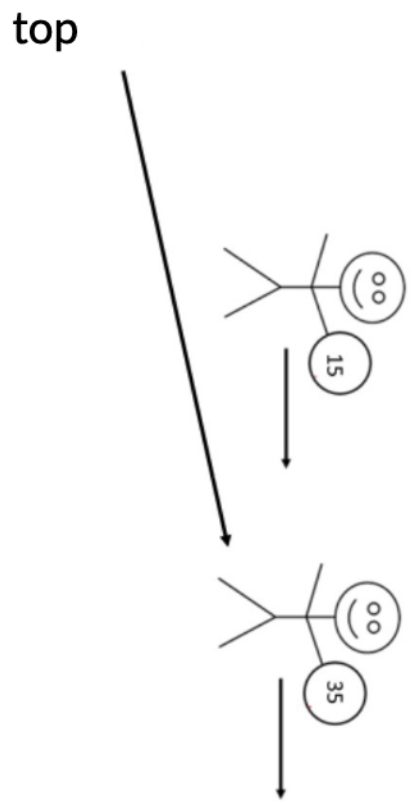
```
IntNode head;
head = new IntNode(35, head);
head = new IntNode (15, head);
head = new IntNode (10, head);
```

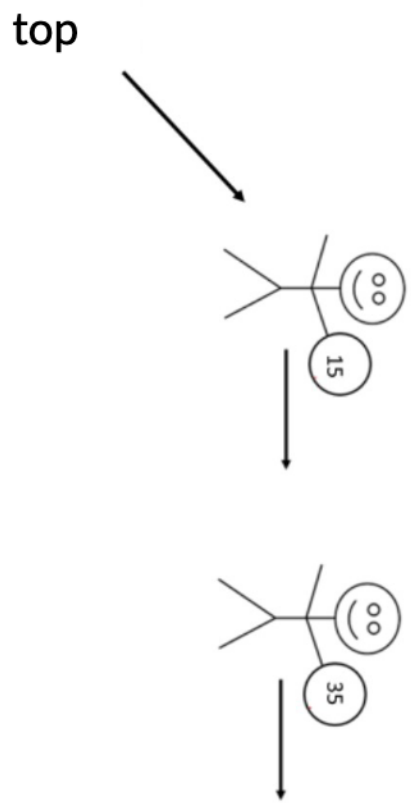


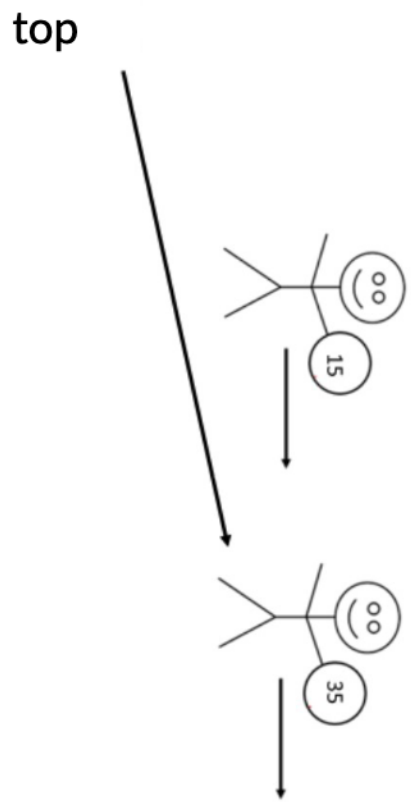
Instead of using head, use top

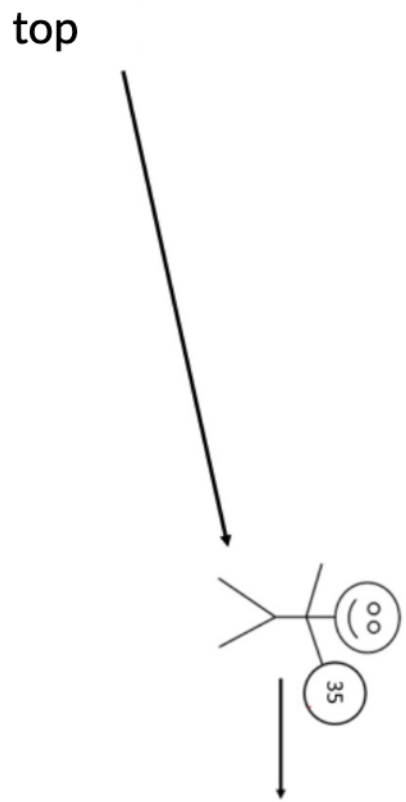

top
35

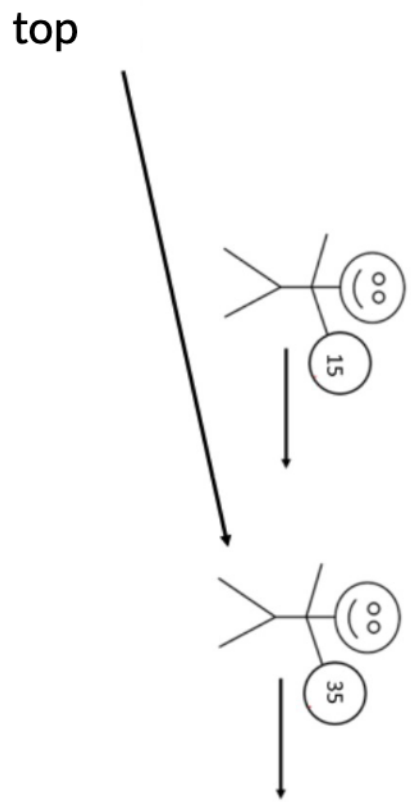

top
35

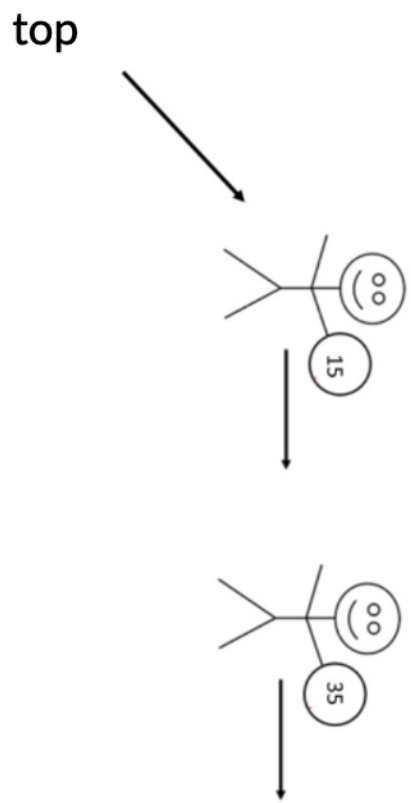

top
15
35

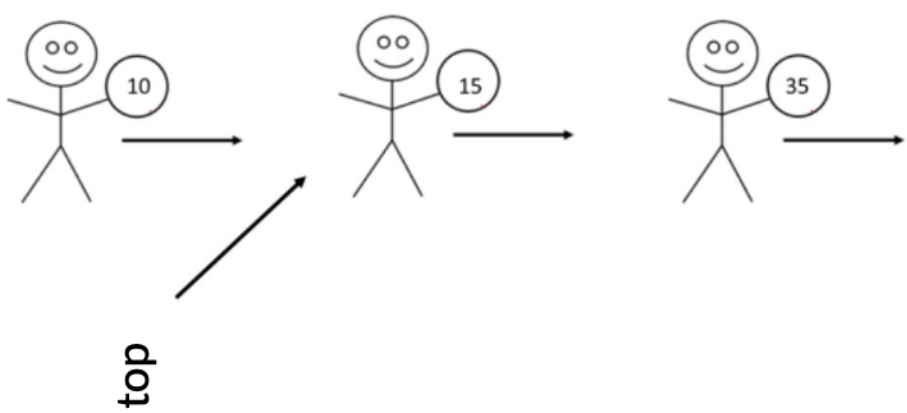

top
15
35

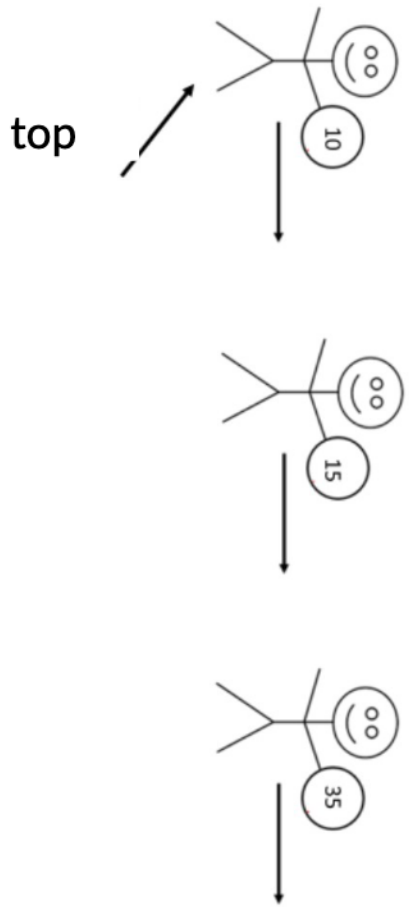

top
15
35


top
35

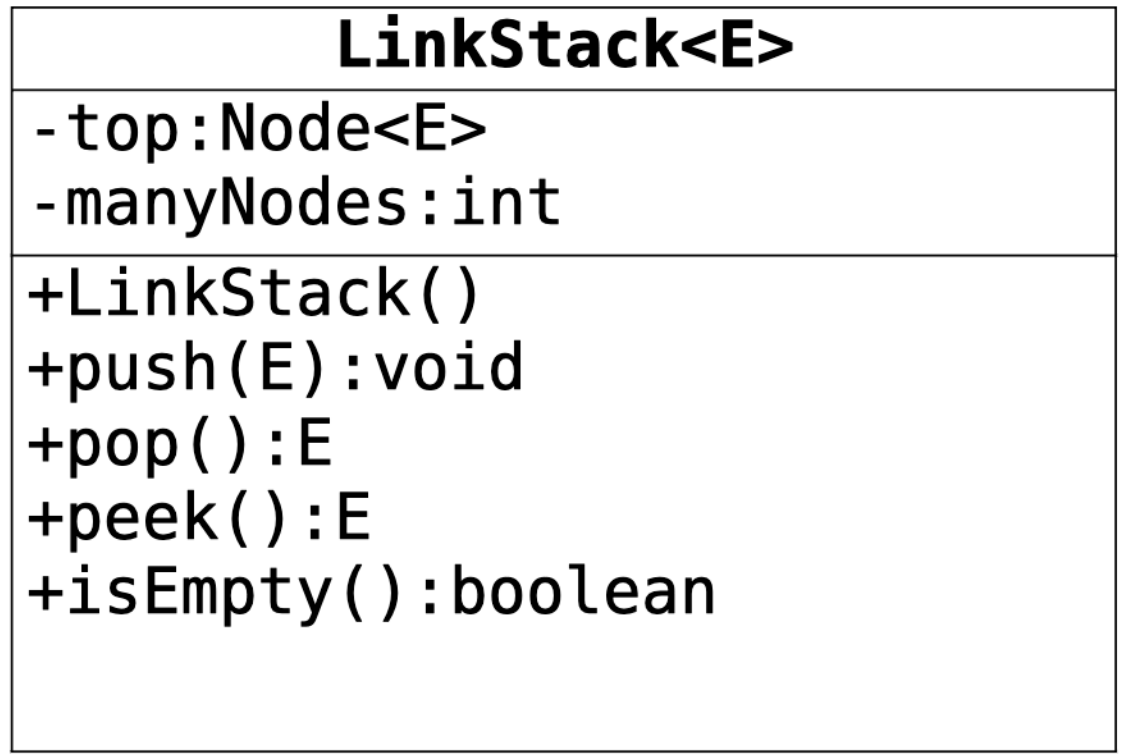

top
15
35


top
15
35


10
15
35
top


top
10
15
35

LinkStack Class Diagram



`LinkStack` Invariant

```
/**
Invariant:

1. The items in the stack are stored in a linked list, with the top
of the stack stored at the head node, down to the bottom of the
stack at the final node.
2. The instance variable top is the head reference of the linked
list of items.
3. The instance variable manyNodes indicates the size of the Stack

*/
```

Adding or Deleting Elements

Add element to stack (push)

```
top = new Node<E>(E elementToAdd, top);
manyNodes++;
```

Remove element from stack (pop)

```
E element = top.getData();
top = top.getLink();
manyNodes--;
return element;
```

Try it!

- Create a generic stack class. Test it with a driver.

Queue

Abstract Data Type

Examples of Queues

What is a Queue?

- A queue is a data structure of ordered items such that items can be inserted at one end (called **tail**) and removed only at the other end (called **head**)
- The defining property of a queue data structure is First-In-First-Out **(FIFO)**
- You can add (or **enqueue**) elements in the queue at any point of time
- You can remove/read only the element at the front of the queue (**dequeue**)
- A queue can be implemented using either an array or a linked list

Queue Methods

Modifier and Type	Method and Description
`boolean`	**`isEmpty`**`()` Determine whether this queue is empty.
`E`	**`peek`**`()` Get the front item of this queue, without removing the item.
`E`	**`dequeue`**`()` Get the front item, removing it from this queue.
`void`	**`enqueue`**`(E item)` Put a new item into the rear of this queue.

Applications that Use a Queue

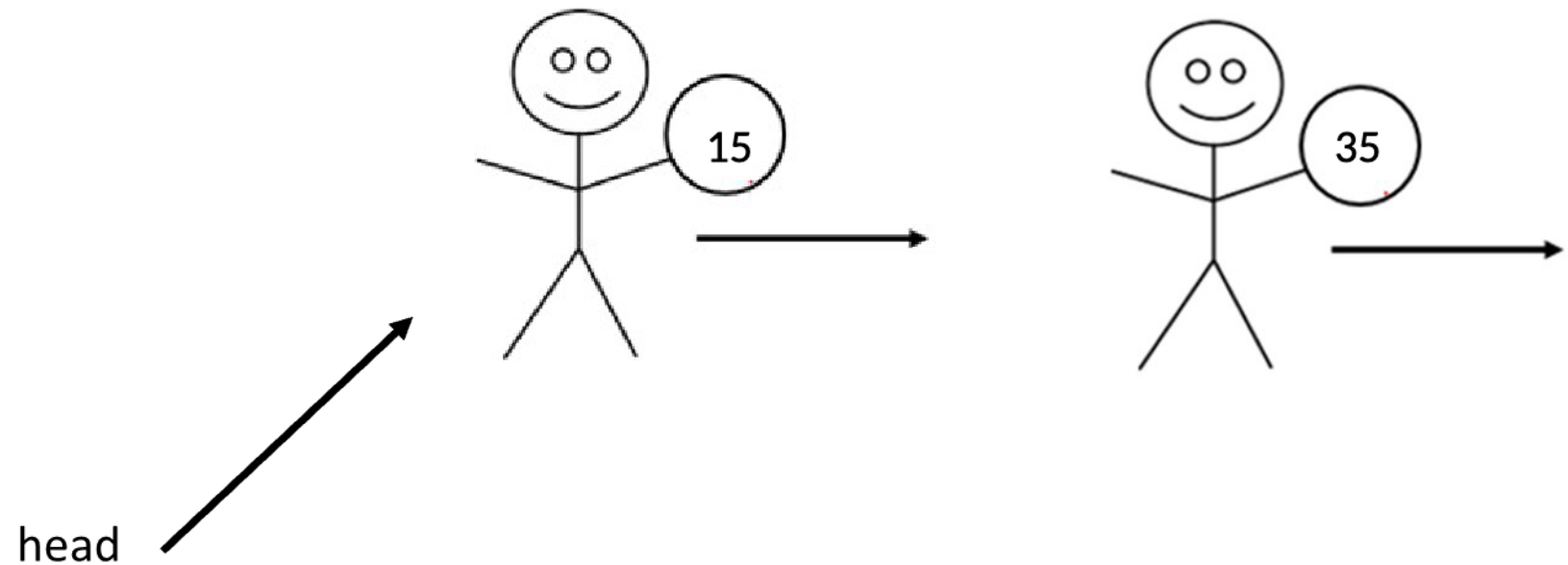
- A printer keeps a queue of documents to be printed
- A queue is used in simulations to simulate:
  - Cars waiting in traffic lines
  - Customers waiting in a bank or cashier queues
  - Internet traffic

Linked List Implementation

How would you implement a queue with a linked list?

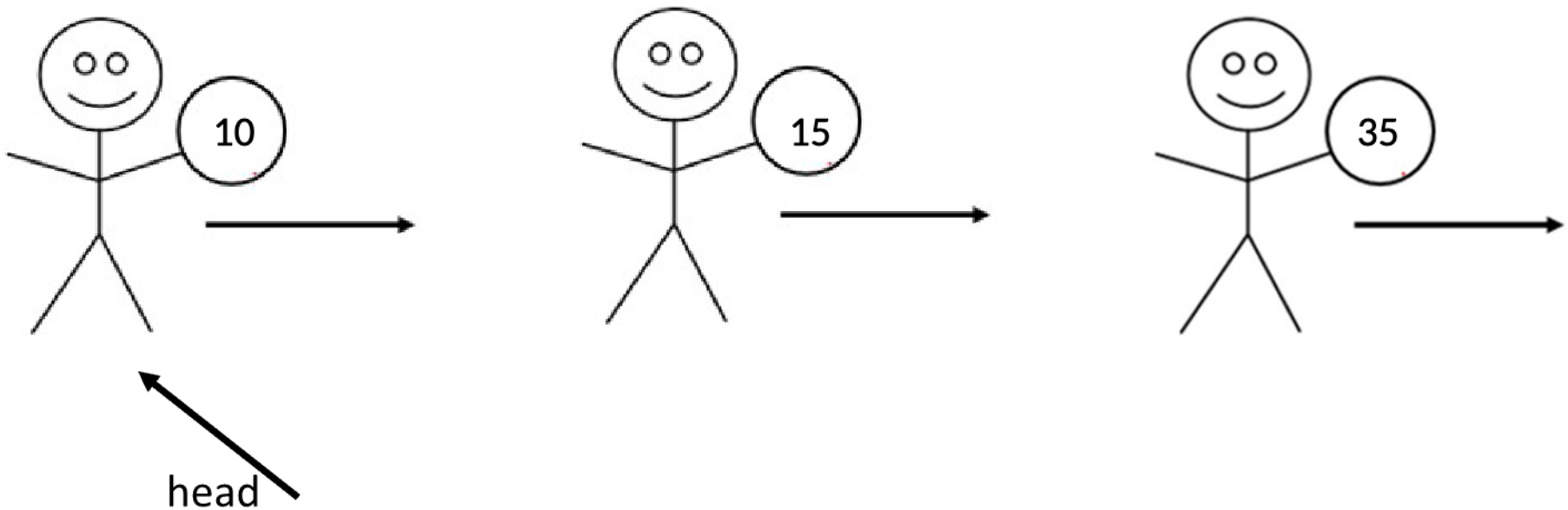
Recall: Add at the Front of the Linked List

```
head = new Node<E> (E elementToAdd, head);
```



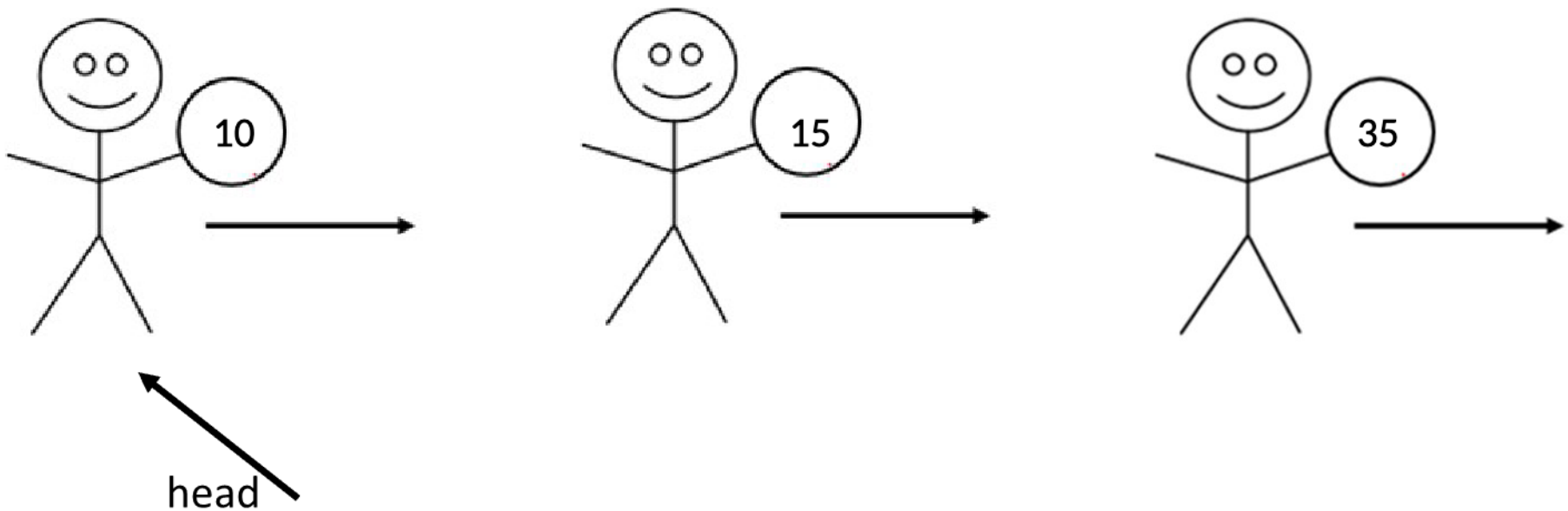
Recall: Add at the Front of the Linked List

```
head = new Node<Integer> (10, head);
```



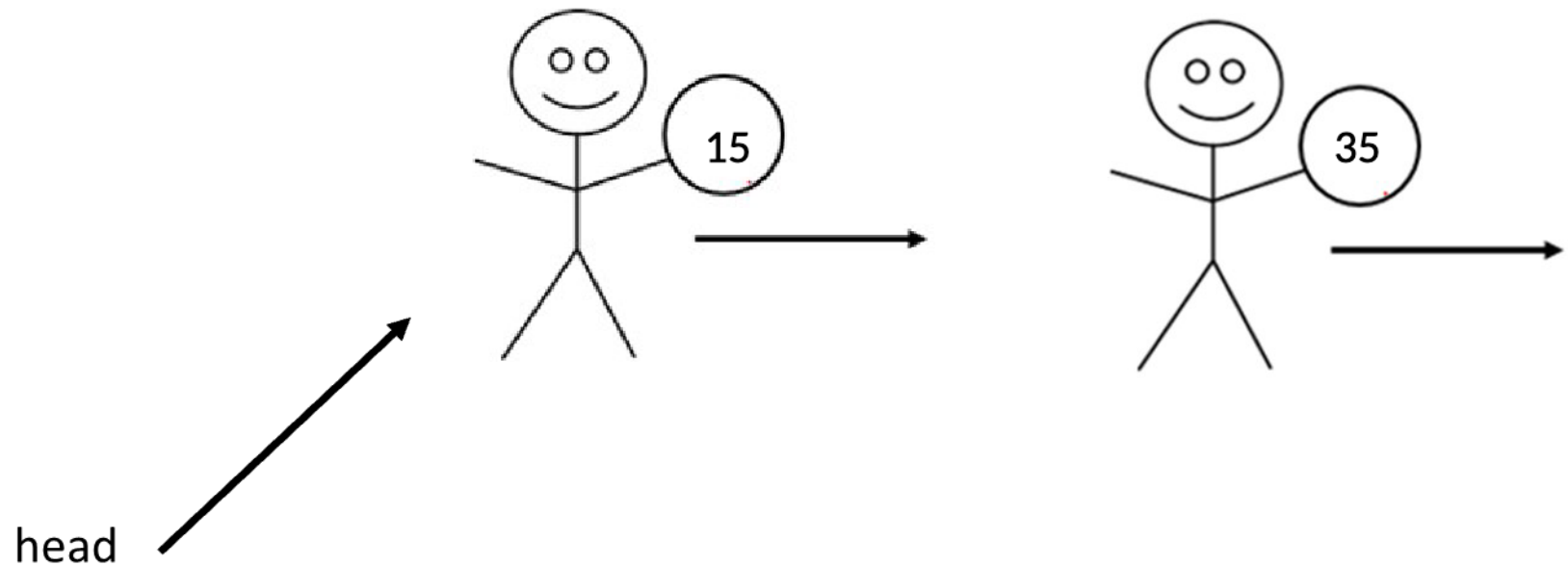
Recall: Remove from the Front of the Linked List

```
head = head.getLink();
```



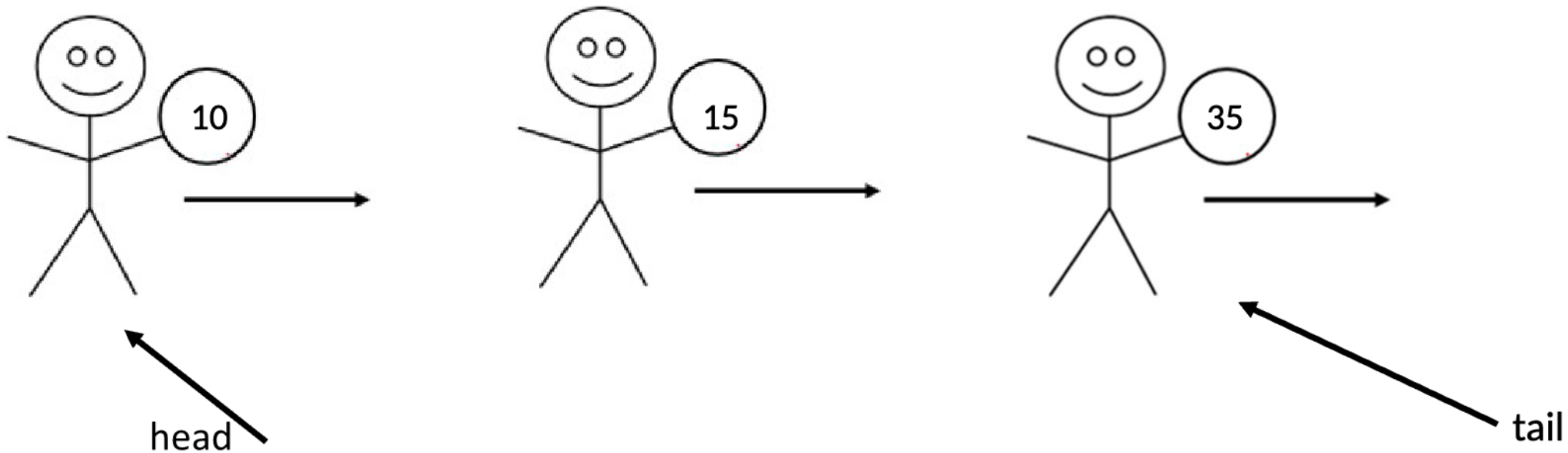
Recall: Remove from the Front of the Linked List

```
head = head.getLink()
```



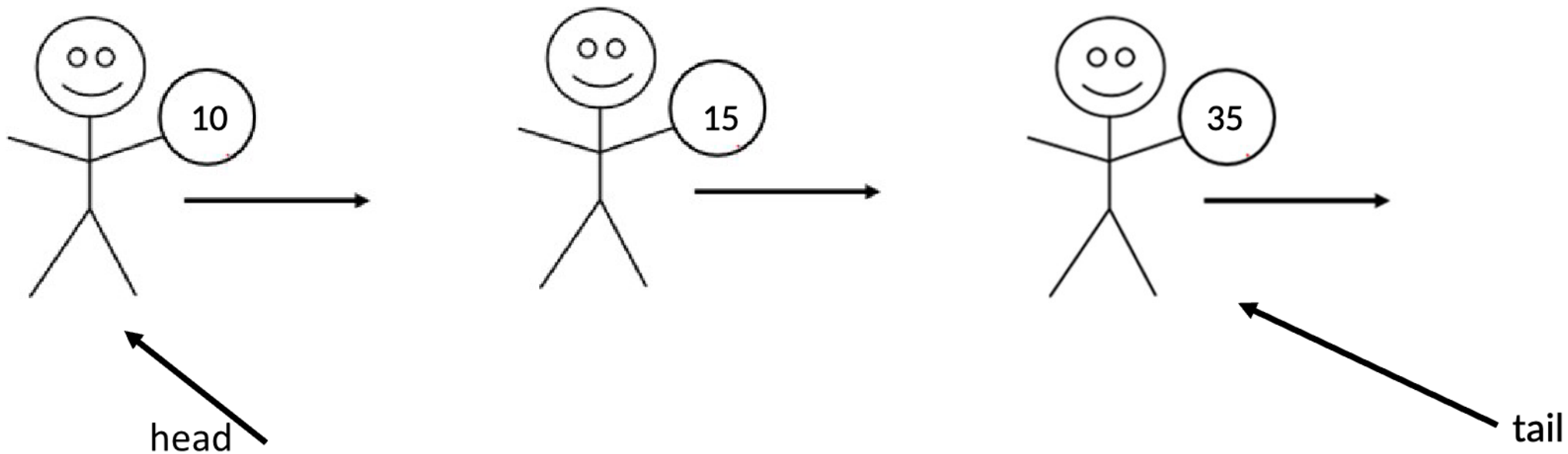
What if we could access both ends?

Keep track of tail



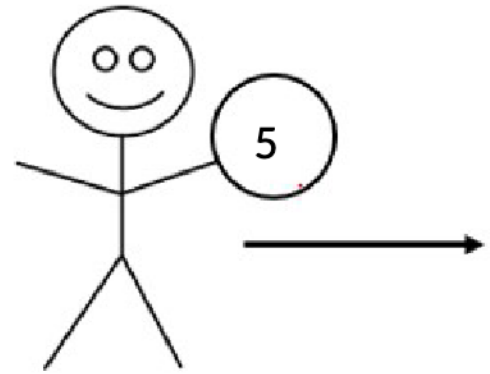
Add to tail

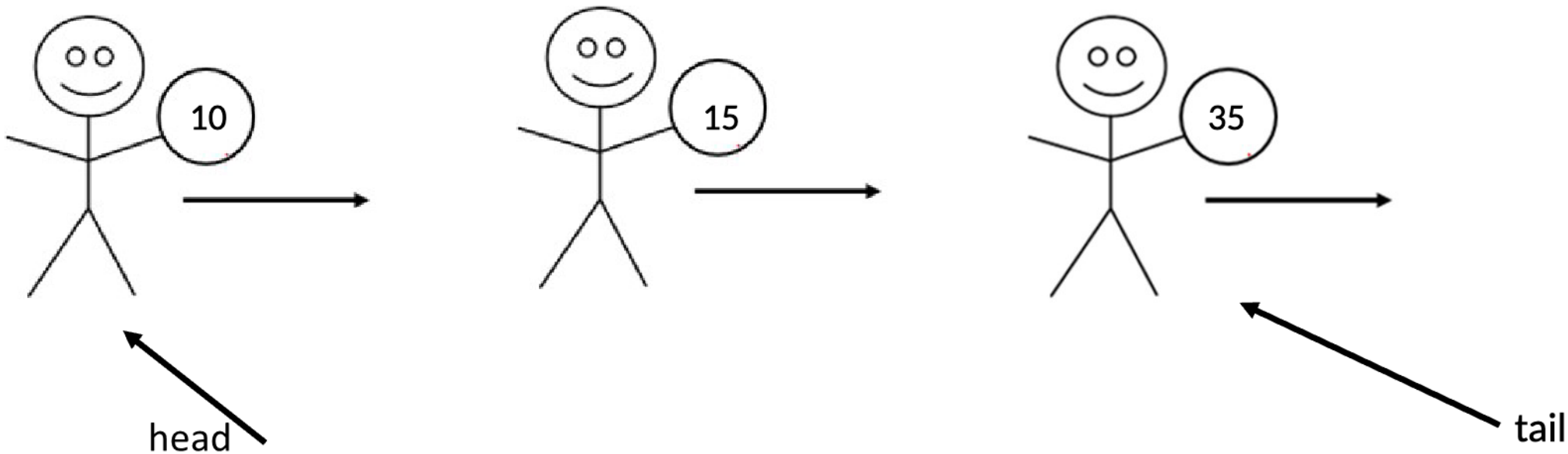
```
tail.setLink(new Node<E>(E elementToAdd, null));
tail = tail.getLink();
```



Add to tail

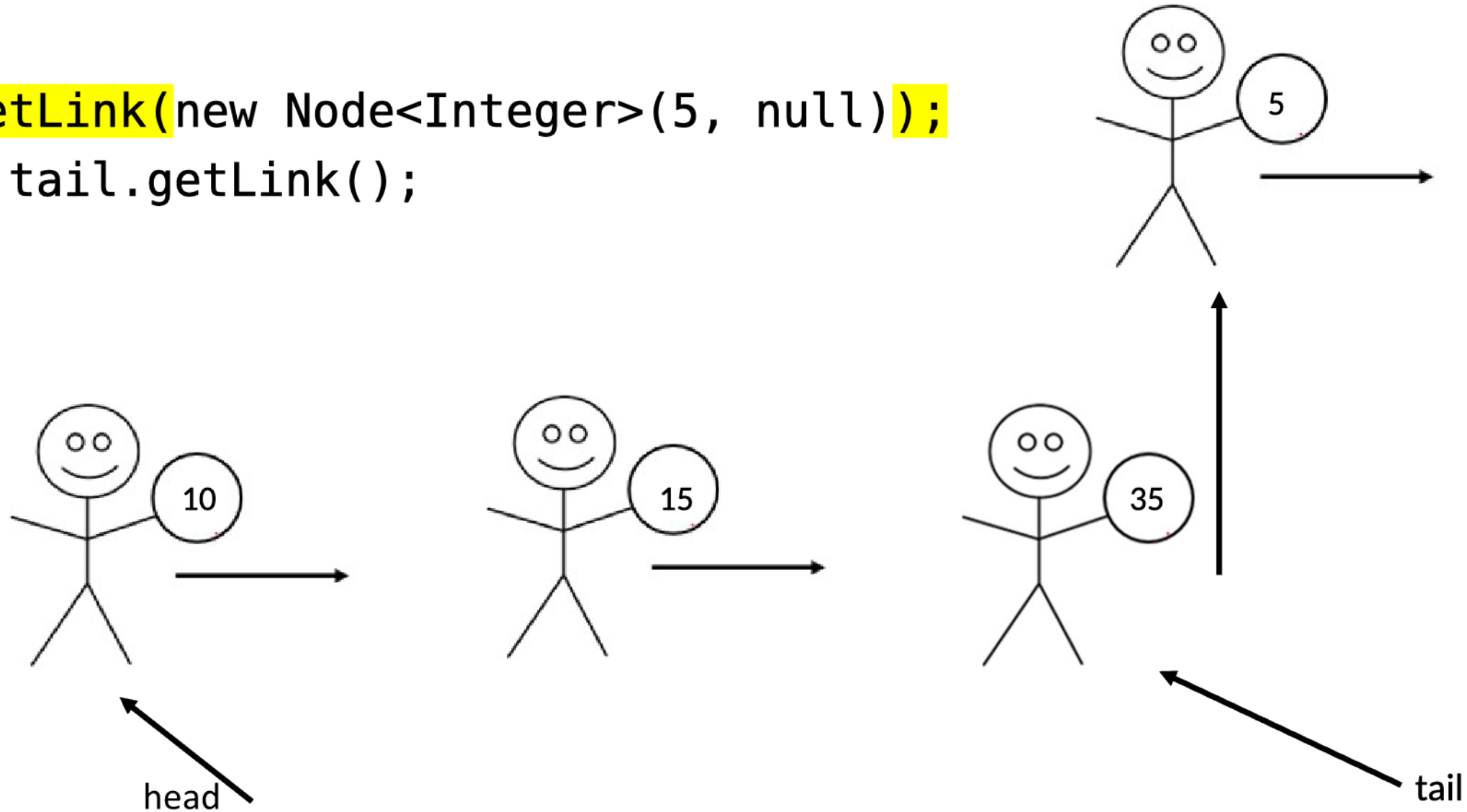
```
tail.setLink(new Node<Integer>(5, null));
tail = tail.getLink();
```





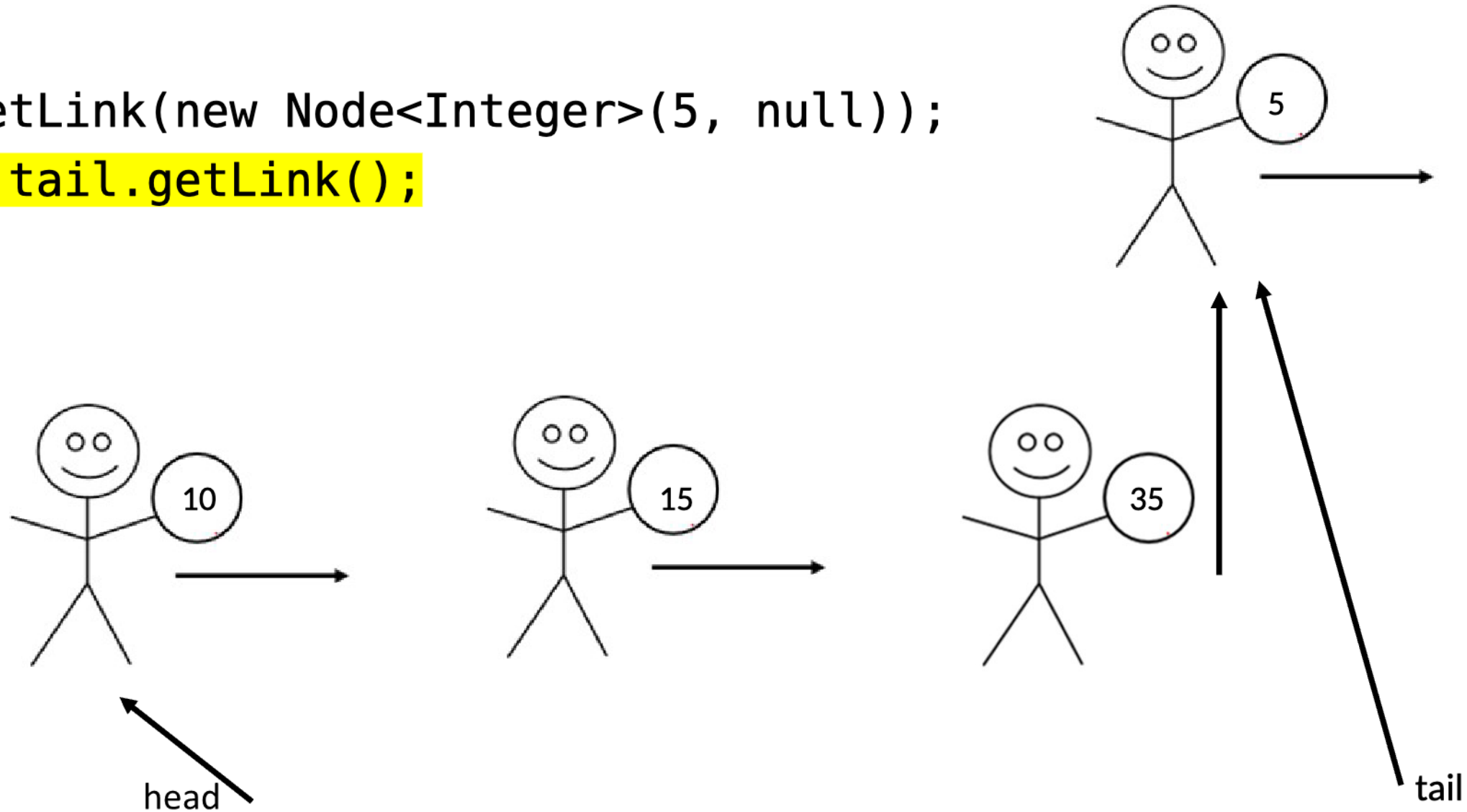
Add to tail

```
tail.setLink(new Node<Integer>(5, null));
tail = tail.getLink();
```



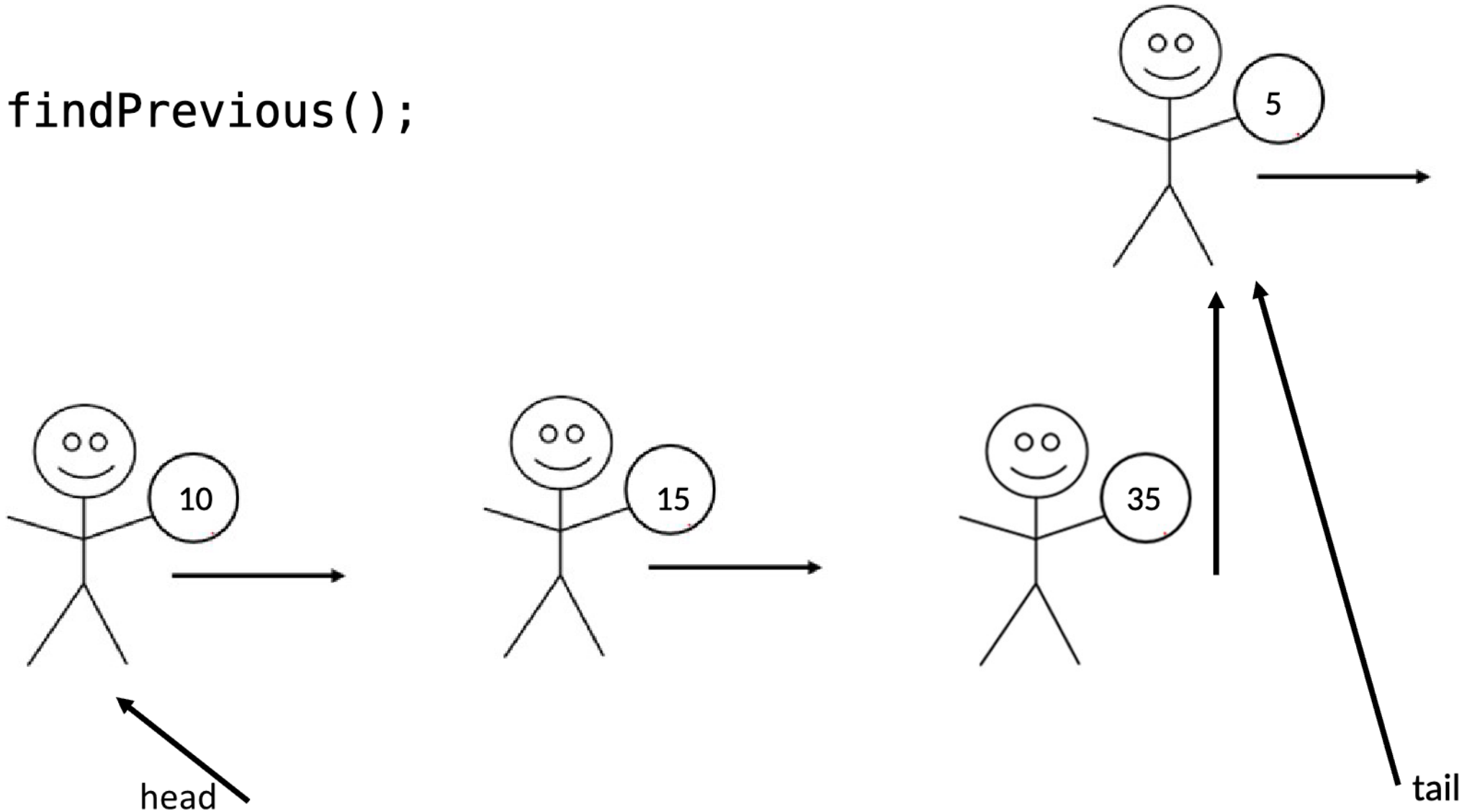
Add to tail

```
tail.setLink(new Node<Integer>(5, null));
tail = tail.getLink();
```



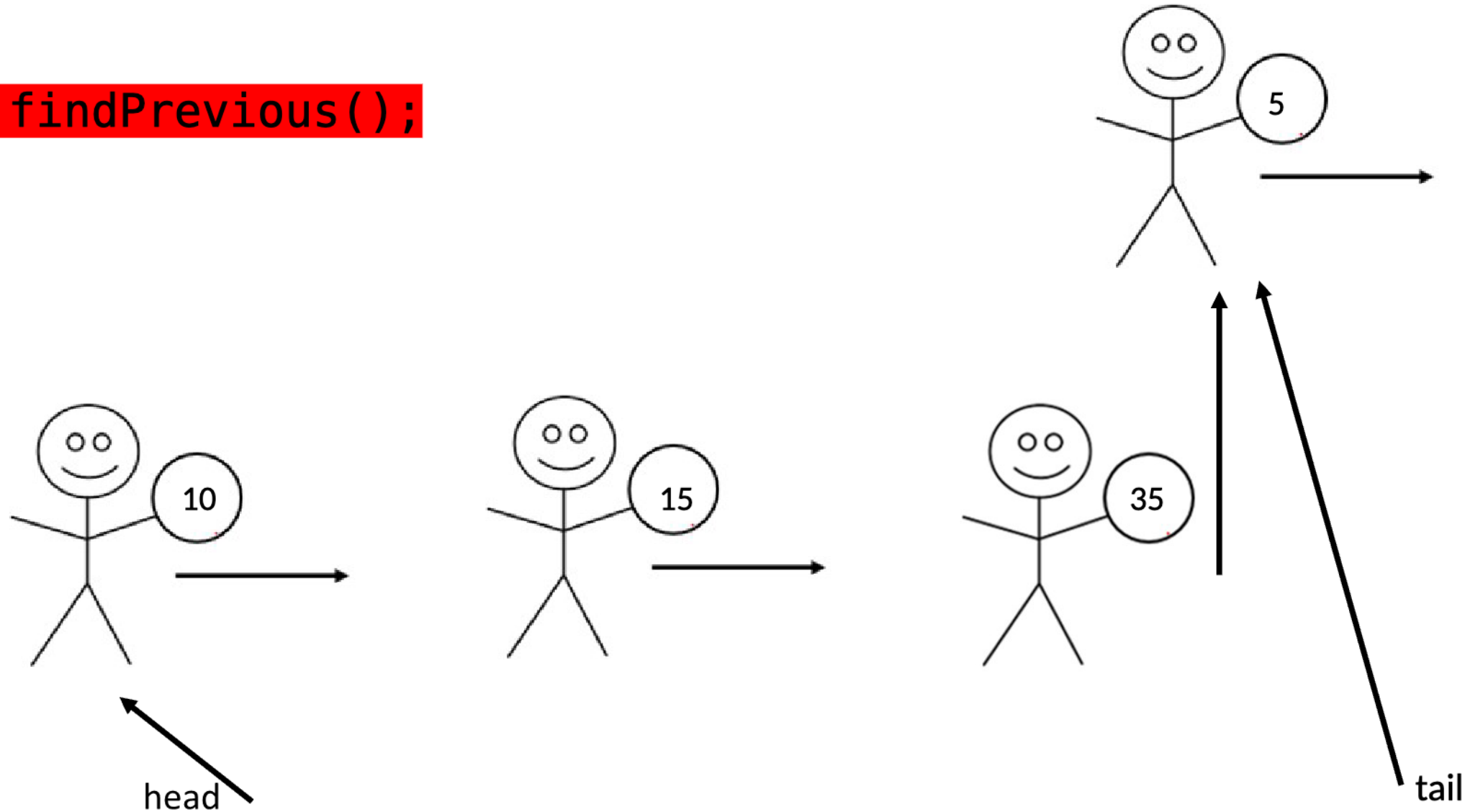
Remove from tail

```
tail = findPrevious();
```

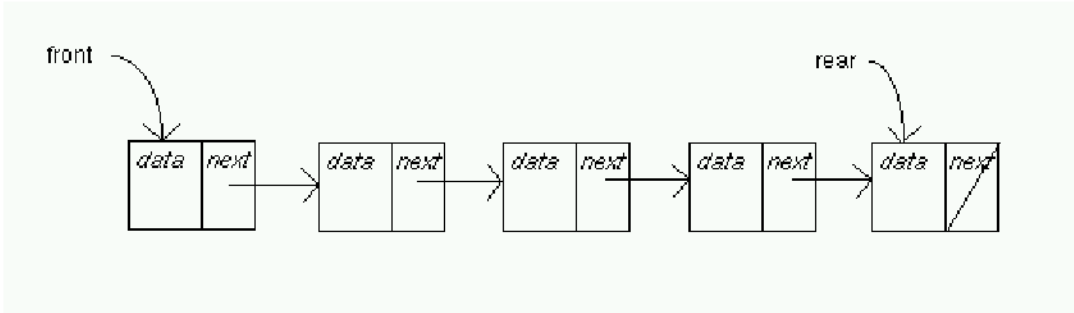


Remove from tail

```
tail = findPrevious();
```



LinkQueue Implementation




```
public class LinkQueue<E> {
        private int manyNodes;
        private Node<E> head;
        private Node<E> tail;

}
```

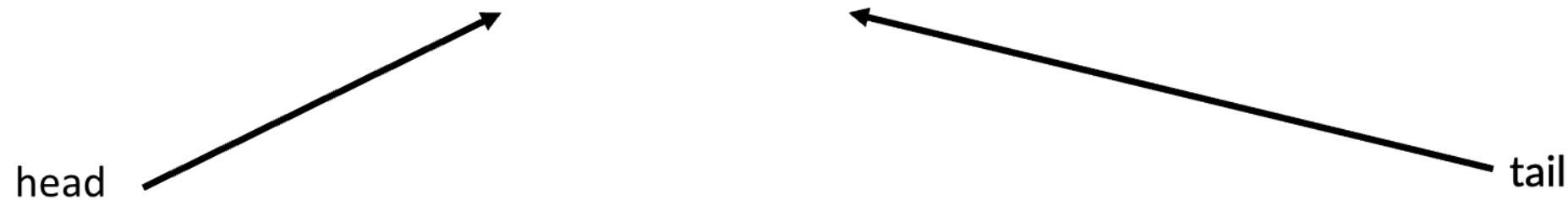
- Keep two pointers, head and tail, to the first and last elements in the queue respectively.
- How to perform `enqueue()` and `dequeue()`?
  - Case 1: add at **tail**, remove at **head**:
    - Insert O(1)
    - Delete O(1)
  - Case 2: add at **head**, remove at **tail**: **not used in practice – listed here just for comparison purposes**
    - Insert O(1)
    - Delete O(n) ==> **why?**

Danger!!!

- What about the border cases?
  - Adding to empty queue
  - Removing last element in the queue

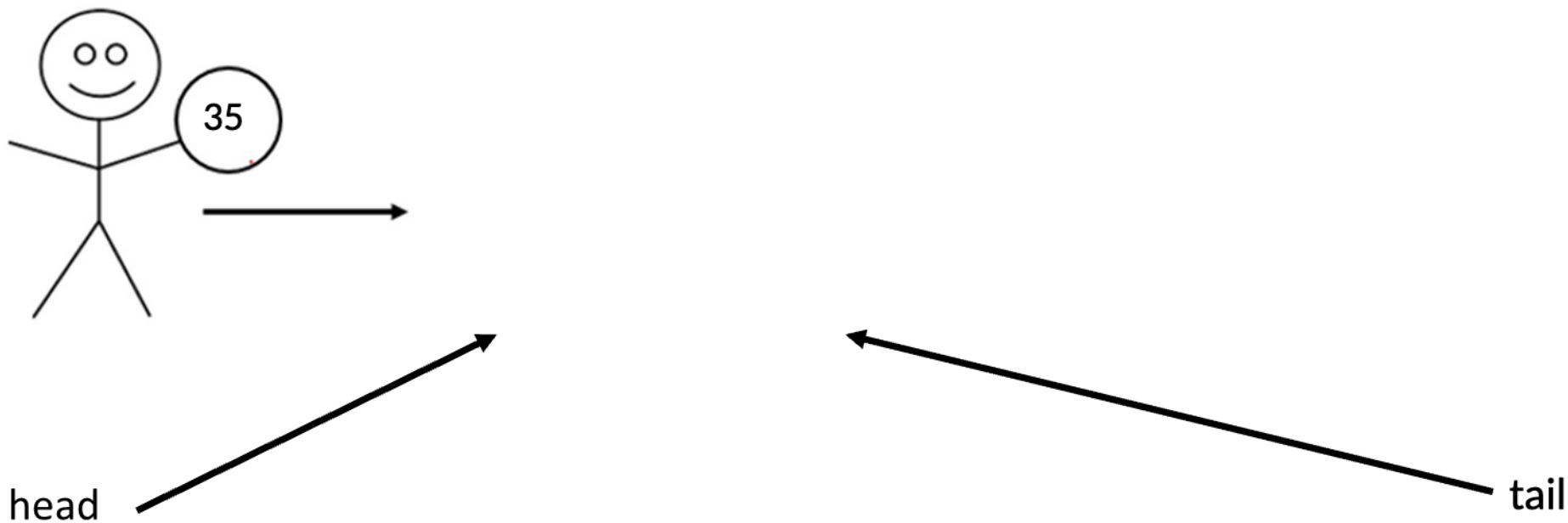
Add to empty queue

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



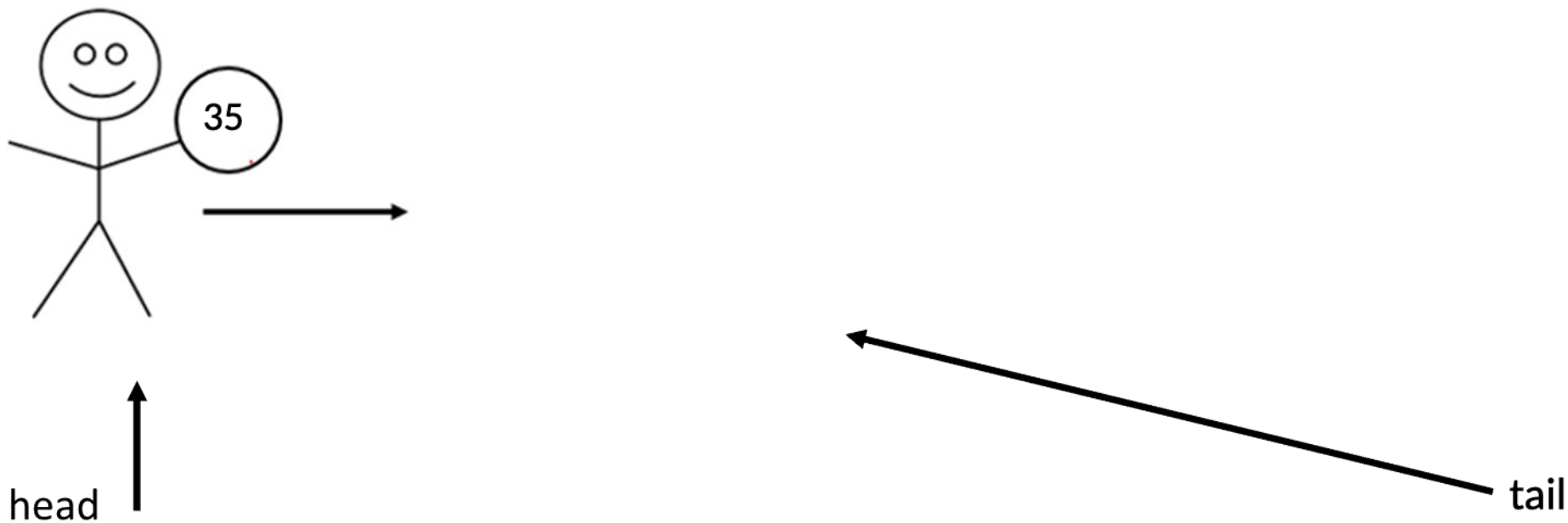
Add to empty queue

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



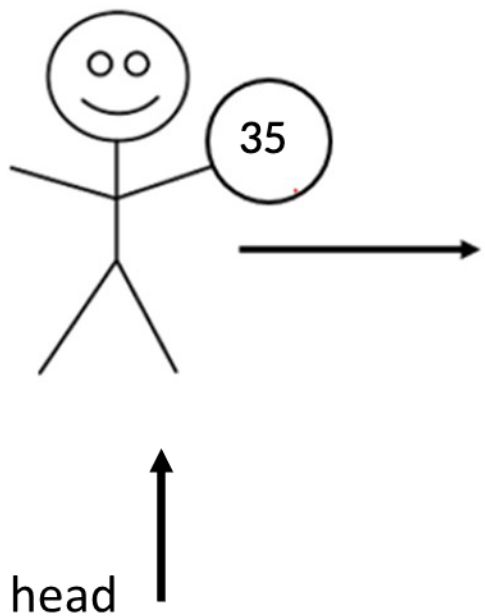
Add to empty queue

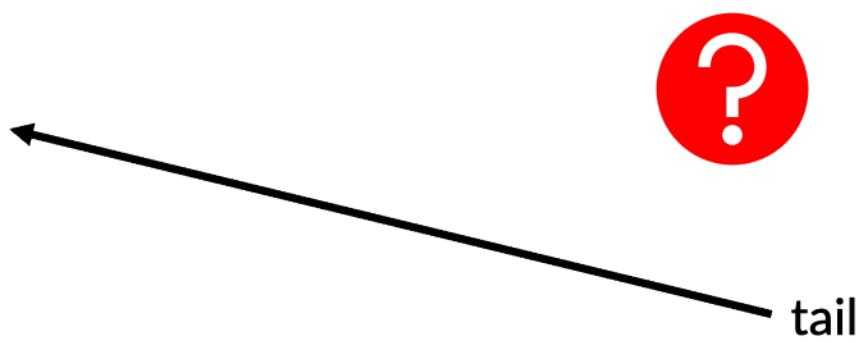
```
head = new IntNode(35, head);
```



Add to empty queue

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



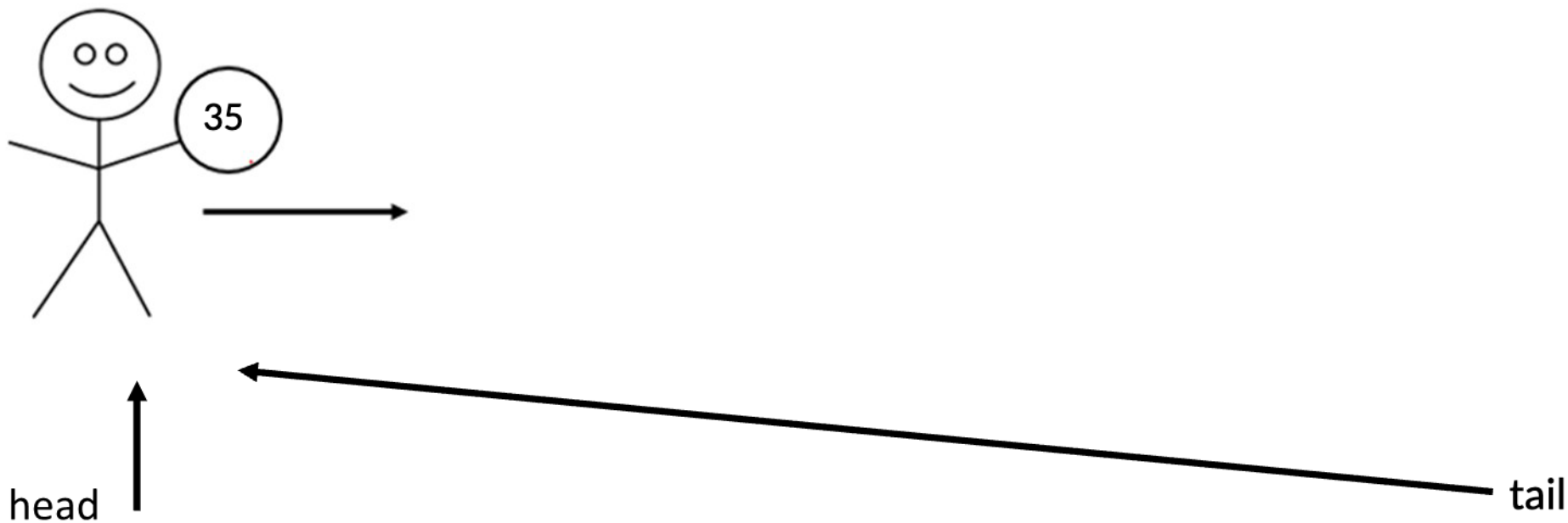


enqueue()

```
public void enqueue(E item){
      if (manyNodes == 0){  {
            head = new Node<E>(item, null);
            tail = head;
      } else     {
            tail.setLink(new Node<E>(item,
null));
            tail = tail.getLink();
      }
      manyNodes++;
}
```

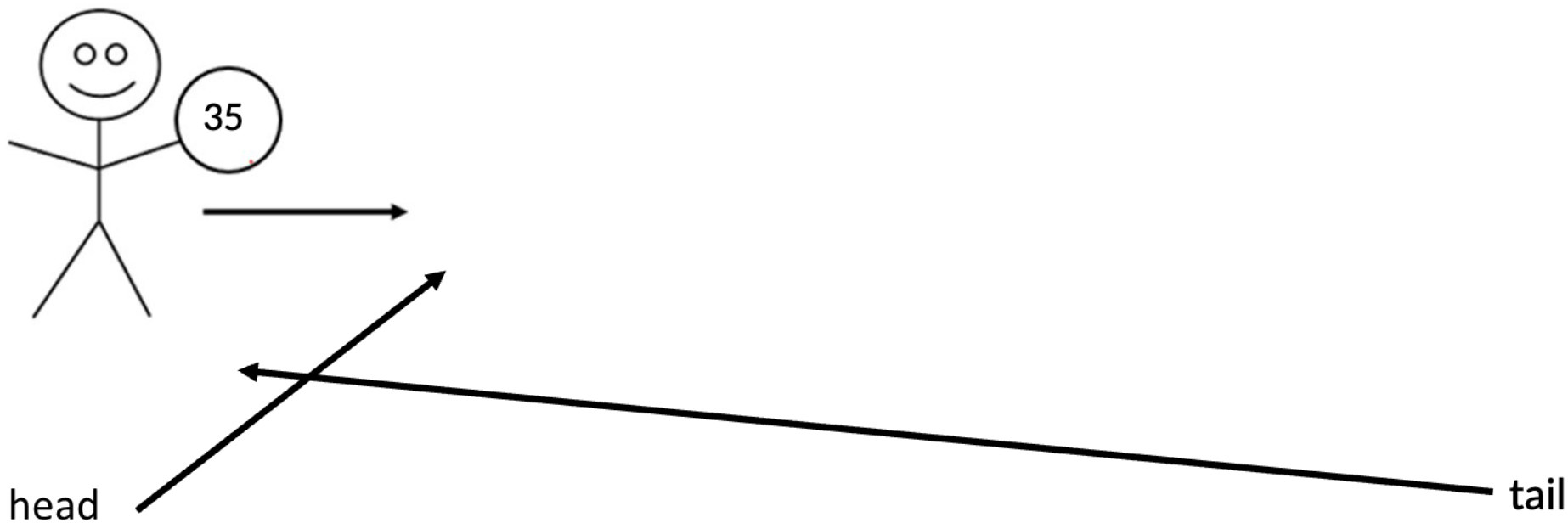
Remove last element from queue

head = head.getLink();



Remove last element from queue

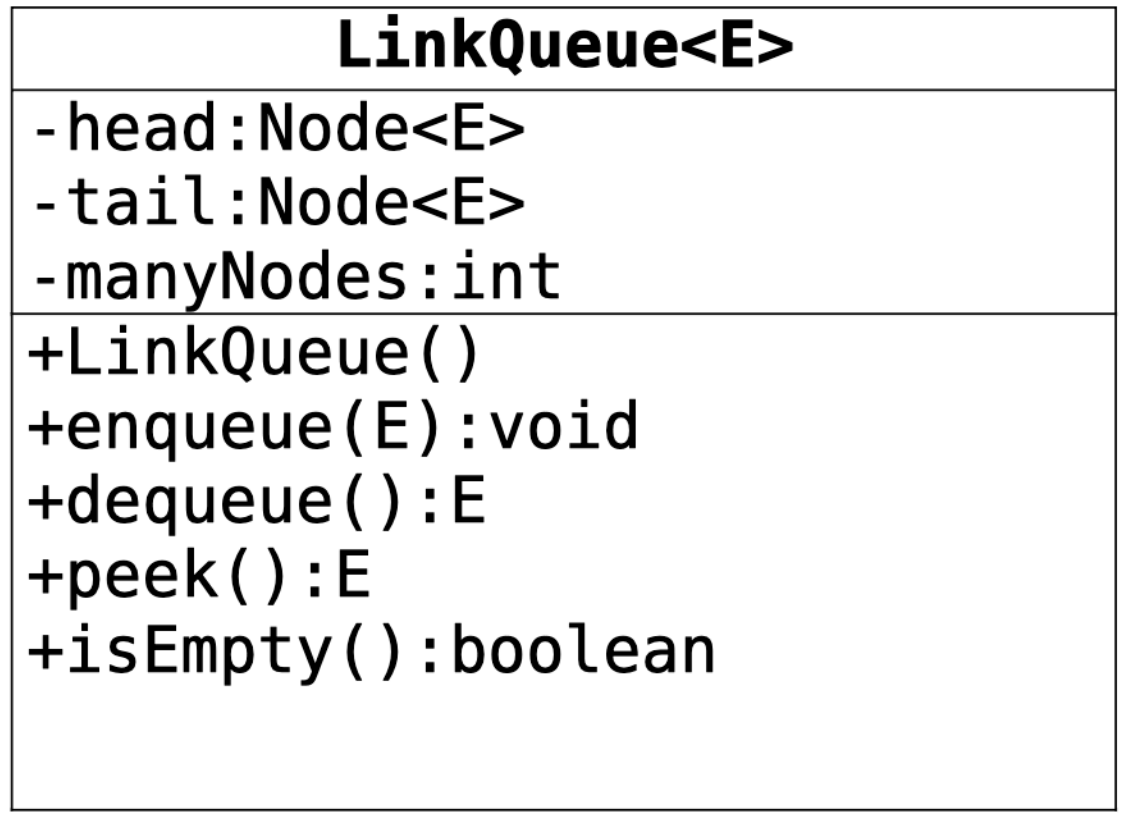
```
head = head.getLink();
```



dequeue()

```
public E dequeue()    {
      E element = head.getData();
      head = head.getLink();
      manyNodes--;
      if (manyNodes==0){
            tail = null;
      }
      return element;
}
```

LinkQueue Class Diagram



Invariant for `LinkQueue`

```
/** Invariant of the LinkQueue class:
* 1. The number of items in the queue is stored in the instance
variable
*    manyNodes
* 2. The items in the queue are stored in a linked list. The front
of the
*    queue is at the head of the list and is referenced by instance
*    variable head. The instance variable tail points to the end of
*    the queue and references the most recently added item.
* 3. For an empty queue, both head and tail are null.
*/
```

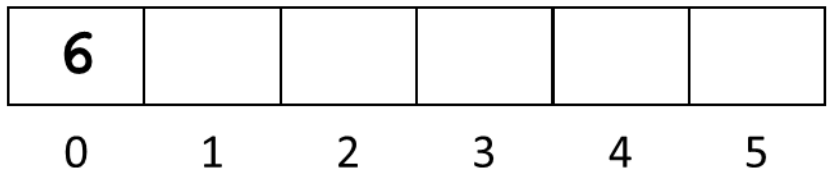
Array Implementation

How would you implement a queue with an array?

Challenges in `ArrayQueue` Implementation

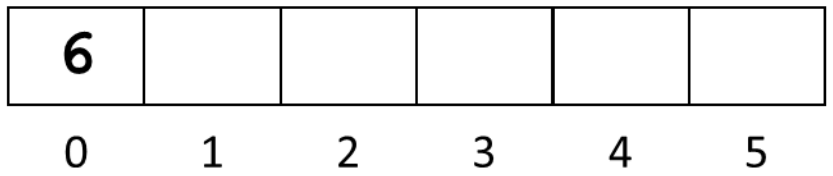
- How to maintain `head` and `manyItems`?
  - Solution 1:
    - Make `head` always at position 0 of the array
    - Always insert at `manyItems` position ␣ insert is constant time **O(1)**
    - When an item is removed (i.e., data[0] is removed):
      - move all the remaining items up to fill the empty slot at position 0 ␣ delete is linear time **O(n)**
  - Solution 2: Use a **circular array**
    - **Goal is to avoid the O(n) cost of deletion**
    - When using a circular array, then both insertion and deletion will be O(1)
    - Be careful about where the queue elements are stored in the array

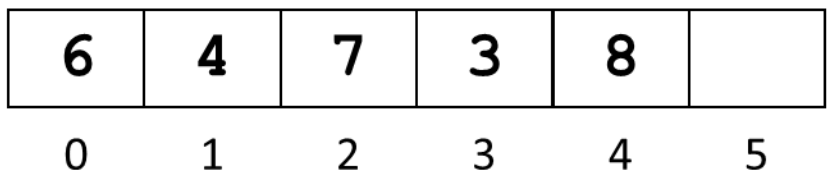
Array implementation



head = 0
manyItems = 1
What spot would we fill in next?

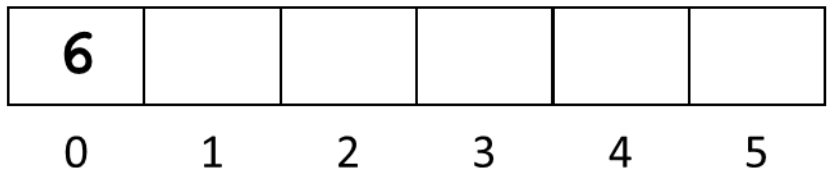
Array implementation

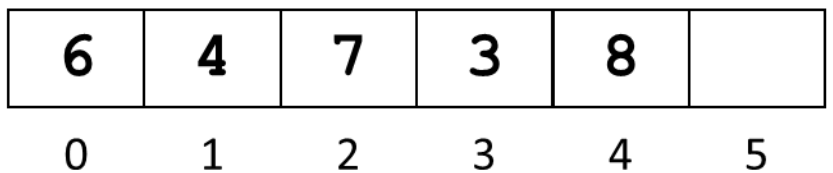


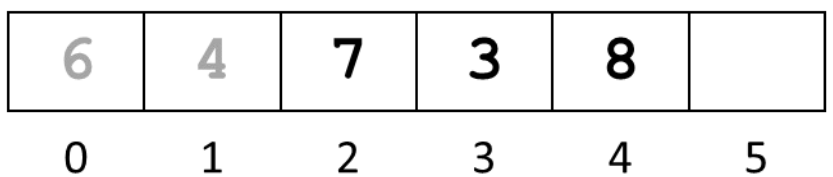


head = 0
manyItems = 5
What spot would we fill in next?

Array implementation

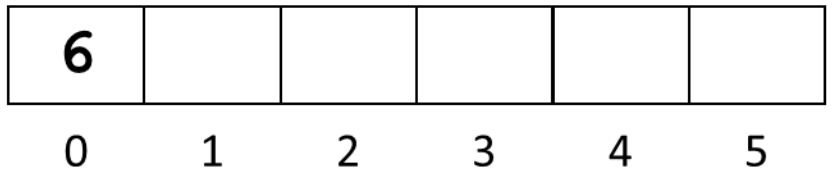


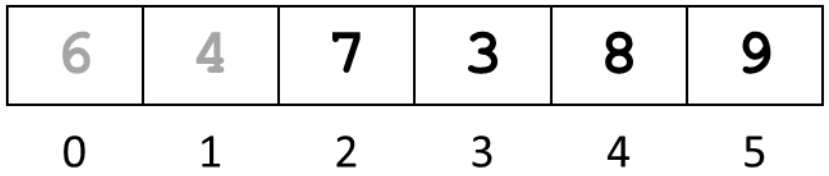


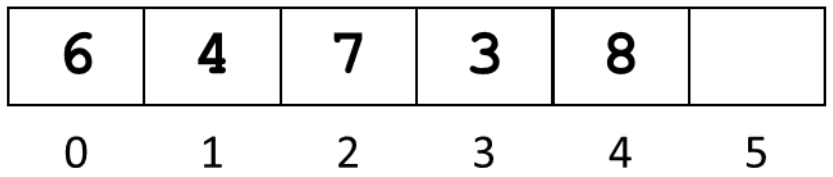


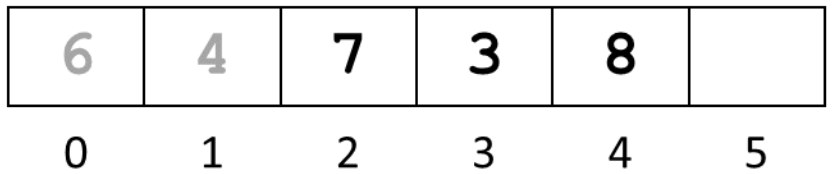
head = 2
manyItems = 3
What spot would we fill in next?

Array implementation



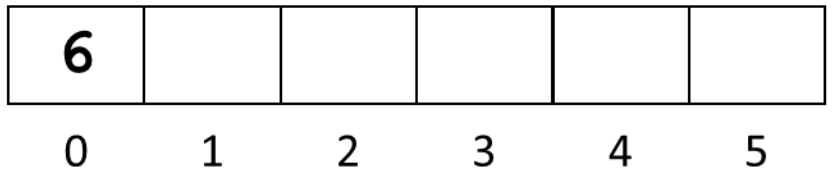


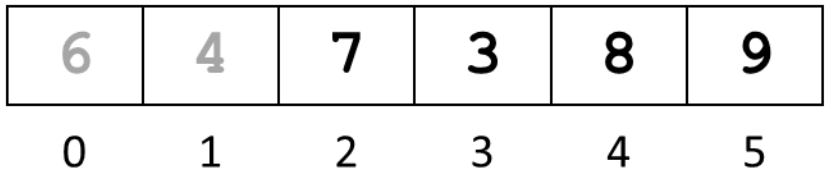


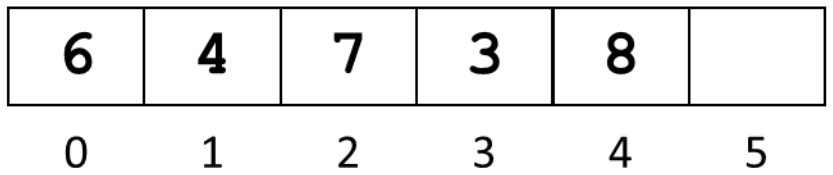


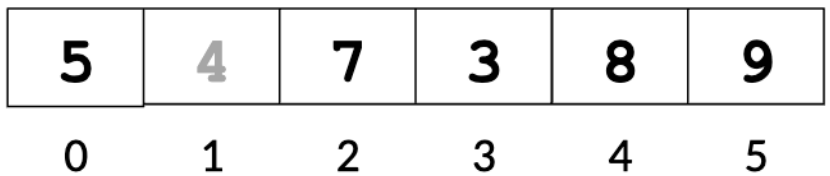
head = 2
manyItems = 4
What spot would we fill in next?

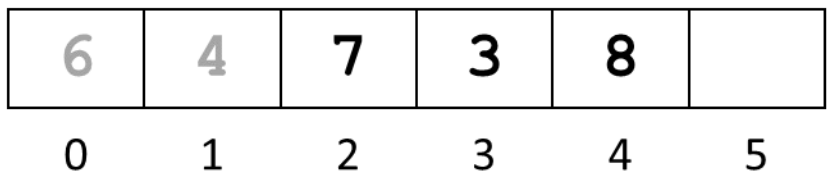
Array implementation







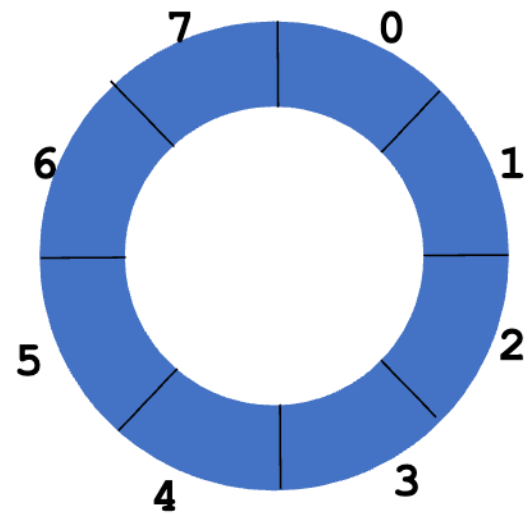




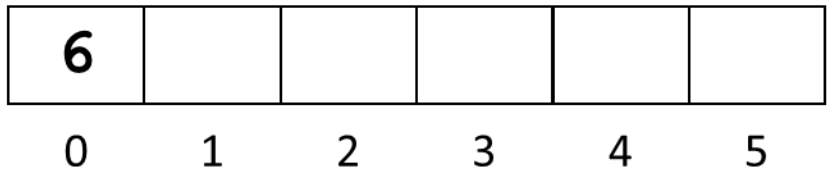
head = 2
manyItems = 5
What spot would we fill in next?

Circular Arrays

- Use a **circular array** to insert and remove items from a queue in constant time
- The idea of a circular array is that the end of the array “wraps around” to the start of the array



Array implementation

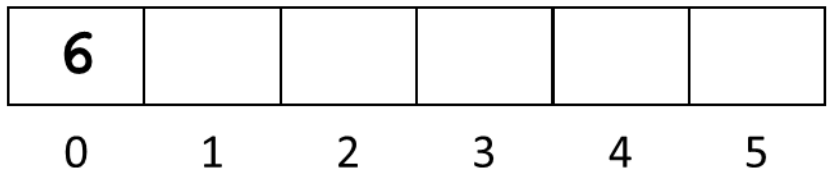


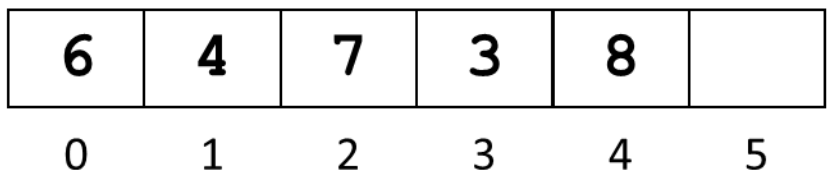
(head + manyItems) % array.length

(0 + 1) % 6 = 1

head = 0
manyItems = 1
What spot would we fill in next? 1

Array implementation



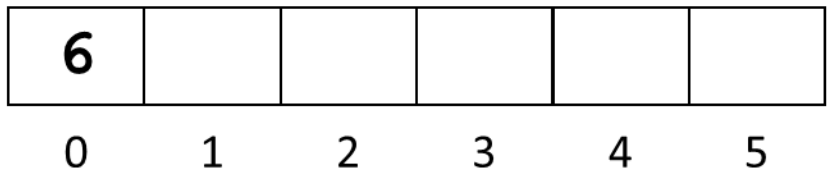


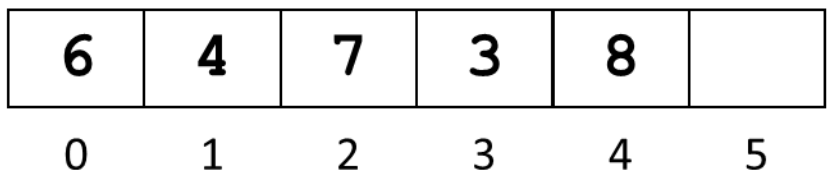
(head + manyItems) % array.length

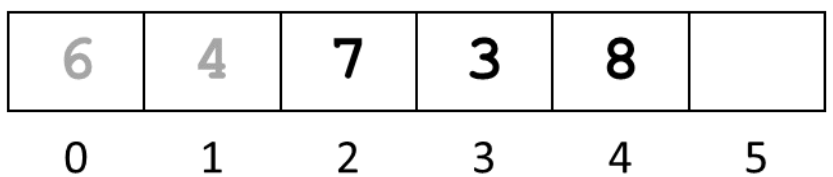
(0 + 5) % 6 = 5

head = 0
manyItems = 5
What spot would we fill in next? 5

Array implementation





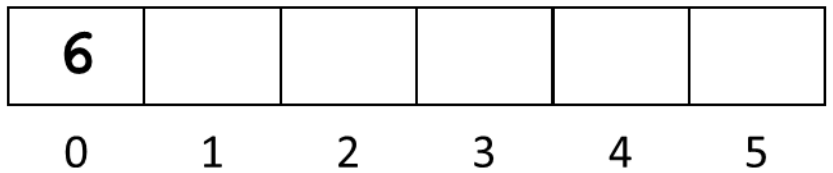


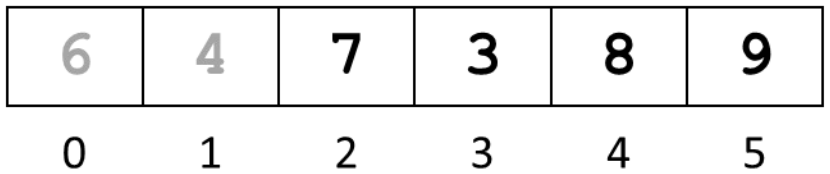
(head + manyItems) % array.length

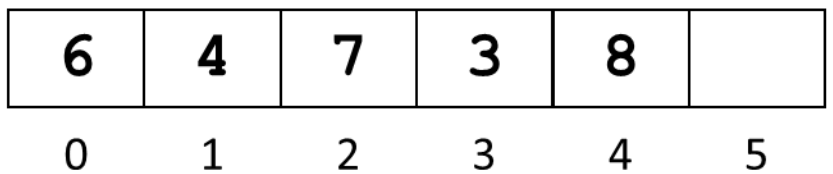
(2 + 3) % 6 = 5

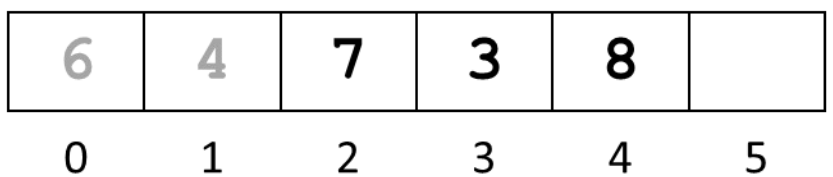
head = 2
manyItems = 3
What spot would we fill in next? 5

Array implementation







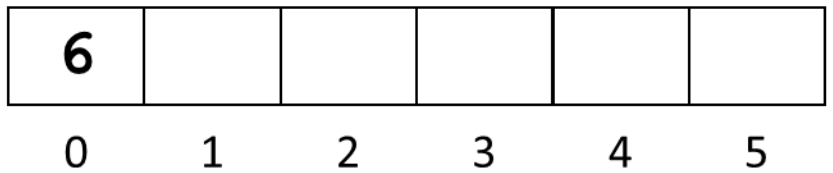


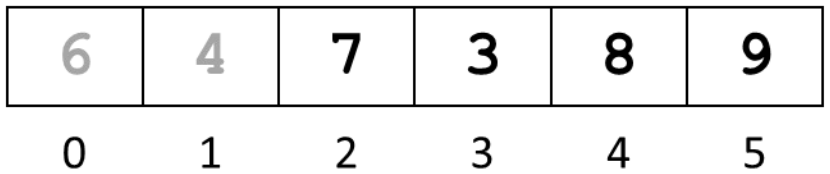
(head + manyItems) % array.length

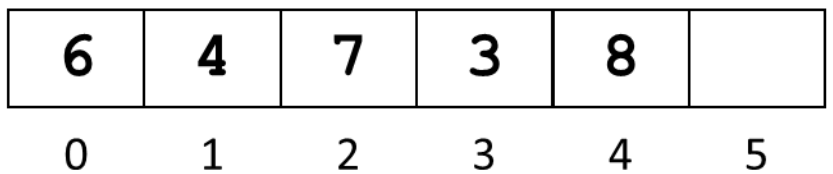
(2 + 4) % 6 = 0

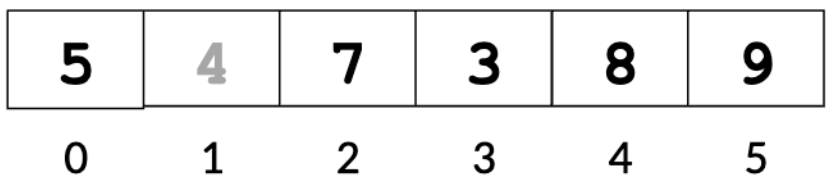
head = 2
manyItems = 4
What spot would we fill in next? 0

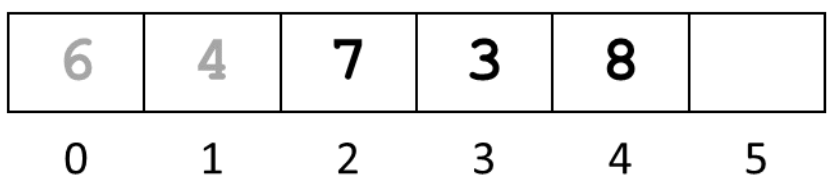
Array implementation











(head + manyItems) % array.length

(2 + 5) % 6 = 1

head = 2
manyItems = 5
What spot would we fill in next? 1

Adding or Deleting Elements

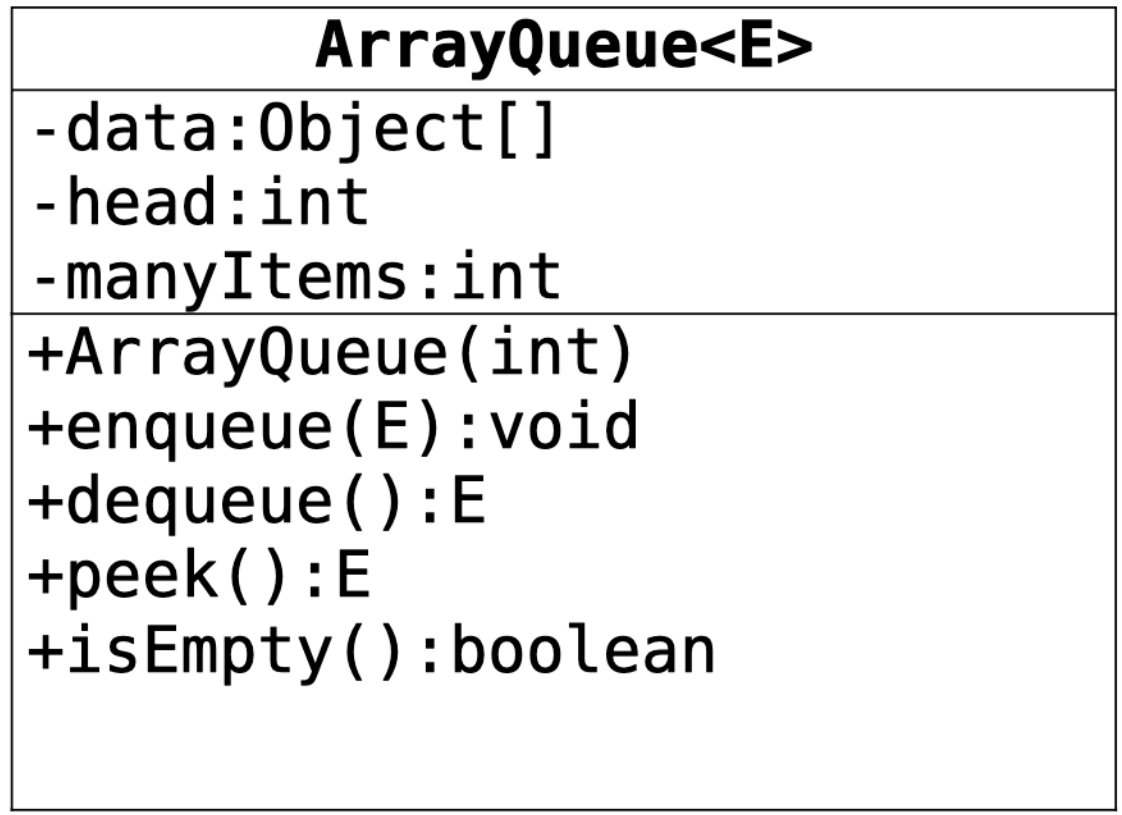
Add element to queue (enqueue)

```
data[(head + manyItems) % data.length] = elementToAdd;
manyItems++;
```

Remove element from queue (dequeue)

```
E temp = (E) data[head];
head = (head + 1) % data.length;
manyItems--;
return temp;
```

ArrayQueue Class Diagram



ArrayQueue Invariant

```
/** Invariant of the ArrayQueue class:
* 1. The number of items in the queue is stored in the instance
*    variable manyItems. 0 <= manyItems <= data.length
* 2. For a non-empty queue, the items are stored in a circular array
*    beginning at data[head] and continuing through
*    data[(head + manyItems) % data.length]
* 3. For an empty queue, manyItems is zero and data is a reference to
*    an array, but we do not care about head position
*/
```

Try it!

- Create a generic queue class. Test it with a driver.