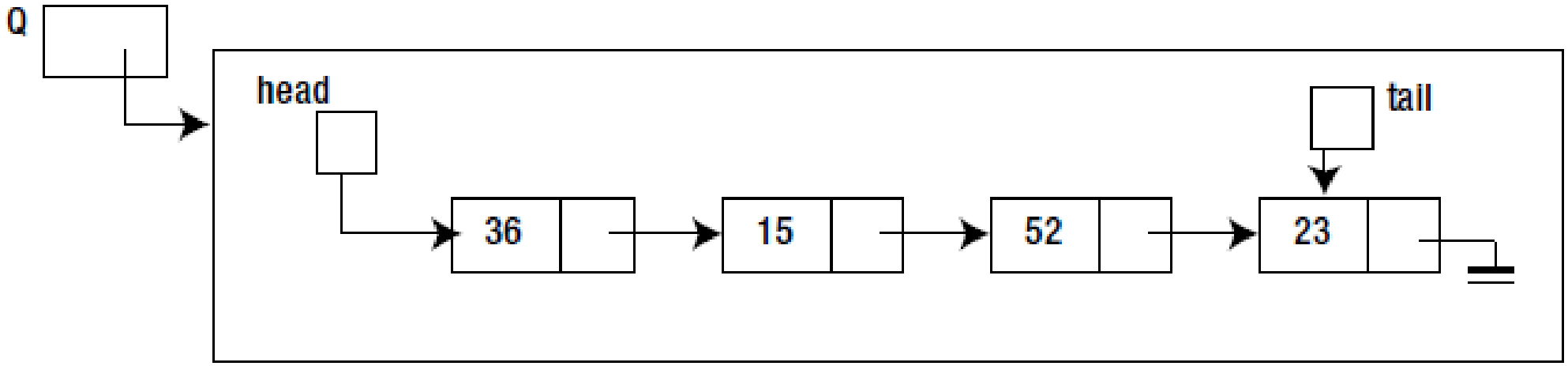
Data Structures and Algorithms

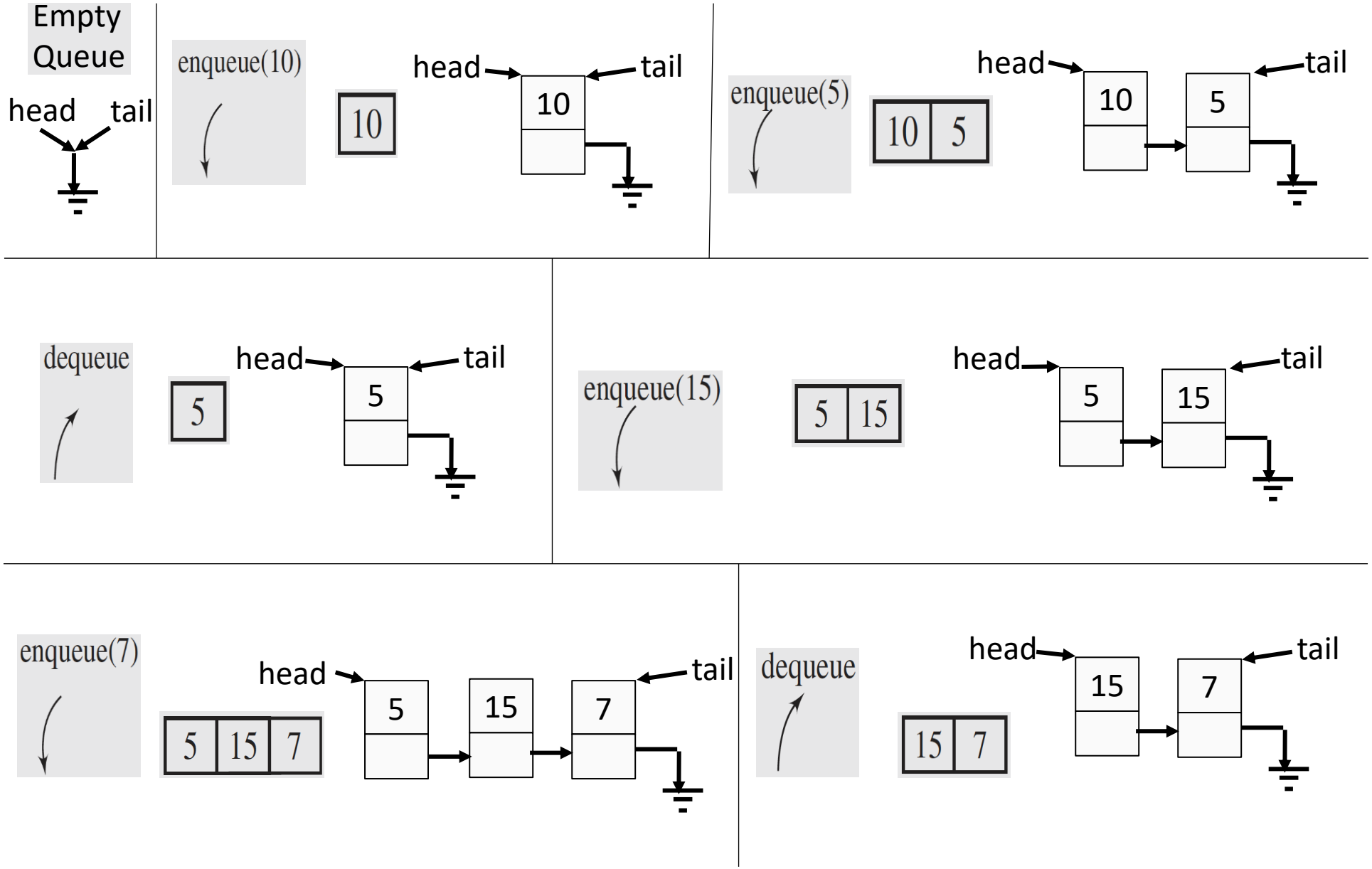
2.5.2 Implementing a Queue using Linked Lists

Implementing a Queue Using a Linked List

- As with stacks, we can implement a queue using linked lists.
- This has the advantage of us not having to decide beforehand the size of the queue.
- We use two links, head and tail, to refer to the first and last nodes in the queue.
- The figure below shows a queue with four items (36, 15, 52, and 23).



Demonstration of enqueue() and dequeue()



Code for the Queue ADT using Nodes

- We will use the same classes NodeData and Node used in the implementation of the Linked List class and the Stack class.

- The class Queue has two data fields head and tail, both are references to Node.

- The Queue ADT has the following operations:
  1. Create an empty queue
  2. isEmpty()
  3. enqueue()
  4. dequeue()
  5. head()
  6. tail()

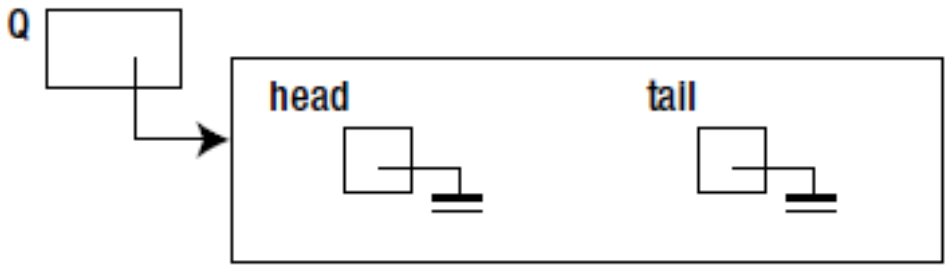
Data fields and constructor

```
public class Queue
{
    private Node head, tail;

    //create an empty queue
    public Queue()
    {
        head = tail = null;
    }
```

We can create an empty queue with the following statement:

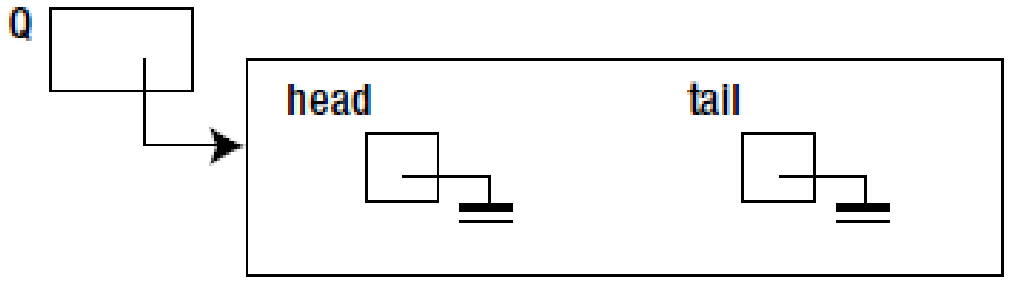
```
Queue Q = new Queue();
```



Operation isEmpty( )

The queue is empty if either the head or the tail is null.

```
// check if the queue is empty
    public boolean isEmpty ()
    {
        return(head == null);    // or return (tail == null)
    }
```



Operation enqueue( )

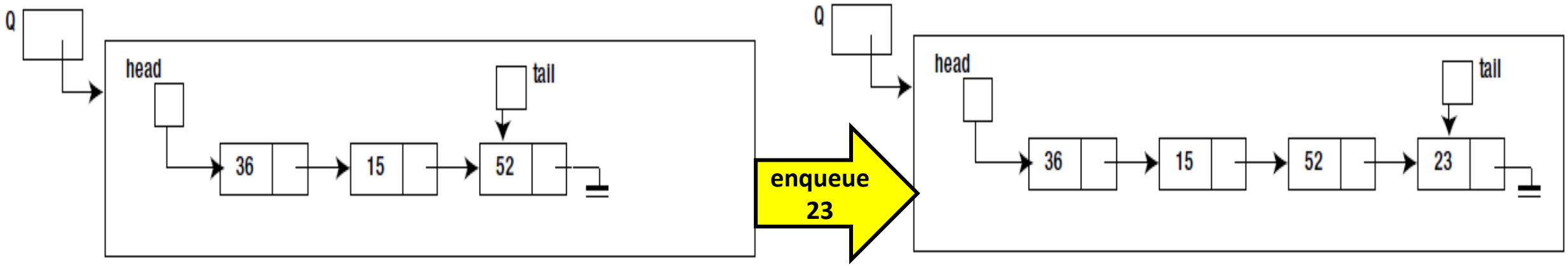
```
// Adds an element at the tail
public void enqueue(NodeData item){
    Node newNode = new Node(item);
    if (isEmpty()) {
        head = newNode;
        tail = newNode;
    }
    else {
        tail.next = newNode;
        tail = newNode;
    }
} // end enqueue
```

Create a new node

Special case: Empty queue

Have next of tail pointing to new node

Update tail to point to new node



Operation dequeue( )

```
// Removes and returns the element at the head
public NodeData dequeue() {
    if(isEmpty()) {
        System.out.println("The queue is empty.");
        System.exit(1);
    }
    NodeData nd = head.data;
    head = head.next;
    if (head == null) // if the queue becomes empty
         tail = null;
    return nd;
}   // end dequeue
```

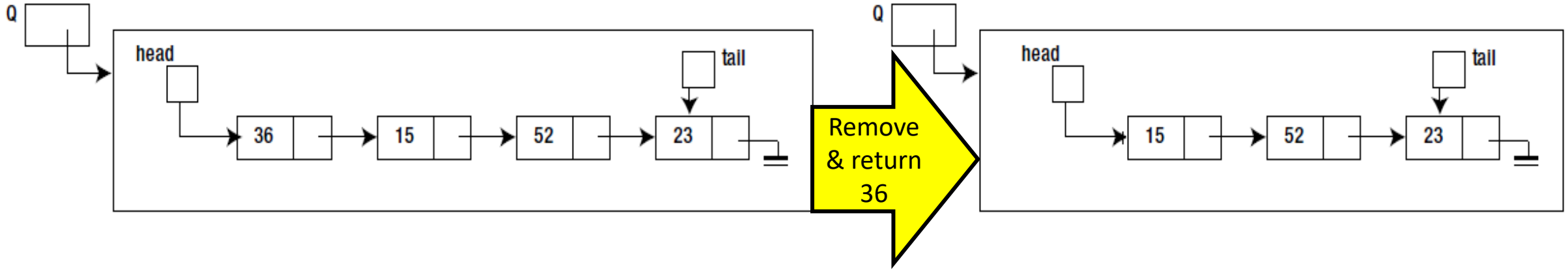
Special case: Empty queue

Save data of old head

Update head to point to its next (2nd node in the list)

Special case: List consisting of a single node

Return the data of the old head



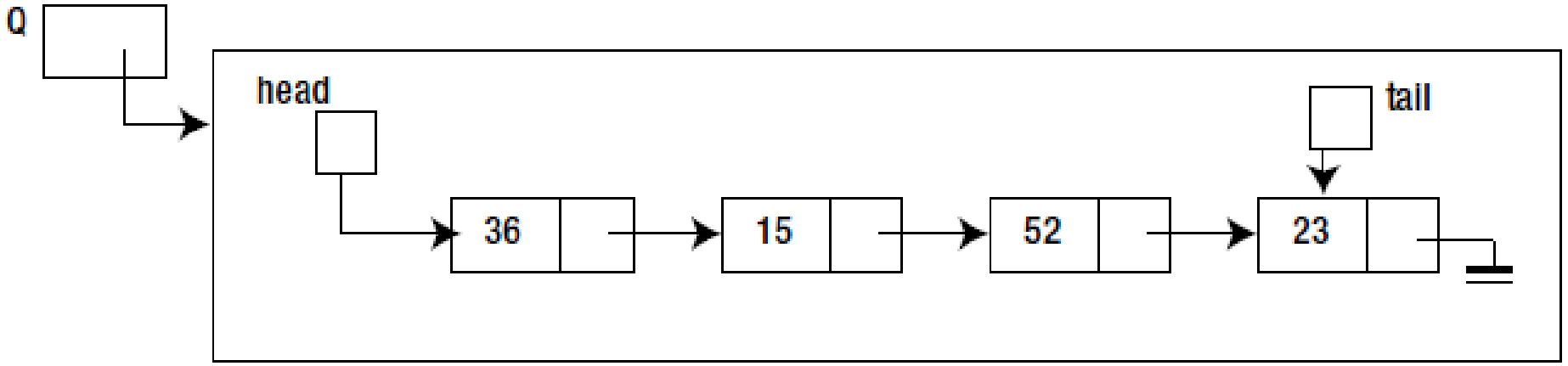
Operation head( )

```
// Returns the element at the head without removing it
public NodeData head()  {
    if(isEmpty()) {
        System.out.println("The queue is empty.");
        System.exit(1);
    }
    return head.data;
}   // end head
```

Special case: Empty queue

Return the data of the head

For the below example the method head( ) will return 36.



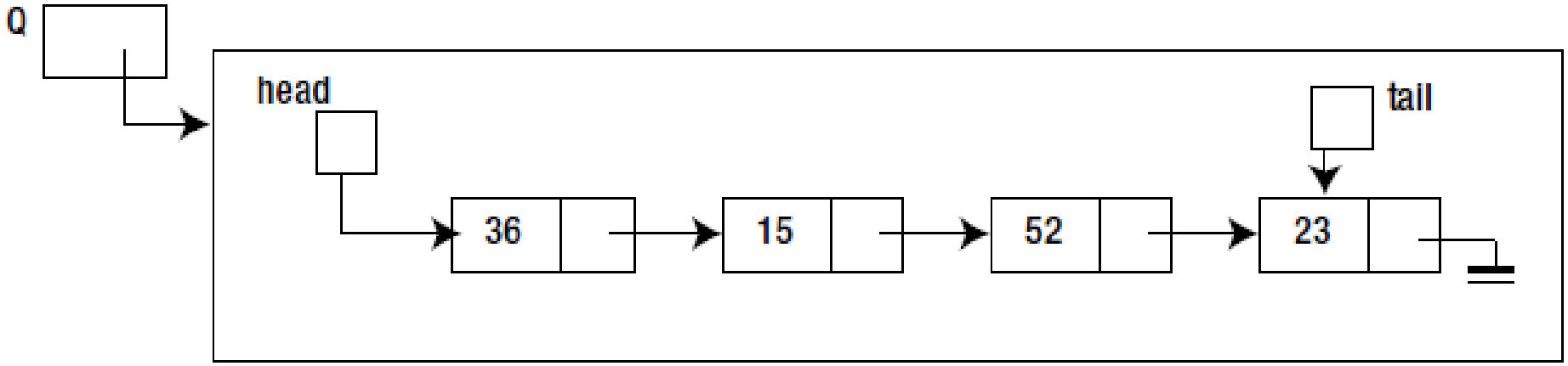
Operation tail( )

```
// Returns the element at the tail without removing it
public NodeData tail() {
    if(isEmpty()) {
        System.out.println("The queue is empty.");
        System.exit(1);
    }
    return tail.data;
}   // end tail
```

Special case: Empty queue

Return the data of the tail

For the below example the method tail( ) will return 23.



Code of Queue (Linked List Implementation)

Download the code of the Queue ADT (Linked List Implementation) posted to your CSCI378 Google Classroom along with this presentation. Run the application and test it.