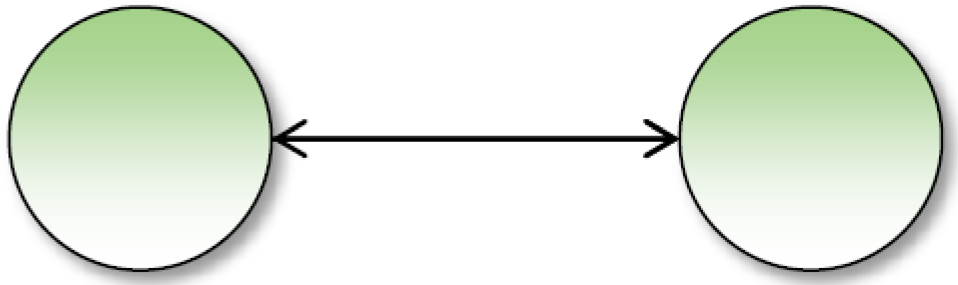
Session 16

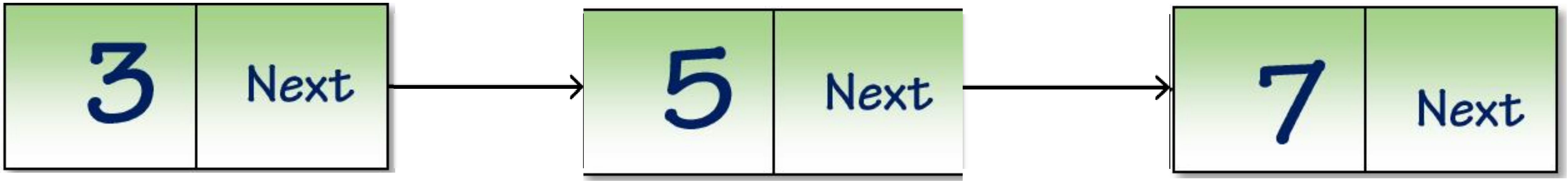
Linked List

Outline

- Node
- Node chains
- Linked List
- Doubly Linked List
- Modern Implementations



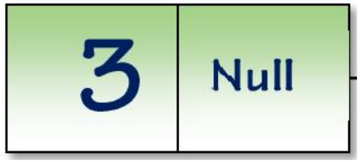
The Node



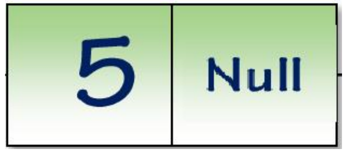
Node Chains

```
public class Node
{
    public int Value { get; set; }
    public Node Next { get; set; }
}
```

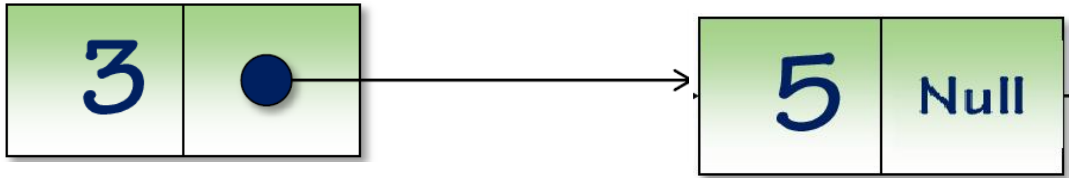
```
Node first = new Node { Value = 3 };
```



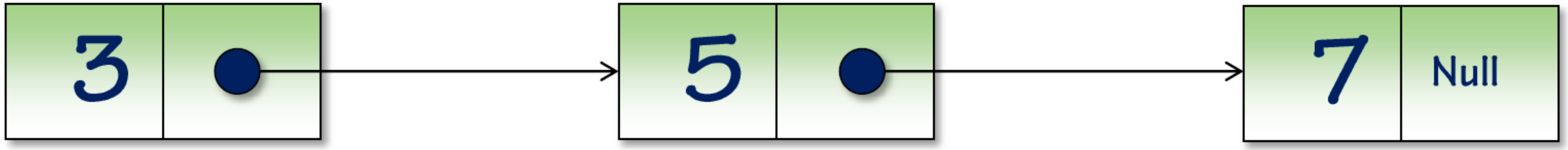
```
Node middle = new Node { Value = 5 };
```



```
first.Next = middle;
```



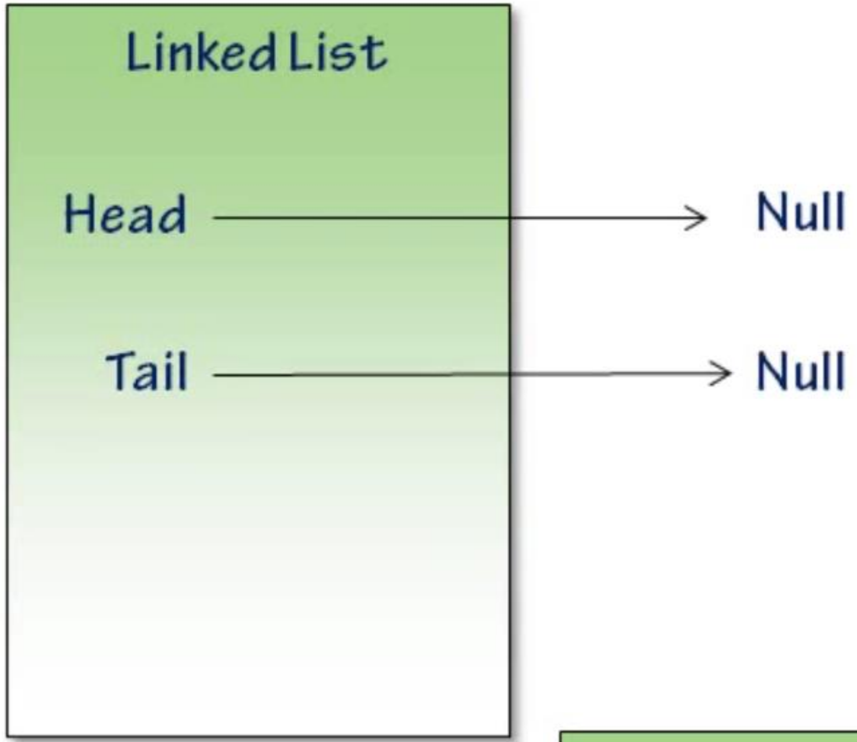
```
Node last = new Node { Value = 7 };
middle.Next = last;
```

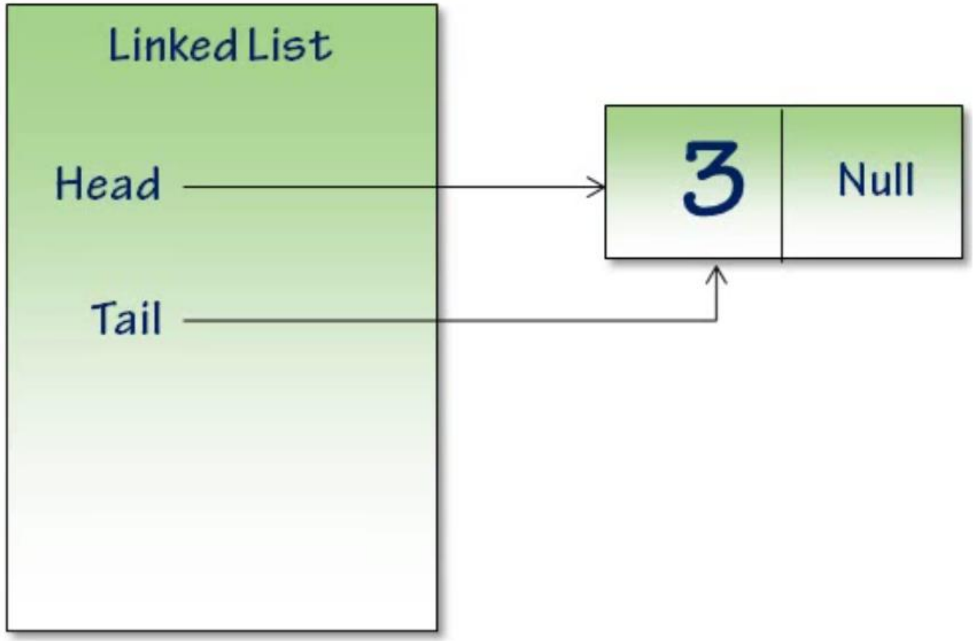


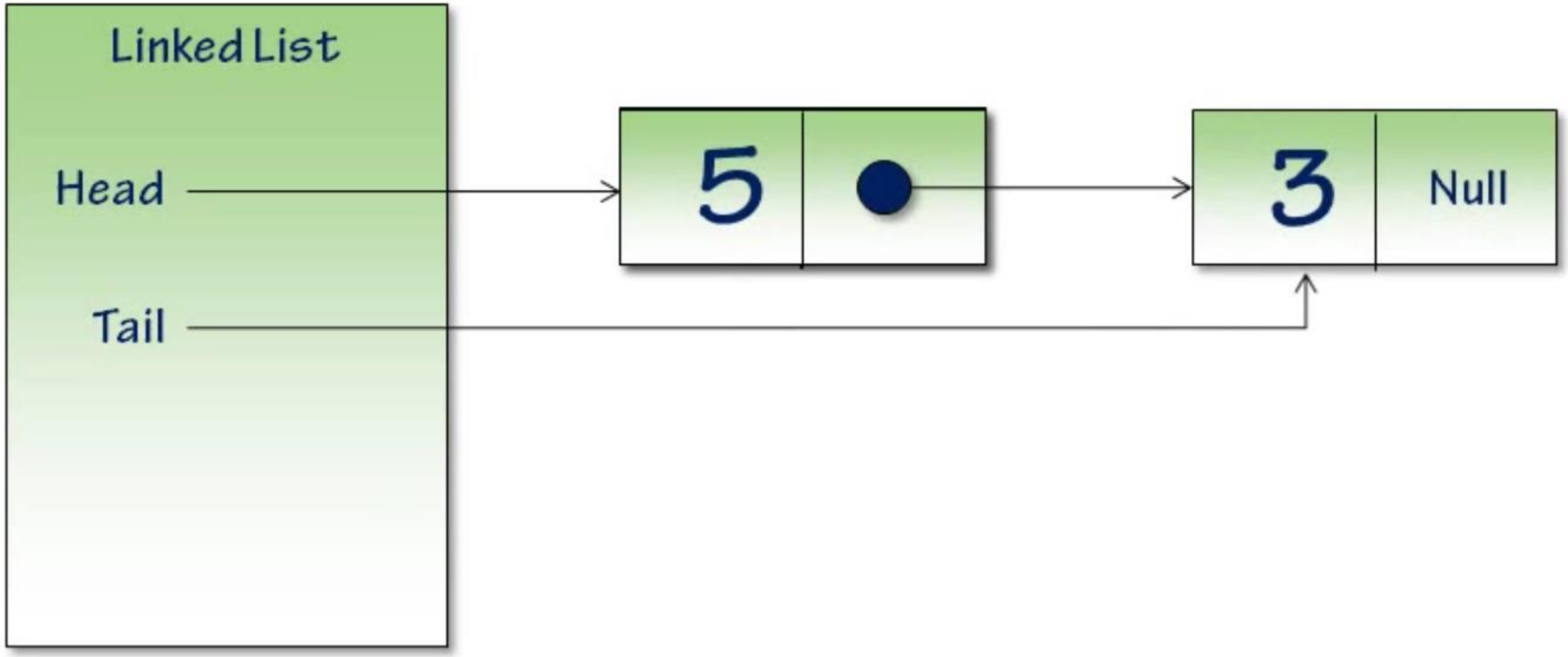
Linked List

- Single chain of nodes
- Head Pointer
- Tail Pointer
- Operations
  - Add
  - Remove
  - Find
  - Enumerate

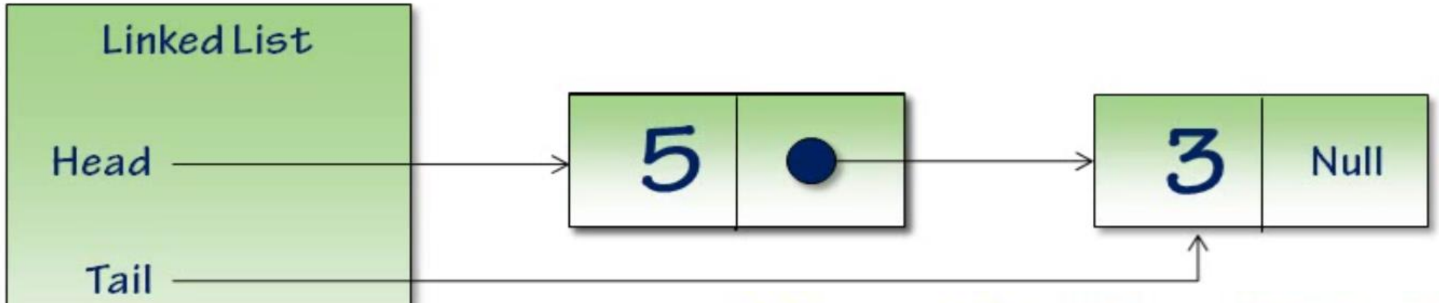
Add to Front

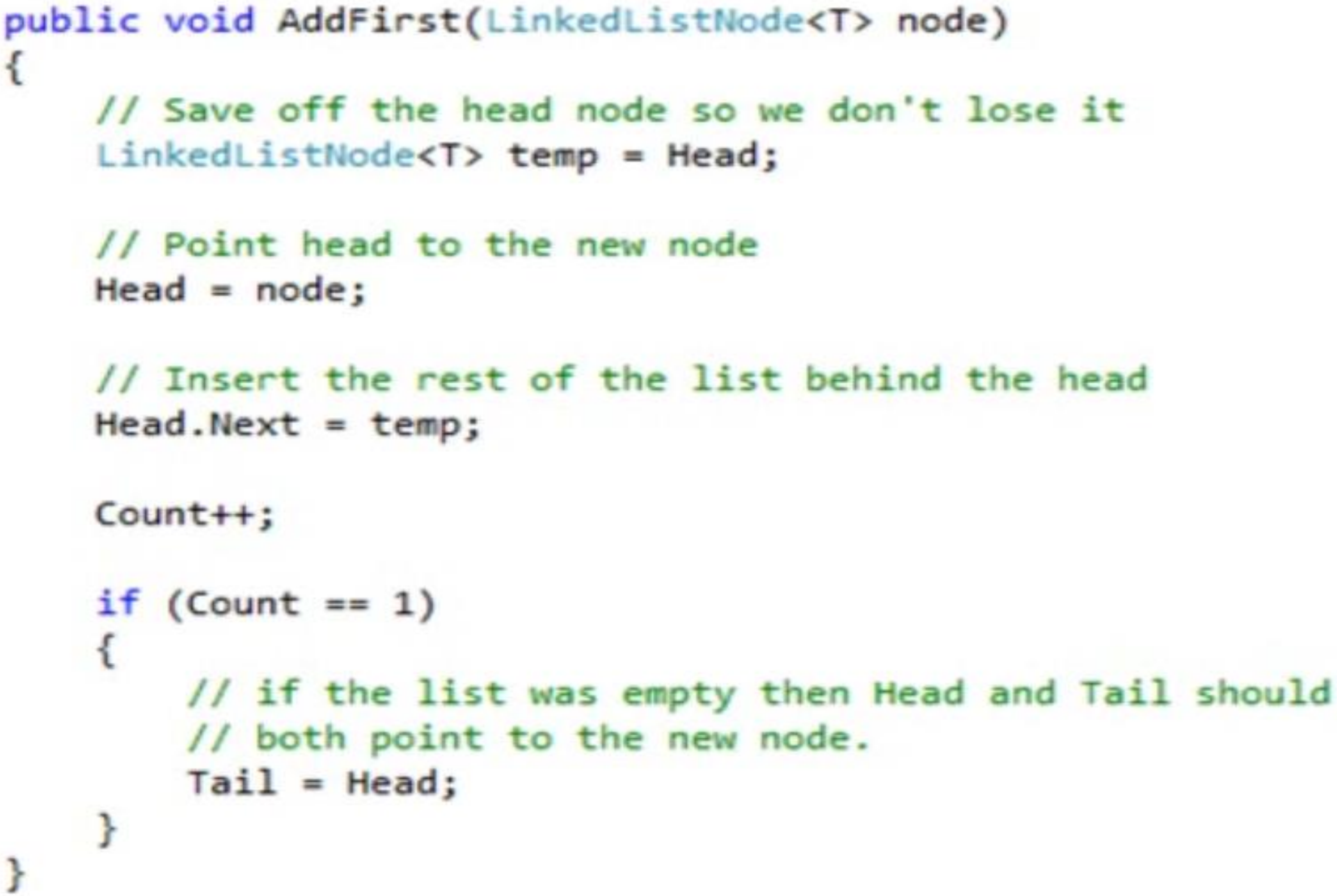

Linked List
Head
Null
Tail
Null


Linked List
Head
Tail
3
Null


Linked List
Head
Tail
5
3
Null

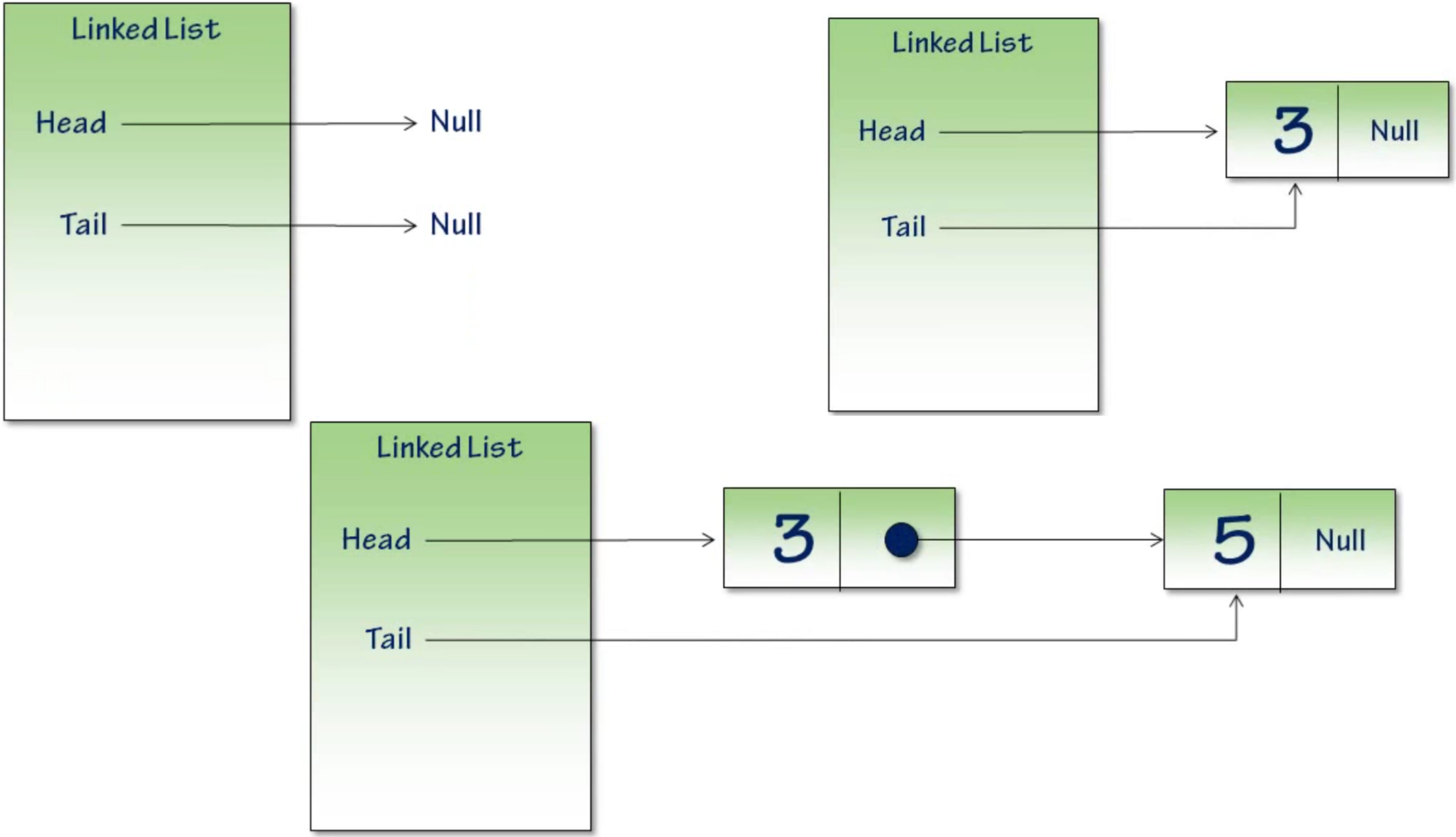
Add to Front





```
public void AddFirst(LinkedListNode<T> node)
{
    // Save off the head node so we don't lose it
    LinkedListNode<T> temp = Head;

    // Point head to the new node
    Head = node;

    // Insert the rest of the list behind the head
    Head.Next = temp;

    Count++;

    if (Count == 1)
    {
        // if the list was empty then Head and Tail should
        // both point to the new node.
        Tail = Head;
    }
}
```

Add to End



Linked List

Head → Null

Tail → Null

Linked List

Head → 3 | Null

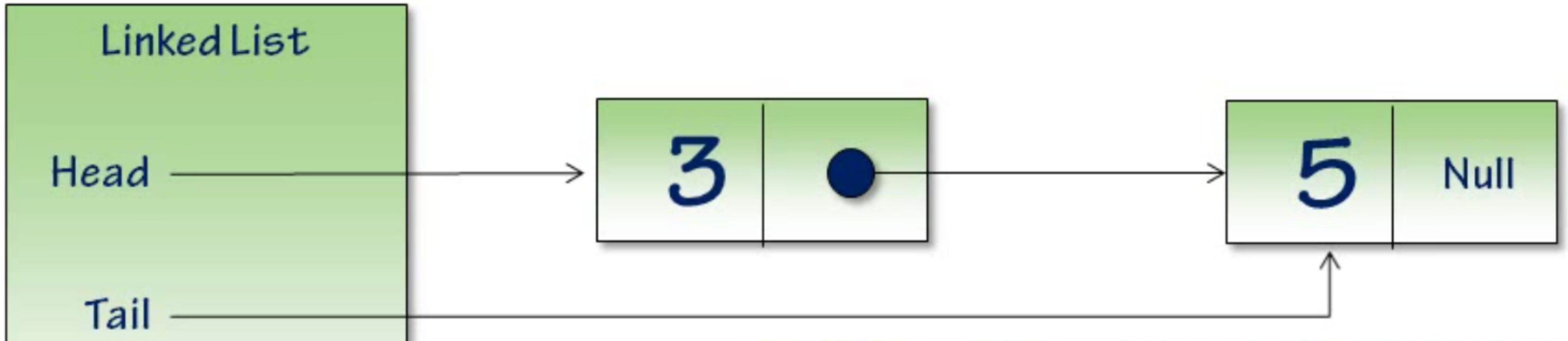
Tail → 3

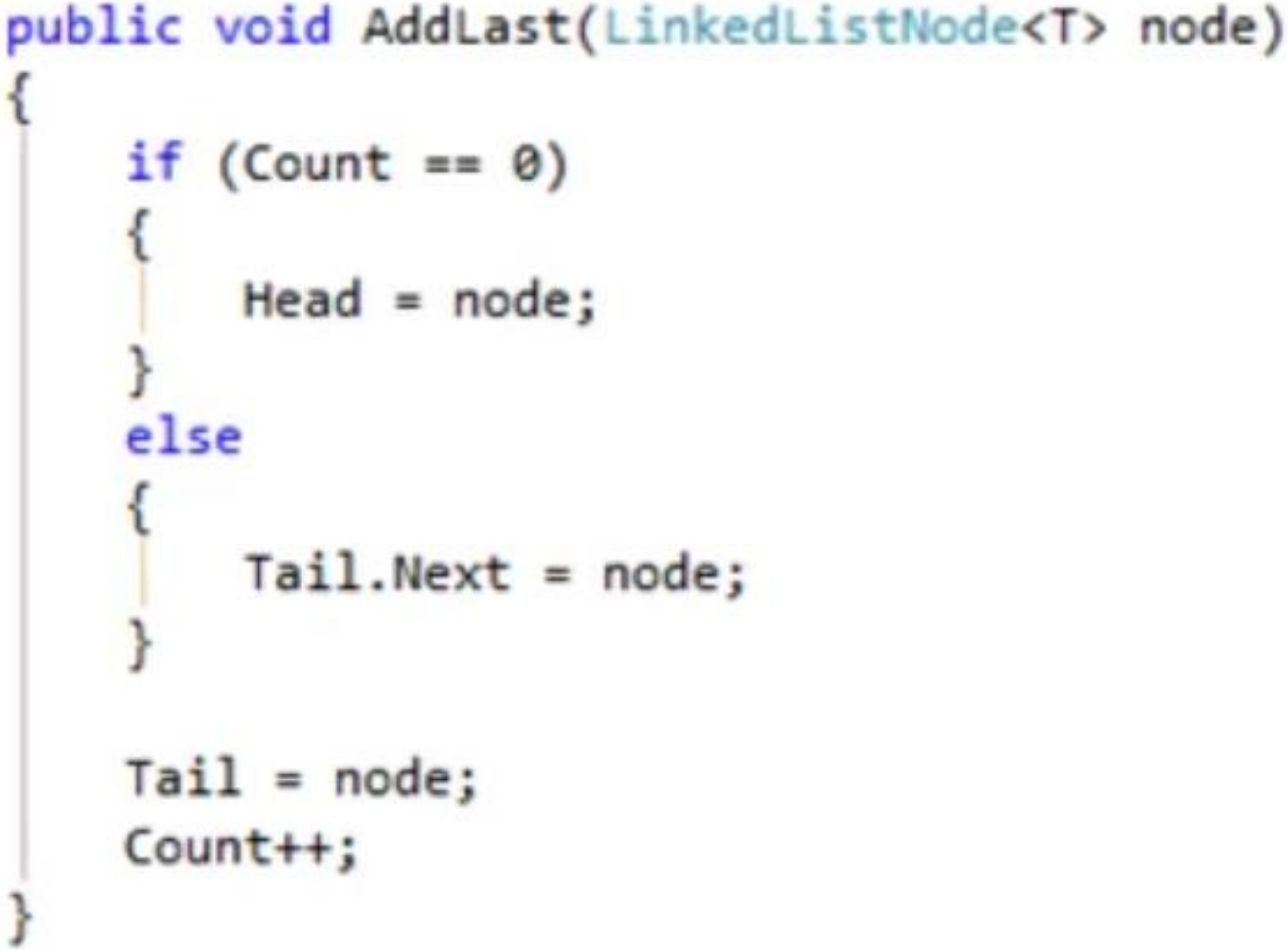
Linked List

Head → 3 | ● → 5 | Null

Tail → 5

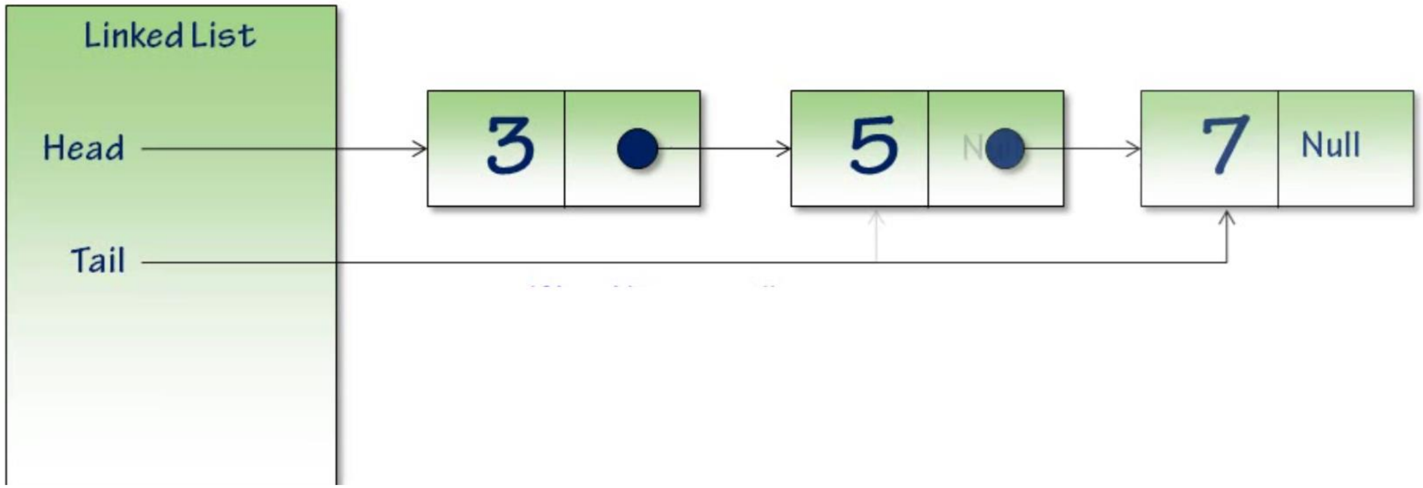
Add to End

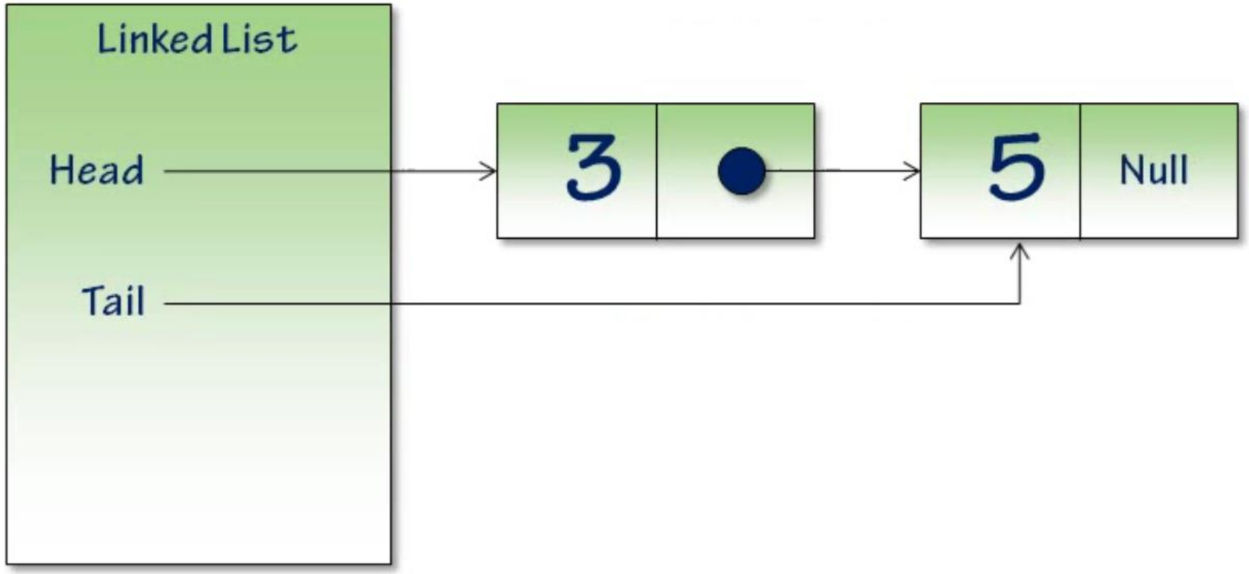


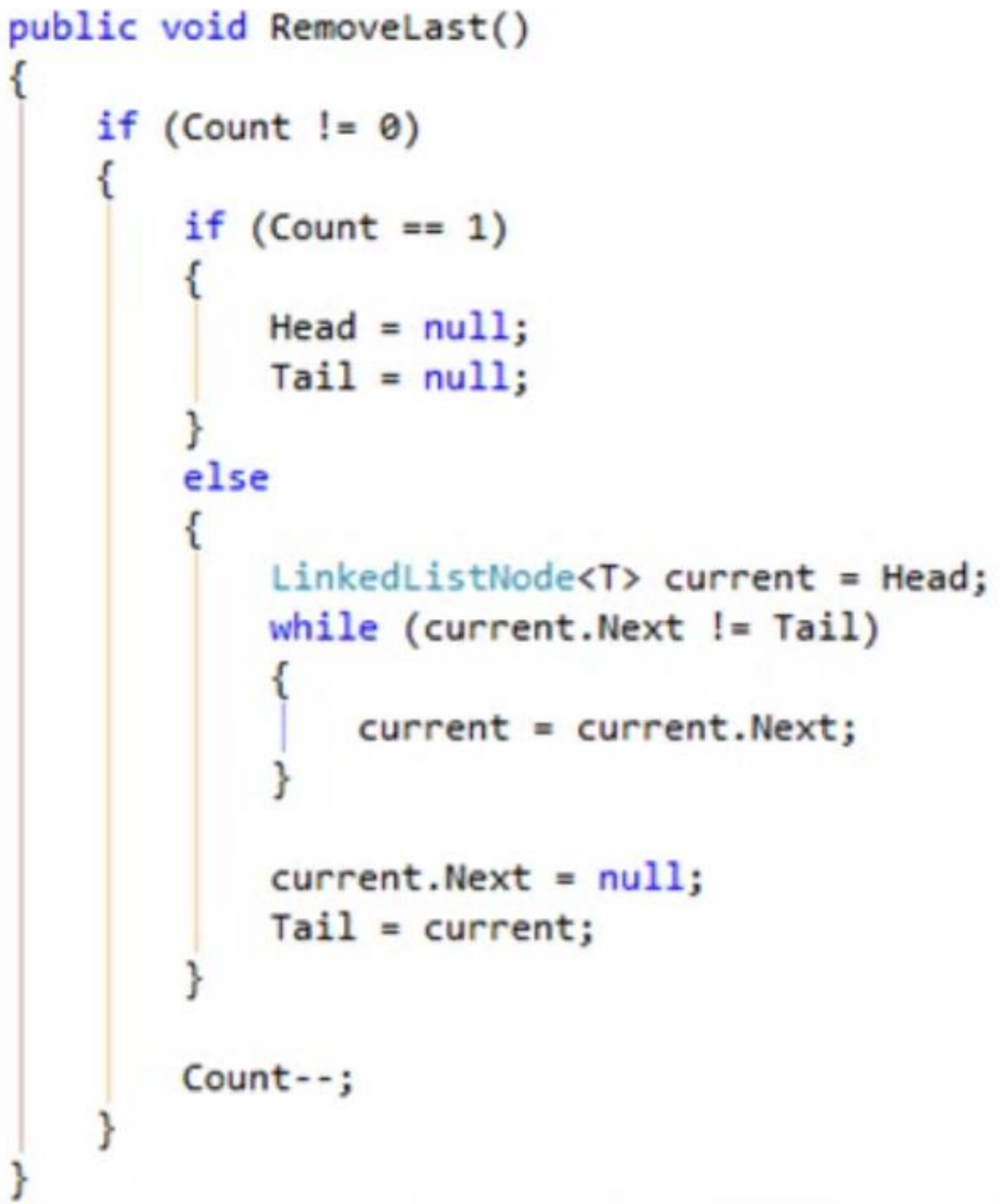


```
public void AddLast(LinkedListNode<T> node)
{
    if (Count == 0)
    {
        Head = node;
    }
    else
    {
        Tail.Next = node;
    }

    Tail = node;
    Count++;
}
```

Remove Last Node

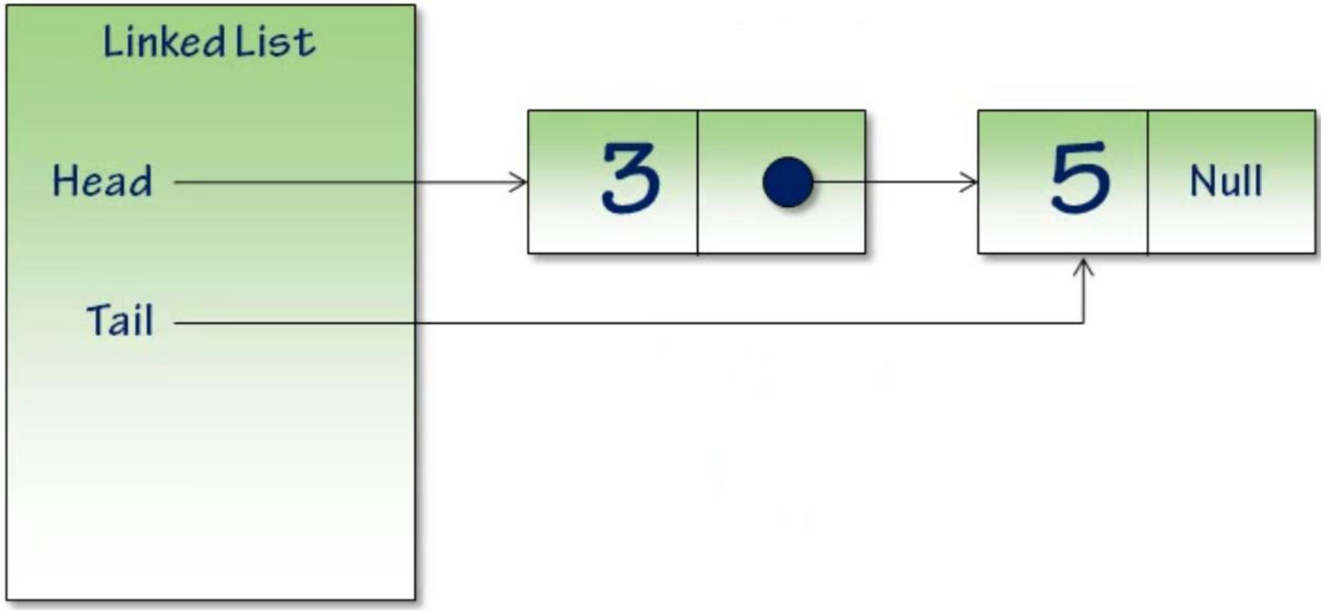


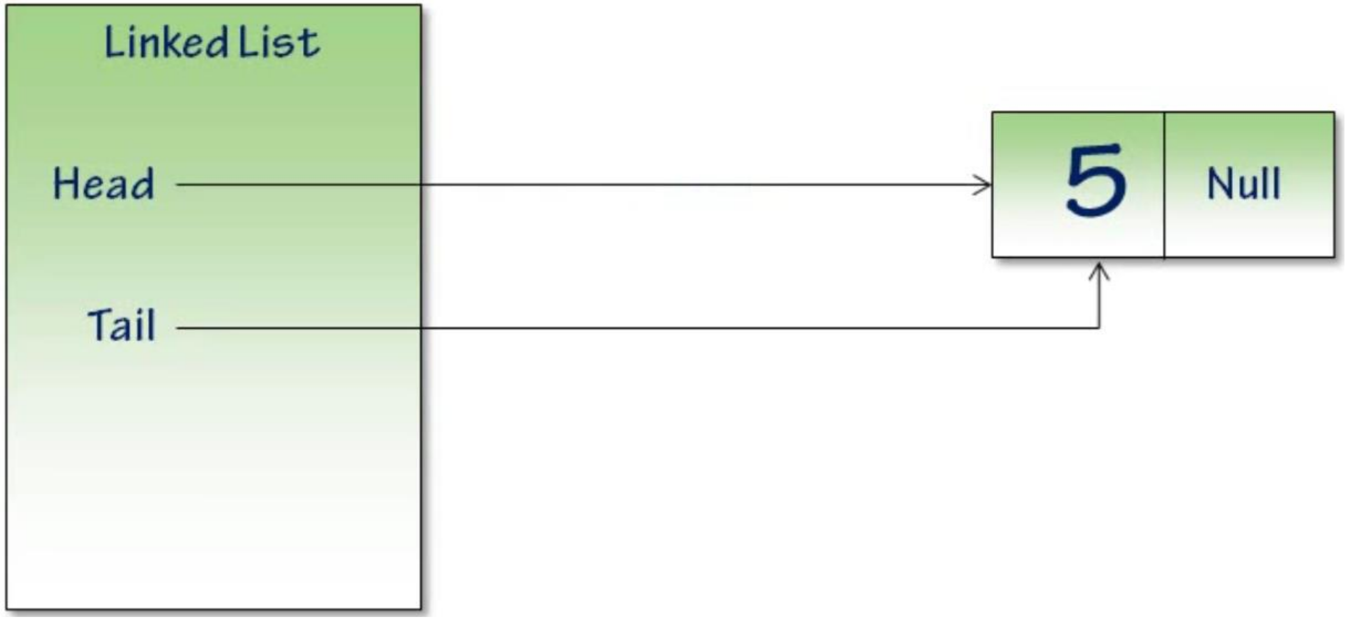


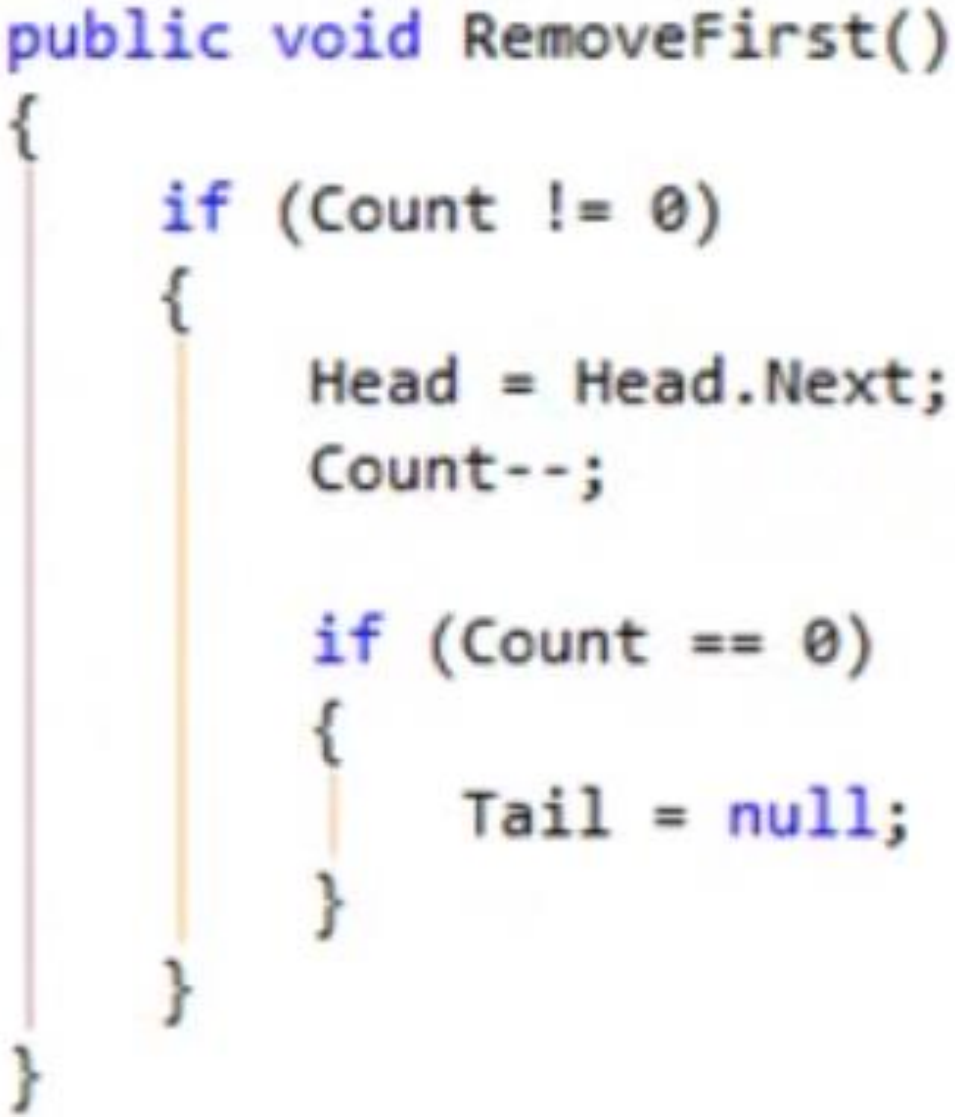


```
public void RemoveLast()
{
    if (Count != 0)
    {
        if (Count == 1)
        {
            Head = null;
            Tail = null;
        }
        else
        {
            LinkedListNode<T> current = Head;
            while (current.Next != Tail)
            {
                current = current.Next;
            }

            current.Next = null;
            Tail = current;
        }

        Count--;
    }
}
```

Remove First Node

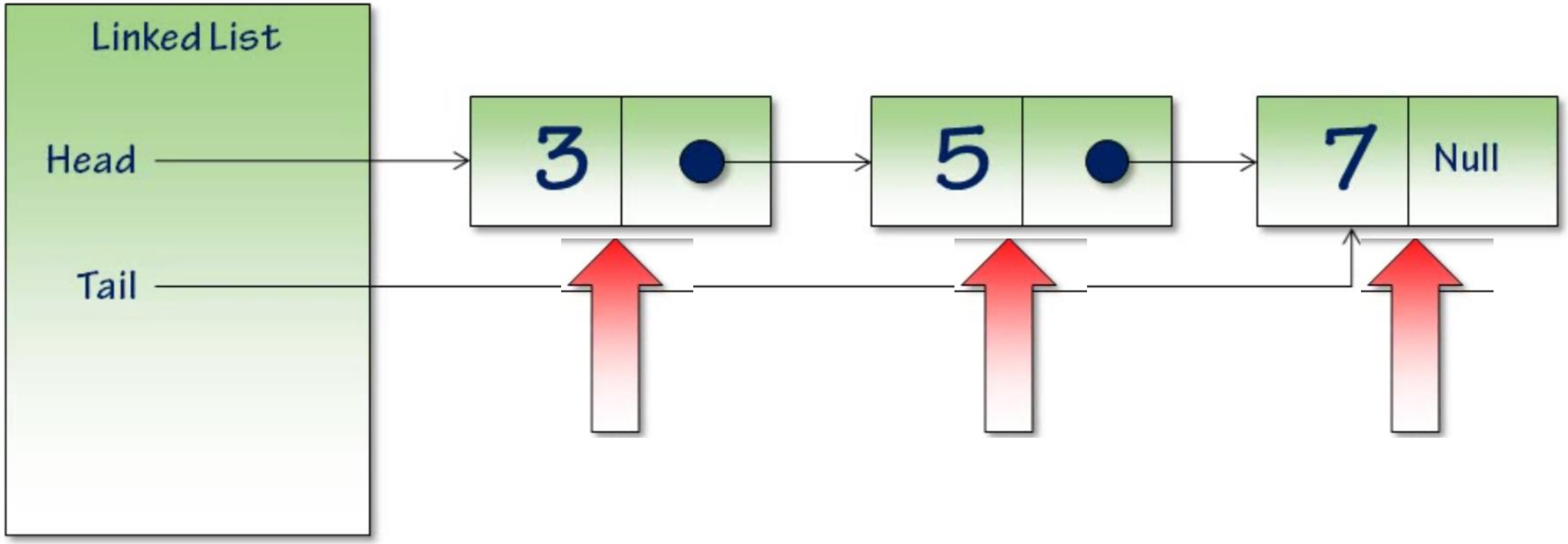


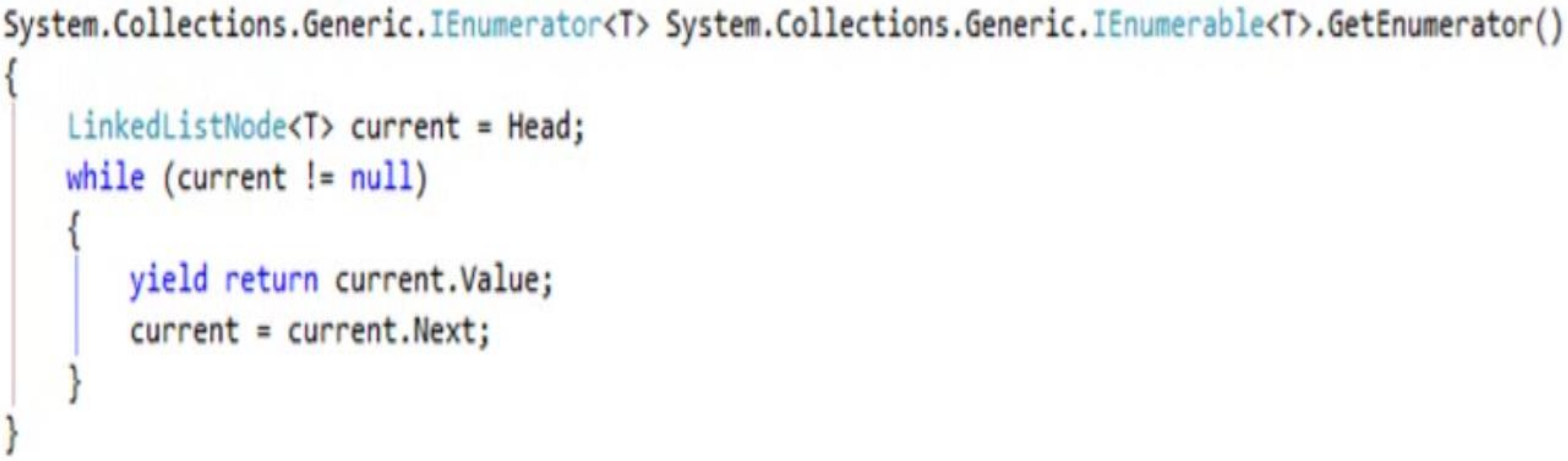




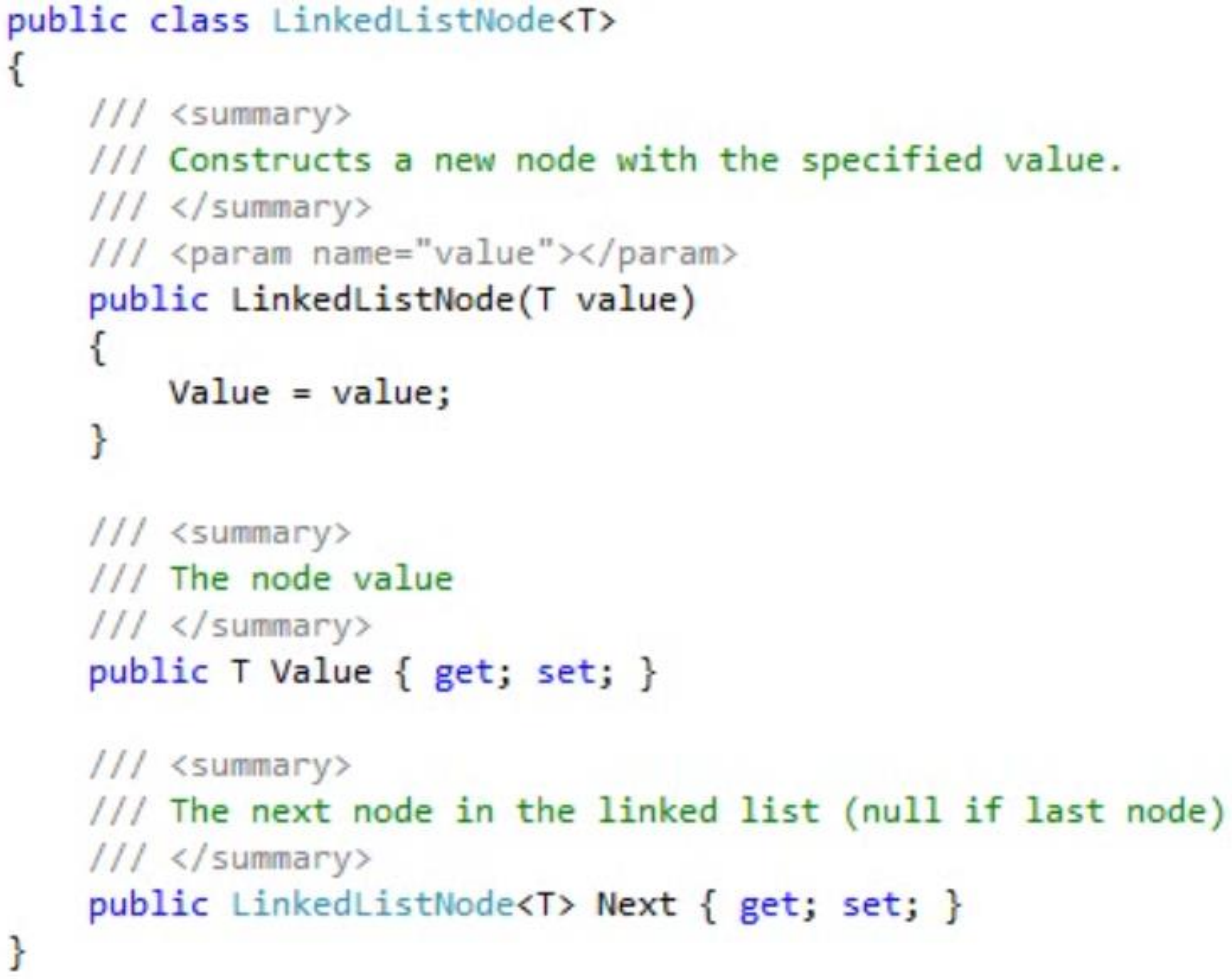
```
public void RemoveFirst()
{
    if (Count != 0)
    {
        Head = Head.Next;
        Count--;

        if (Count == 0)
        {
            Tail = null;
        }
    }
}
```

Enumerate

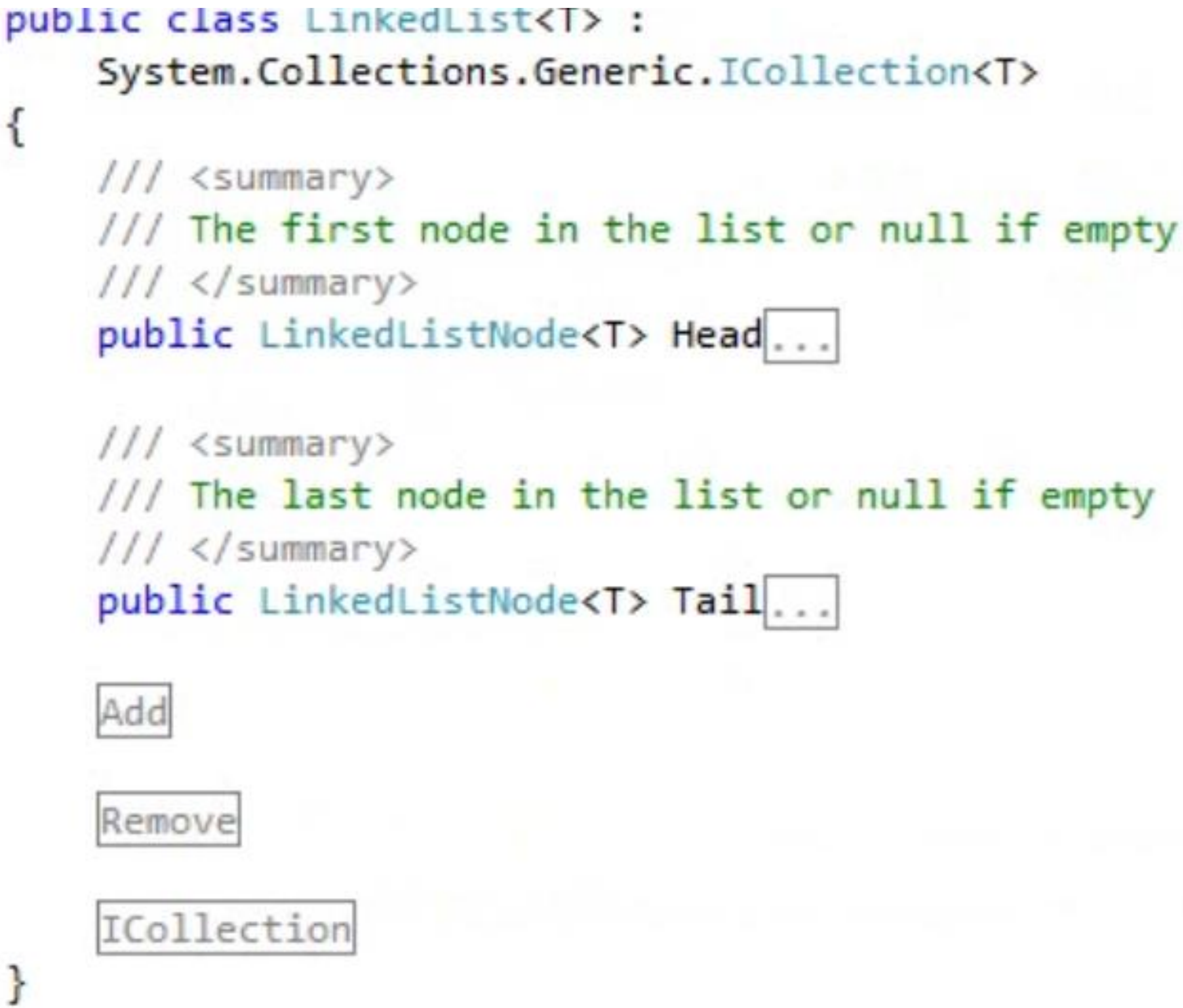




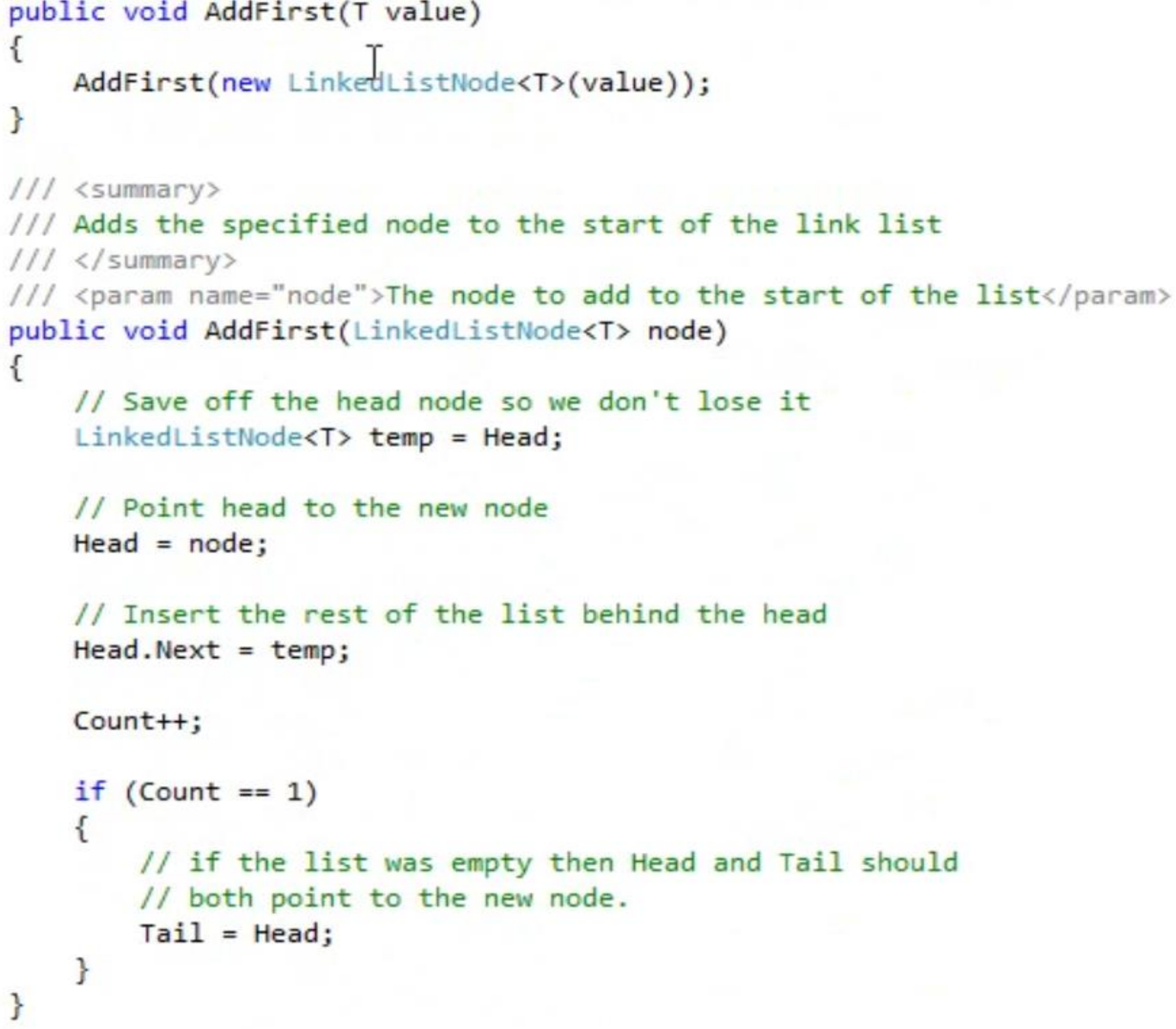
```
System.Collections.Generic.IEnumerator<T> System.Collections.Generic.IEnumerable<T>.GetEnumerator()
{
    LinkedListNode<T> current = Head;
    while (current != null)
    {
        yield return current.Value;
        current = current.Next;
    }
}
```



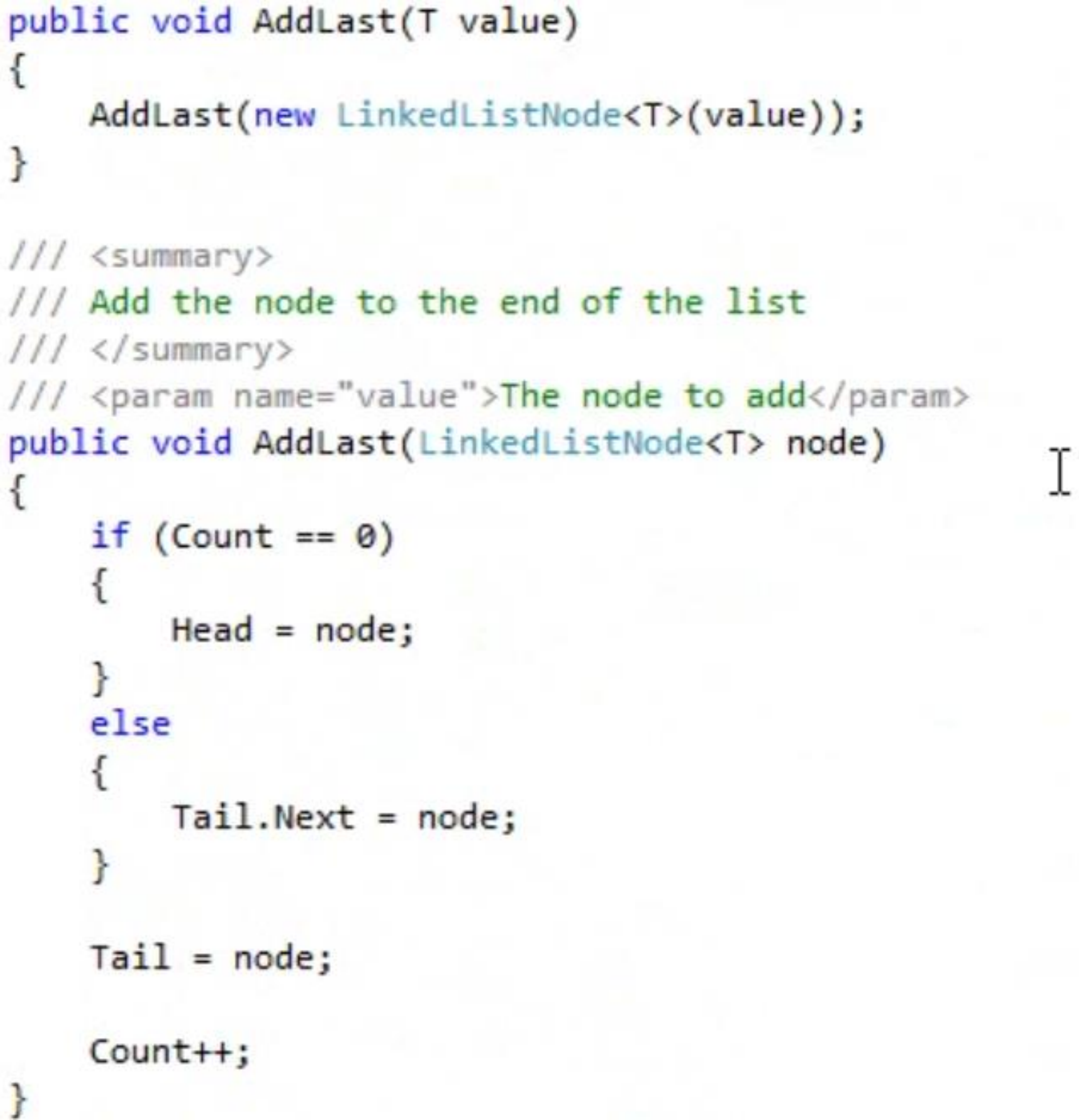
```
public class LinkedListNode<T>
{
    /// <summary>
    /// Constructs a new node with the specified value.
    /// </summary>
    /// <param name="value"></param>
    public LinkedListNode(T value)
    {
        Value = value;
    }

    /// <summary>
    /// The node value
    /// </summary>
    public T Value { get; set; }

    /// <summary>
    /// The next node in the linked list (null if last node)
    /// </summary>
    public LinkedListNode<T> Next { get; set; }
}
```



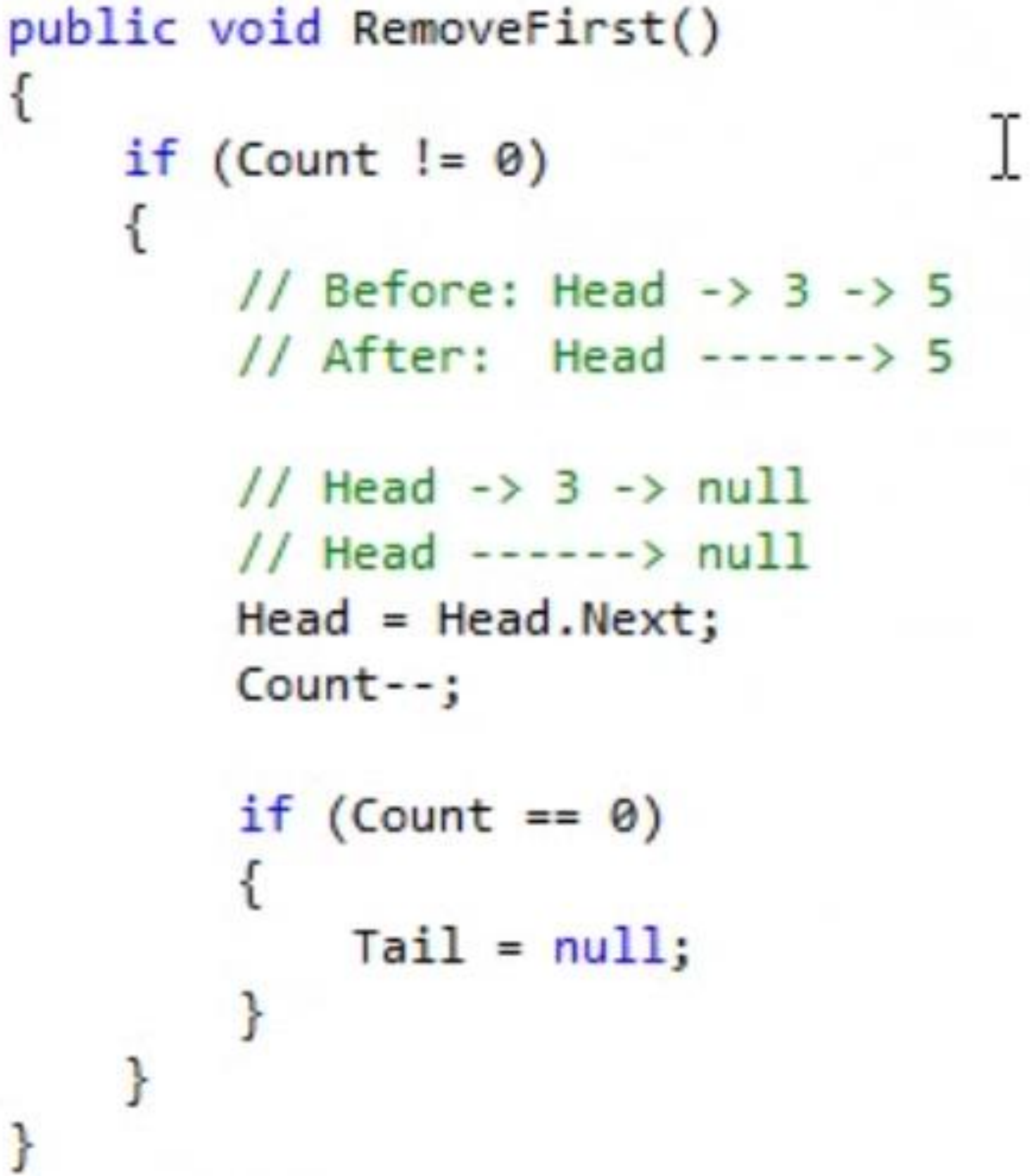
```
public class LinkedList<T> :
    System.Collections.Generic.ICollection<T>
{
    /// <summary>
    /// The first node in the list or null if empty
    /// </summary>
    public LinkedListNode<T> Head...

    /// <summary>
    /// The last node in the list or null if empty
    /// </summary>
    public LinkedListNode<T> Tail...

    Add

    Remove

    ICollection
}
```



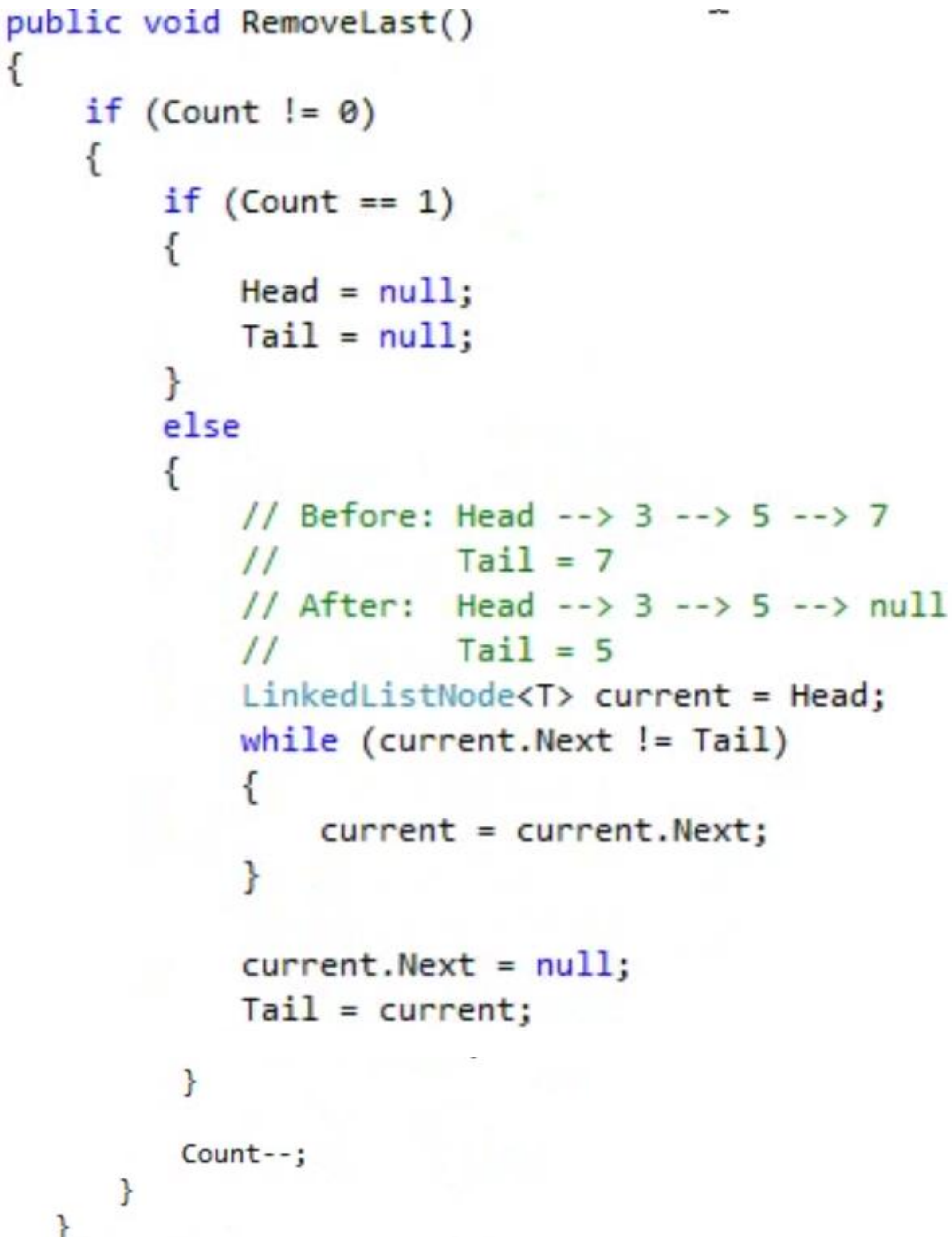
```
public void AddFirst(T value)
{
    AddFirst(new LinkedListNode<T>(value));
}

/// <summary>
/// Adds the specified node to the start of the link list
/// </summary>
/// <param name="node">The node to add to the start of the list</param>
public void AddFirst(LinkedListNode<T> node)
{
    // Save off the head node so we don't lose it
    LinkedListNode<T> temp = Head;

    // Point head to the new node
    Head = node;

    // Insert the rest of the list behind the head
    Head.Next = temp;

    Count++;

    if (Count == 1)
    {
        // if the list was empty then Head and Tail should
        // both point to the new node.
        Tail = Head;
    }
}
```



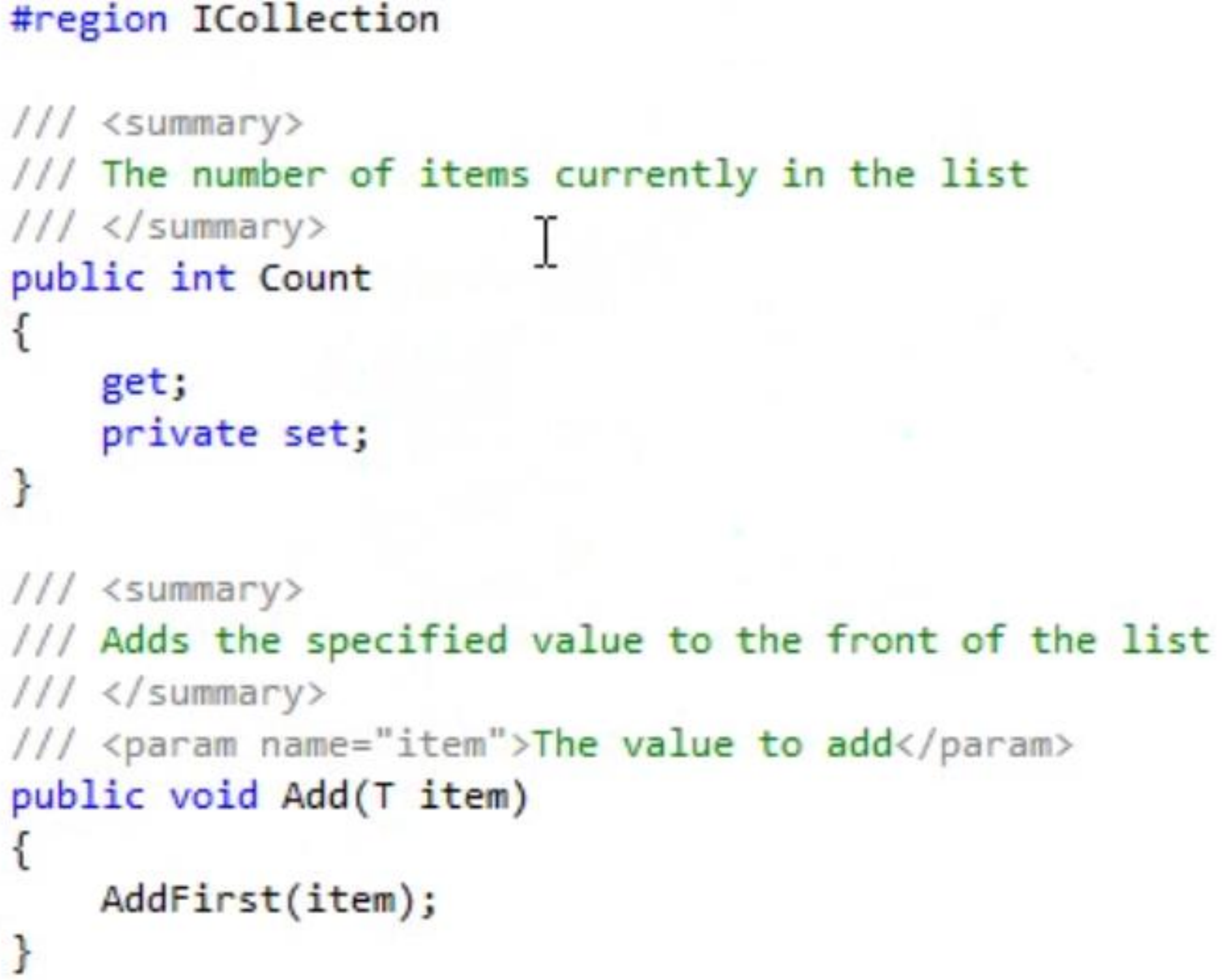
```
public void AddLast(T value)
{
    AddLast(new LinkedListNode<T>(value));
}

/// <summary>
/// Add the node to the end of the list
/// </summary>
/// <param name="value">The node to add</param>
public void AddLast(LinkedListNode<T> node)
{
    if (Count == 0)
    {
        Head = node;
    }
    else
    {
        Tail.Next = node;
    }

    Tail = node;

    Count++;
}
```



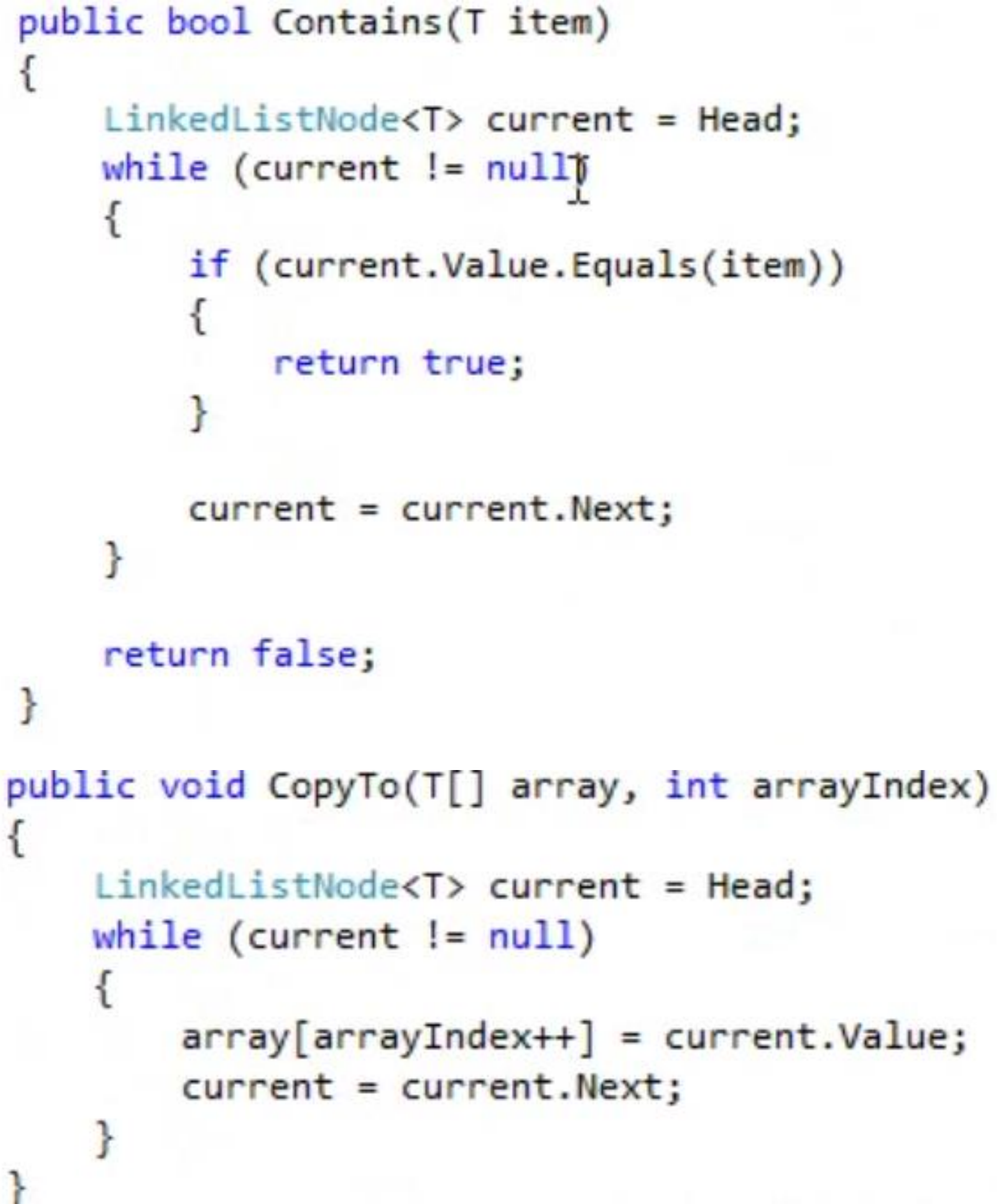
```
public void RemoveFirst()
{
    if (Count != 0)
    {
        // Before: Head -> 3 -> 5
        // After:  Head ------> 5

        // Head -> 3 -> null
        // Head ------> null
        Head = Head.Next;
        Count--;

        if (Count == 0)
        {
            Tail = null;
        }
    }
}
```



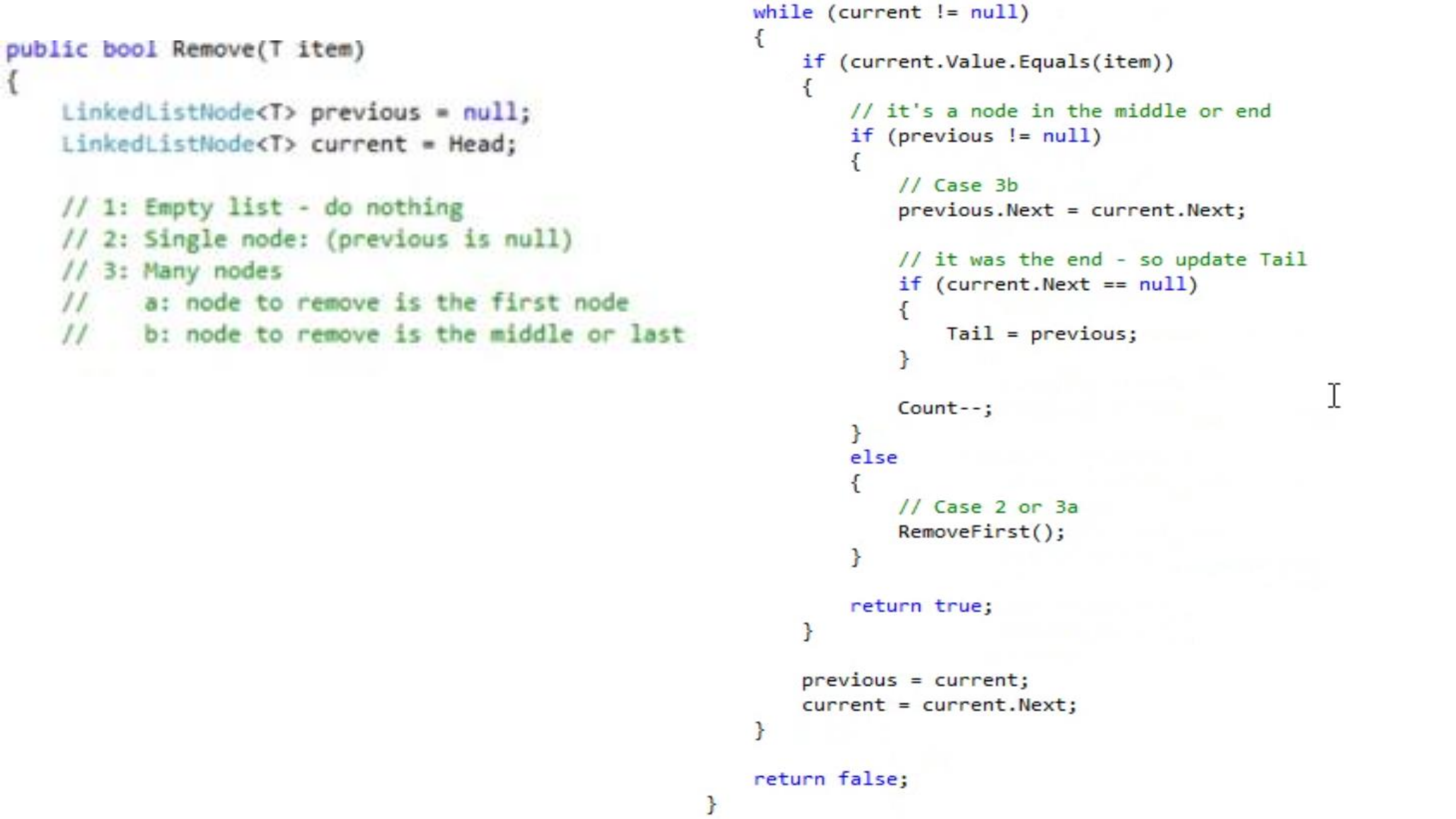
```
public void RemoveLast()
{
    if (Count != 0)
    {
        if (Count == 1)
        {
            Head = null;
            Tail = null;
        }
        else
        {
            // Before: Head --> 3 --> 5 --> 7
            //         Tail = 7
            // After:  Head --> 3 --> 5 --> null
            //         Tail = 5
            LinkedListNode<T> current = Head;
            while (current.Next != Tail)
            {
                current = current.Next;
            }

            current.Next = null;
            Tail = current;

         }

         Count--;
      }
   }
```



```
#region ICollection

/// <summary>
/// The number of items currently in the list
/// </summary>
public int Count
{
    get;
    private set;
}

/// <summary>
/// Adds the specified value to the front of the list
/// </summary>
/// <param name="item">The value to add</param>
public void Add(T item)
{
    AddFirst(item);
}
```

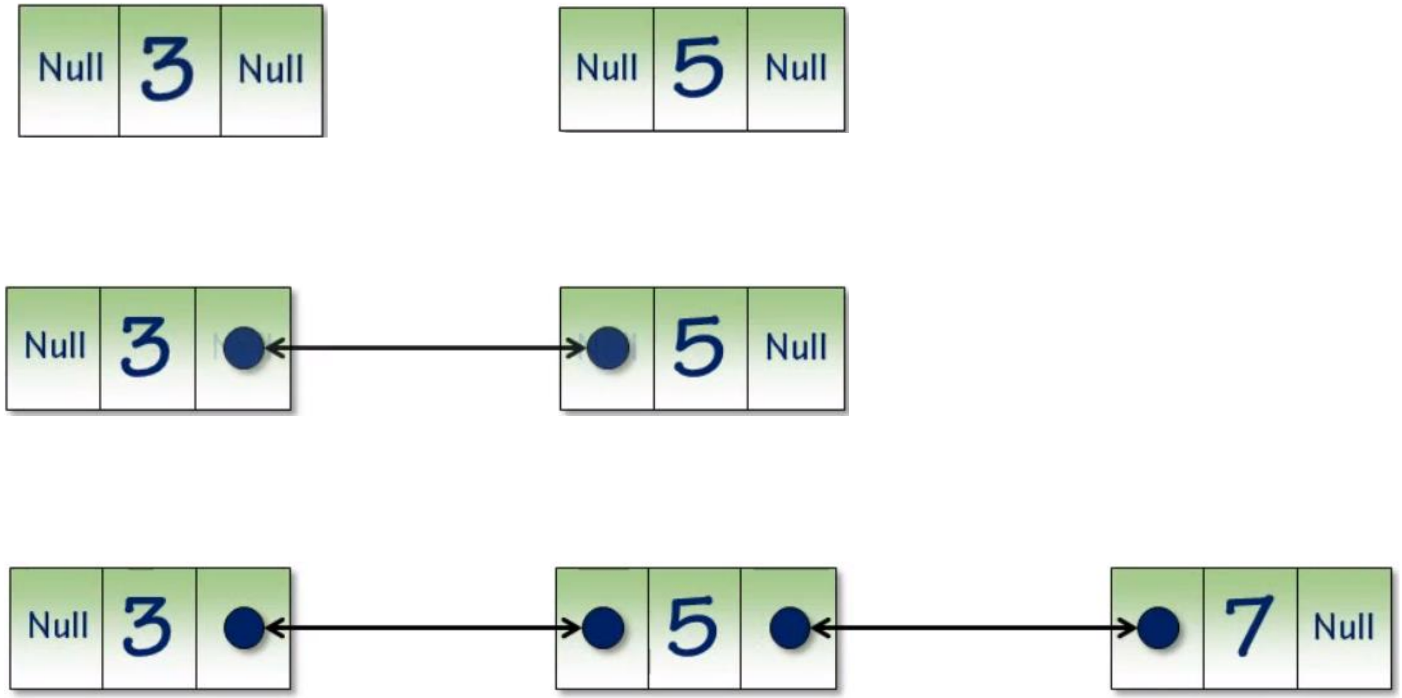


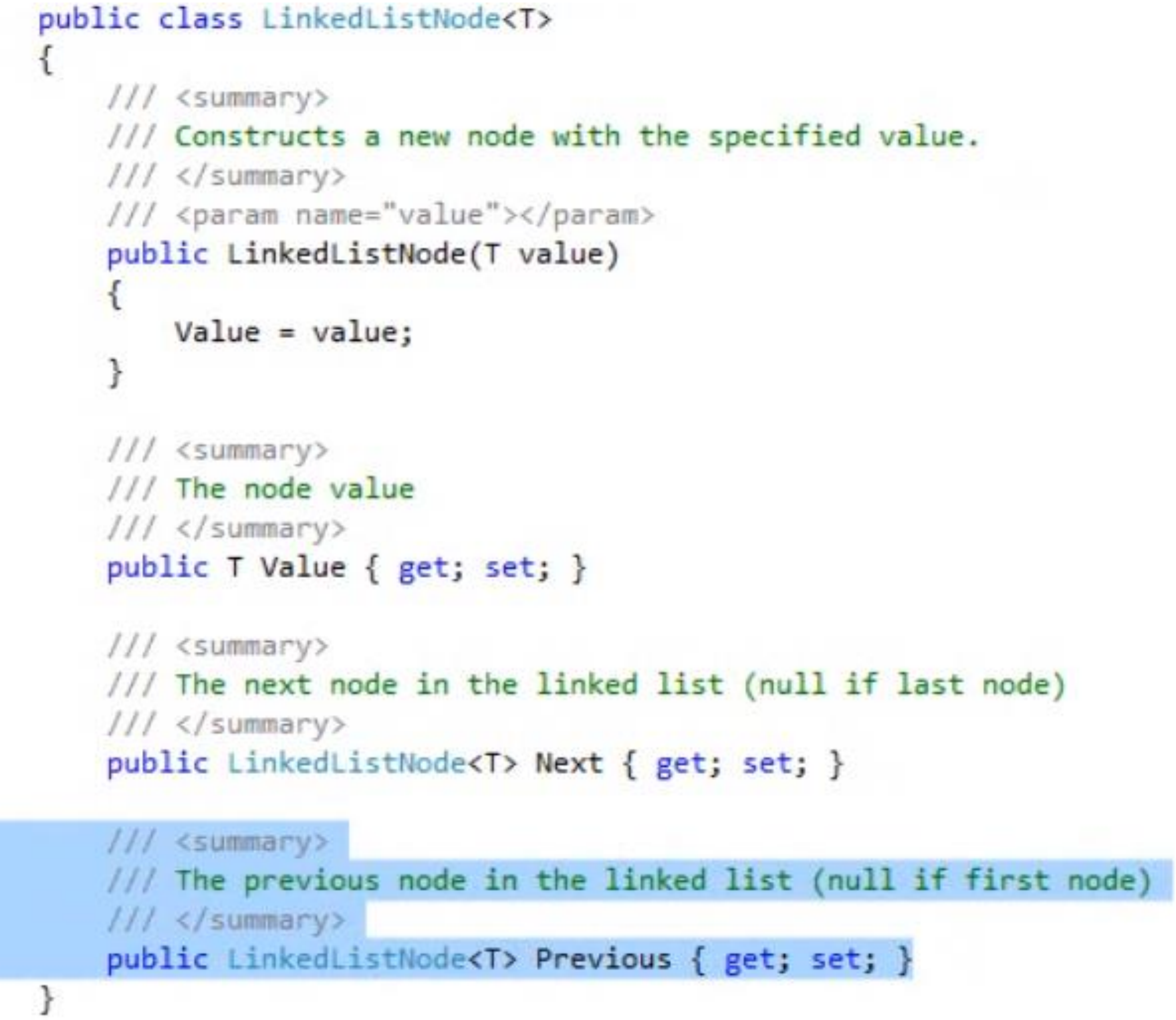
```
public bool Contains(T item)
{
    LinkedListNode<T> current = Head;
    while (current != null)
    {
        if (current.Value.Equals(item))
        {
            return true;
        }

        current = current.Next;
    }

    return false;
}

public void CopyTo(T[] array, int arrayIndex)
{
    LinkedListNode<T> current = Head;
    while (current != null)
    {
        array[arrayIndex++] = current.Value;
        current = current.Next;
    }
}
```



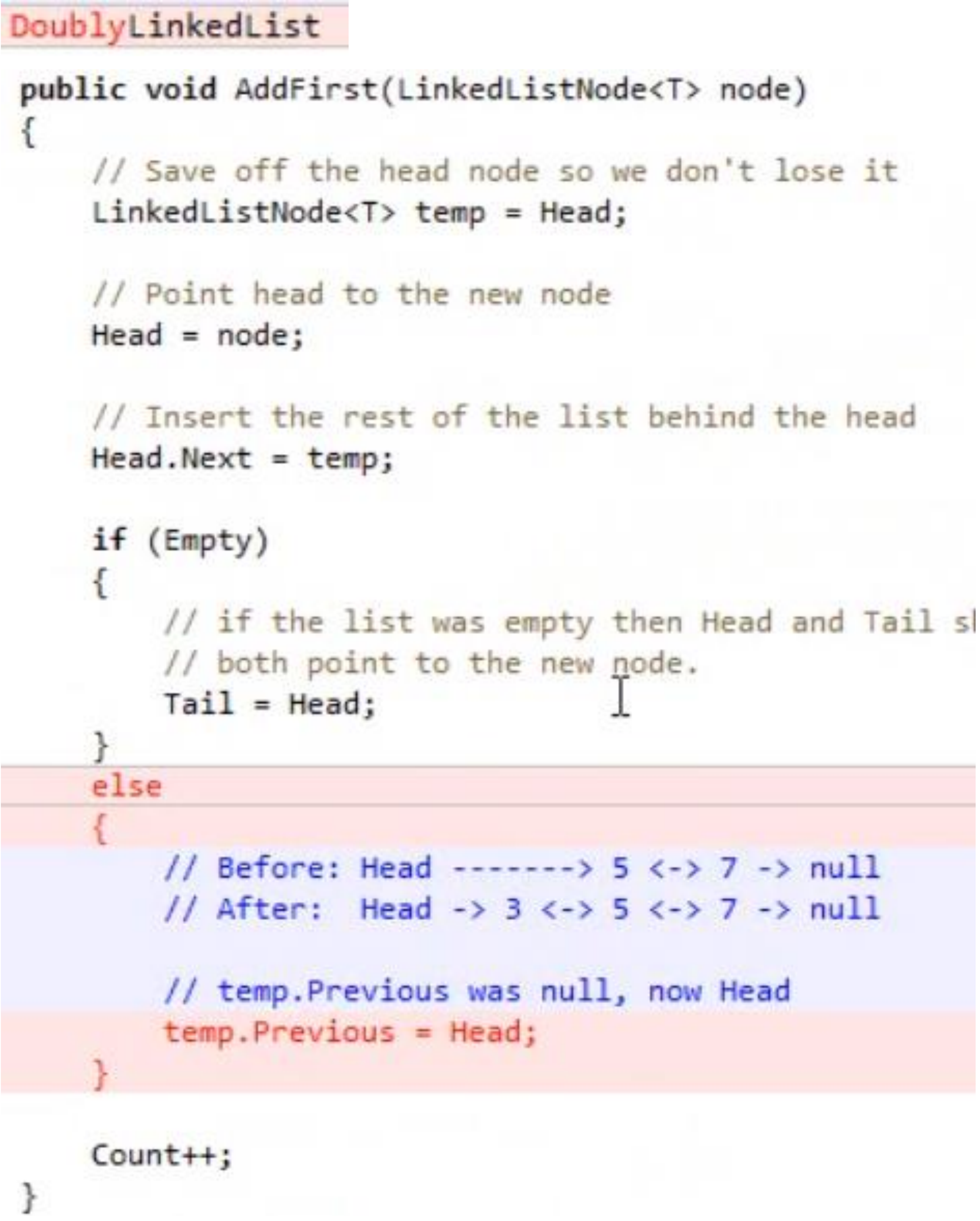
```
public bool Remove(T item)
{
    LinkedListNode<T> previous = null;
    LinkedListNode<T> current = Head;

    // 1: Empty list - do nothing
    // 2: Single node: (previous is null)
    // 3: Many nodes
    //    a: node to remove is the first node
    //    b: node to remove is the middle or last

    while (current != null)
    {
        if (current.Value.Equals(item))
        {
            // it's a node in the middle or end
            if (previous != null)
            {
                // Case 3b
                previous.Next = current.Next;

                // it was the end - so update Tail
                if (current.Next == null)
                {
                    Tail = previous;
                }

                Count--;
            }
            else
            {
                // Case 2 or 3a
                RemoveFirst();
            }

            return true;
        }

        previous = current;
        current = current.Next;
    }

    return false;
}
```

Doubly Linked List



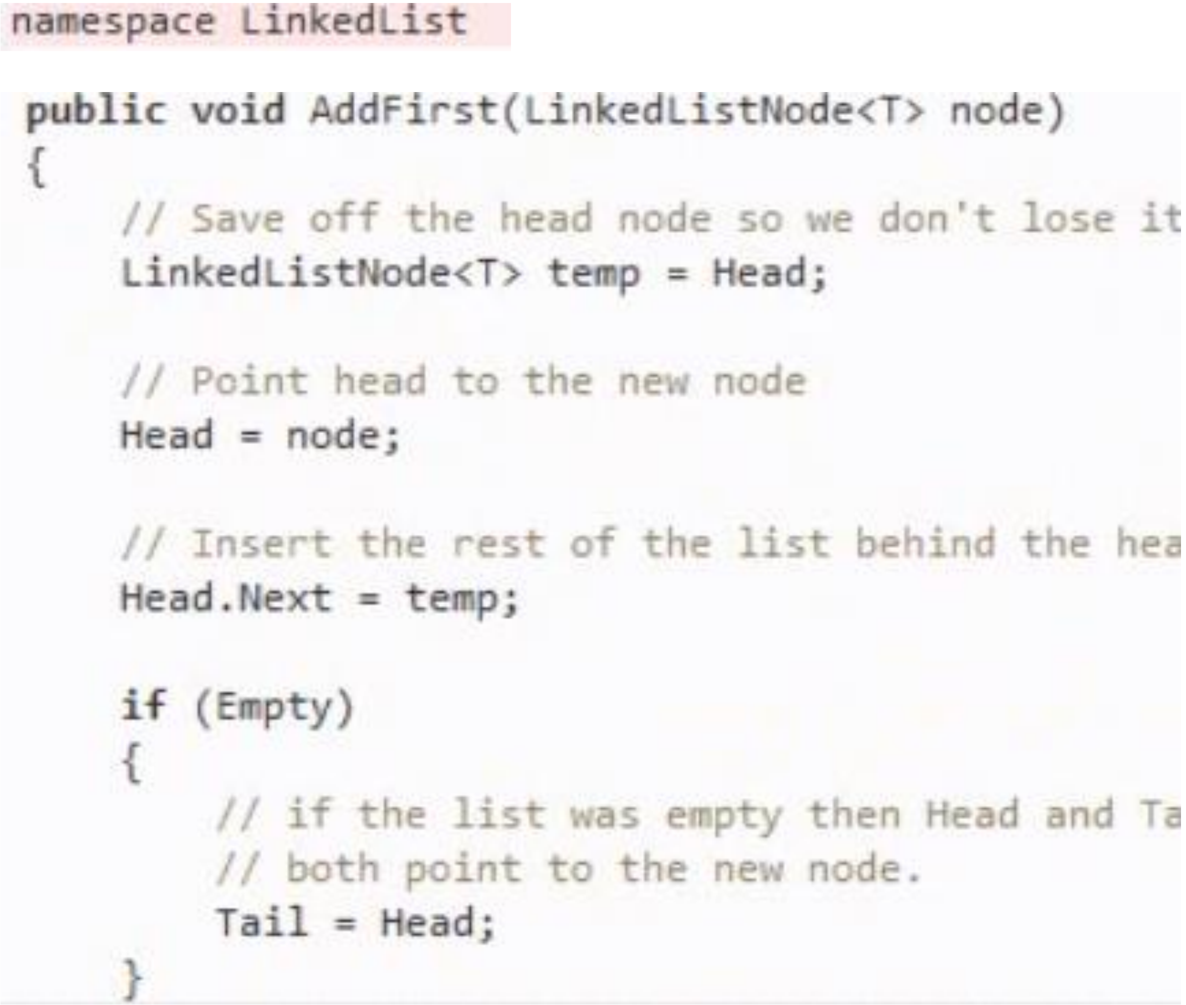


```
public class LinkedListNode<T>
{
    /// <summary>
    /// Constructs a new node with the specified value.
    /// </summary>
    /// <param name="value"></param>
    public LinkedListNode(T value)
    {
        Value = value;
    }

    /// <summary>
    /// The node value
    /// </summary>
    public T Value { get; set; }

    /// <summary>
    /// The next node in the linked list (null if last node)
    /// </summary>
    public LinkedListNode<T> Next { get; set; }

    /// <summary>
    /// The previous node in the linked list (null if first node)
    /// </summary>
    public LinkedListNode<T> Previous { get; set; }
}
```



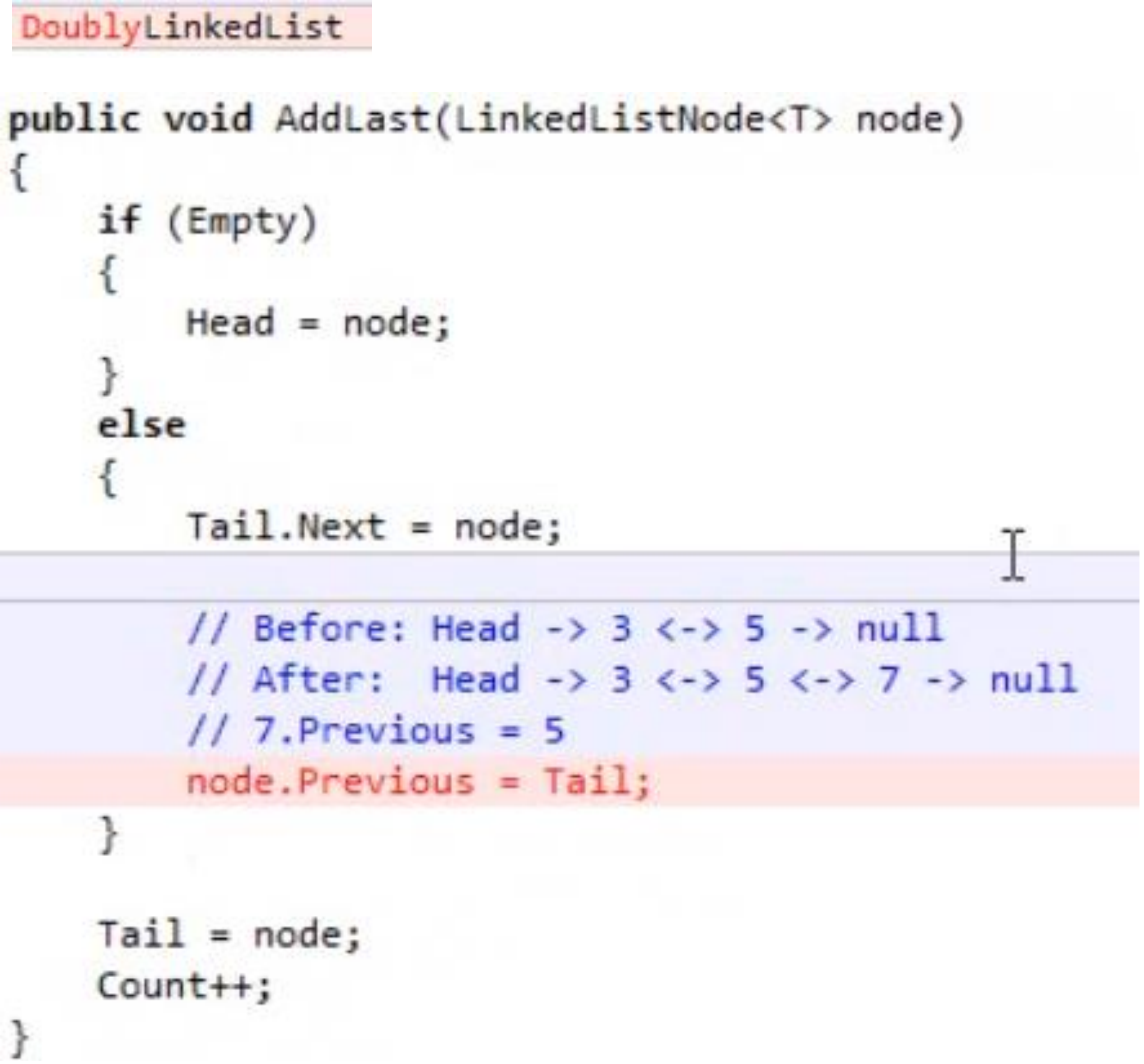
DoublyLinkedList

```
public void AddFirst(LinkedListNode<T> node)
{
    // Save off the head node so we don't lose it
    LinkedListNode<T> temp = Head;

    // Point head to the new node
    Head = node;

    // Insert the rest of the list behind the head
    Head.Next = temp;

    if (Empty)
    {
        // if the list was empty then Head and Tail s
        // both point to the new node.
        Tail = Head;
    }
    else
    {
        // Before: Head -------> 5 <-> 7 -> null
        // After:  Head -> 3 <-> 5 <-> 7 -> null

        // temp.Previous was null, now Head
        temp.Previous = Head;
    }

    Count++;
}
```



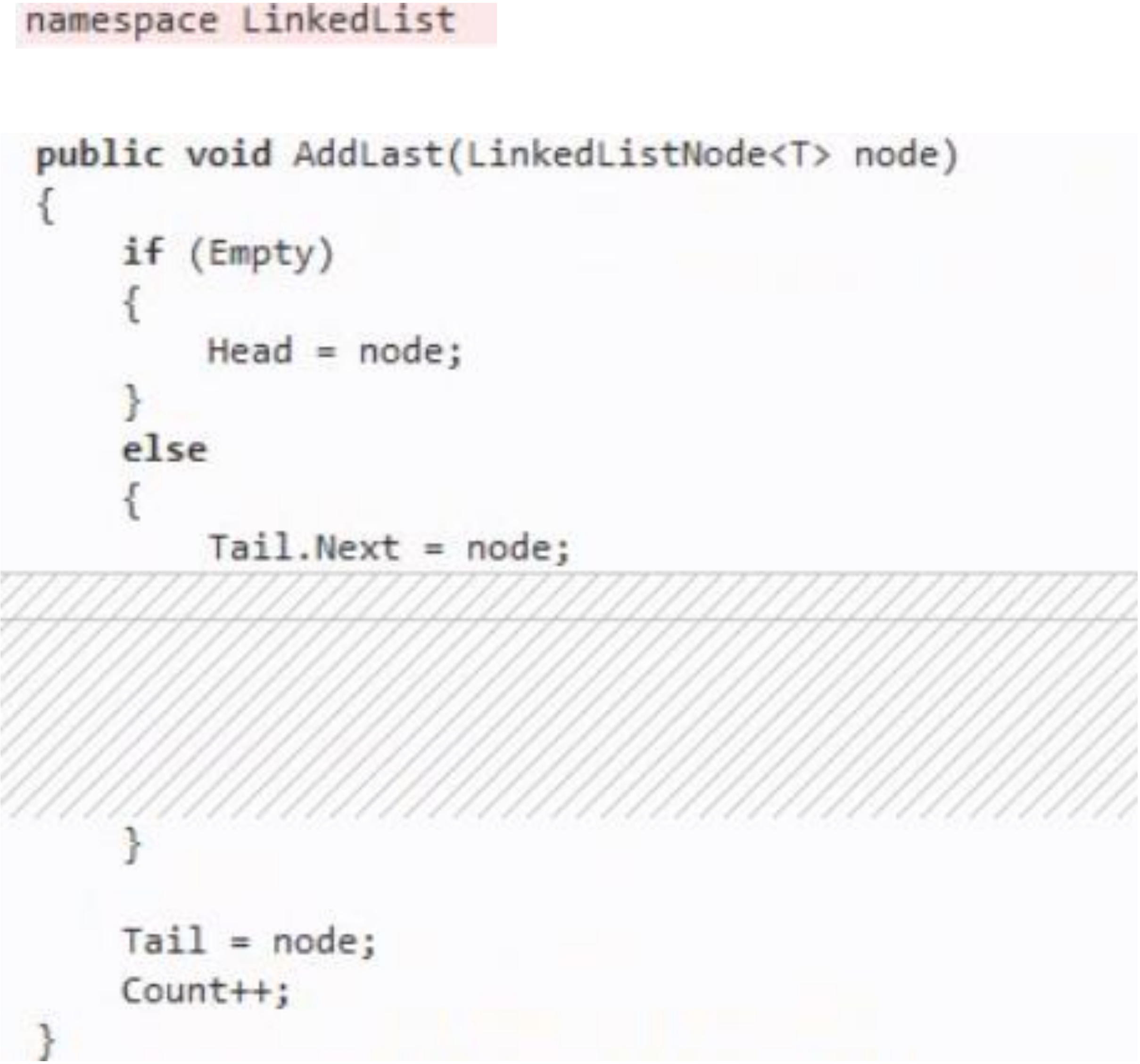
namespace LinkedList

```
public void AddFirst(LinkedListNode<T> node)
{
    // Save off the head node so we don't lose it
    LinkedListNode<T> temp = Head;

    // Point head to the new node
    Head = node;

    // Insert the rest of the list behind the hea
    Head.Next = temp;

    if (Empty)
    {
        // if the list was empty then Head and Ta
        // both point to the new node.
        Tail = Head;
    }
```



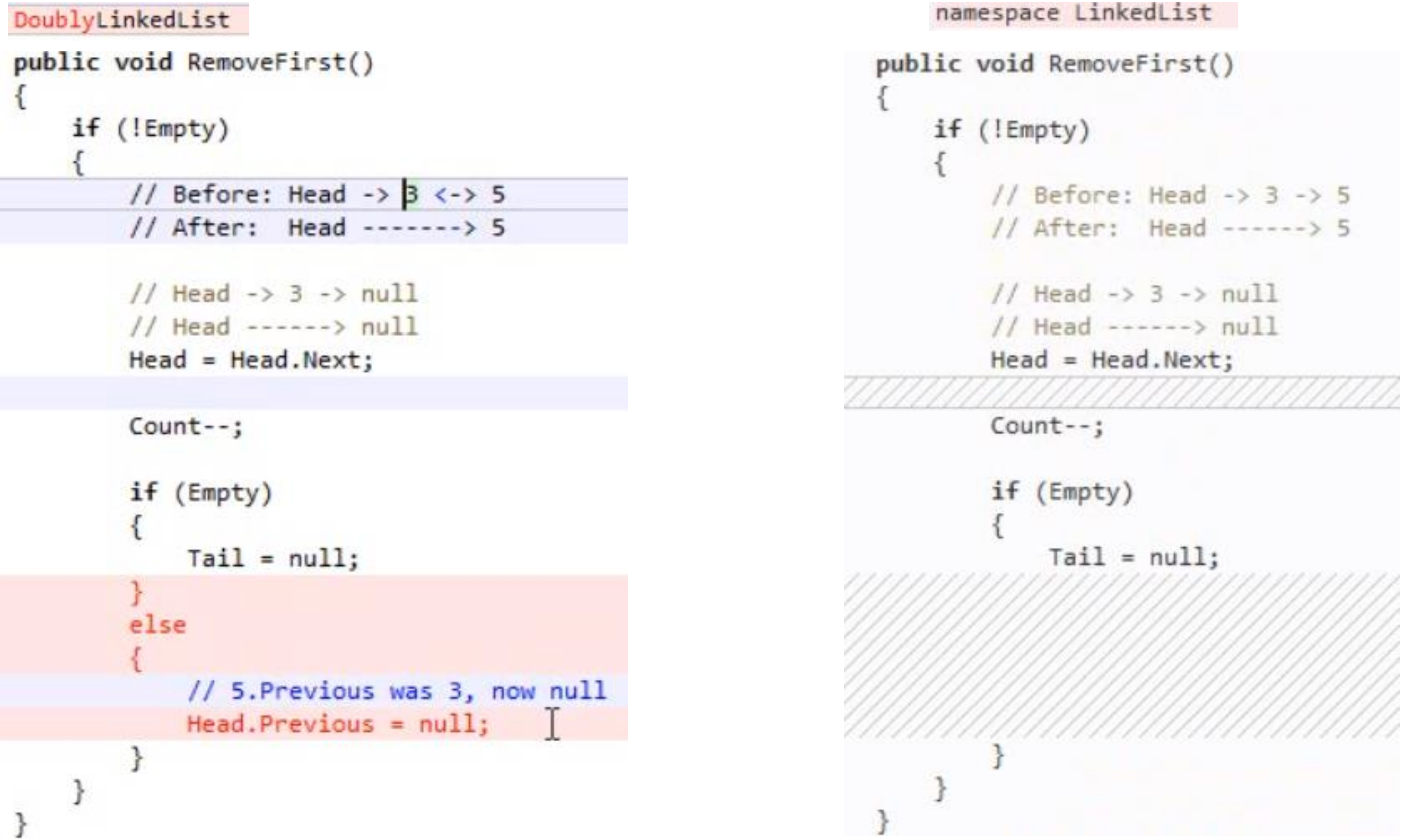
DoublyLinkedList

```
public void AddLast(LinkedListNode<T> node)
{
    if (Empty)
    {
        Head = node;
    }
    else
    {
        Tail.Next = node;

        // Before: Head -> 3 <-> 5 -> null
        // After:  Head -> 3 <-> 5 <-> 7 -> null
        // 7.Previous = 5
        node.Previous = Tail;
    }

    Tail = node;
    Count++;
}
```



namespace LinkedList

```
public void AddLast(LinkedListNode<T> node)
{
    if (Empty)
    {
        Head = node;
    }
    else
    {
        Tail.Next = node;
    }

    Tail = node;
    Count++;
}
```

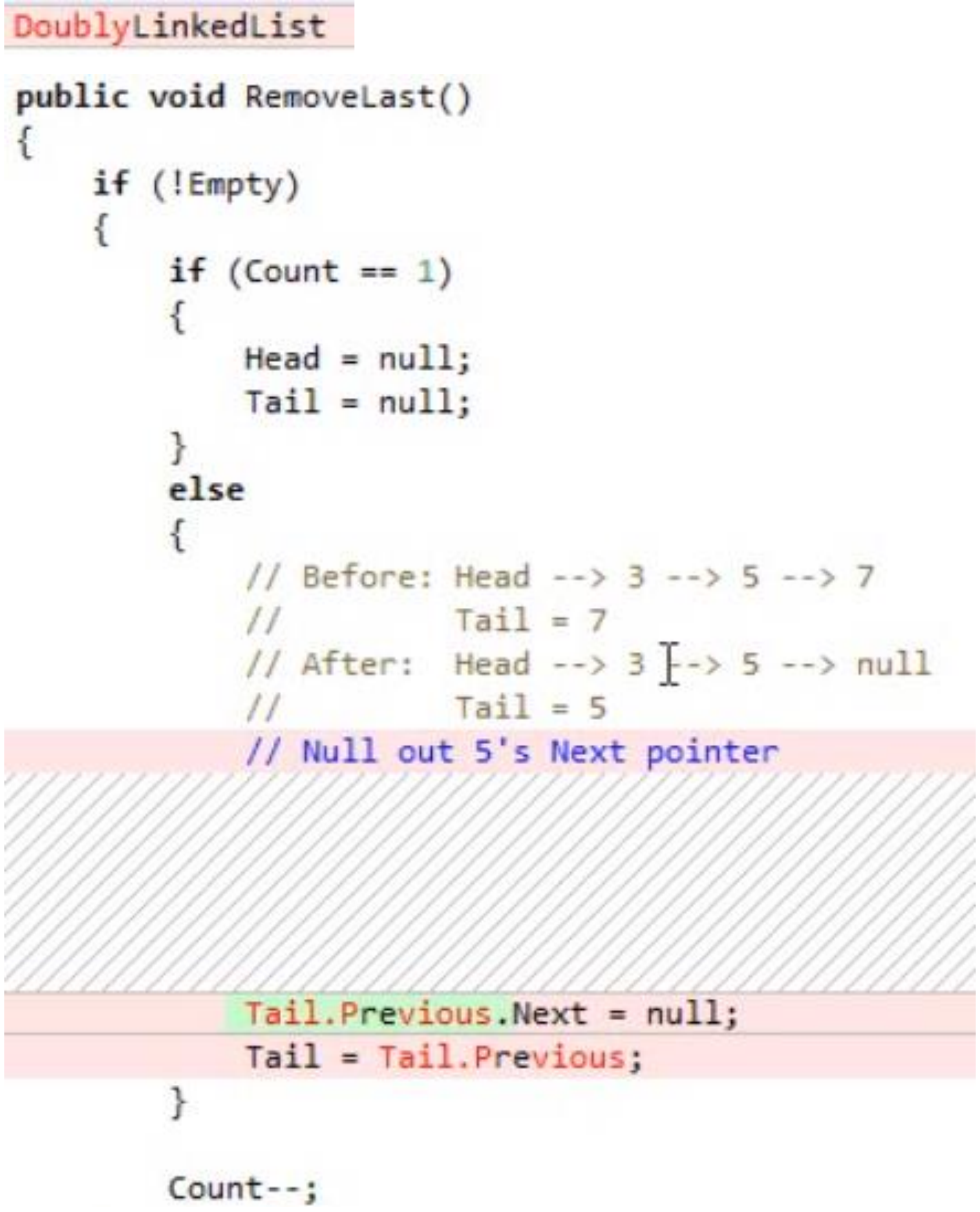


DoublyLinkedList

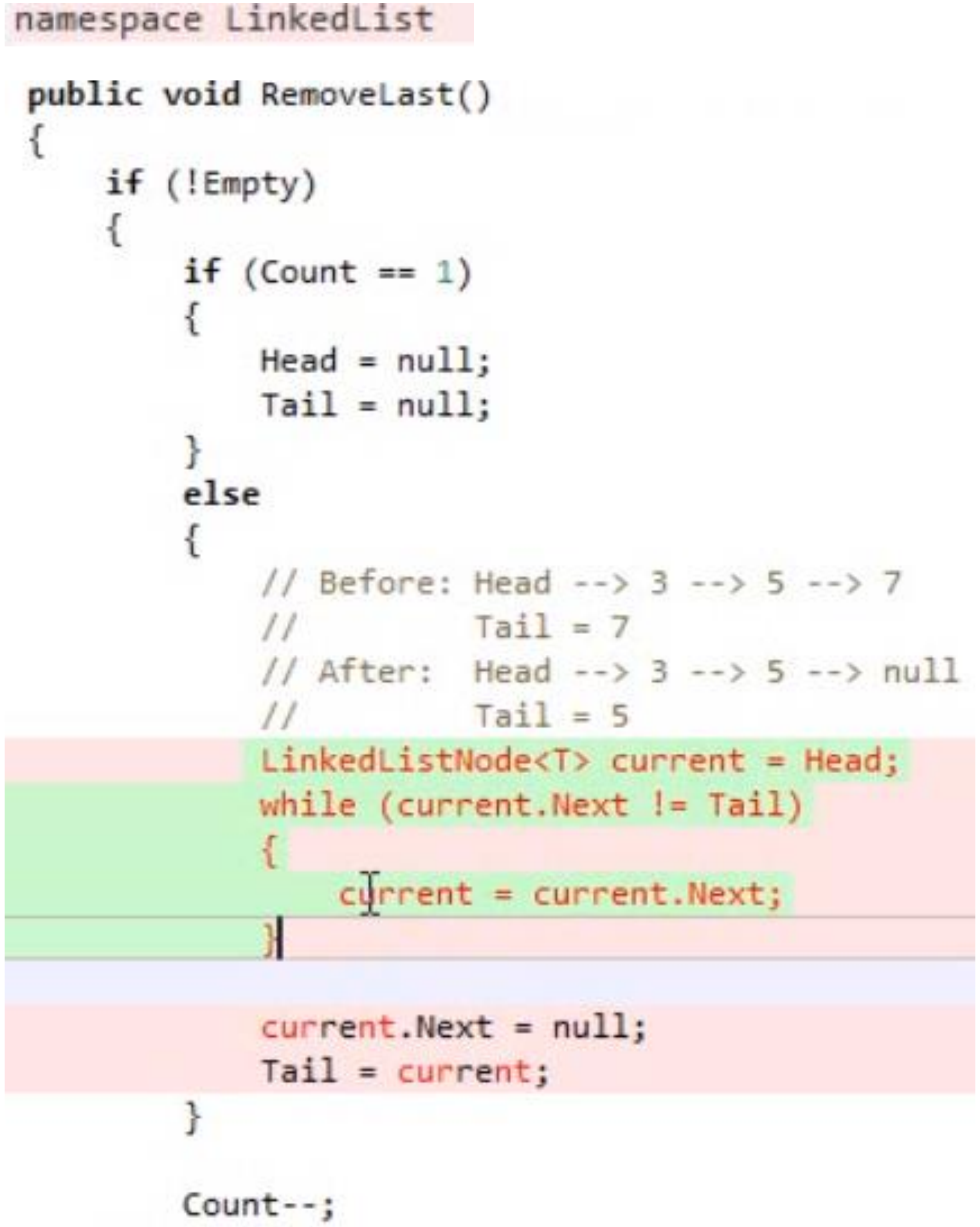
```
public void RemoveFirst()
{
    if (!Empty)
    {
        // Before: Head -> 3 <-> 5
        // After:  Head -------> 5

        // Head -> 3 -> null
        // Head ------> null
        Head = Head.Next;


        Count--;

        if (Empty)
        {
            Tail = null;
        }
        else
        {
            // 5.Previous was 3, now null
            Head.Previous = null;
        }
    }
}
```

namespace LinkedList

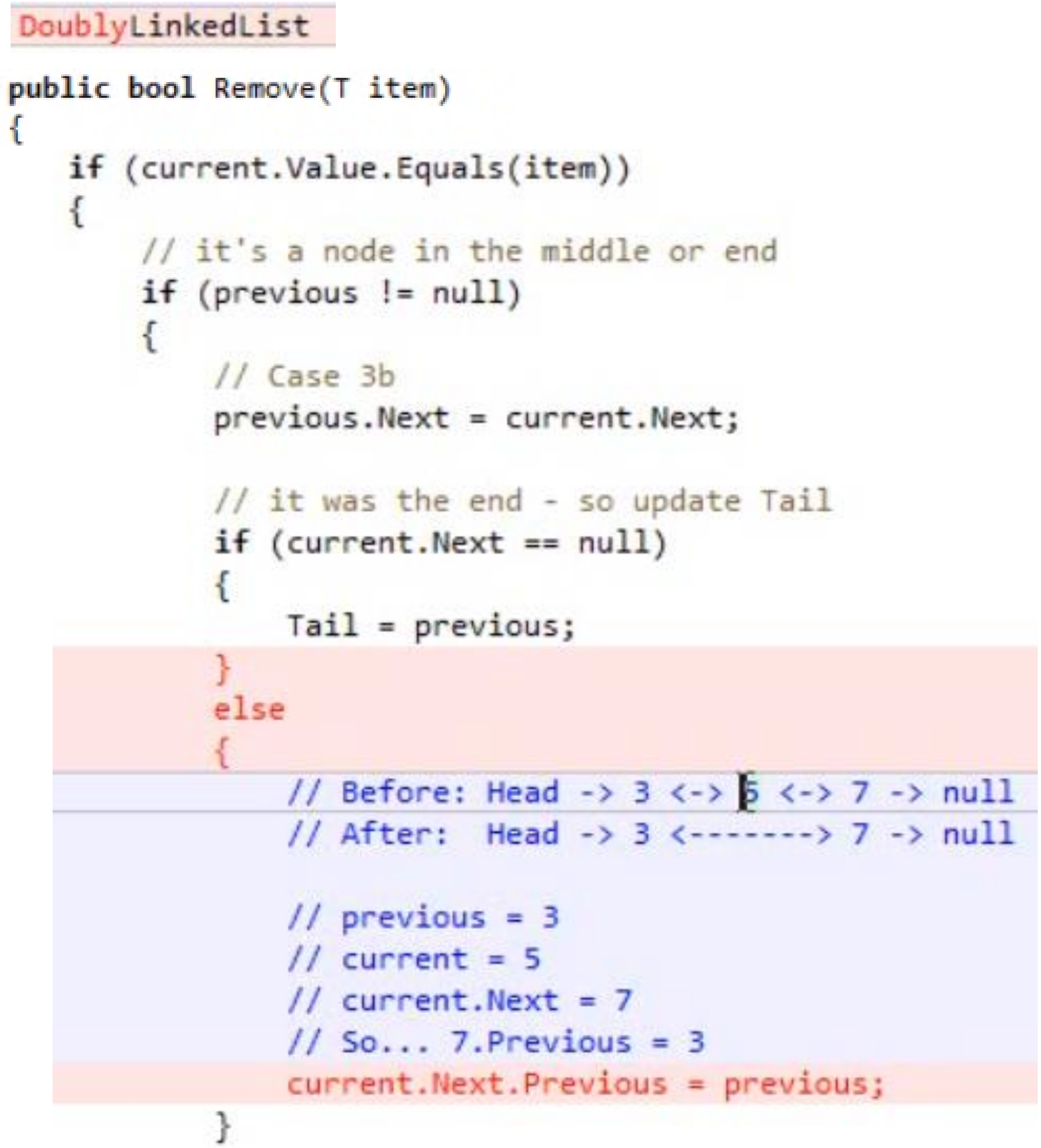
```
public void RemoveFirst()
{
    if (!Empty)
    {
        // Before: Head -> 3 -> 5
        // After:  Head ------> 5

        // Head -> 3 -> null
        // Head ------> null
        Head = Head.Next;

        Count--;

        if (Empty)
        {
            Tail = null;
        }
    }
}
```



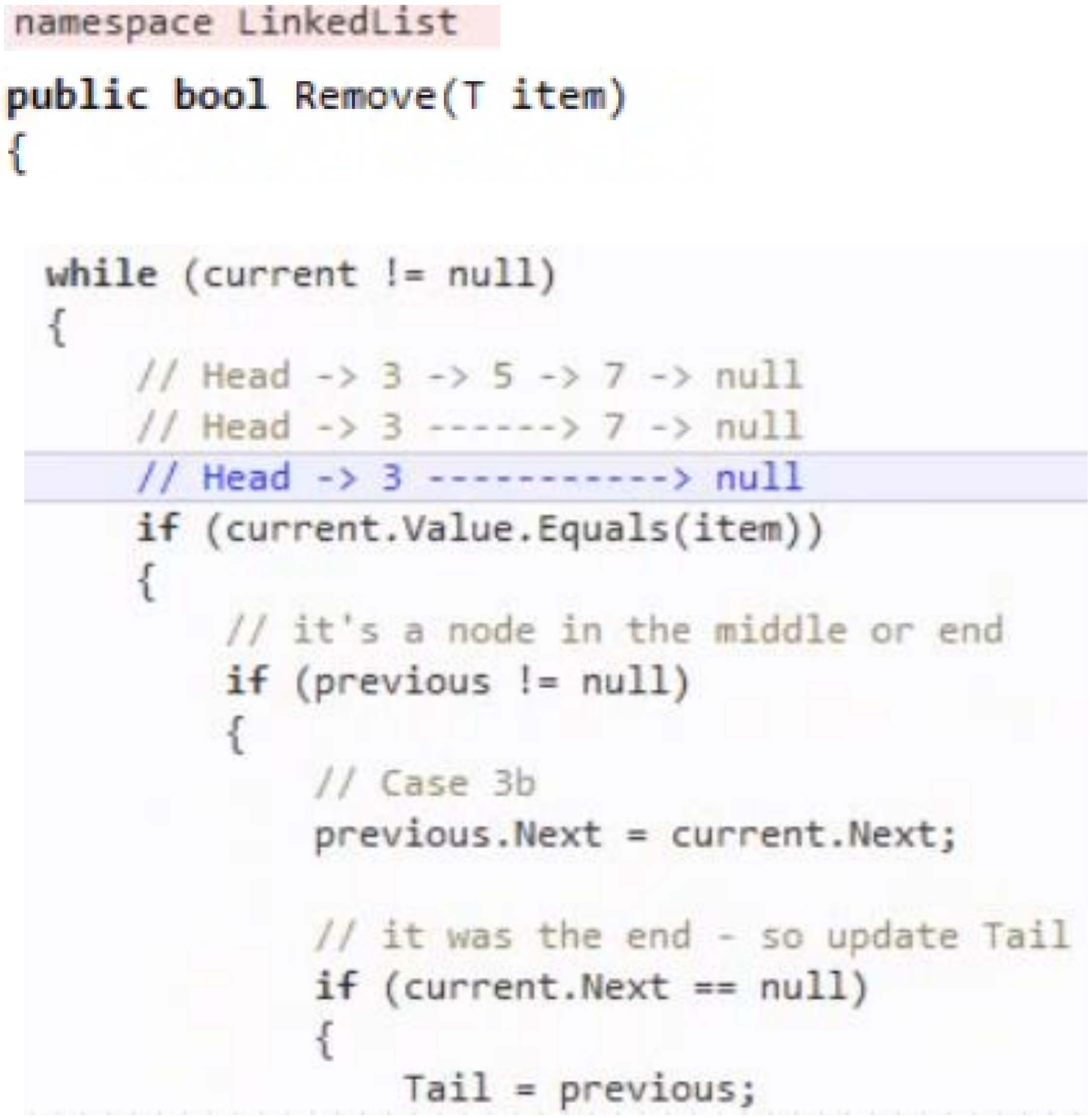
```
DoublyLinkedList

public void RemoveLast()
{
    if (!Empty)
    {
        if (Count == 1)
        {
            Head = null;
            Tail = null;
        }
        else
        {
            // Before: Head --> 3 --> 5 --> 7
            //         Tail = 7
            // After:  Head --> 3 --> 5 --> null
            //         Tail = 5
            // Null out 5's Next pointer




            Tail.Previous.Next = null;
            Tail = Tail.Previous;
        }

        Count--;
```



```
namespace LinkedList

public void RemoveLast()
{
    if (!Empty)
    {
        if (Count == 1)
        {
            Head = null;
            Tail = null;
        }
        else
        {
            // Before: Head --> 3 --> 5 --> 7
            //         Tail = 7
            // After:  Head --> 3 --> 5 --> null
            //         Tail = 5
            LinkedListNode<T> current = Head;
            while (current.Next != Tail)
            {
                current = current.Next;
            }

            current.Next = null;
            Tail = current;
        }

        Count--;
```



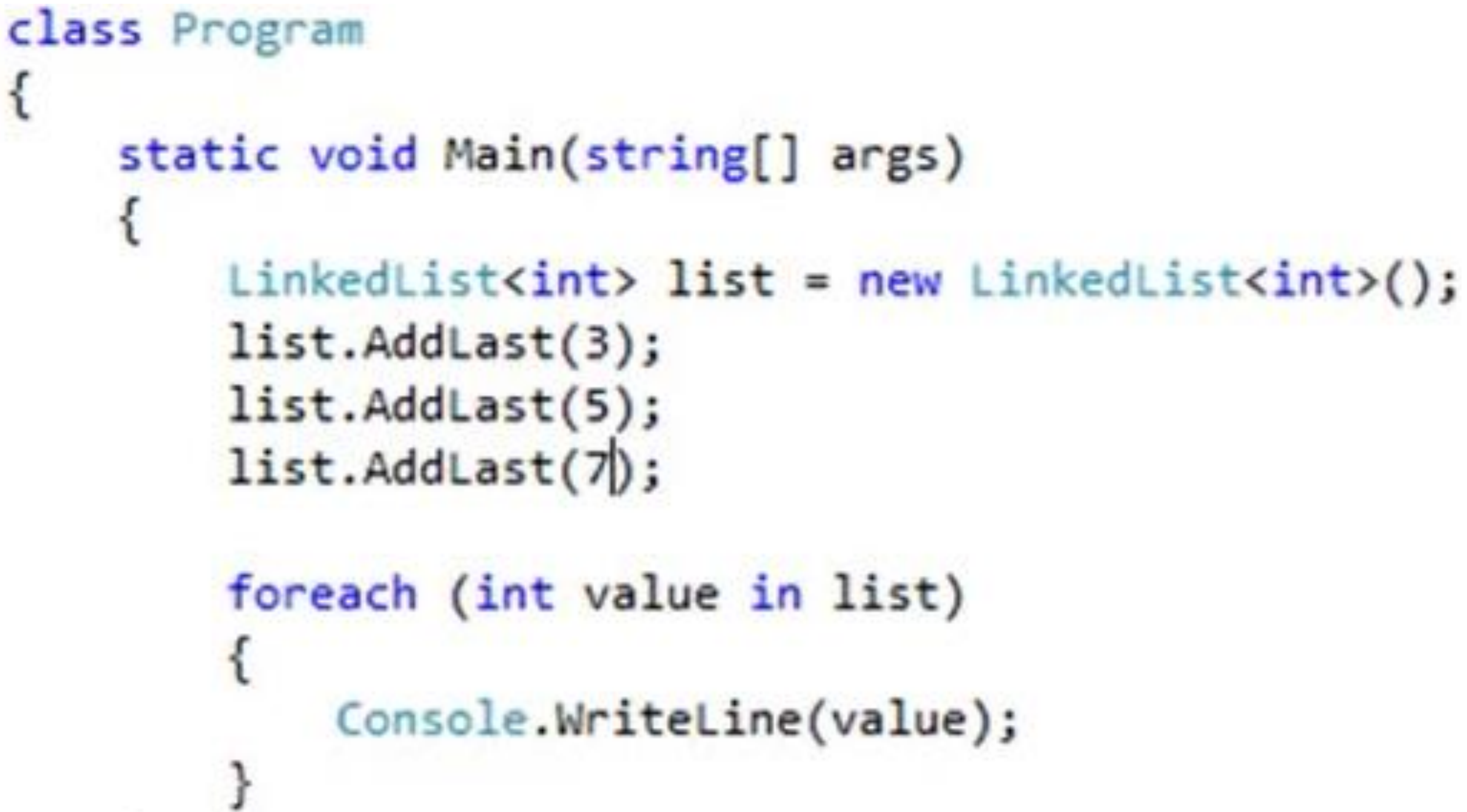
```
DoublyLinkedList

public bool Remove(T item)
{
    if (current.Value.Equals(item))
    {
        // it's a node in the middle or end
        if (previous != null)
        {
            // Case 3b
            previous.Next = current.Next;

            // it was the end - so update Tail
            if (current.Next == null)
            {
                Tail = previous;
            }
            else
            {
                // Before: Head -> 3 <-> 5 <-> 7 -> null
                // After:  Head -> 3 <-------> 7 -> null

                // previous = 3
                // current = 5
                // current.Next = 7
                // So... 7.Previous = 3
                current.Next.Previous = previous;
            }
```



```
namespace LinkedList

public bool Remove(T item)
{

    while (current != null)
    {
        // Head -> 3 -> 5 -> 7 -> null
        // Head -> 3 ------> 7 -> null
        // Head -> 3 -----------> null
        if (current.Value.Equals(item))
        {
            // it's a node in the middle or end
            if (previous != null)
            {
                // Case 3b
                previous.Next = current.Next;

                // it was the end - so update Tail
                if (current.Next == null)
                {
                    Tail = previous;
```

Modern Implementations

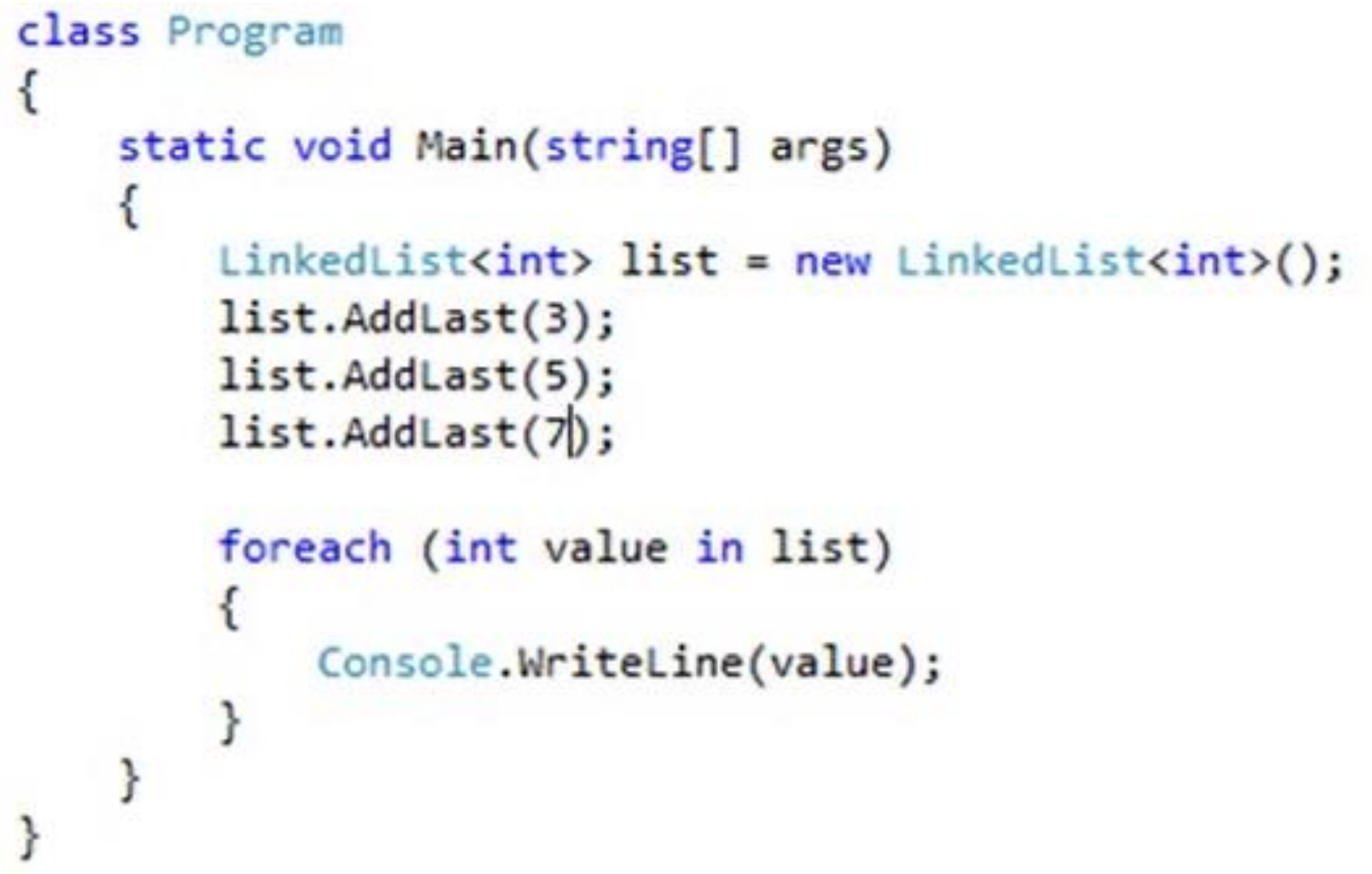
- .NET Framework
  - LinkedList<T>



```
class Program
{
    static void Main(string[] args)
    {
        LinkedList<int> list = new LinkedList<int>();
        list.AddLast(3);
        list.AddLast(5);
        list.AddLast(7);

        foreach (int value in list)
        {
            Console.WriteLine(value);
        }
```

.NET Framework

- **System.Collections.Generic**
- **Doubly linked list**
- **Common Operations**
  - AddFirst, AddLast
  - RemoveFirst, RemoveLast
  - Find, FindLast



```
class Program
{
    static void Main(string[] args)
    {
        LinkedList<int> list = new LinkedList<int>();
        list.AddLast(3);
        list.AddLast(5);
        list.AddLast(7);

        foreach (int value in list)
        {
            Console.WriteLine(value);
        }
    }
}
```

Summary

- Nodes and node chaining
- Singly and doubly linked lists
- Operations
  - Add
  - Remove
  - Enumerate
  - Find
- Modern Implementations
  - LinkedList<T>

References

- LinkedList<T> on MSDN
  - http://msdn.microsoft.com/en-us/library/he2s3bh7.aspx
- LinkedList on Wikipedia
  - http://en.wikipedia.org/wiki/Linked_list