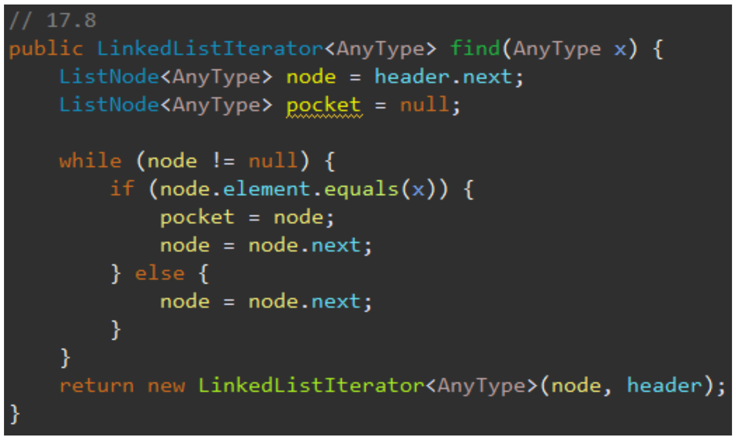
CS 2420
Fall semester section 01
Ricardo Gonzalez Mendez

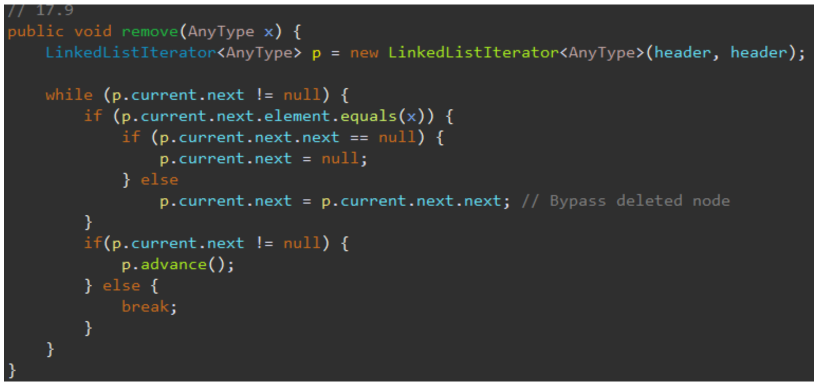
Chapter 17 Programming

17.8: made the code go through the whole link list with a while loop. If I hit an x then I would keep it in a variable then check to see if there was a next one until I found the last instance of x.



```java
// 17.8
public LinkedListIterator<AnyType> find(AnyType x) {
    ListNode<AnyType> node = header.next;
    ListNode<AnyType> pocket = null;

    while (node != null) {
        if (node.element.equals(x)) {
            pocket = node;
            node = node.next;
        } else {
            node = node.next;
        }
    }
    return new LinkedListIterator<AnyType>(node, header);
}
```

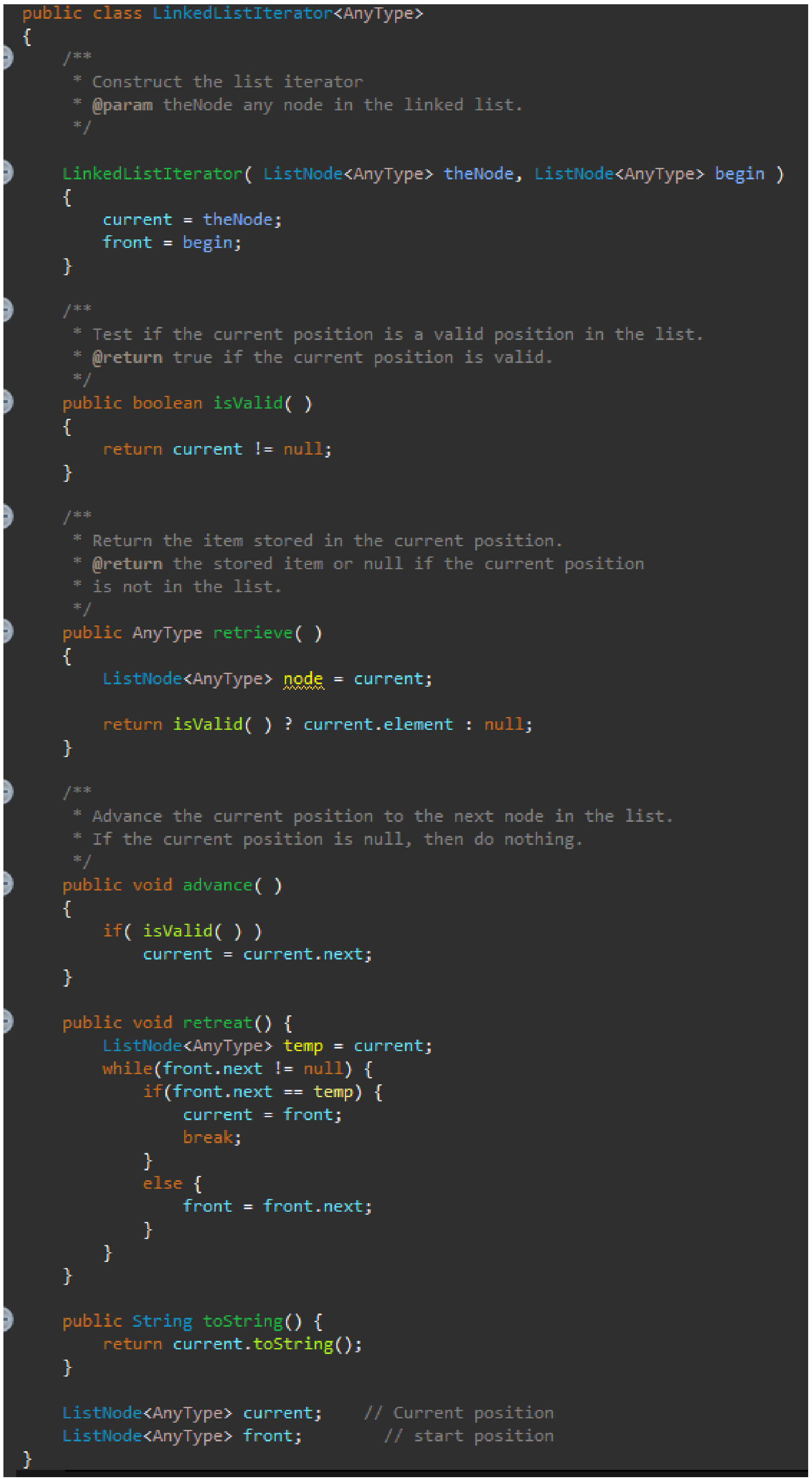
17.9: I go through the linked list checking to see if the .next is equal to what I am looking for and if it is then I skip it by going next next twice.



```java
// 17.9
public void remove(AnyType x) {
    LinkedListIterator<AnyType> p = new LinkedListIterator<AnyType>(header, header);

    while (p.current.next != null) {
        if (p.current.next.element.equals(x)) {
            if (p.current.next.next == null) {
                p.current.next = null;
            } else
                p.current.next = p.current.next.next; // Bypass deleted node
        }
        if(p.current.next != null) {
            p.advance();
        } else {
            break;
        }
    }
}
```

17.14: for this one I did end up adding the front to the constructor so I could go through the link list from the beginning and find the spot before x by iterating through it and having a temp variable that was the current position so when I found the one before the current I just made the current equal the one before so it retreats.

CS 2420
Fall semester section 01
Ricardo Gonzalez Mendez

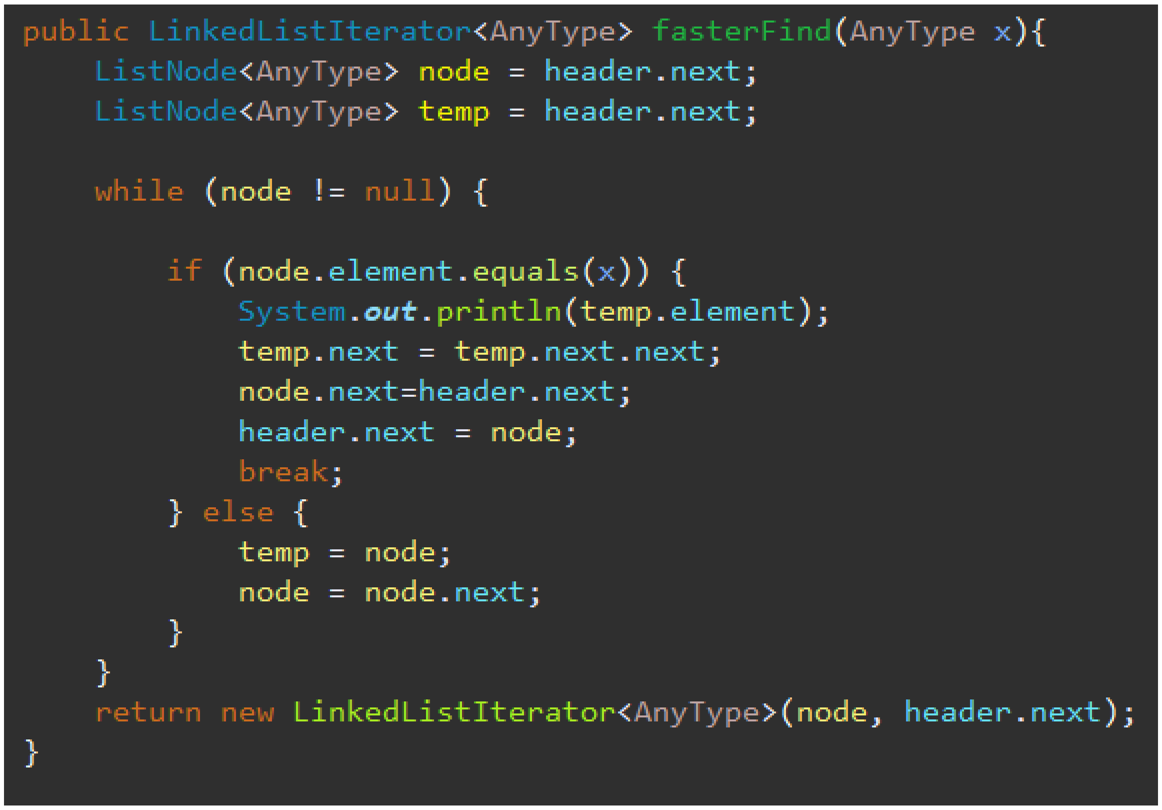


```java
public class LinkedListIterator<AnyType>
{
    /**
     * Construct the list iterator
     * @param theNode any node in the linked list.
     */

    LinkedListIterator( ListNode<AnyType> theNode, ListNode<AnyType> begin )
    {
        current = theNode;
        front = begin;
    }

    /**
     * Test if the current position is a valid position in the list.
     * @return true if the current position is valid.
     */
    public boolean isValid( )
    {
        return current != null;
    }

    /**
     * Return the item stored in the current position.
     * @return the stored item or null if the current position
     * is not in the list.
     */
    public AnyType retrieve( )
    {
        ListNode<AnyType> node = current;

        return isValid( ) ? current.element : null;
    }

    /**
     * Advance the current position to the next node in the list.
     * If the current position is null, then do nothing.
     */
    public void advance( )
    {
        if( isValid( ) )
            current = current.next;
    }

    public void retreat() {
        ListNode<AnyType> temp = current;
        while(front.next != null) {
            if(front.next == temp) {
                current = front;
                break;
            }
            else {
                front = front.next;
            }
        }
    }

    public String toString() {
        return current.toString();
    }

    ListNode<AnyType> current;    // Current position
    ListNode<AnyType> front;        // start position
}
```

CS 2420
Fall semester section 01
Ricardo Gonzalez Mendez

17.17:

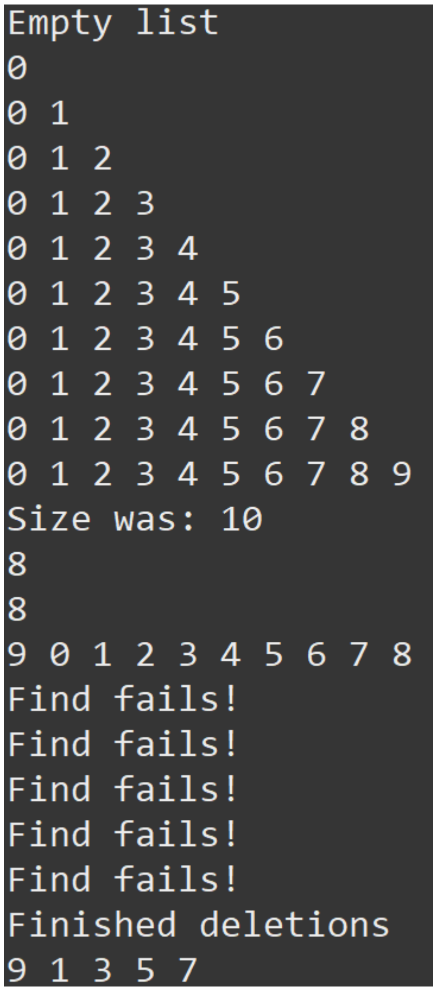
in my ,method I had it start from the front of the list to check if it was there and it was I had a temp variable lag behind one to it could be the previous and if I found what I was looking for then I would make my temp point to the next next so it wasn't pointing to nothing and I would have what I found the .next to point to the header to keep the information then I made it the new header.



```
public LinkedListIterator<AnyType> fasterFind(AnyType x){
    ListNode<AnyType> node = header.next;
    ListNode<AnyType> temp = header.next;

    while (node != null) {

        if (node.element.equals(x)) {
            System.out.println(temp.element);
            temp.next = temp.next.next;
            node.next=header.next;
            header.next = node;
            break;
        } else {
            temp = node;
            node = node.next;
        }
    }
    return new LinkedListIterator<AnyType>(node, header.next);
}
```

here is my main where I tested some of the methods I made

they all worked.



```
Empty list
0
0 1
0 1 2
0 1 2 3
0 1 2 3 4
0 1 2 3 4 5
0 1 2 3 4 5 6
0 1 2 3 4 5 6 7
0 1 2 3 4 5 6 7 8
0 1 2 3 4 5 6 7 8 9
Size was: 10
8
8
9 0 1 2 3 4 5 6 7 8
Find fails!
Find fails!
Find fails!
Find fails!
Find fails!
Finished deletions
9 1 3 5 7
```