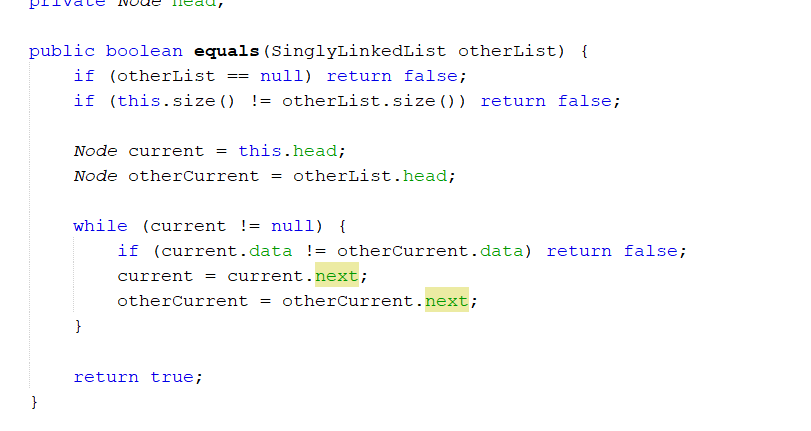
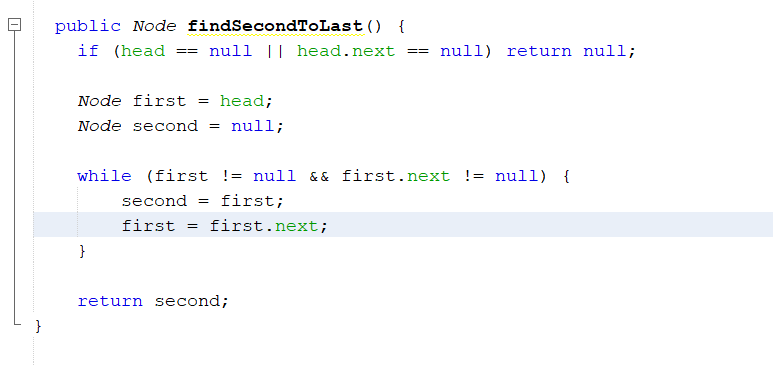
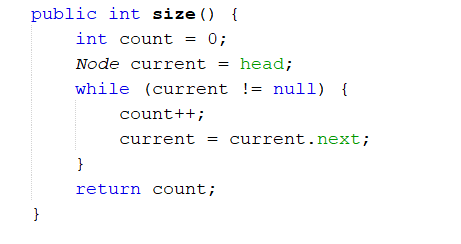
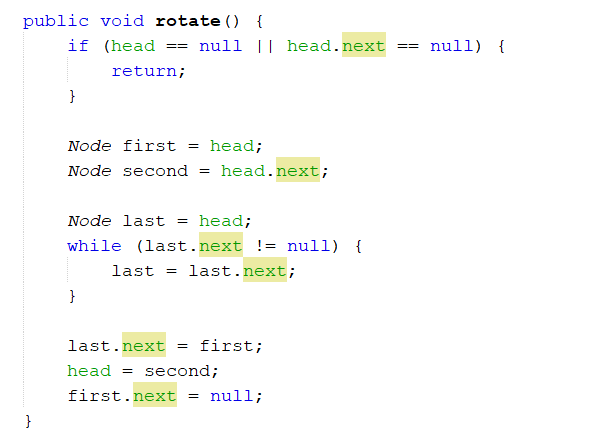
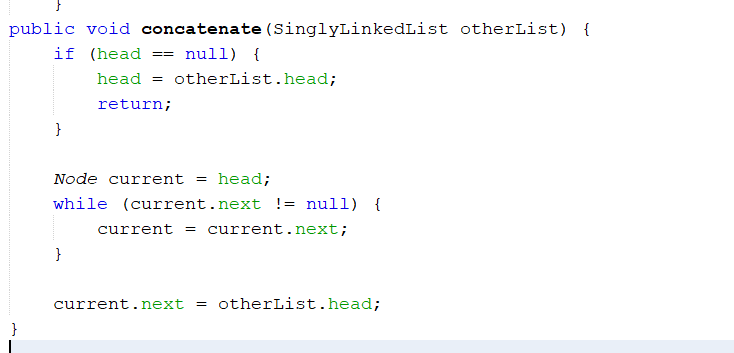
1. **Develop an implementation of the equals method in the context of the SinglyLinkedList class.**
2. **Give an algorithm for finding the second-to-last node in a singly linked list in which the last node is indicated by a null next reference.**

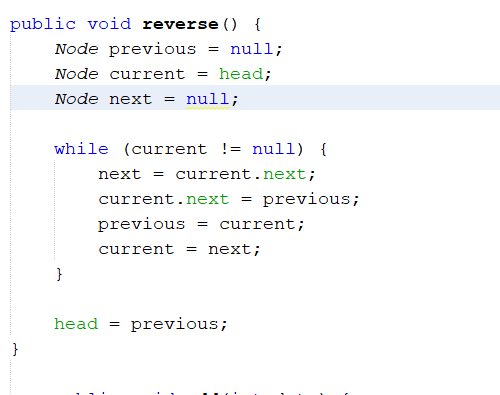
* Initialize two pointers, first and second, both pointing to the head of the list.
* Traverse the list until first reaches the last node (first.next == null).
* While traversing, move both pointers one step forward at a time so that second will be one node behind first.
* When first reaches the last node, second will point to the second-to-last node.

1. **Give an implementation of the size() method for the SinglyLinkedList class, assuming that we did not maintain size as an instance variable.**
2. **Implement a rotate() method in the SinglyLinkedList class, which has semantics equal to addLast(removeFirst()), yet without creating any new node.**
3. **Describe an algorithm for concatenating two singly linked lists L and M, into a single list L′ that contains all the nodes of L followed by all the nodes of M.**

* If list L is empty, return M.
* Traverse L until the last node.
* Set the last node of L to point to the head of M.
* If M is empty, the list remains unchanged.

1. **Describe in detail an algorithm for reversing a singly linked list L using only a constant amount of additional space.**

* Initialize three pointers: previous, current, and next. Set previous to null, and current to the head of the list.
* Traverse the list, updating the next pointer of each node to point to the previous node.
* Move the previous and current pointers one step forward.
* After the traversal, set the head of the list to previous.



Testing:

