Linked List Cycle I (LeetCode 141 - Easy)

Problem Description

Given a linked list, determine if it has a cycle in it.

Problem Analysis

Two pointers – slow and fast, return if two pointers meet.

Solution

```
/**
 * LeetCode 141
 * Two Pointers.
 * @param head
 * @return true if linked list has a cycle
 * @timecomplexity - O(n)
 */
public static boolean hasCycle(ListNode head) {
  if (head == null) {
        return false;
    }
    ListNode slow = head;
    ListNode fast = head;
    while (fast != null && fast.next != null) {
        slow = slow.next;
        fast = fast.next.next;
        if (slow == fast) {
            return true;
        }
    }
    return false;
}
```

Linked List Cycle II (LeetCode 142 - Medium)

Problem Description

Given a linked list, return the node where the cycle begins. If there is no cycle, return null.

Problem Analysis

Using two pointers: One of them takes one step at a time. The other pointer takes two steps at a time. Suppose they first met at step k, the length of the Cycle is r. -> 2k - k = nr -> k = nr

Now, let's say the distance between the start node of list and the start node of cycle is s. The distance between the start of list and the first meeting node is k. The distance between the start node of cycle and the first meeting node is m. -> s = k - m -> s = nr – m = (n - 1)*r + (r - m)
Therefore, using one pointer start from the start node of list, the other pointer start from the first meeting node. All of them take one step at a time, the first time they meeting each other is at the start of the cycle.

Solution

```
/**
 * LeetCode 142
 * Two Pointers.
 * @param head
 * @return The node where the cycle begins.
           If there is no cycle, return null.
 * @timecomplexity - O(n)
 */
public static ListNode detectCycle(ListNode head) {

    ListNode slow = head;
    ListNode fast = head;
    while (fast != null && fast.next != null) {
        slow = slow.next;
        fast = fast.next.next;
        if (slow == fast) {
            break;
        }
    }

    //No cycle.
    if (fast == null || fast.next == null) {
        return null;
    }

    slow = head;
    while (slow != fast) {
        slow = slow.next;
        fast = fast.next;
    }
    return slow;
}
```