Java Intersection of Linked Lists

Write java code for Given the heads of two singly linked-lists headA and headB, return the node at which the two lists intersect. If the two linked lists have no intersection at all, return null.
Use   public ListNode getIntersectionNode(ListNode headA, ListNode headB) {
}

Here's a Java solution to find the intersection node of two singly linked lists. The idea is to traverse both lists, and if one list reaches the end, it switches to the head of the other list. This way, both pointers will traverse the same number of nodes when an intersection exists.

```java
class ListNode {
    int val;
    ListNode next;
    ListNode(int x) {
        val = x;
        next = null;
    }
}

public class Solution {
    public ListNode getIntersectionNode(ListNode headA, ListNode headB) {
        if (headA == null || headB == null) {
            return null;
        }

        ListNode pointerA = headA;
        ListNode pointerB = headB;

        // Traverse both lists. Once either pointer reaches the end, reset it to the head
of the other list.
        while (pointerA != pointerB) {
            pointerA = (pointerA == null) ? headB : pointerA.next;
            pointerB = (pointerB == null) ? headA : pointerB.next;
        }

        // Either the intersection node or null if there's no intersection
        return pointerA;
    }
}
```

Explanation:

- **Pointer Resetting**: Both pointers traverse through their respective lists. When they reach the end of a list, they are reset to the head of the other list. If there is an intersection, the pointers will meet at the intersection node after one pass through both lists.
- **No Intersection Case**: If the lists don't intersect, both pointers will eventually become `null` at the same time, and the loop will exit.

This solution runs in **O(n + m)** time, where `n` and `m` are the lengths of the two linked lists. It uses **O(1)** space.