Singly and Doubly Linked List

Overview

- A sequential list of nodes that holds data that point to other nodes
    - The last node points to null to determine when to terminate
- **Terminology**
    - **Head**, **Tail** - Beginning node and end node of a linked list
    - **Pointer** - a reference to a different node
    - **Node** - an object containing data and pointers
- **Singly and Doubly Linked List**
    - **Singly** - only uses next pointers
        - **Pros**
            - Simple Implementation
            - Uses less memory
        - **Cons**
            - List cannot be traversed backward
    - **Doubly** - contains next and previous pointers
        - **Pros**
            - Can be traversed backward
        - **Cons**
            - Takes 2x the memory
            - More complicated implementation

Why Use It?

- Used in many List, Queue, and Stack implementations
- Great for circular lists

- Can model real-world objects such as trains
- Used for hashing collisions
- Used in adjacency list for graphs

Big O Analysis

Aa Operation	SL	DL	Explanation
Insertion	O(1)	O(1)	Head And Tail: Constant time due to their pointers
Deletion (Head)	O(1)	O(1)	Constant time due to the head pointer
Deletion (Tail)	O(n)	O(1)	Without a previous pointer, the tail pointer cannot be readjusted in constant time, therefore, the list must be traversed to remove the tail with a singly linked list. This is not necessary with a doubly-linked list.
Deletion (Middle)	O(n)	O(n)	
Search	O(n)	O(n)	

Code Implementation

- https://www.geeksforgeeks.org/implementing-a-linked-list-in-java-using-class/