What is a linked list in Java, and how does it differ from arrays?

-In Java, a linked list is a linear data structure where elements are stored in "nodes," each containing data and a pointer to the next node in the sequence. Unlike arrays, linked list elements are not stored contiguously in memory but can be scattered. Arrays, on the other hand, store elements in a contiguous block of memory, allowing for direct access using an index.

What are the main types of linked lists?

**Singly Linked List (SLL)**

* Nodes contain data and a reference to the next node.
* The last node points to null, marking the end of the list.
* Traversal is one-directional, from the head to the last node.
* Advantages: Simple structure and memory-efficient.
* Disadvantages: Cannot traverse backward; inserting or deleting at the middle or end requires traversal.

**Doubly Linked List (DLL)**

* Nodes contain data, a reference to the next node, and a reference to the previous node.
* Traversal is bidirectional (forward and backward).
* Advantages: Flexible traversal and easier insertion/deletion at both ends.
* Disadvantages: Higher memory usage due to the additional reference.

**Circular Linked List (CLL)**

* The last node connects back to the first node, forming a circular structure.
* Can be implemented as a singly or doubly linked list.
* Traversal can continue indefinitely unless explicitly stopped.
* Advantages: Useful for scenarios like round-robin scheduling or continuous iteration.
* Disadvantages: Requires careful handling to avoid infinite loops.

How do you implement a basic linked list in Java?

// Node class

class Node {

int data;

Node next;

// Constructor to initialize a node

Node(int data) {

this.data = data;

this.next = null;

}

}

// Singly Linked List class

class SinglyLinkedList {

private Node head;

// Method to insert a new node at the end

public void insert(int data) {

Node newNode = new Node(data);

if (head == null) {

head = newNode;

} else {

Node current = head;

while (current.next != null) {

current = current.next;

}

current.next = newNode;

}

}

// Method to display the linked list

public void display() {

if (head == null) {

System.out.println("The list is empty.");

return;

}

Node current = head;

System.out.print("Linked List: ");

while (current != null) {

System.out.print(current.data + " -> ");

current = current.next; // Move to the next node

}

System.out.println("null");

}

// Main method to test the linked list

public static void main(String[] args) {

SinglyLinkedList list = new SinglyLinkedList();

// Insert nodes

list.insert(10);

list.insert(20);

list.insert(30);

// Display the list

list.display();

}

}

What are the advantages and disadvantages of linked lists compared to other data structures?

**Advantages of Linked Lists**

* Linked lists have a dynamic size, efficient insertion/deletion without shifting, and flexibility for custom data structures, though they require more memory for pointers and sequential traversal.

**Disadvantages of Linked Lists**

* They have slower access times, use more memory than arrays, and are cache-unfriendly due to non-contiguous memory allocation.

**Preferred Scenarios**

* Linked lists are ideal when the size is dynamic, frequent insertions/deletions occur, or custom structures like stacks and queues are needed.

**When to Use Arrays**

* Arrays are better for fast random access, memory efficiency, and operations requiring sorting or binary search.

What is a static stack, and how is it implemented in Java?

-A static stack is a stack data structure implemented using a fixed-size array. Because the size is fixed, the stack has a maximum capacity defined at the time of its creation. This means that a static stack can only hold a predetermined number of elements, and attempting to add elements beyond this limit will result in a stack overflow.

What is a dynamic stack, and how is it implemented in Java?

A dynamic stack is a stack data structure that can grow or shrink in size during runtime, unlike a static stack which has a fixed size. This flexibility is achieved by using a linked list as the underlying data structure.

How does a queue differ from a stack, and what are their primary operations?

A queue is a linear data structure that works on the **FIFO (First In, First Out)** principle, where elements are added at the rear and removed from the front, much like a line of people waiting their turn. In contrast, a stack follows the **LIFO (Last In, First Out)** principle, where the most recently added element is the first to be removed, similar to stacking plates and removing the top one first.

The main operations for a queue include enqueue, which adds an element to the rear, and dequeue, which removes an element from the front. For stacks, the primary operations are push, which adds an element to the top, and pop, which removes the top element. Both structures also allow you to view the next element with peek, but the element retrieved depends on whether it's a queue or a stack.

Queues are ideal for scenarios like task scheduling and buffering, while stacks are useful for backtracking, undo actions, and solving expressions.

What is a static queue?

A Static Queue is a queue of fixed size implemented using array.

What is a dynamic queue?

A dynamic queue is a queue whose size can change during runtime.

What are some real-world applications of linked lists, stacks, and queues?

**Linked Lists**

1. **Dynamic Memory Allocation**:  
   Linked lists are used in memory management systems to allocate memory dynamically. They allow efficient allocation and deallocation of memory blocks, especially when the size is not known in advance.
2. **Implementing Graphs**:  
   Linked lists are used to represent adjacency lists in graph structures. Each node points to other nodes, which makes traversing the graph and performing operations like searching or pathfinding efficient.

**Stacks**

1. **Undo/Redo Functionality**:  
   Stacks are used in applications like text editors, where the most recent action is stored and can be undone or redone by popping or pushing actions on the stack.
2. **Function Call Management (Recursion)**:  
   The call stack in programming languages stores information about function calls. Each call is pushed onto the stack, and when the function completes, it is popped off, ensuring the correct order of execution.

**Queues**

1. **Task Scheduling**:  
   Queues are used in operating systems to manage tasks for CPU processing. Tasks are handled in the order they arrive, ensuring fair and efficient task execution (FIFO).
2. **Print Spoolers**:  
   Queues manage print jobs in printers. Documents are added to the print queue and processed in the order they were received, ensuring each job is printed in sequence.