**day14\_107856406\_dsdipt\_sudipto\_2july2025**

**Employee Code:** 107856406

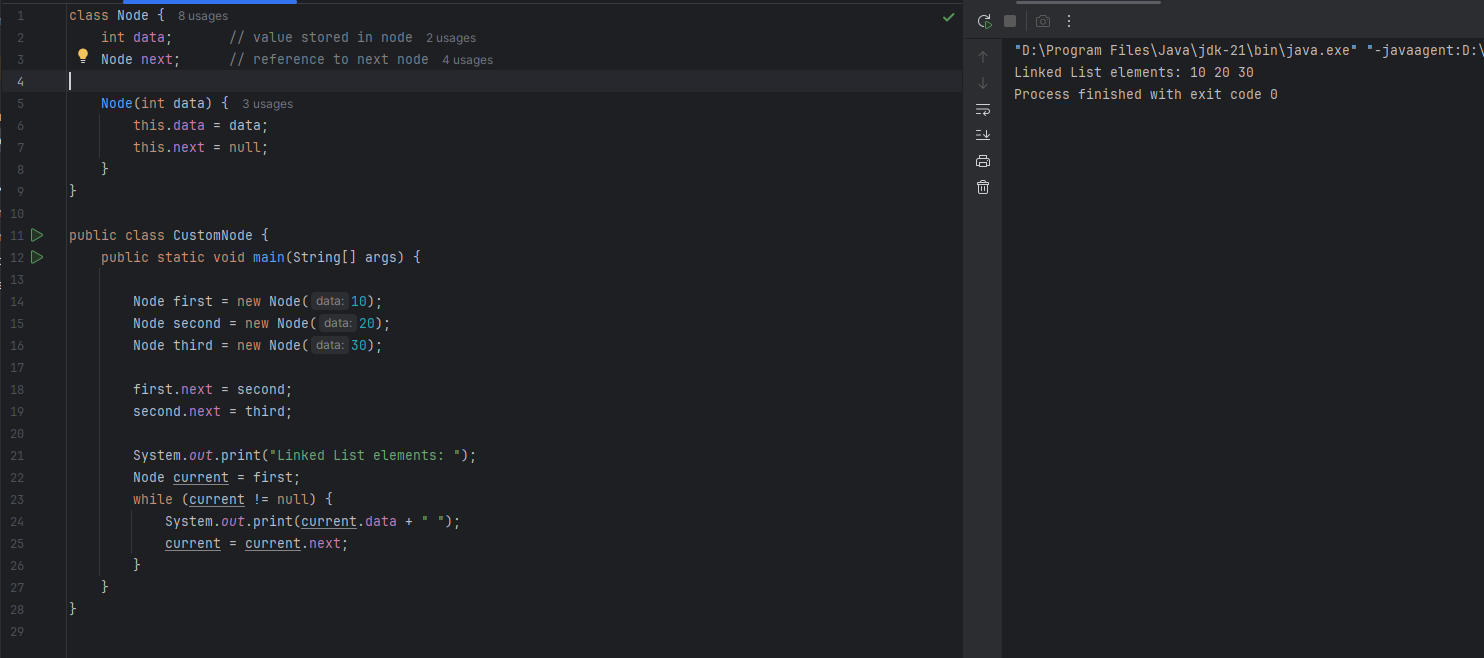
**Login ID:** dsdipt

**Email :** dsdipt@amazon.com

**Name:** Sudipto Das

**Date:** 2 July 2025 (Day 14)

### ***Task 1: Custom Node with Linked List Traversal***



### ***Task 2: What is Traversing a Linked List?***

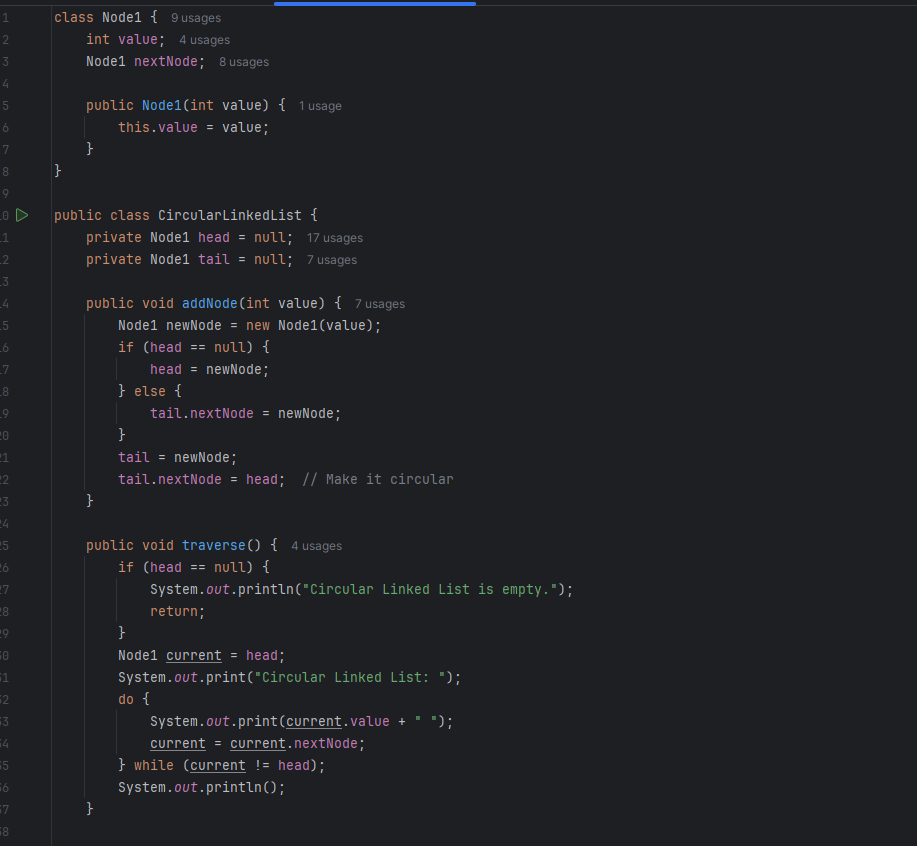
Traversing a linked list means visiting each node one by one starting from the head (first node), and performing an action on each node (like printing the value, summing, searching, etc.).

**Why is traversal important?**

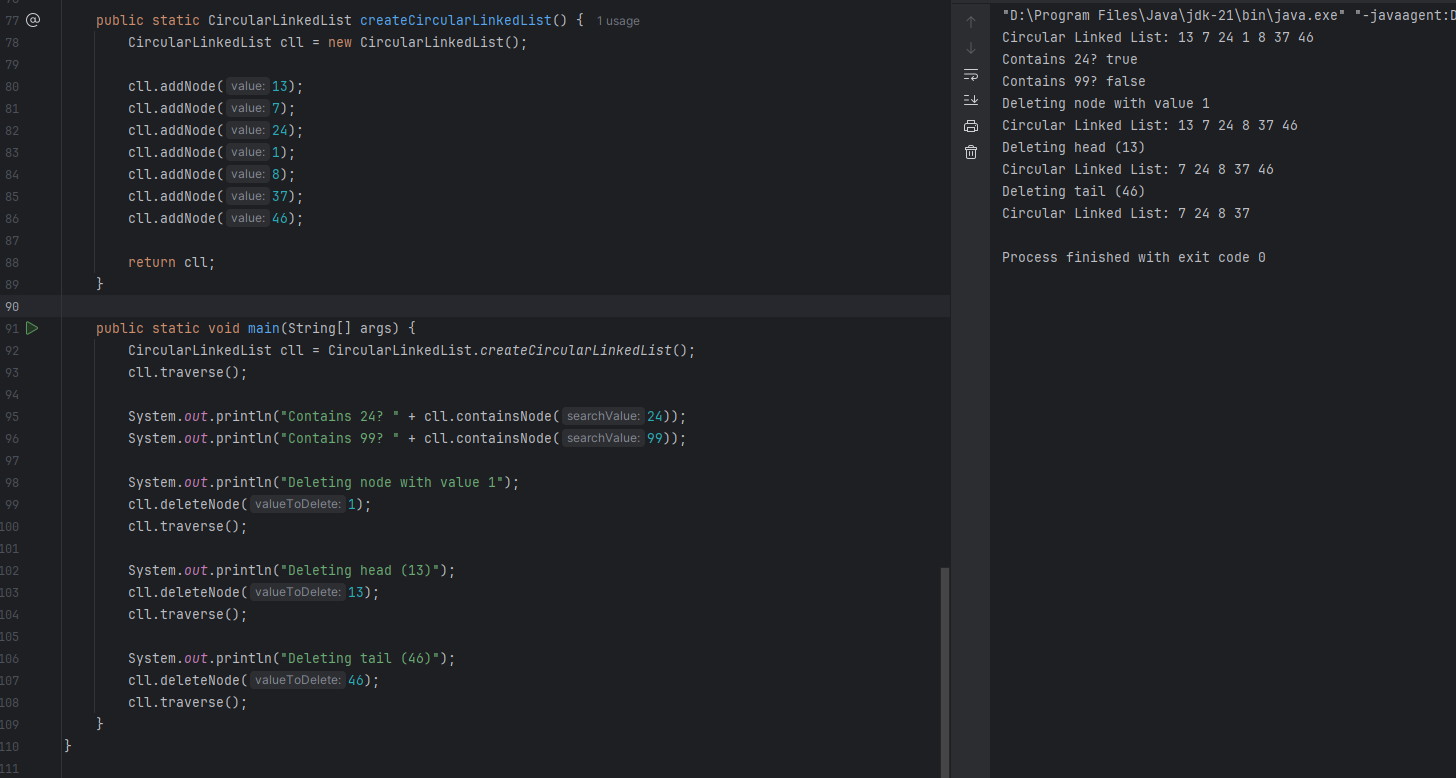
Because in a linked list:

* Elements are not stored in continuous memory.
* We can’t access elements directly by index (like in an array).
* So we must start from the head and follow each node using .next.

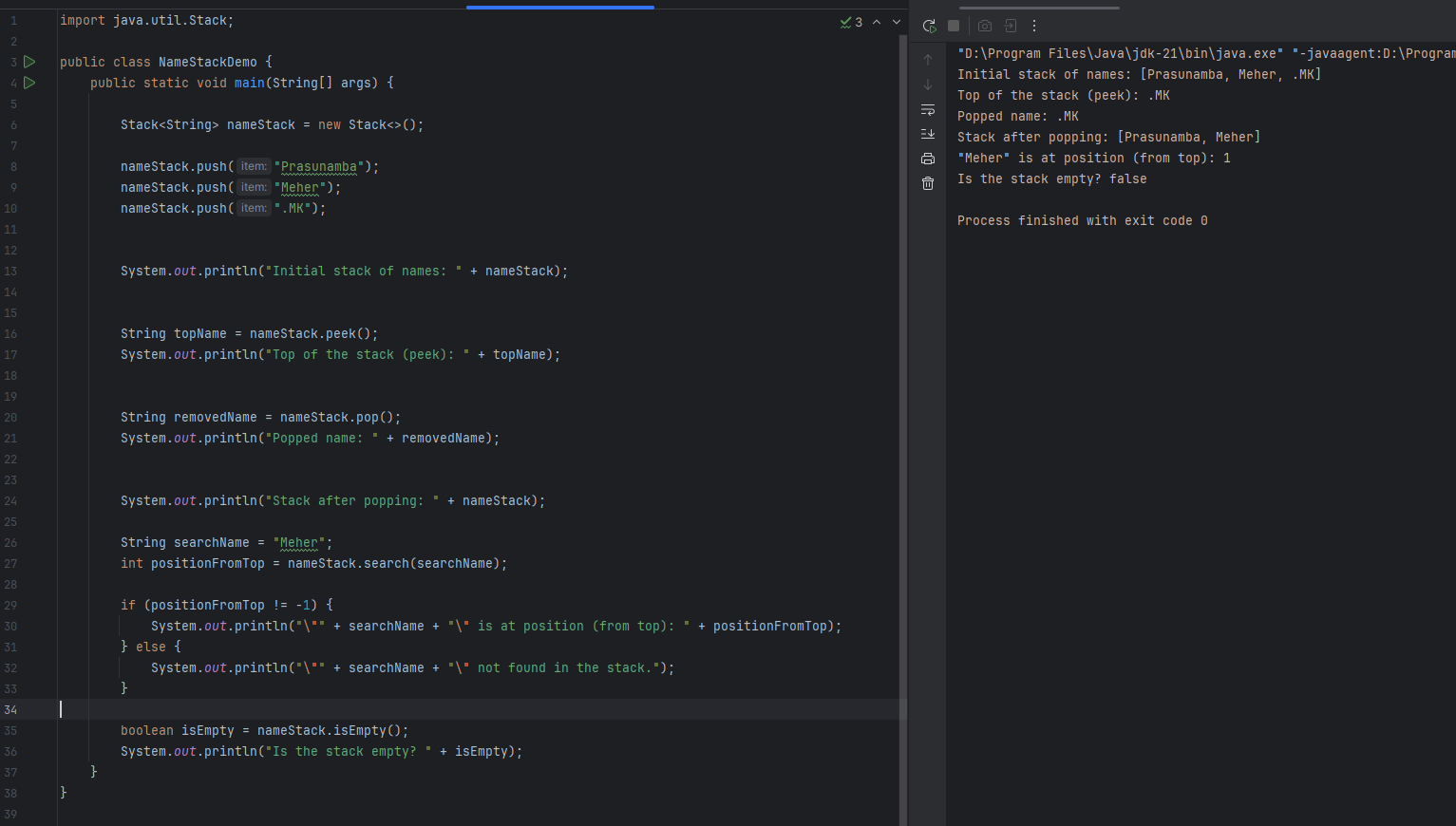
***Task 3: Circular Linked List Creation***



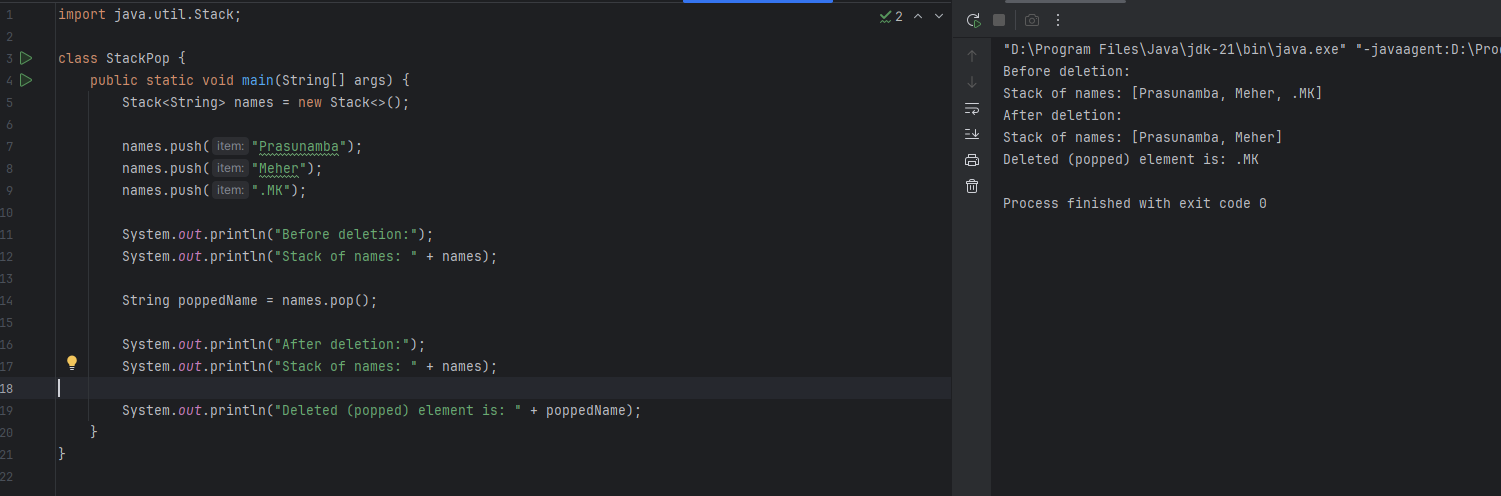




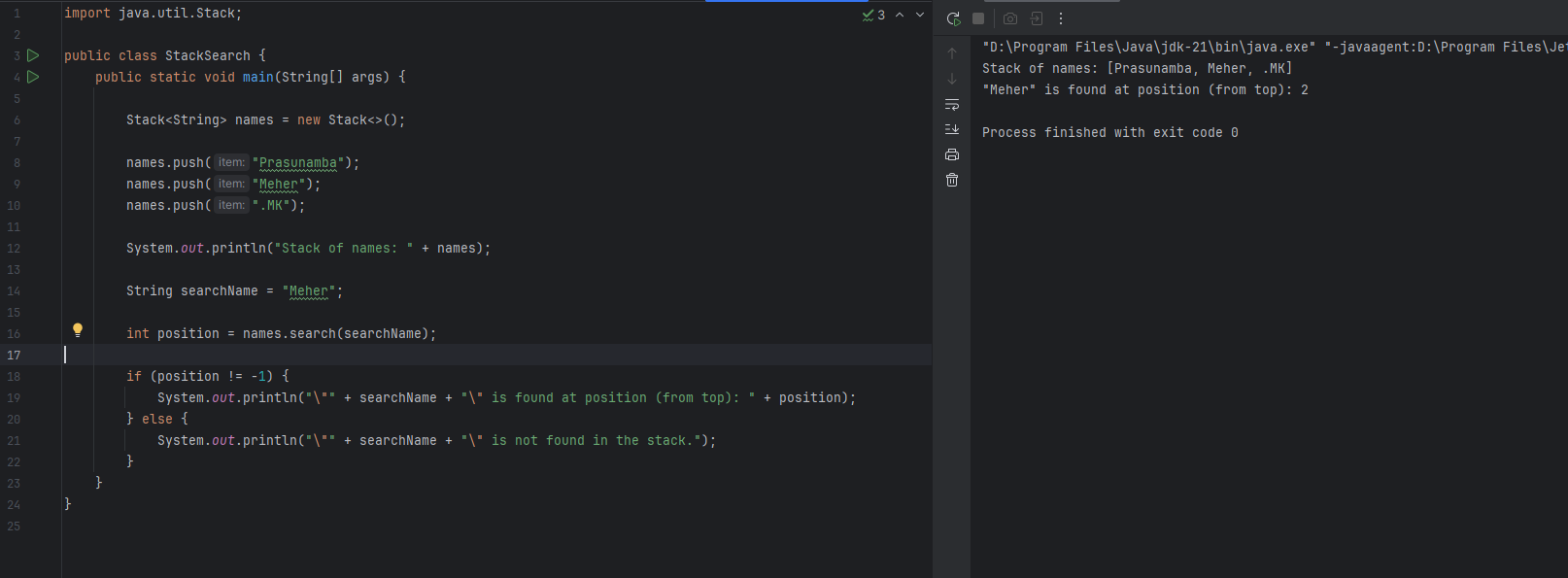
***Task 4: Implementing Stack Operations***

******

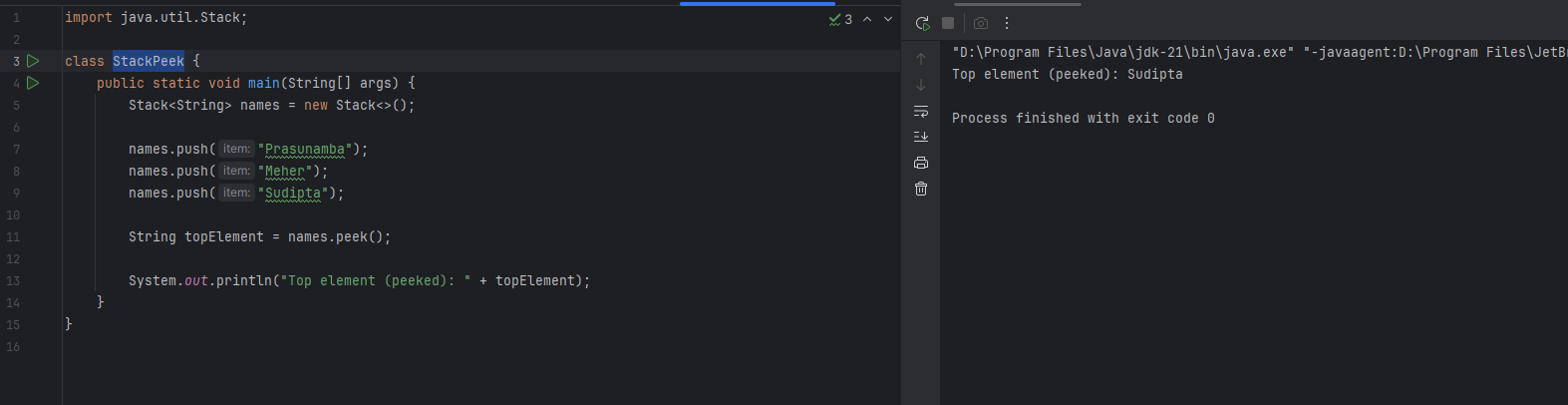
***Task 5: Create a Stack, Pop an Element, and Print It***



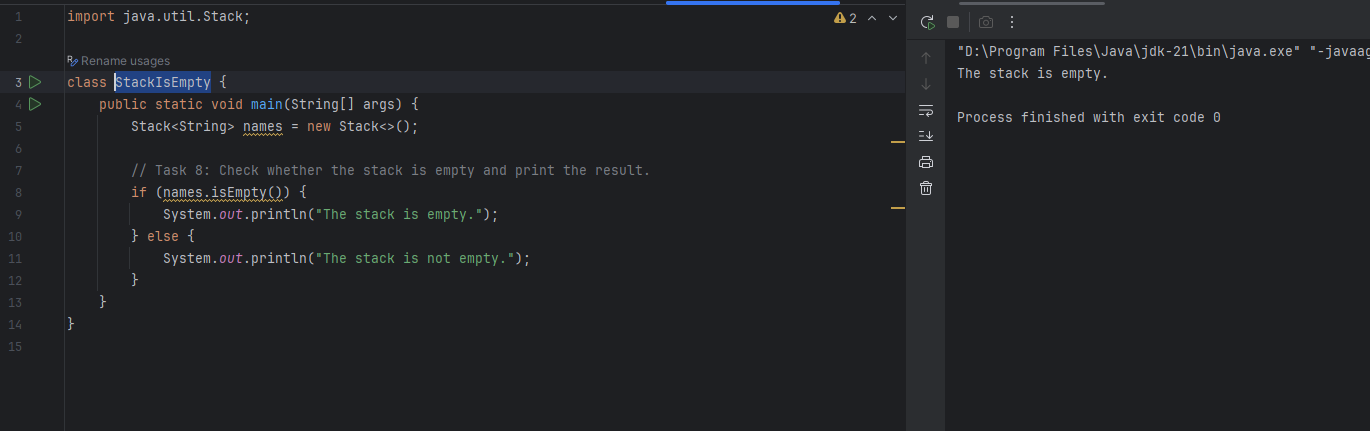
***Task 6: Find an Element in the Stack and Display Its Position***



***Task 7: Peek the top element of the stack***



***Task 8: Check if the stack is empty or not***



***Task 9: What are the methods of the Stack class in Java?***

* **push(E item):**  
   This method is used to add an element to the top of the stack. It increases the stack size by one.  
   **Example: stack.push("Apple");**
* **pop():**  
   This method removes and returns the top element of the stack. If the stack is empty, it throws an EmptyStackException.  
   **Example: String item = stack.pop();**
* **peek():**  
   This method returns the top element without removing it from the stack. Useful for inspection.  
   **Example: String top = stack.peek();**
* **isEmpty():**  
   Returns true if the stack contains no elements; otherwise returns false.  
   **Example: if (stack.isEmpty()) { ... }**
* **search(Object o):**  
   Returns the 1-based position of the element from the top of the stack. If not found, it returns -1.  
   **Example: int pos = stack.search("Banana");**
* **size() (inherited from Vector):**  
   Returns the total number of elements currently in the stack.  
   **Example: int count = stack.size();**
* **clear() (inherited from Vector):**  
   Removes all elements from the stack, making it empty.  
   **Example: stack.clear();**
* **get(int index) (inherited from Vector):**  
   Allows you to access the element at a specific index (not typical for stack behavior).  
   **Example: stack.get(0);**

***Task 10: What are the Common Operations in Queue?***

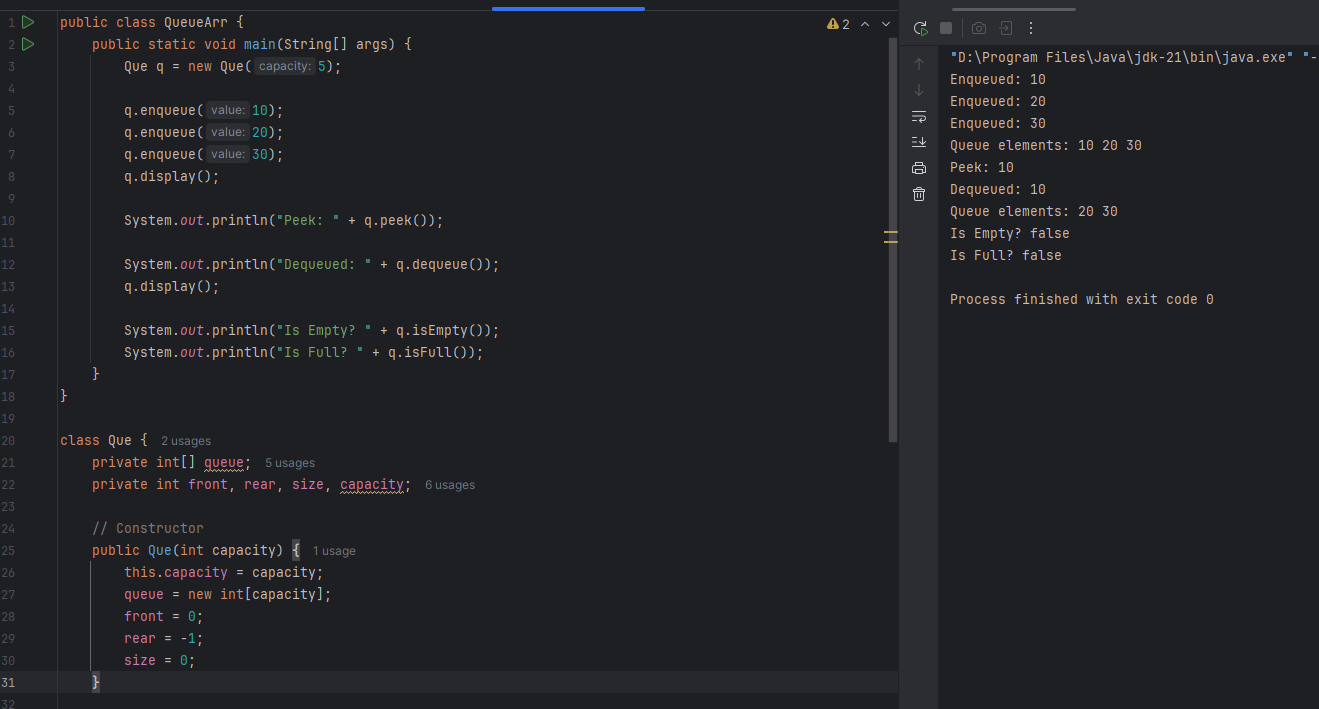
A **Queue** in data structures follows the **FIFO** principle (First-In, First-Out). The element added first is the one removed first.

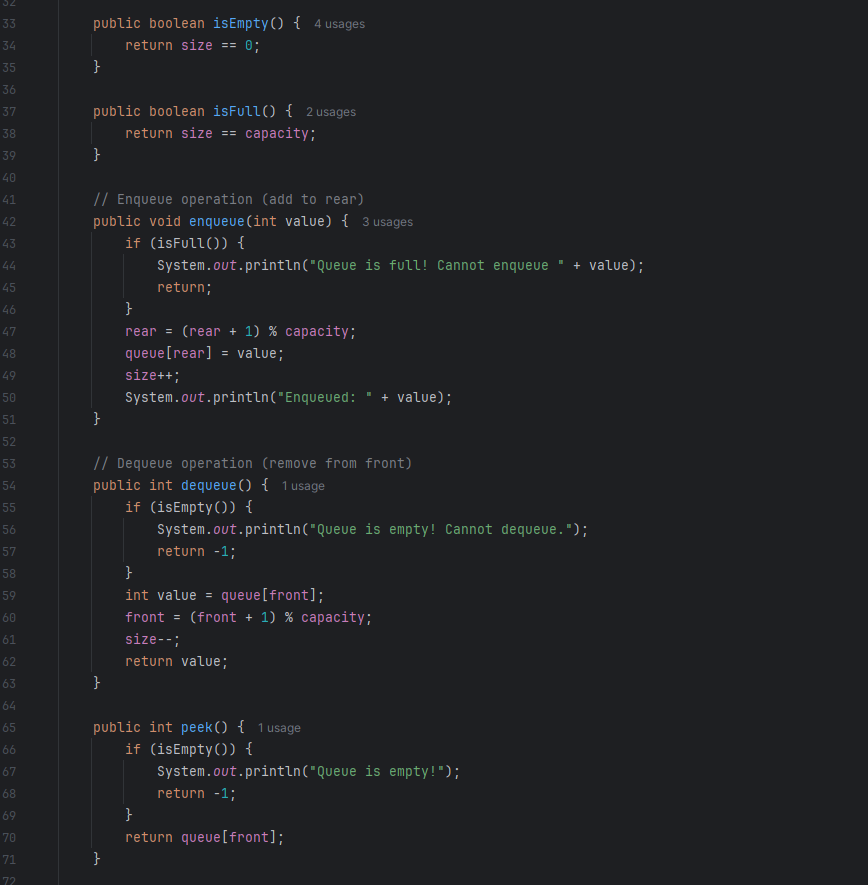
* **enqueue(item)**:  
   This operation **adds an element to the rear (end)** of the queue. It increases the size of the queue by one.  
   👉 *Example:* queue.add("Apple"); or queue.offer("Apple");
* **dequeue()**:  
   Removes and returns the **front element** of the queue. If the queue is empty, this operation may throw an exception (like NoSuchElementException).  
   👉 *Example:* queue.remove(); or queue.poll();
* **peek() / front()**:  
   Returns the **element at the front** of the queue **without removing** it. Useful for inspection.  
   👉 *Example:* queue.peek();
* **isEmpty()**:  
   Checks whether the queue contains **no elements**. Returns true if the queue is empty.  
   👉 *Example:* queue.isEmpty();
* **size()**:  
   Returns the **number of elements** currently present in the queue.  
   👉 *Example:* queue.size();

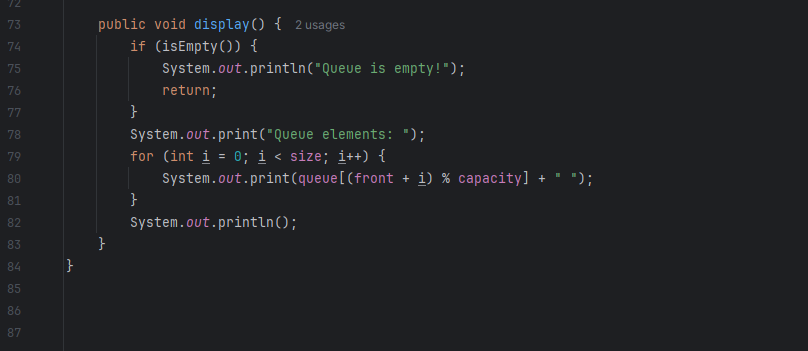
**Additional Notes:**

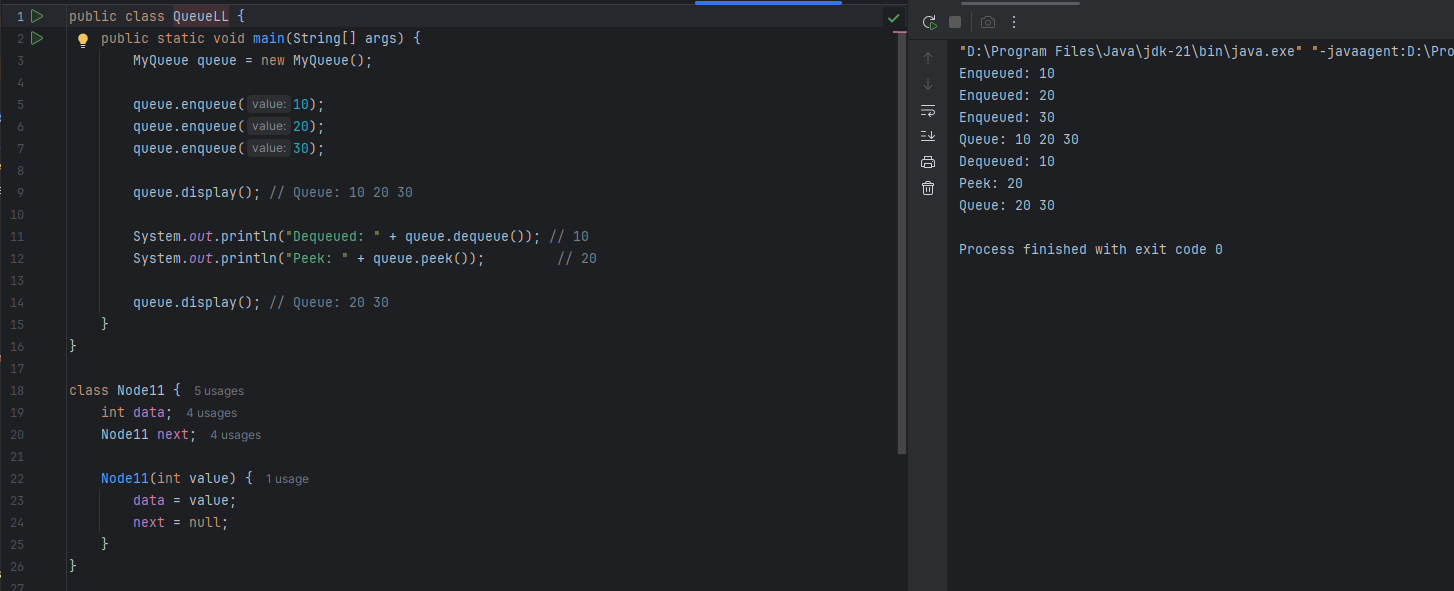
* In Java, the Queue interface is implemented by classes like LinkedList, PriorityQueue, and ArrayDeque.
* For basic FIFO behavior, LinkedList or ArrayDeque is commonly used.

***Task 11: Queue Implementation Using Array & Linked List***

***Using Array:***  






***Using LinkedList***  


******

