**Common Use Cases of Deque in Java - 2025**

A Deque, short for Double-Ended Queue, is a linear collection that supports element insertion and removal at both ends.

Example on Deque

public class DequeExample {

public static void main(String[] args) {

Deque<String> deque = new ArrayDeque<>();

// Adding elements to the Deque

deque.addFirst("Element 1");

deque.addLast("Element 2");

deque.addFirst("Element 3");

// Removing elements from the Deque

String firstElement = deque.removeFirst();

String lastElement = deque.removeLast();

// Accessing elements

String peekFirst = deque.peekFirst();

String peekLast = deque.peekLast();

System.out.println("First Element: " + firstElement);

System.out.println("Last Element: " + lastElement);

System.out.println("Peek First: " + peekFirst);

System.out.println("Peek Last: " + peekLast);

}

}

**Common Pitfalls**

* **Null Elements:** Deque implementations like *ArrayDeque* do not allow null elements. Attempting to add a null element will result in a *NullPointerException*.
* **Concurrent Modification:** Modifying a Deque while iterating over it can lead to *ConcurrentModificationException*. Use an iterator's *remove* method to safely remove elements during iteration.

**best practices to follow**

* **Choose the Right Implementation:** Use *ArrayDeque* for better performance in most cases, but consider *LinkedList* if you need frequent insertions and deletions from both ends.
* **Thread Safety:** Deque implementations are not thread-safe. Use *ConcurrentLinkedDeque* for concurrent access.
* **Exception Handling:** Always handle exceptions like *NoSuchElementException* when performing operations that might fail.

**Advanced Usage**

Deque can be used in more advanced scenarios, such as implementing a LRU (Least Recently Used) Cache. An LRU Cache evicts the least recently used items first, making it useful for memory management.

public class LRUCache {

private final int capacity;

private final Deque<Integer> deque;

private final Set<Integer> set;

public LRUCache(int capacity) {

this.capacity = capacity;

this.deque = new LinkedList<>();

this.set = new HashSet<>();

}

public void refer(int page) {

if (!set.*contains*(page)) {

if (deque.*size*() == capacity) {

int last = deque.*removeLast*();

set.*remove*(last);

}

} else {

deque.*remove*(page);

}

deque.*addFirst*(page);

set.*add*(page);

}

public void display() {

for (int page : deque) {

System.out.*print*(page + " ");

}

System.out.*println*();

}

public static void main(String[] args) {

LRUCache cache = new *LRUCache*(4);

cache.*refer*(1);

cache.*refer*(2);

cache.*refer*(3);

cache.*refer*(1);

cache.*refer*(4);

cache.*refer*(5);

cache.*display*();

}

}

Ref: <https://www.machinet.net/tutorial-eng/common-use-cases-of-deque-in-java-applications>