QUEUE VISUALIZATION WITH LINKED LIST IMPLEMENTATION

Presented by Aaron Loeb

ABSTRACT

This project visualizes a queue implemented with a linked list using the FIFO (First In, First Out) principle. Queues are linear structures where elements are added at the end and removed from the front. Linked lists offer efficient memory management and flexible insertions and deletions. The project graphically demonstrates queue operations and highlights the advantages of linked-list implementations.

THE TERM OF QUEUE

REAL-WORLD EXAMPLE

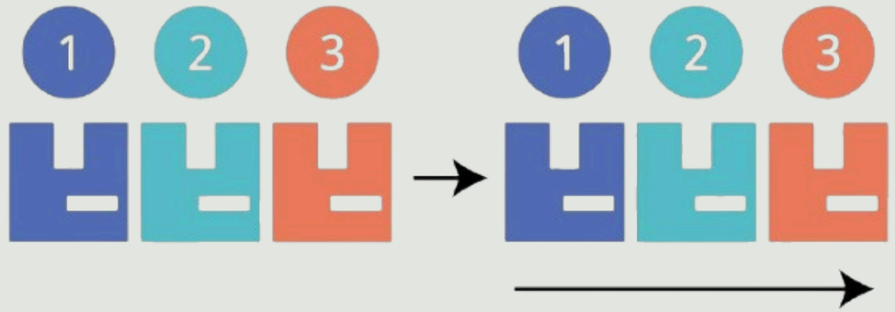
A SUPERMARKET CHECKOUT LINE IS A CLASSIC FIFO QUEUE: CUSTOMERS JOIN AT THE BACK, AND THE CASHIER SERVES WHOEVER'S BEEN WAITING THE LONGEST AT THE FRONT. THIS ENSURES EVERYONE IS TREATED FAIRLY AND IN THE EXACT ORDER THEY ARRIVED.

WHY FIFO WAS INTEGRATED INTO COMPUTERS:

COMPUTERS OFTEN RECEIVE STREAMS OF TASKS OR DATA THAT MUST BE HANDLED IN ARRIVAL ORDER TO AVOID LOSS, STARVATION, OR OUT-OF-ORDER PROCESSING. THE FIFO ABSTRACTION MIRRORS REAL-WORLD LINES, PROVIDING A SIMPLE YET POWERFUL WAY TO BUFFER AND SCHEDULE WORK.

CONTENT

- Simple Queue (FIFO)
- Double-Ended Queue (Deque)
- Circular Queue
- Priority Queue
- Multi-Level Feedback Queue
- Lock-Free Queue



FIFO
(First-in, First-out)

SIMPLE QUEUE (FIFO)

- First-In, First-Out
- Like a line in a supermarket
- All of the queues has same class methods (exept dubble ended queue

```
class Queue {
    boolean add(E e)        // insert element (throws if full)
    E element()             // get head (throws if empty)
    boolean offer(E e)      // insert element (returns false if full)
    E peek()                // get head (null if empty)
    E poll()                // remove and return head (null if empty)
    E remove()              // remove and return head (throws if empty)
}
```

```
// Extra deque methods (optional)
public void addFirst(E e) { list.addFirst(e); }
public void addLast(E e) { list.addLast(e); }
```

DETAILED EXPLANATION

Simple FIFO Queue

Behavior: First-in, first-out
Usage: Think of it like people standing in line. First person in is the first out.
Example Use: Print jobs, task scheduling

Double-Ended Queue

Behavior: Add or remove from both ends
Usage: More flexible than FIFO. Can act like a stack or a queue.
Example Use: Undo/redo stacks, sliding window algorithms

DETAILED EXPLANATION

Circular Queue

Behavior: FIFO, but when you reach the end, you loop to the beginning
Usage: Fixed size, good for saving space
Example Use: Network buffers, CPU time slicing

Priority Queue

Behavior: The element with highest priority comes out first, not the one that entered first
Usage: Needs a comparator to define priority
Example Use: Emergency rooms, Dijkstra's algorithm, job execution priority

DETAILED EXPLANATION

Multi-Level Feedback Queue

Behavior: Multiple queues for different priorities. Tasks move between queues based on their behavior

Usage: Used in Operating Systems to schedule processes fairly

Example Use: A process that runs too long gets "demoted" to a lower priority queue

Lock-Free Queue

Behavior: Like a normal queue but designed for multi-threaded environments

Usage: Safe and fast access by multiple threads without using locks (uses atomic operations)

Example Use: Concurrent programming, real-time systems, event queues

COMPARISON TABLE

Queue Type	Order	Ends Used	Fixed Size	Priority	Threadsafe	Used In
Simple FIFO Queue	FIFO	FIFOOne End Add, One End Remove	No	No	No	General task queues
Deque (Double Ended)	FIFO or LIFO	Add/Remove from Both Ends	No	No	No	Undo/redo, palindrome check
Circular Queue	FIFO or LIFO	Circular Buffer	Yes	No	No	Memory buffers, OS schedulers
Priority Queue	Priority	One End Add, Head is Highest Priority	No	Yes	No	Pathfinding, job schedulers
Multi-Level Feedback Queue	Priority + Aging	Multiple Queues	No	Yes	No	CPU scheduling (OS)
Lock-Free Queue	FIFO	One End Add, One End Remove	No	No	Yes	High-performance apps

CONCLUSION

- Queues are fundamental building blocks in computer science.
- Each type solves a specific kind of problem — from basic task ordering to complex OS scheduling.

- Knowing the right queue to use makes your system more efficient, scalable, and reliable.
- Whether you're building a simple app or a high-performance system —
- there’s always a queue involved behind the scenes.

Thank You

For your attention