# Fundamentals of Data Structure

Mahesh Shirole

VJTI, Mumbai-19

Slides are prepared from

1. Data Structures and Algorithms in Java, 6th edition, by M. T. Goodrich, R. Tamassia, and M. H. Goldwasser, Wiley, 2014
2.Data Structures and Algorithms in Java, by Robert Lafore, Second Edition, Sams Publishing

## Queues

- The word queue is British for line. In Britain, to “queue up” means to get in line
- A queue works like the line at the movies: The first person to join the rear of the line is the first person to reach the front of the line and buy a ticket
- Queues are used to model real-world situations such as people waiting in line at a bank, airplanes waiting to take off

# Queues in Computer

- Data packets waiting to be transmitted over the Internet
- Printer queue where print jobs wait for the printer to be available
- A queue also stores keystroke data as you type at the keyboard
- A Web server responding to requests
- Queues are used in the implementation of breadth-first search algorithm

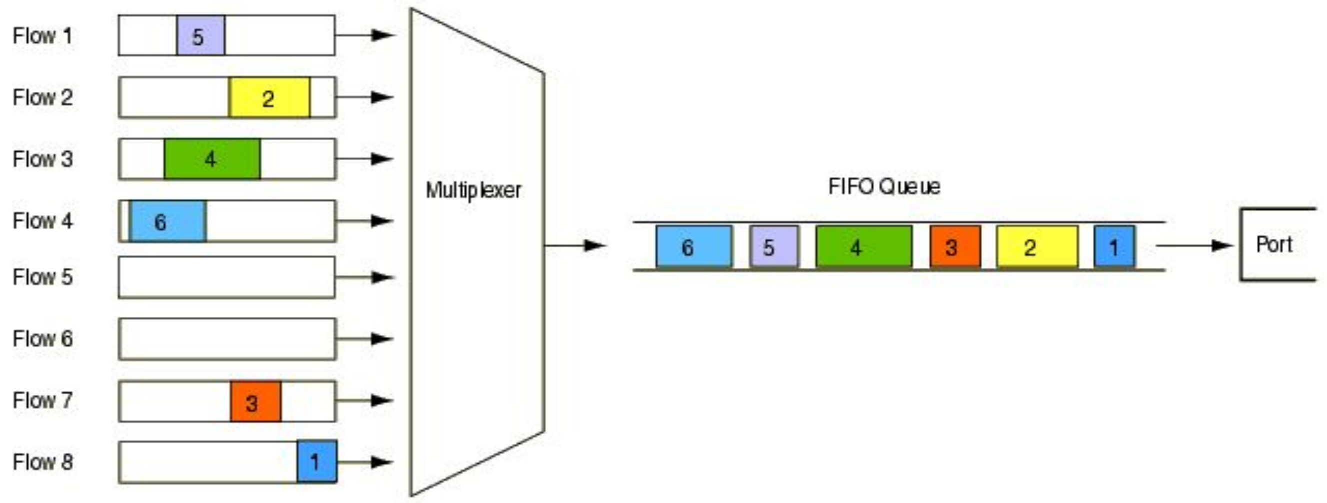
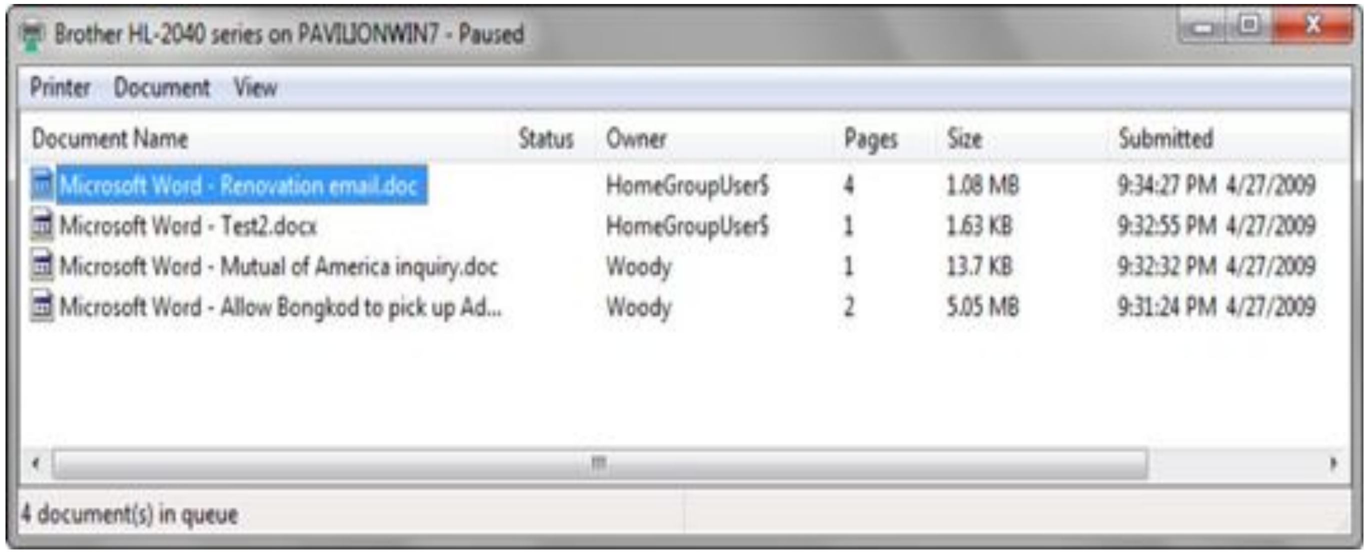


Figure 2.2.1



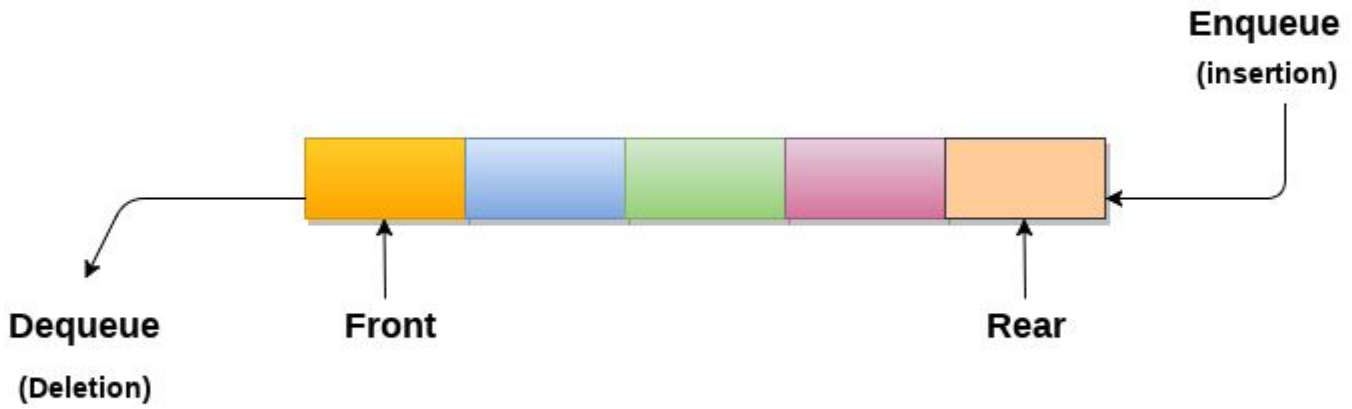
Brother HL-2040 series on PAVILIONWIN7 - Paused

| Document Name | Status | Owner | Pages | Size | Submitted |
|---|---|---|---|---|---|
| Microsoft Word - Renovation email.doc | | HomeGroupUser$ | 4 | 1.08 MB | 9:34:27 PM 4/27/2009 |
| Microsoft Word - Test2.docx | | HomeGroupUser$ | 1 | 1.63 KB | 9:32:55 PM 4/27/2009 |
| Microsoft Word - Mutual of America inquiry.doc | | Woody | 1 | 13.7 KB | 9:32:32 PM 4/27/2009 |
| Microsoft Word - Allow Bongkod to pick up Ad... | | Woody | 2 | 5.05 MB | 9:31:24 PM 4/27/2009 |

4 document(s) in queue

## The Queue Abstract Data Type

- Formally, the queue abstract data type defines a collection that keeps objects in a sequence, where element access and deletion are restricted to the first element in the queue, and element insertion is restricted to the back of the sequence.
- This restriction enforces the rule that items are inserted and deleted in a queue according to the **first-in first-out (FIFO) principle**
- **Insert** is also called *put or add or queue*, while **remove** may be called *delete or get or dequeue*



## The Queue Abstract Data Type

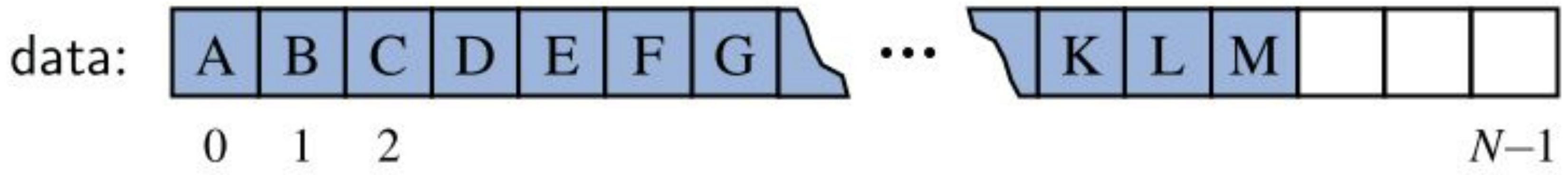
- The queue abstract data type (ADT) supports the following two *update methods*
  - **enqueue(e)**: Adds element ***e*** to the back of queue
  - **dequeue()**: Removes and returns the first element from the queue (or null if the queue is empty)
- The queue ADT also includes the following *accessor methods*
  - **first()**: Returns the first element of the queue, without removing it (or null if the queue is empty)
  - **size()**: Returns the number of elements in the queue
  - **isEmpty()**: Returns a boolean indicating whether the queue is empty

# Queue ADT - Java interface

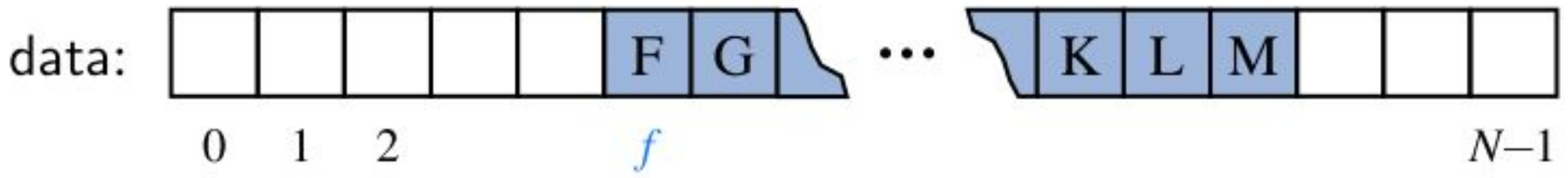
```
public interface Queue<E> {
    int size();   //Returns the number of elements in the queue.
    boolean isEmpty(); // Tests whether the queue is empty
    void enqueue(E e); // Inserts an element at the rear of the queue
    E first(); // Returns, but does not remove, the first element of the queue (null if empty)
    E dequeue(); // Removes and returns the first element of the queue (null if empty)
}
```

# Array-Based Queue Implementation

- Let’s assume that as elements are inserted into a queue, we store them in an array such that the first element is at index 0, the second element at index 1, and so on
- How  to implement the dequeue operation?
  - The element to be removed is stored at index 0 of the array
  - **Strategy 1**- shift all other elements of the queue one cell to the left [*O(n)* running time]

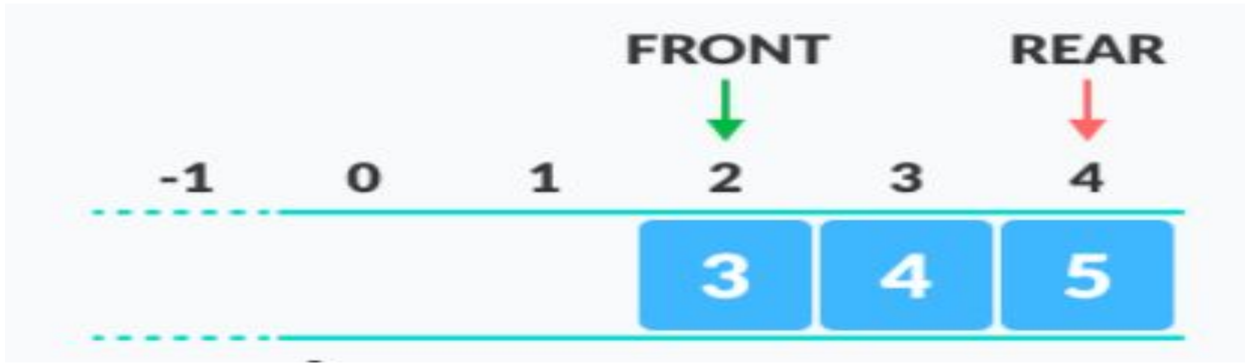


  - **Strategy 2** - maintain an explicit variables *front*  and *rear* to represent the index of the element that is currently at the front of the queue [*O(1)* running time]



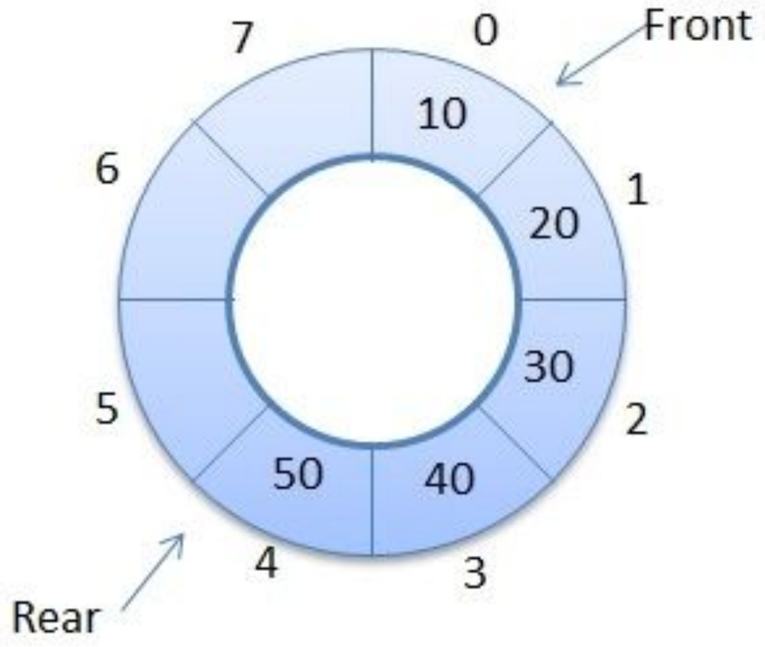
## Challenge with strategy 2

- With an array of capacity ***N***, we should be able to store up to ***N*** elements before reaching any exceptional case
- **FRONT** tracks the first element of the queue
- **REAR** tracks the last elements of the queue
- Initially, set value of FRONT and REAR to -1
- After a bit of enqueuing and dequeuing, the size of the queue reduces



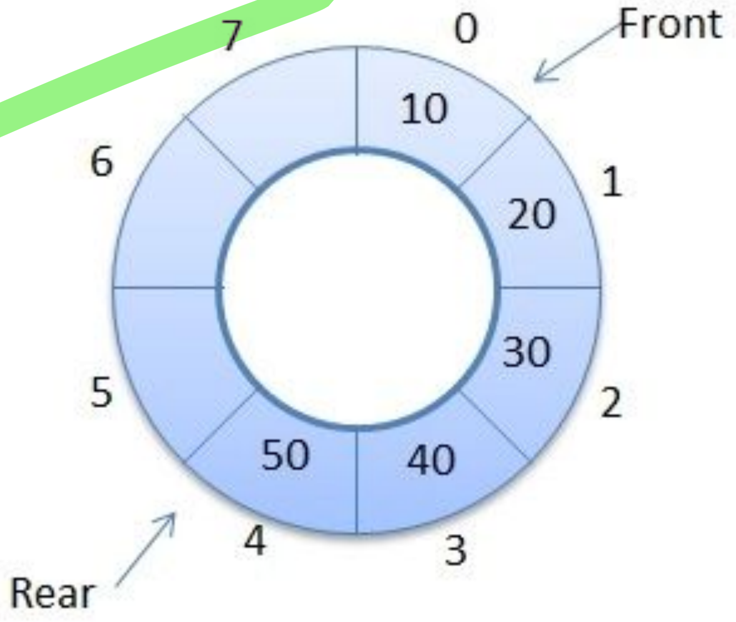
# A Circular Queue- Solution to strategy 2

- The circular queue is a *linear data structure*. It follows FIFO principle. In circular queue, the last node is connected back to the first node to make a circle
- Elements are ***added at the rear end*** and the elements are ***deleted at the front end*** of the queue
- Circular Queue works by the process of circular increment i.e., when we try to increment the pointer and we reach the end of the queue, we start from the beginning of the queue
- Implementing such a circular view is relatively easy with the ***modulo operator***, denoted with the symbol ***%*** in Java
- The modulo operator is ideal for treating an array circularly



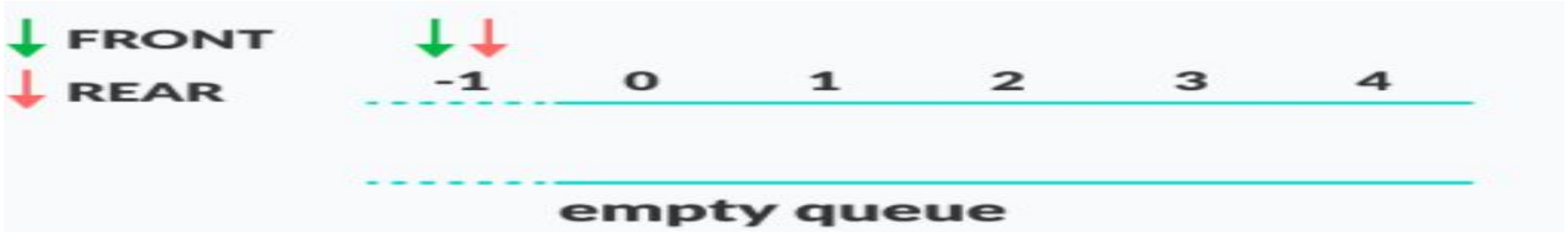
# A Java Queue Implementation

- Internally, the queue class maintains the following three instance variables:
  - **data** - a reference to the underlying array
  - **front** - index of the first element of the queue
  - **rear** - index of current (last) number of elements stored in the queue
- In given example
  - capacity of queue is 8 elements
  - front index is 0
  - rear index is 4



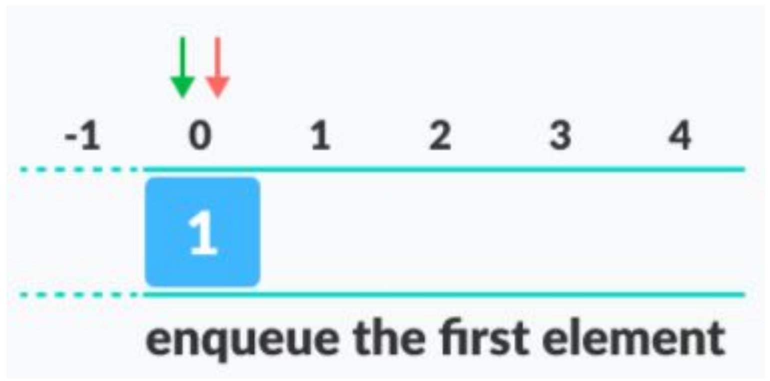
# A Java Queue Implementation

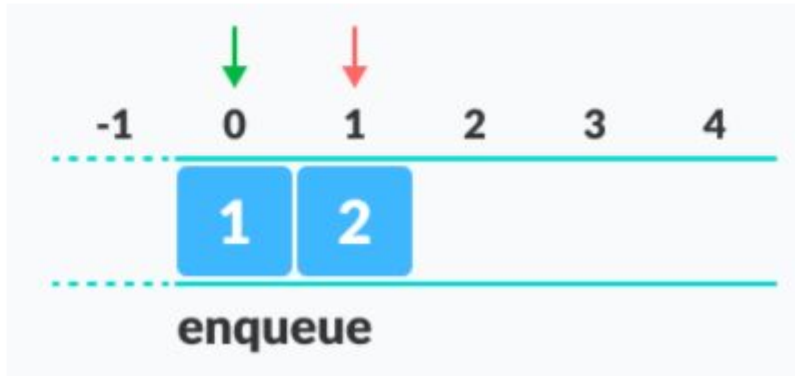
```
public class ArrayQueue<E> implements Queue<E> {
      private E[ ] data;                              // generic array used for storage
      private int front = 0;                          // index of the front element
      private int rear = -1;                      // index of the rear element
      private int size =0;                            // size of the quque
      public ArrayQueue(int capacity) {               //constructs queue with given capacity
            data = (E[ ]) new Object[capacity];
      }
      public int size() { //Returns the number of elements in the queue [-1 if queue is empty]
         return size; }
      public boolean isEmpty() {
           return (size == 0); }  //Tests whether the queue is empty
```

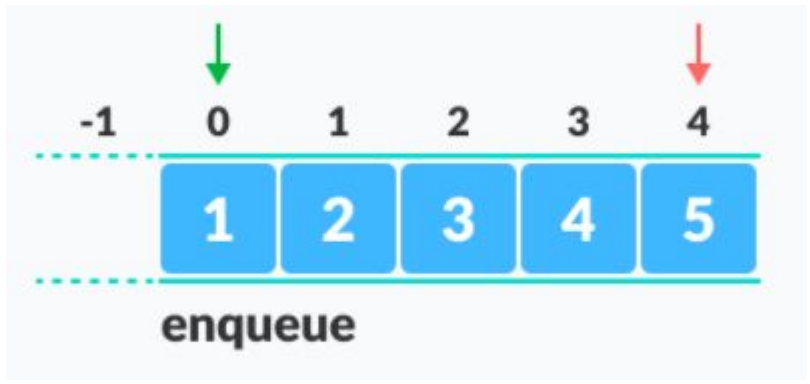


# A Java Queue Implementation

```
public void enqueue(E e) throws IllegalStateException {
   /*Inserts an element at the rear of the queue */
   if (size == data.length) throw new
   IllegalStateException("Queue is full");
   rear= (rear+1) % data.length;
   data[rear] = e;
   size++;
 }
   public E first() { /*Returns, but does not remove, the first
   element of the queue (null if empty)*/
        if (isEmpty()) return null;
        return data[front];
}
```

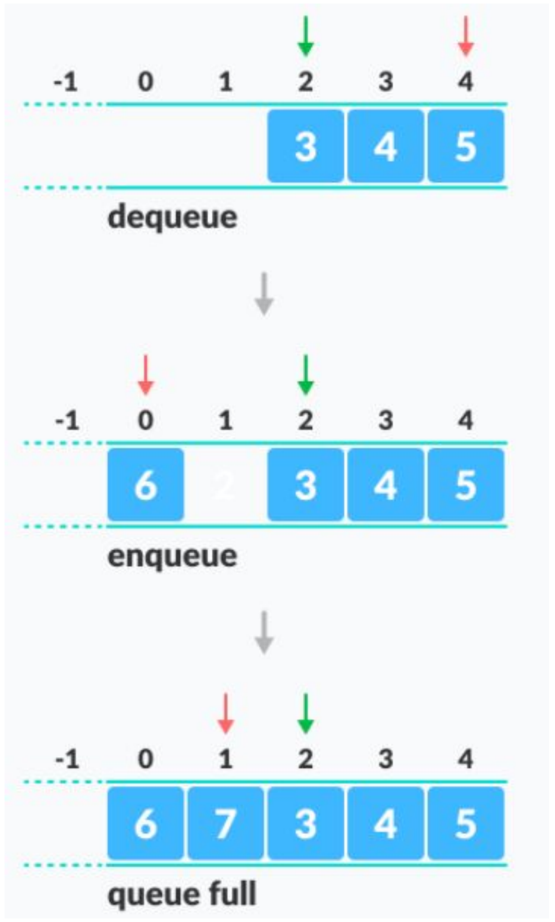






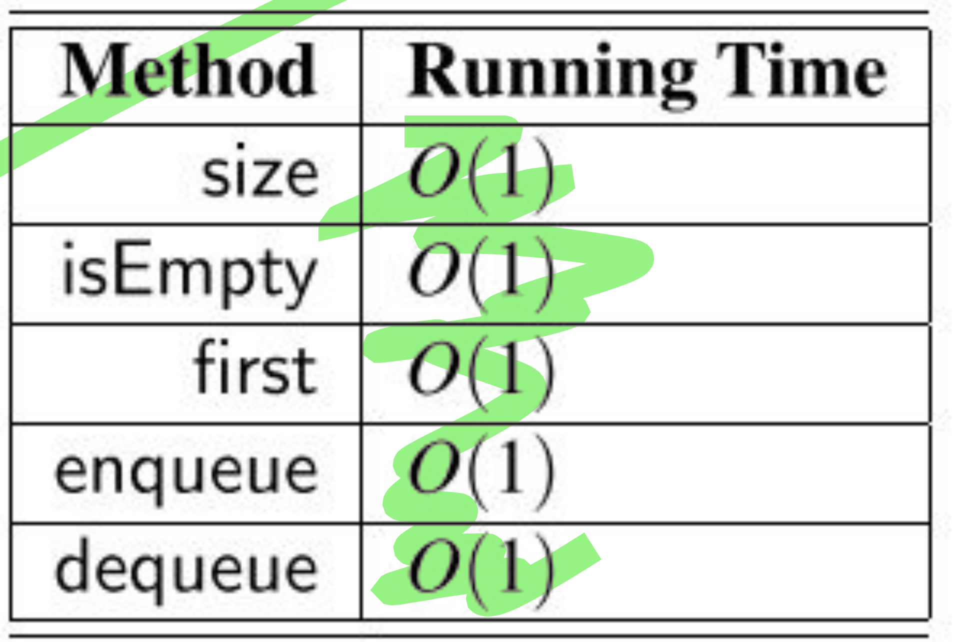
# A Java Queue Implementation

```
public E dequeue() {
   if (isEmpty()) return null;
   E answer = data[front];
   data[front] = null;
   front = (front + 1) % data.length;
   size−−;
   return answer;
}
```



## Analyzing the Efficiency of an Array-Based Queue

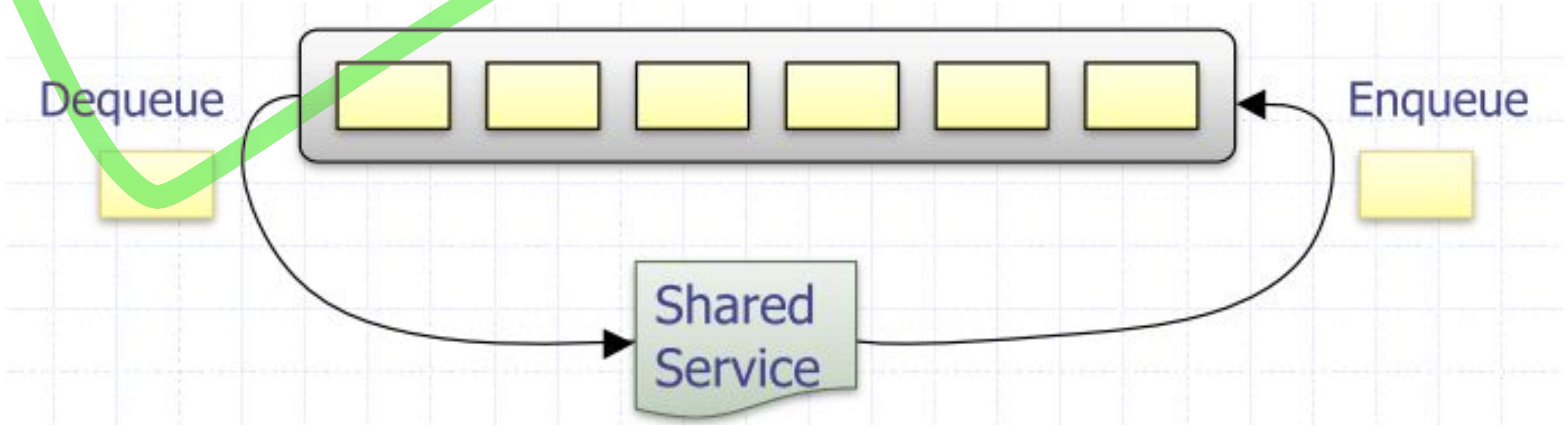
- Each of the queue methods in the array realization executes a constant number of statements involving arithmetic operations, comparisons, and assignments
- Thus, each method in this implementation runs in *O(1)* time

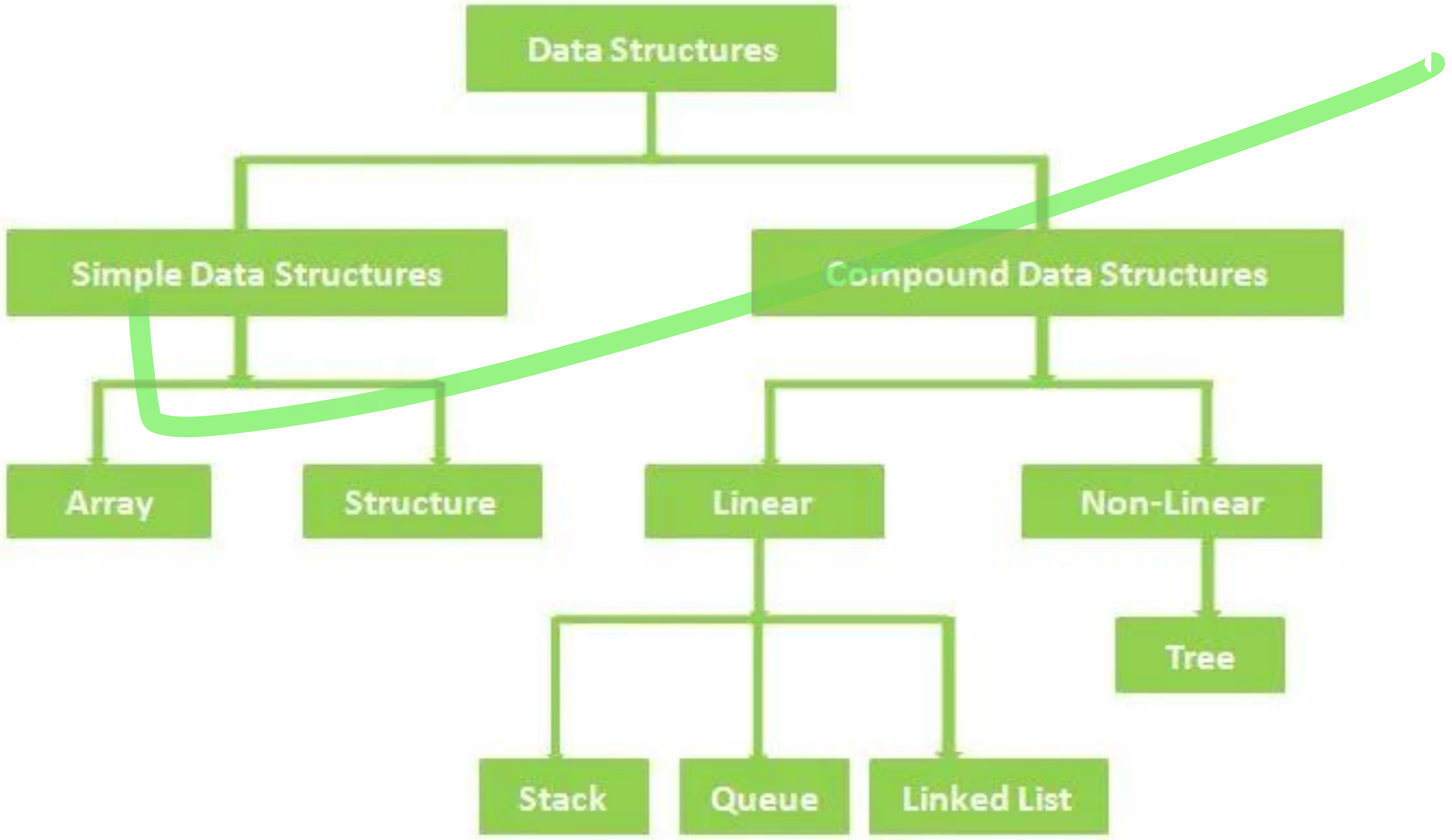


| Method | Running Time |
|---|---|
| size | $O(1)$ |
| isEmpty | $O(1)$ |
| first | $O(1)$ |
| enqueue | $O(1)$ |
| dequeue | $O(1)$ |

# Application: Round Robin Schedulers

- We can implement a round robin scheduler using a queue Q by repeatedly performing the following steps:
  - e = Q.dequeue()
  - Service element e
  - Q.enqueue(e)




Data Structures
Simple Data Structures
Compound Data Structures
Array
Structure
Linear
Non-Linear
Stack
Queue
Linked List
Tree