Queues
• First in First Out (FIFO)
  – Processing Data in Order
  – Time
  – “Waiting in a Line”

• Queue Operations
  – Enqueue: Add new element to the end of the queue
  – Dequeue: Remove the first element from the queue
  – Peek: Observe but not remove the first element in a queue
  – Print: Print all elements in a Queue
• Queue Implementations
  – Array
  – Linked List
• Two Major References
  – Head
  – Tail

**Array Queue**

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>-</td>
<td>5</td>
<td>8</td>
<td>7</td>
<td>6</td>
</tr>
</tbody>
</table>

**Linked List Queue**

5 → 8 → 7 → 6

Head: 5
Tail: 6
Queues

Queue <T> <<Interface>>
+enqueue(T): void
+dequeue(): T
+peek(): T
+print(): void

ArrayQueue <T>
-queue[]: T
+enqueue(T): void
+dequeue(): T
+peek(): T
+print(): void

LLQueue <T>
-head: ListNode
-tail: ListNode
+enqueue(T): void
+dequeue(): T
+peek(): T
+print(): void

“Implements”
• References
  – Head Index (First Element)
  – Tail Index (First NULL Element)
  – Items in the Queue start from the Head Index and end Tail Index - 1

• Moves forward in a Circular Way
  – “Next Index = (Index + 1) % Array.Length”
  – Avoids “Shifting”
Array Queue

- References
  - Head Index (First Element)
  - Tail Index (First NULL Element)
  - Items in the Queue start from the Head Index and end Tail Index - 1

- Moves forward in a Circular Way
  - “Next Index = (Index + 1)%Array.Length”
  - Avoids “Shifting”
• Enqueue
  – Add new element at the Tail Index
  – Advance Tail Index Circularly
- **Enqueue**
  - Add new element at the Tail Index
  - Advance Tail Index Circularly

---

**Array Queue**

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>-</td>
<td>5</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>4</td>
</tr>
</tbody>
</table>
• **Enqueue**
  – Add new element at the Tail Index
  – Advance Tail Index Circularly

---

**Array Queue**

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>-</td>
<td>5</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>4</td>
</tr>
</tbody>
</table>
• Dequeue
  – Save Reference to item at the Head Index
  – Advance Head Index Circularly
  – Return Saved Reference
• Dequeue
  – Save Reference to item at the Head Index
  – Advance Head Index Circularly
  – Return Saved Reference
• Dequeue
  – Save Reference to item at the Head Index
  – Advance Head Index Circularly
  – Return Saved Reference

<table>
<thead>
<tr>
<th>Index</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value</td>
<td>-</td>
<td>5</td>
<td>8</td>
<td>7</td>
<td>6</td>
<td>-</td>
</tr>
</tbody>
</table>

Return Value 5
• Enqueue
  – Create a new List Node with the Data
  – Add new element after the Tail Reference
- Enqueue
  - Create a new List Node with the Data
  - Add new element after the Tail Reference
• Enqueue
  – Create a new List Node with the Data
  – Add new element after the Tail Reference
• Enqueue
  – Create a new List Node with the Data
  – Add new element after the Tail Reference
• Dequeue
  – Save Reference to the Data in the Head
  – Move the Head forward
  – Return Saved Reference
• Dequeue
  – Save Reference to the Data in the Head
  – Move the Head forward
  – Return Saved Reference
Linked List Queue

- Dequeue
  - Save Reference to the Data in the Head
  - Move the Head forward
  - Return Saved Reference
• Dequeue
  – Save Reference to the Data in the Head
  – Move the Head forward
  – Return Saved Reference