A queue is consistent with the general concept of
- a waiting line to buy movie tickets
- a request to print a document
- crawling the web to retrieve documents

A queue is a linear collection whose elements are added on one end and removed from the other

Queue elements are processed in a first in, first out (FIFO) manner

Elements are removed from the queue in the same order in which they are placed on the queue
15.6 – Queues

Adding an element

Removing an element

rear of queue

front of queue

javafoundations.Queue

```java
package javafoundations;

public interface Queue<T> {
    // Adds element to rear of the queue
    public void enqueue(T element);
    // Removes and returns element at front of queue
    public T dequeue();
    // Returns reference to first element without removing
    public T first();
    // Returns true if queue contains no elements
    public boolean isEmpty();
    // Returns number of elements
    public int size();
    // Returns string representation
    public String toString();
}
```
Example using a Queue

```java
classes = new Queue<String>();
classes.enqueue ("RNaSG");
classes.enqueue ("Gl2HoSi1N");
classes.enqueue ("GSs");
classes.enqueue ("StCf0GST");
if (classes.first.equals("RNaSG"))
    classes.enqueue(classes.dequeue());
System.out.println("Today:"+ classes.dequeue());
System.out.println("Sunday:"+ classes.dequeue());
```

Radix Sort

- A **radix sort** is unusual because it does **not** involve **comparisons** between keys!

- The technique used in the radix sort is based on the **structure** of the sort key.

- Separate queues are created for each possible value of each digit or character of the sort key.

- **Radix** = The number of queues, or the number of possible values
  - if we were sorting strings made up of lowercase alphabetic characters, the radix would be 26, one for each possible character
  - if we were sorting decimal numbers, then the radix would be 10, one for each digit 0 to 9

- The radix sort makes a pass through the values for each position in the sort key.
Radix Sort (1\textsuperscript{st} pass)

<table>
<thead>
<tr>
<th>Digit</th>
<th>1s position</th>
<th>front of queue</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Original list: 442, 503, 312, 145, 250, 341, 325, 102, 420, 143

Radix Sort (2\textsuperscript{nd} pass begins)

<table>
<thead>
<tr>
<th>Digit</th>
<th>1s position</th>
<th>front of queue</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Original list
Radix Sort (2\textsuperscript{nd} pass results)

<table>
<thead>
<tr>
<th>Digit</th>
<th>10s position</th>
<th>front of queue</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>503 102</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>312</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>325 420</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>145 143 442 341</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>250</td>
</tr>
</tbody>
</table>

Radix Sort (3\textsuperscript{rd} pass results)

<table>
<thead>
<tr>
<th>Digit</th>
<th>100s position</th>
<th>front of queue</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>145 143 102</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>250</td>
</tr>
<tr>
<td>3</td>
<td>341 325 312</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>442 420</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>503</td>
</tr>
</tbody>
</table>

Visualization: https://visualgo.net/en/sorting
RadixSort.java

```java
int[] list = {...} // the input array
for (int digitVal = 0; digitVal <= 9; digitVal++) // create 10 Qs
digitQueues[digitVal] = new ArrayQueue<Integer>();

// sort the list that contains the numbers
for (int position=0; position <= 3; position++) { // 4-digit nums
    for (int scan = 0; scan < list.length; scan++) {
        temp = String.valueOf(list[scan]);
        digit = Character.digit(temp.charAt(3-position), 10);
        digitQueues[digit].enqueue(list[scan]);
    }
}

// gather numbers from the queues back into list
num = 0;
for (int digitVal = 0; digitVal <= 9; digitVal++) {
    while (!digitQueues[digitVal].isEmpty()) {
        list[num] = digitQueues[digitVal].dequeue().intValue();
        num++;
    }
}
}
```

Implementing Queues with Arrays

- Adds a new element to the rear of the queue, which is stored at the high end of the array

```
0 1 2 3 4 5 6 7 ... ...
```

```
A B C D E Left-shift to correct queue
```

```
Enqueue 'E'
count 4
```
package javafoundations;
import javafoundations.exceptions.*;

public class ArrayQueue<T> implements Queue<T> {
    private final int DEFAULT_CAPACITY = 10;
    private int count;
    private T[] queue;

    // 0-args constructor: Creates empty queue
    public ArrayQueue(){
        count = 0;
        queue = (T[]) (new Object[DEFAULT_CAPACITY]);
    }

    public boolean isEmpty() {
    }
    public int size() {
    }

    public T first() throws EmptyCollectionException {
    }

    public void enqueue(T element) {
    }
}
public T dequeue() throws EmptyCollectionException {
    if (count == 0) throw new EmptyCollectionException
             ("Dequeue failed. Queue empty");
    // shift elements to keep the front at element 0
    for (int index = 0; index < count; index++)
        queue[index] = queue[index + 1];
    queue[count] = null;
    return result;
}

// Left as programming project:
// public String toString() {}

Implementing Queues with Circular Arrays

- As elements are dequeued, the front of the queue will move further into the array
- As elements are enqueued, the rear of the queue will also move further into the array
- The challenge comes when the rear of the queue reaches the end of the array
- When this occurs, it “wraps around” to the front of the array
- Use two variables, front and rear, to represent the location where the first element is stored, and where the next available slot in the array is located (respectively)
15.9 – Implementing Queues with Circular Arrays

15.9 – The Changing State of a Circular Array Q
package javafoundations;

import javafoundations.exceptions.*;

public class CircularArrayQueue<T> implements Queue<T> {
    private final int DEFAULT_CAPACITY = 10;
    private int front, rear, count;
    private T[] queue;

    (more...)

    //-----------------------------------------
    //  Creates an empty queue using the default capacity.
    //-----------------------------------------
    public CircularArrayQueue() {
    }

    //-----------------------------------------
    //  Adds the specified element to the rear of this queue, expanding
    //  the capacity of the queue array if necessary.
    //-----------------------------------------
    public void enqueue (T element) {
        if (count == queue.length) expandCapacity();
    }

    (more...)
javafoundations.CircularArrayQueue

// Creates a new array to store the contents of this queue with twice the capacity of the old one.
public void expandCapacity() {
    T[] larger = (T[])(new Object[queue.length*2]);
    for (int index=0; index < count; index++)
        larger[index] = queue[(front+index) % queue.length];
    front = 0;
    rear = count;
    queue = larger;
}

// The following methods are left as Programming Projects.

public T dequeue () throws EmptyCollectionException { }
public T first () throws EmptyCollectionException { }
public int size() { }
public boolean isEmpty() { }
public String toString() { }

Distributed Denial of Service (DDOS)

- To connect two computers on the internet we use routers: simple devices that control a queue
  - They’re CircularArrayQueues without expandCapacity()

- A router connects to several channels, chooses the best to send a message it received

- When (portions of) a message arrives, the router stores it temporarily, decides which channel to use next, dequeues and sends the message.

- A DDOS is caused when attackers flood routers with messages for delivery!
15.10 – Implementing Queues with Links

```java
javafoundations.LinkedQueue

//********************************************************************
// LinkedQueue.java Java Foundations
//********************************************************************
package javafoundations;
import javafoundations.exceptions.*;
public class LinkedQueue<T> implements Queue<T>
{
    private int count;
    private LinearNode<T> front, rear;
    //---
    // Creates an empty queue.
    //---
    public LinkedQueue()
    {
        count = 0;
        front = rear = null;
    }
    (more...)
```
javafoundations.LinkedQueue

// Adds the specified element to the rear of this queue.
public void enqueue (T element) {
    LinearNode<T> node = new LinearNode<T>(element);
    if (count == 0)
        front = node;
    else
        rear.setNext(node);
    rear = node;
    count++;
}

// The following methods are left as Programming Projects.
// public T dequeue () throws EmptyCollectionException { }
// public T first () throws EmptyCollectionException { }
// public boolean isEmpty() { }
// public int size() { }
// public String toString() { }

Analysis of Stack and Queue Implementations

➤ All operations for a stack (push, pop, peek, etc.) are O( )

➤ Almost all operations for a queue are O( )

➤ The only exception is the dequeue operation for the ArrayQueue implementation – the shifting of elements makes it O( )

➤ The dequeue operation for the CircularArrayQueue is O( ) because of the ability to eliminate the shifting of elements

➤ Both stacks and queues can be implemented very efficiently