CSC 1052 – Algorithms & Data Structures II:

Queue Variations

Professor Henry Carter
Spring 2017

Recap

- Linked Lists allow for convenient implementation of queues
- Maintaining a head and tail simplifies implementation (but is not necessary)
- Array/Linked List tradeoffs between time and space are consistent with previous examples
  - Large queue relative to max size: array
  - Small or variable size queue: linked list

One size fits…most

- Queues are a widely applicable data structure
  - Real life
  - Computing
- Our queue implementation is somewhat restrictive
  - What added functions might we want?
- Today's lecture is all about variations and applications

Exceptions

- Overflow/underflow exceptions were a great teaching tool
  - Are they always necessary?
- If no exception is thrown, should the method behavior change?
  - Enqueue behavior
  - Dequeue behavior

Exception-less method headers

- `void enqueue(T element)`
- `boolean enqueue(T element)`
- `T dequeue()`
  - `if(isEmpty())`
    `return null;`

Peeking

- Our stack implementation offered a convenient way to check the next element without modifying the stack
  - Superfluous in the case of the stack
- Why would we need to peek at the first/last element of a queue?
- Can we emulate this behavior with only enqueue and dequeue?

GlassQueue Interface

```
//-----------------------------------------------------------------
// GlassQueueInterface.java      by Dale/Joyce/Weems                Chapter 4
//
// Interface for a class that implements a queue of T and includes
// operations for peeking at the front and rear elements of the queue.
//-----------------------------------------------------------------
package ch04.queues;

public interface GlassQueueInterface<T> extends QueueInterface<T>
{
  public T peekFront();
  // If the queue is empty, returns null.
  // Otherwise, returns the element at the front of this queue.

  public T peekRear();
  // If the queue is empty, returns null.
  // Otherwise, returns the element at the rear of this queue.
}
```

GlassQueue Implementation

```
package ch04.queues;
public class LinkedGlassQueue<T> extends LinkedQueue<T>
                                 implements GlassQueueInterface<T>
{
  public LinkedGlassQueue()
  {
    super();
  }

  public T peekFront()
  {
    if (isEmpty())
       return null;
    else
       return front.getInfo();
  }

  public T peekRear()
  {
    if (isEmpty())
      return null;
    else
      return rear.getInfo();
  }
}
```

Exercise

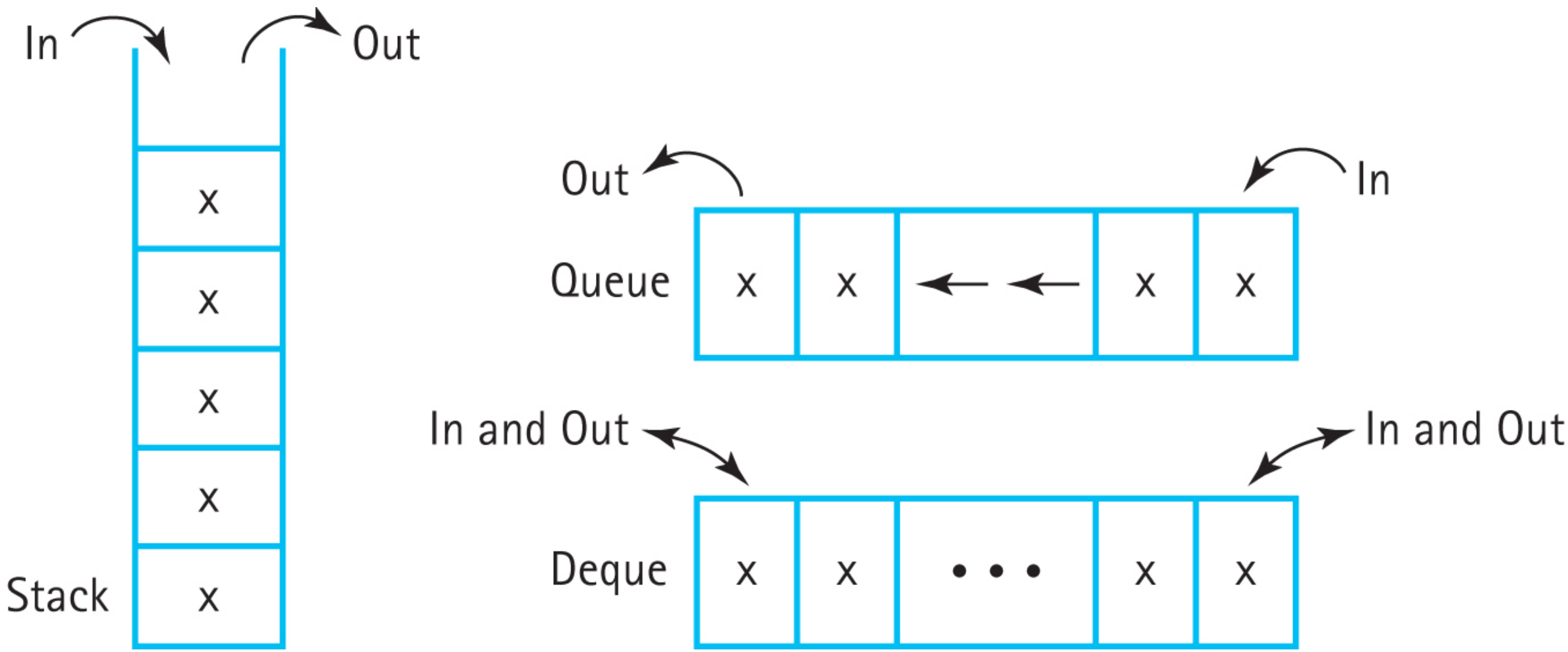
- Show what is written by the following segment:

```
elt1 = 1;elt2 = 0; elt3 = 4;
glassQ.enqueue(elt2); glassQ.enqueue(elt1);
glassQ.enqueue(elt3);
System.out.println(glassQ.peekFront());
System.out.println(glassQ.peekRear());

elt2 = glassQ.dequeue();
glassQ.enqueue(elt3*elt3);
elt1 = glassQ.peekRear(); glassQ.enqueue(elt1);

System.out.println(elt1 + " " + elt2 + " " + elt3);
while(!glassQ.isEmpty())
   System.out.println(glassQ.dequeue());
```

Double-ended Queues (Deque)

- One-stop data structure
  - Can be used as a queue or stack
- Enqueue and Dequeue split into two versions
  - Front and back
- Makes reverting enqueue or dequeue simple
- Pronounced “deck”

Deque

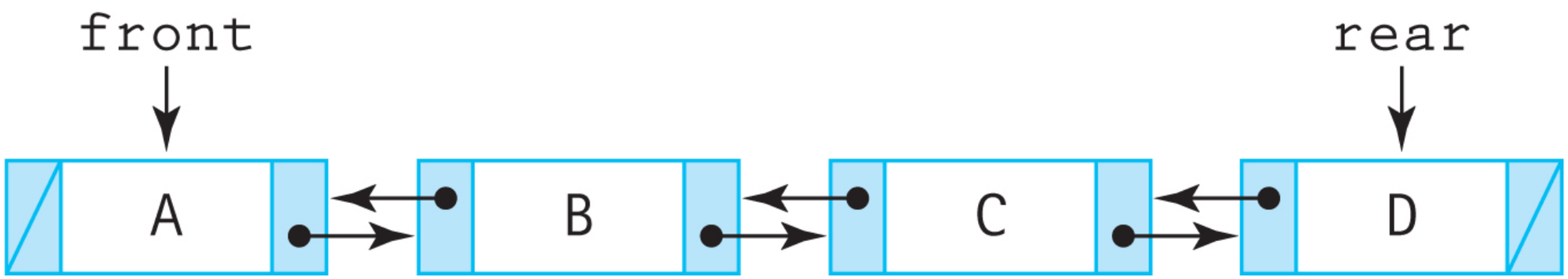

In
Out
x
x
x
x
x
Stack
Out
In
Queue
x
x
x
x
In and Out
In and Out
Deque
x
x
• • •
x
x

Implementation Hangups

- Consider a linked list implementation
- Enqueue/dequeue at the front?
- Enqueue at the back?
- Dequeue at the back?

Doubly Linked Lists

- Each node stores a link to the following node AND the previous node
- Allows traversing in both directions
- Does this solve the dequeue problem?



Dequeue from the rear code

```
public T dequeueRear() {
    if(isEmpty())
        return null;
    else{
        T element = rear.getInfo();
        if(rear.getPrev() == null)
            front = null; rear = null;
        else{
            rear.getPrev().setLink(null);
            rear = rear.getPrev();
        }
        numElements--;
        return element;
    }
}
```

Java Queues

- Java 5 introduced the Queue interface
- Matches most of our simple queue operations
  - Exception AND non-exception versions of enqueue and dequeue
  - Peek operations available
  - Priority queueing possible
- Java 6 added the deque for more featured implementations

Application: Wait Time

- Queueing is commonly used to provide first-come, first-served service
  - Real-world
  - Computer processes
  - Message delivery
  - Print jobs
- How do we design and optimize these systems?

Queueing Simulator 2017

- Generate some number of queues
- Randomly generate customers being served in the queues
- Calculate the waiting (queued) time for each customer and average
- The number of queues and expected customers can be modified to fit the application
and optimize for cost/queues

Customer Modeling

- What values are inherent to each customer?
  - Generated up front
- What values are inherent to the queue?
  - Generated during simulation

Customer Objects (abstract)

Customer	Arrival Time	Service Time
1	3	10
2	4	3
3	5	10
4	25	7

Simulator Properties

- Generating service times
  - Random values in a range
- Generating arrival times
  - Random inter-arrival times
- Distributions can be refined depending on the setting
- Number of customers
- Number of queues

Queue Properties

- Enqueue updates
  - Finish time
- Dequeue updates
  - Remove the lowest finish time customer
- Which queue will we need?

The Queues

Time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
Q0			cust1											cust3											
Q1				cust2																					

Calculated Results

Customer	Arrival Time	Service Time	Finish Time	Wait Time
1	3	10	13	0
2	4	3	7	0
3	5	10	23	8
4	25	7	32	0

Putting it together: example execution

```
Enter minimum interarrival time: 0
Enter maximum interarrival time: 10
Enter minimum service time: 5
Enter maximum service time: 20
Enter number of queues: 2
Enter number of customers: 2000
Average waiting time is 1185.632

Evaluate another simulation instance? (Y=Yes): y
Enter number of queues: 3
Enter number of customers: 2000
Average waiting time is 5.7245

Evaluate another simulation instance? (Y=Yes): n
Program completed.
```

Practice

Customer	Arrival Time	Service Time	Finish Time	Wait Time
1	0	10		
2	8	3		
3	8	10		
4	9	40		
5	20	15		
6	32	18		

- Finish the table for one queue. What is the average wait time?

Recap

- Queues allow for FIFO behavior in an ADT
- Applications may require atypical functionality or variations on the standard ADT
  - Exception-less queues
  - Glass queues
  - Deques
  - Doubly-linked lists
- Java provides both queue and deque interfaces and implementations
- Using data structures to simulate real-world applications is a valuable application

Next Time...

- Dale, Joyce, Weems Chapter 5.1-5.3
  - Remember, you need to read it BEFORE you come to class!
- Check the course webpage for practice problems
- Peer Tutors
  - http://www.csc.villanova.edu/help/