Waiting Lines: Queues, Deques and Priority Queues

Chapter 10

Pre-requisite Chapter 5: Stack

See http://www.guardian.co.uk/technology/video/2012/mar/16/new-ipad-queues-apple-video

Javadoc: java.util.[Queue | Deque | PriorityQueue]
http://docs.oracle.com/javase/7/docs/api/java/util/Queue.html
http://docs.oracle.com/javase/7/docs/api/java/util/Deque.html
http://docs.oracle.com/javase/7/docs/api/java/util/PriorityQueue.html
Motivation: Figure 10-1 Some waiting lines

Computer Science Motivation
- Servers queue requests
  - Web-server, print server, ...
  - Operating Systems
- Simulate physical service centers
  - Decide #runways
  - Decide #checkout lanes in retail stores, banks, ...
- Queueing theory models
Classifying Waiting Lines

• Who gets served next? Which entry is removed?
  ▪ **Queue**: First in, first out (FIFO), (remove oldest entry, i.e., from front)
  ▪ **Priority Queue**: remove highest priority entry

• Where may one add new entry?
  ▪ **Queue**: Rear only
  ▪ **Deque**: Rear or Front

• **Ex.** Classify following waiting lines into **Queue, Deque or Priority Queue**:
  ▪ Post-office
  ▪ Hospital
  ▪ Movie Theater
  ▪ Boarding an airplane
• A family of ADTs
• The ADT *Queue*
  ▪ Interface java.util.Queue
  ▪ Use case: waiting line, capital gain in stock sale
• The ADT *Deque*
  ▪ Use case: Capital Gain in a stock sale
  ▪ Interface java.util.Deque
  ▪ Class java.util.ArrayDeque
• The ADT *Priority Queue*
  ▪ Use case: Tracking Your Assignments
  ▪ Class java.util.PriorityQueue
Objectives

- **Describe operations and use cases**
- **ADT Queue**
  - Describe operations of ADT queue
  - Use queue to simulate waiting line
  - Use queue in program that organizes data in first-in, first-out manner
- **ADT Deque**
  - Describe operations of ADT deque
  - Use deque in program that organizes data chronologically and can operate on both oldest and newest entries
- **ADT Priority Queue**
  - Describe operations of ADT priority queue
  - Use priority queue in program that organizes data objects according to priorities
Abstract Data Type: Queue

• A collection of objects
  ▪ in chronological order (FIFO) and
  ▪ having the same data type

• Operations
  ▪ enqueue(newEntry)
  ▪ dequeue()
  ▪ getFront()
  ▪ isEmpty()
  ▪ clear()

• Interface for Queue, Listing 10-1
public interface QueueInterface<T> {
    public void enqueue(T newEntry); // Adds a new entry to the rear of this queue.
    public T dequeue(); // Remove & return front entry if non-empty
    public T getFront(); // Retrieves front entry if non-empty
    public boolean isEmpty(); // Return true if this queue has no entry
    public void clear(); // Removes all entries
}

• **Ex.** Test operations via tracing entries and their order in myQueue
  QueueInterface<String> myQueue = new LinkedQueue<String>();
  myQueue.enqueue(“Jim”);
  myQueue.enqueue(“Jess”);
  myQueue.enqueue(“Jill”);
  myQueue.enqueue(“Jane”);
  myQueue.enqueue(“Joe”);
  String front = myQueue.getFront();
  front = myQueue.dequeue();
  myQueue.enqueue(“Jerry”);
  front = myQueue.getFront();
  front = myQueue.dequeue();
Comarison with java.util.Queue

• Common Methods with textbook Queue interface
  - public boolean add(T newEntry)  // enqueue
  - public T remove()  // dequeue
  - public T peek()  // getFront
  - public boolean isEmpty()
  - public void clear()

• Other methods
  - public int size()
  - public boolean offer(T newEntry)  // similar to add, enqueue
  - public T poll()  // similar to remove, dequeue
  - public T element()  // similar to peek, getFront

• Q. Review pages 259-260 to compare and contrast the following:
  - add() vs. offer()
  - remove() vs. poll()
  - peek() vs. element()
Objectives

• Describe operations and use cases

• ADT Queue
  ▪ Describe operations of ADT queue
  ▪ Use queue to simulate waiting line
  ▪ Use queue in program that organizes data in first-in, first-out manner

• ADT Deque
  ▪ Describe operations of ADT deque
  ▪ Use deque in program that organizes data chronologically and can operate on both oldest and newest entries

• ADT Priority Queue
  ▪ Describe operations of ADT priority queue
  ▪ Use priority queue in program that organizes data objects according to priorities
ADT Priority Queue

- Contrast lines in banks and emergency rooms
  - Latter may not observe FIFO
  - Patients with most urgent needs may be seen first
- Organizes objects according to their priorities
- **Listing 10-5**, pp. 263

```java
public interface PriorityQueueInterface<T> extends Comparable<T> {
    public void add(T newEntry); // Adds a new entry. What is its priority?
    public T remove(); // Remove highest priority entry if non-empty
    public T peek(); // Retrieves highest priority entry if non-empty
    public int getSize(); // Return number of entries
    public boolean isEmpty(); // Return true if this priority queue is empty
    public void clear(); // Removes all entries
}
```
- The API looks same as that for Queue.
  - Q? Why does not add() ask for priority information?
  - Q? Which entry will be remove() return?
Queue vs. Priority Queue

- **Ex.** Determine values for strings a, b, c, and d after following code fragment:

```java
QueueInterface<String> myQueue = new LinkedQueue<String>();
myQueue.enqueue("Jim"); myQueue.enqueue("Jess");
myQueue.enqueue("Jill"); myQueue.enqueue("Jane");
myQueue.enqueue("Joe");
String a = myQueue.dequeue();
myQueue.enqueue("Jerry");
String b = myQueue.dequeue();

QueueInterface<String> pQ = new PriorityQueue<String>();
pQ.add("Jim"); pQ.add("Jess");
pQ.add("Jill"); pQ.add("Jane");
pQ.add("Joe");
String c = pQ.remove();
pQ.add("Jerry");
String d = PQ.remove();
```
Compare with java.util.PriorityQueue

• Common methods
  ▪ public PriorityQueue()
  ▪ public PriorityQueue(int initialCapacity)
  ▪ public boolean add(T newEntry)    // vs. void
  ▪ public T remove()
  ▪ public T peek()
  ▪ public int size()                // vs. getSize
  ▪ public boolean isEmpty()
  ▪ public void clear()

• Additional Methods
  ▪ public boolean offer(T newEntry)    // similar to add
  ▪ public T poll()
  ▪ public T element()               // similar to peek

• Q. Review pages 268-269 to compare and contrast the following:
  ▪ add() vs. offer()
  ▪ Peek() vs. element()
Objectives

• **Describe operations and use cases**

• **ADT Queue**
  - Describe operations of ADT queue
  - Use queue to simulate waiting line
  - Use queue in program that organizes data in first-in, first-out manner

• **ADT Deque**
  - Describe operations of ADT deque
  - Use deque in program that organizes data chronologically and can operate on both oldest and newest entries

• **ADT Priority Queue**
  - Describe operations of ADT priority queue
  - Use priority queue in program that organizes data objects according to priorities
ADT Deque

- Double ended queue: Add, remove, retrieve
  - At both front and back of a queue
- **Listing 10-4**, pp. 261

```java
public interface DequeInterface<T> {
    public void addToFront(T newEntry);   // Adds a new entry to front.
    public void addToBack(T newEntry);   // Adds a new entry to the rear
    public T removeFront();              // Remove & return front entry if non-empty
    public T removeBack();               // Remove & return rear entry if non-empty
    public T getFront();                 // Retrieves front entry if non-empty
    public T getBack();                  // Retrieves last entry if non-empty
    public boolean isEmpty();            // Return true if this stack is empty
    public void clear();                 // Removes all entries from this stack
}
```

**Figure 10-10** An instance d of a deque
FIGURE 10-11 A comparison of operations for a stack s, a queue q, and a deque d:
(a) add; (b) remove; (c) retrieve

The stack s, queue q, or deque d
Comparison with Java.util.Deque (pp. 263-264)

- Common with textbook
  - public void addFirst(T newEntry)  // addToFront
  - public void addLast(T newEntry)   // addToBack
  - public T removeFirst()           // removeFront
  - public T removeLast()            // removeBack
  - public T getFirst()              // getFront
  - public T getLast()               // getBack
  - public boolean isEmpty()         //
  - public void clear()

- Additional
  - public int size()
  - public boolean offerFirst(T newEntry) // Compare with addFirst, addToFront
  - public boolean offerLast(T newEntry)   //
  - public T pollFirst() // Compare w/ removeFirst, removeFront
  - public T pollLast()
  - public T peekFirst() // Compare with getFirst, getFront
  - Public T peekLast()
Java Class Library

- **Deque interface** extends **Queue interface**
  - Inherits `add`, `offer`, `remove`, `poll`, `element`, `peek`
  - Adds additional methods: `push`, `pop`

- Class **ArrayDeque** Implements **Deque interface**
  - has methods appropriate for `deque`, `queue`, and `stack`
  - Could be used for instances of any of these
  - Constructors
    - `public ArrayDeque()`
    - `public ArrayDeque(int initialCapacity)`
QueueInterface<String> myQueue = new LinkedQueue<String>();
myQueue.enqueue("Jim"); myQueue.enqueue("Jess");
myQueue.enqueue("Jill"); myQueue.enqueue("Jane");
String name = myQueue.dequeue(); myQueue.enqueue(name);
myQueue.enqueue(myQueue.getFront()); name = myQueue.dequeue();
PriorityQueueInterface<String> myPriorityQueue = new LinkedPriorityQueue<String>();
myPriorityQueue.add("Jane"); myPriorityQueue.add("Jim");
myPriorityQueue.add("Jill"); myPriorityQueue.add("Jane");
String name = myPriorityQueue.remove(); myPriorityQueue.add(name);
myPriorityQueue.add("Jess");
DequeInterface<String> myDeque = new LinkedDeque<String>();
myDeque.addToFront("Jim"); myDeque.addToBack("Jess");
myDeque.addToFront("Jill"); myDeque.addToBack("Jane");
String name = myDeque.getFront(); myDeque.addToBack(name);
myDeque.removeFront(); myDeque.addToFront(myDeque.removeBack());

Ex. 10-1,3,4. What strings are at front and back after above statements in
• myQueue
• myPriorityQue
• myDeque
Review Ex.: Queue, Priority Queue, Deque

1. Choose an ADT (e.g., Queue, Deque, PriorityQueue) for each of the following:
   (a.) Manage home-works from 4 courses in Spring 2012 with varying due dates
   (b.) Simulate a waiting line at customer service desk in a retail store
   (c.) Simulate a waiting line in a post office
   (d.) Computing taxes for Stock sales using
       (d1) first bought first sold rule
       (d2) sell most underwater (least appreciated) stock first to reduce taxes
       (d3) sell either oldest or newest purchases

2. To simulate people waiting in a line, which data structure would you NOT use?
   (a.) Queue    (b.) Stack    (c.) Priority Queue    (d.) Deque

3. If the characters ‘B', ‘D', ‘C', 'A' are placed in a queue (in that order), and then removed one at a
time, in what order will they be removed?
   (a.) ABCD    (b.) ABDC    (c.) BDCA    (d.) DCBA

4. If the characters ‘B', ‘D', ‘C', 'A' are placed in a priority queue (in that order), and then removed
one at a time, in what order will they be removed?
   (a.) ABCD    (b.) ABDC    (c.) BDAC    (d.) DCBA
Objectives

- **Describe operations and use cases**
- **ADT Queue**
  - Describe operations of ADT queue
  - Use queue to simulate waiting line
  - Use queue in program that organizes data in first-in, first-out manner
- **ADT Deque**
  - Describe operations of ADT deque
  - Use deque in program that organizes data chronologically and can operate on both oldest and newest entries
- **ADT Priority Queue**
  - Describe operations of ADT priority queue
  - Use priority queue in program that organizes data objects according to priorities
Use Case: Simulating a Waiting Line

Figure 10-3 A line, or queue, of people
<table>
<thead>
<tr>
<th>Responsibilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simulate customers entering and leaving a waiting line</td>
</tr>
<tr>
<td>Display number served, total wait time, average wait time, and number left in line</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Collaborations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Customer</td>
</tr>
</tbody>
</table>

Figure 10-4 A CRC card for the class `WaitLine`
Figure 10-5 A diagram of the classes `WaitLine` and `Customer`
Algorithm simulate(duration, arrivalProbability, maxTransactionTime)
transactionTimeLeft = 0
for (clock = 0; clock < duration; clock++)
{
    if (a new customer arrives)
    {
        numberOfArrivals++
        transactionTime = a random time that does not exceed maxTransactionTime
        nextArrival = a new customer containing clock, transactionTime, and
                    a customer number that is numberOfArrivals
        line.enqueue(nextArrival)
    }

    if (transactionTimeLeft > 0) // if present customer is still being served
        transactionTimeLeft--
    else if (!line.isEmpty())
    {
        nextCustomer = line.dequeue()
        transactionTimeLeft = nextCustomer.getTransactionTime() - 1
        timeWaited = clock - nextCustomer.getArrivalTime()
        totalTimeWaited = totalTimeWaited + timeWaited
        numberServed++
    }
}
Figure 10-6 A simulated waiting line
Figure 10-6 A simulated waiting line
Class `WaitLine`

- Implementation of class `WaitLine`

Listing 10-2

- Statements
  - Generate line for 20 minutes
  - 50 percent arrival probability
  - 5-minute maximum transaction time.

- View sample output
Computing Capital Gain for Stock Sale

- Buying $n$ shares at $d$
  - Then selling – gain or lose money
- We seek a way to
  - Record your investment transactions chronologically
  - Compute capital gain of any stock sale.
- We design a class, \texttt{StockPurchase}
Figure 10-7 A CRC card for the class **StockLedger**

<table>
<thead>
<tr>
<th>Responsibilities</th>
</tr>
</thead>
<tbody>
<tr>
<td>Record the shares of a stock purchased, in chronological order</td>
</tr>
<tr>
<td>Remove the shares of a stock sold, beginning with the ones held the longest</td>
</tr>
<tr>
<td>Compute the capital gain (loss) on shares of a stock sold</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Collaborations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Share of stock</td>
</tr>
</tbody>
</table>
Figure 10-8 A diagram of the classes **StockLedger** and **StockPurchase**

**StockLedger**

- ledger—a collection of shares owned, in order of their purchase
- buy(sharesBought, pricePerShare)
- sell(sharesSold, pricePerShare)

**StockPurchase**

- cost—cost of one share
- getCostPerShare()
Objectives

- Describe operations and use cases
- **ADT Queue**
  - Describe operations of ADT queue
  - Use queue to simulate waiting line
  - Use queue in program that organizes data in first-in, first-out manner
- **ADT Deque**
  - Describe operations of ADT deque
  - Use deque in program that organizes data chronologically and can operate on both oldest and newest entries
- **ADT Priority Queue**
  - Describe operations of ADT priority queue
  - Use priority queue in program that organizes data objects according to priorities
Computing Capital Gain for Stock Sale

• View class implementation

**Listing 10-3**

Figure 10-9 A queue of (a) individual shares of stock; (b) grouped shares
Computing Capital Gain for Stock Sale

• Revise implementation of class StockLedger
  ▪ Data field ledger now an instance of deque
  ▪ Note method buy

```java
public void buy(int sharesBought, double pricePerShare) {
    StockPurchase purchase = new StockPurchase(sharesBought, pricePerShare);
    ledger.addToBack(purchase);
} // end buy
```

• View method sell, Listing 10-A, pp. 263
Objectives

• Describe operations and use cases
  - ADT Queue
    ▪ Describe operations of ADT queue
    ▪ Use queue to simulate waiting line
    ▪ Use queue in program that organizes data in first-in, first-out manner
  - ADT Deque
    ▪ Describe operations of ADT deque
    ▪ Use deque in program that organizes data chronologically and can operate on both oldest and newest entries
  - ADT Priority Queue
    ▪ Describe operations of ADT priority queue
    ▪ Use priority queue in program that organizes data objects according to priorities
Problem: Tracking Your Assignments

• Consider tasks assigned with due dates
• We use a priority queue to organize in due date order

<table>
<thead>
<tr>
<th>Assignment</th>
</tr>
</thead>
<tbody>
<tr>
<td>course—the course code</td>
</tr>
<tr>
<td>task—a description of the assignment</td>
</tr>
<tr>
<td>date—the due date</td>
</tr>
<tr>
<td>getCourseCode()</td>
</tr>
<tr>
<td>getTask()</td>
</tr>
<tr>
<td>getDueDate()</td>
</tr>
<tr>
<td>compareTo()</td>
</tr>
</tbody>
</table>

Figure 10-12 A diagram of the class Assignment
Tracking Your Assignments

• Note implementation of class AssignmentLog, **Listing 10-6**, pp. 267

<table>
<thead>
<tr>
<th>AssignmentLog</th>
</tr>
</thead>
<tbody>
<tr>
<td>log—a priority queue of assignments</td>
</tr>
<tr>
<td>addProject(newAssignment)</td>
</tr>
<tr>
<td>addProject(courseCode, task, dueDate)</td>
</tr>
<tr>
<td>getNextProject()</td>
</tr>
<tr>
<td>removeNextProject()</td>
</tr>
</tbody>
</table>

Figure 10-13 A diagram of the class AssignmentLog
End

Chapter 10