Queues, Deques and Priority Queues

Chapter 10
Contents

• The ADT Queue
  ▪ A Problem Solved: Simulating a Waiting Line
  ▪ A Problem Solved: Computing the Capital Gain in a Sale of Stock
  ▪ Java Class Library: The Interface Queue
Contents

• The ADT Deque
  ▪ A Problem Solved: Computing the Capital Gain in a Sale of Stock
  ▪ Java Class Library: The Interface Deque
  ▪ Java Class Library: The Class ArrayDeque

• The ADT Priority Queue
  ▪ A Problem Solved: Tracking Your Assignments
  ▪ Java Class Library: The Class PriorityQueue
Objectives

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

• Use deque in program that organizes data chronologically and can operate on both oldest and newest entries
• Describe operations of ADT priority queue
• Use priority queue in program that organizes data objects according to priorities
Queue

- Another name for a waiting line
  - Used within operating systems
  - Simulate real world events
  - First in, first out (FIFO)
- Consider double ended queue (deque)
  - Possible to manipulate both ends of queue
- When multiple queues exist, priority can be established
Figure 10-1 Some everyday queues
Abstract Data Type: Queue

- A collection of objects in chronological order and having the same data type
- Operations
  - enqueue(newEntry)
  - dequeue()
  - getFront()
  - isEmpty()
  - clear()
- Interface for Queue, **Listing 10-1**
Figure 10-2 A queue of strings after (a) enqueue adds *Jim*; (b) enqueue adds *Jess*; (c) enqueue adds *Jill*; (d) enqueue adds *Jane*;
Figure 10-2 A queue of strings after (e) enqueue adds Joe; (f) dequeue retrieves and removes Jim; (g) enqueue adds Jerry; (h) dequeue retrieves and removes Jess;
Question 1 After the following nine statements execute, what string is at the front of the queue and what string is at the back?

```java
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();
```
1. *Jill* is at the front, *Jess* is at the back.
Simulating a Waiting Line

Figure 10-3 A line, or queue, of people
### WaitLine

<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**

**WaitLine**
- line—a queue of customers
- numberOfArrivals—number of customers
- numberOfServed—number of customers actually served
- totalTimeWaited—total time customers have waited

```java
simulate(duration, arrivalProbability, maxTransactionTime)
displayResults()
```

**Customer**
- arrivalTime
- transactionTime
- customerNumber

```java
getArrivalTime()
getTransactionTime()
getCustomerNumber()
```
Algorithm for simulate

```java
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++
    }
}
```
Transaction time left: 5
Time: 0
Wait: 0

Customer 1 enters line with a 5-minute transaction.
Customer 1 begins service after waiting 0 minutes.

Transaction time left: 4
Time: 1

Customer 1 continues to be served.

Transaction time left: 3
Time: 2

Customer 1 continues to be served.
Customer 2 enters line with a 3-minute transaction.

Transaction time left: 2
Time: 3

Customer 1 continues to be served.

Transaction time left: 1
Time: 4

Customer 1 continues to be served.
Customer 3 enters line with a 1-minute transaction.

Figure 10-6 A simulated waiting line
Figure 10-6 A simulated waiting line
Question 2 Consider the simulation begun in Figure 10-6.

a. At what time does Customer 4 finish and depart?

b. How long does Customer 5 wait before beginning the transaction?
2. a. 11.
b. 4.
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**

```java
WaitLine customerLine = new WaitLine();
customerLine.simulate(20, 0.5, 5);
customerLine.displayResults();
```
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}
### 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 10-7 A CRC card for the class **StockLedger**
**Figure 10-8** A diagram of the classes **StockLedger** and **StockPurchase**
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](image)

Copyright ©2012 by Pearson Education, Inc. All rights reserved
Java Class Library

• Interface Queue
  - public boolean add(T newEntry)
  - public boolean offer(T newEntry)
  - public T remove()
  - public T poll()
  - public T element()
  - public T peek()
  - public boolean isEmpty()
  - public void clear()
  - public int size()
ADT Deque

• Need for an ADT which offers
  ▪ Add, remove, retrieve
  ▪ At both front and back of a queue

• Double ended queue
  ▪ Called a *deque*
  ▪ Pronounced “deck”

• Actually behaves more like a double ended stack
ADT Deque

- Note deque interface,
  **Listing 10-4**

---

**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
Question 3 After the following nine statements execute, what string is at the front of the deque and what string is at the back?

```java
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.addToListFront(myDeque.removeBack());
```
3. *Jill* is at the front, *Jane* is at the back.
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](#)
Java Class Library

- Interface Deque
  - public void addFirst(T newEntry)
  - public boolean offerFirst(T newEntry)
  - public void addLast(T newEntry)
  - public boolean offerLast(T newEntry)
  - public T removeFirst()
  - public T pollFirst()
  - public T removeLast()
  - public T pollLast()
Java Class Library

• **Interface Deque**
  - `public T getFirst()`
  - `public T peekFirst()`
  - `public T getLast()`
  - `Public T peekLast()`
  - `public boolean isEmpty()`
  - `public void clear()`
  - `public int size()`
Java Class Library

• Deque extends Queue
• Thus inherits
  ▪ add, offer, remove, poll, element, peek
• Adds additional methods
  ▪ push, pop
Java Class Library

• Class `ArrayDeque`
  ▪ Implements `Deque`

• Note – has methods appropriate for `deque`, `queue`, and `stack`
  ▪ Could be used for instances of any of these

• Constructors
  ▪ `public ArrayDeque()`
  ▪ `public ArrayDeque(int initialCapacity)`
ADT Priority Queue

• Contrast bank queue and emergency room queue(s)
• ADT priority queue organizes objects according to their priorities
• Note interface, Listing 10-5
Question 4 After the following statements execute, what string is at the front of the priority queue and what string is at the back?

```java
PriorityQueueInterface<String> myPriorityQueue = new LinkedPriorityQueue<String>();
myPriorityQueue.add("Jane");
myPriorityQueue.add("Jim");
myPriorityQueue.add("Jill");
String name = myPriorityQueue.remove();
myPriorityQueue.add(name);
myPriorityQueue.add("Jess");
```
4. *Jane* is at the front, *Jim* is at the back.
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>
</tbody>
</table>

getCourseCode()  
getTask()  
getDueDate()  
compareTo()

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

- Note implementation of class `AssignmentLog`, **Listing 10-6**

<table>
<thead>
<tr>
<th>AssignmentLog</th>
</tr>
</thead>
<tbody>
<tr>
<td>log—a priority queue of assignments</td>
</tr>
</tbody>
</table>

addProject(newAssignment)
addProject(courseCode, task, dueDate)
getNextProject()
removeNextProject()

Figure 10-13 A diagram of the class `AssignmentLog`
Class `PriorityQueue` constructors and methods

- `public PriorityQueue()`
- `public PriorityQueue(int initialCapacity)`
- `public boolean add(T newEntry)`
- `public boolean offer(T newEntry)`
- `public T remove()`
- `public T poll()`
Java Class Library

• Class `PriorityQueue` methods, ctd.
  ▪ `public T element()`
  ▪ `public T peek()`
  ▪ `public boolean isEmpty()`
  ▪ `public void clear()`
  ▪ `public int size()`
End

Chapter 10