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

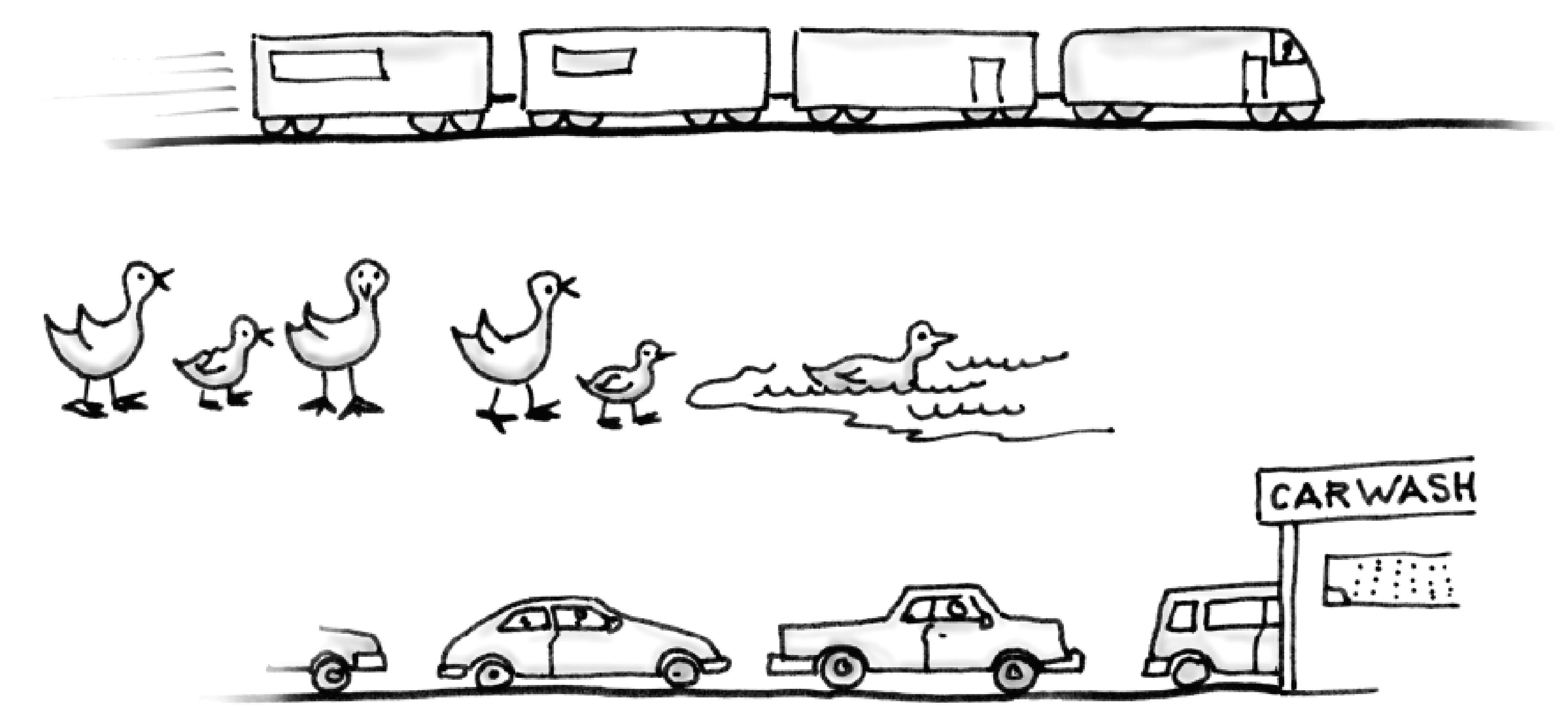


Figure 10-1 Some everyday queues

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

Note: Code listing files must be in same folder as PowerPoint files for links to work

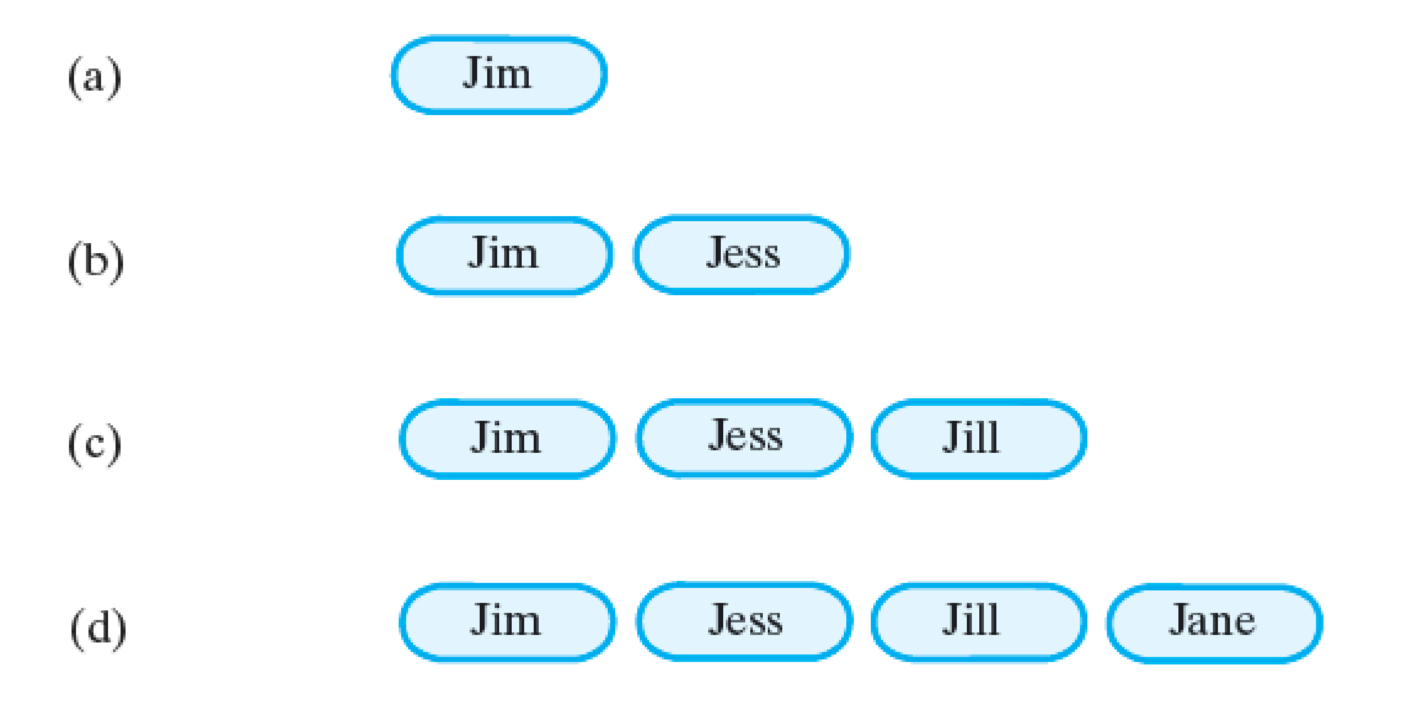


Figure 10-2 A queue of strings after (a) enqueue adds *Jim*; (b) enqueue adds *Jess*; (c) enqueue adds *Jill*; (d) enqueue adds *Jane*;

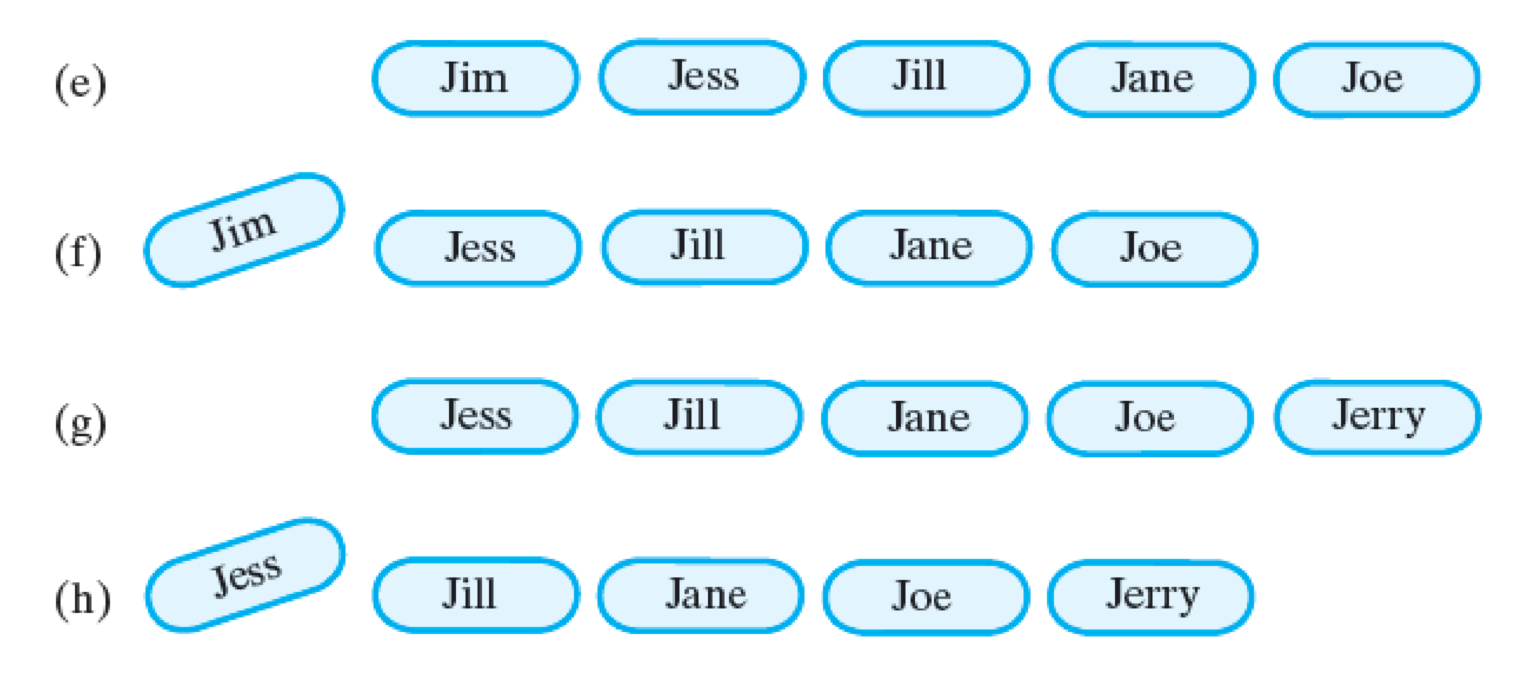


Figure 10-2 A queue of strings after (e) enqueue adds *Joe*; (f) dequeue retrieves and removes *Jim*; (g) enqueue adds *Jerry*; (h) dequeue retrieves and removes *Jess*;

Question 1 After the following nine statements execute, what string is at the front of the queue and what string is at the back?

```
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

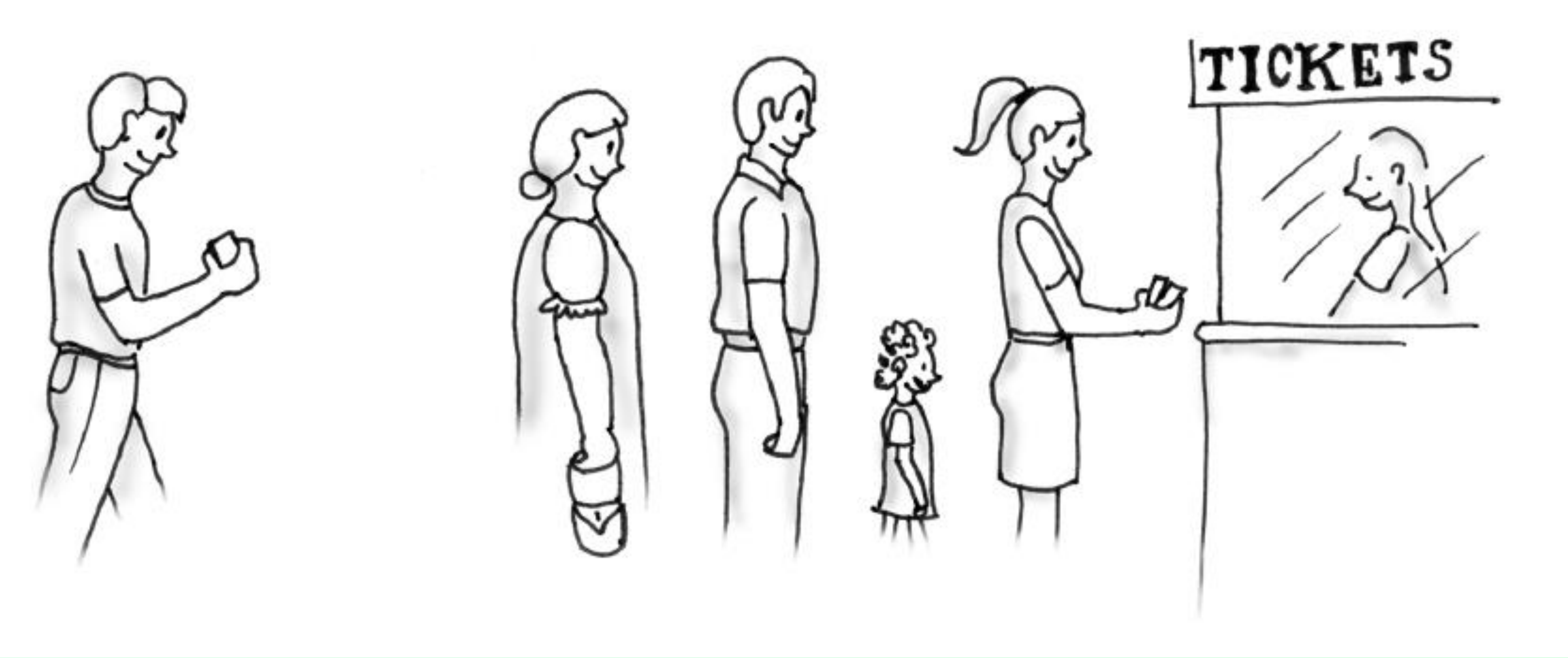
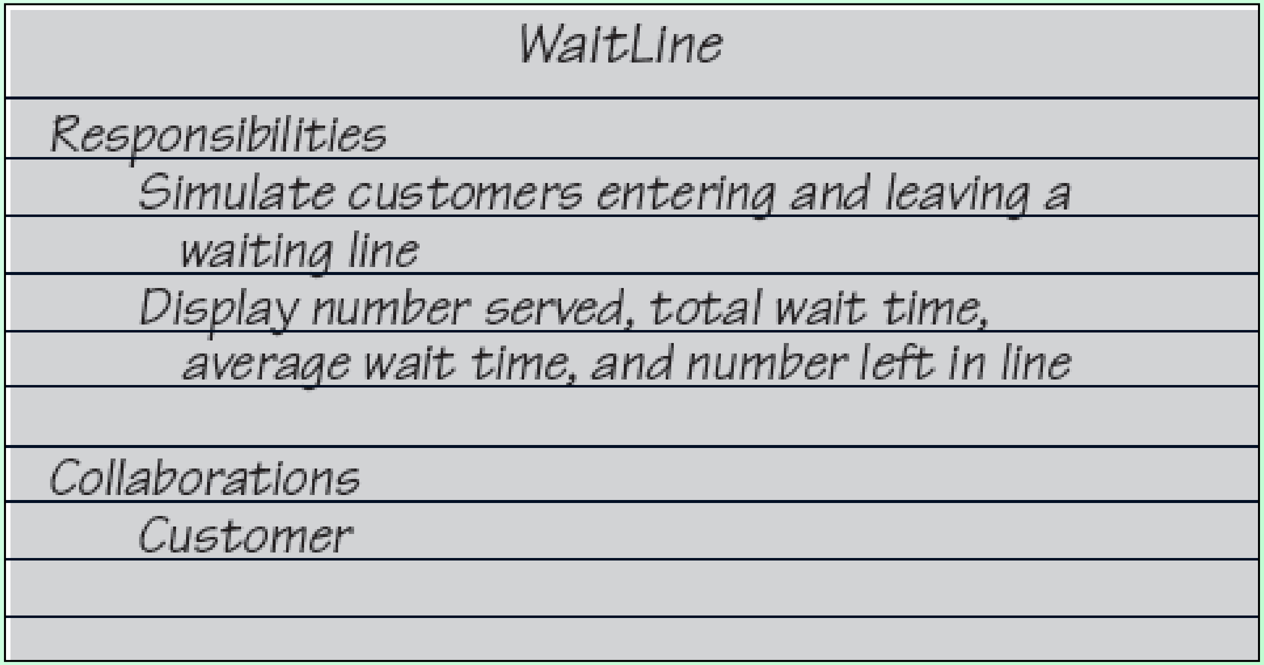


Figure 10-3 A line, or queue, of people



WaitLine

Responsibilities

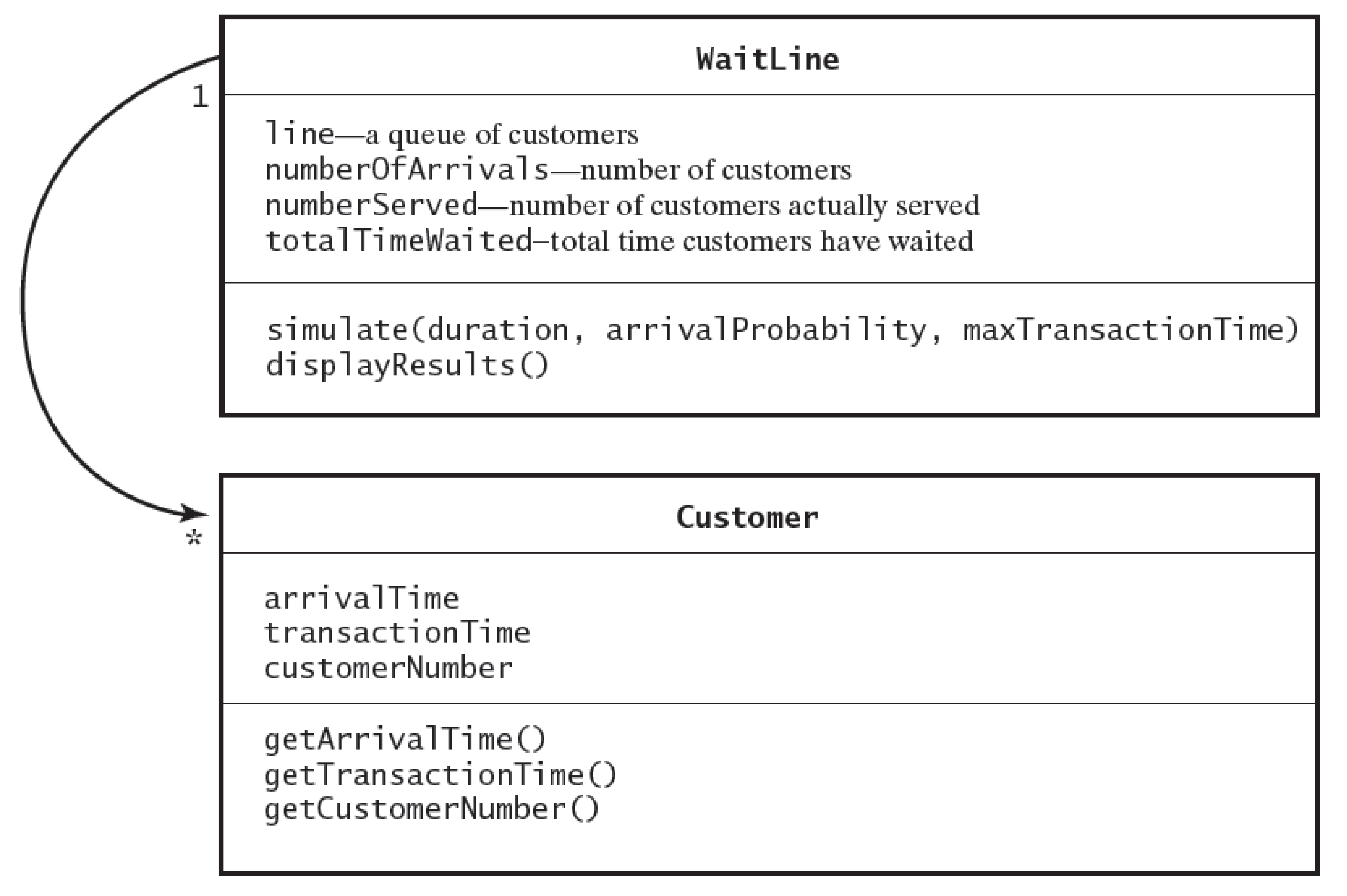
Simulate customers entering and leaving a waiting line

Display number served, total wait time, average wait time, and number left in line

Collaborations

Customer

Figure 10-4 A CRC card for the class **`WaitLine`**



1

WaitLine

line—a queue of customers
numberOfArrivals—number of customers
numberServed—number of customers actually served
totalTimeWaited–total time customers have waited

simulate(duration, arrivalProbability, maxTransactionTime)
displayResults()

*

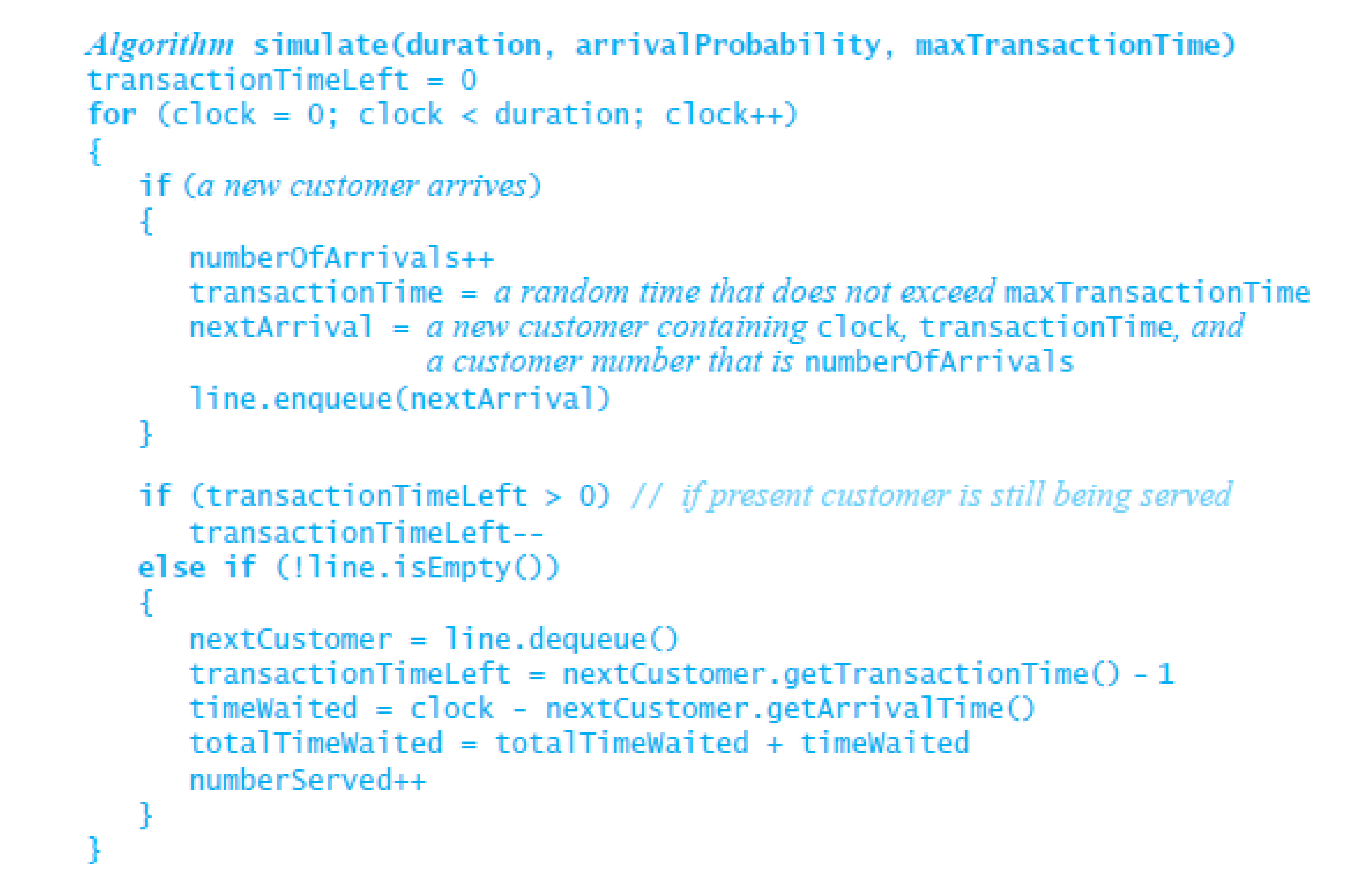
Customer

arrivalTime
transactionTime
customerNumber

getArrivalTime()
getTransactionTime()
getCustomerNumber()

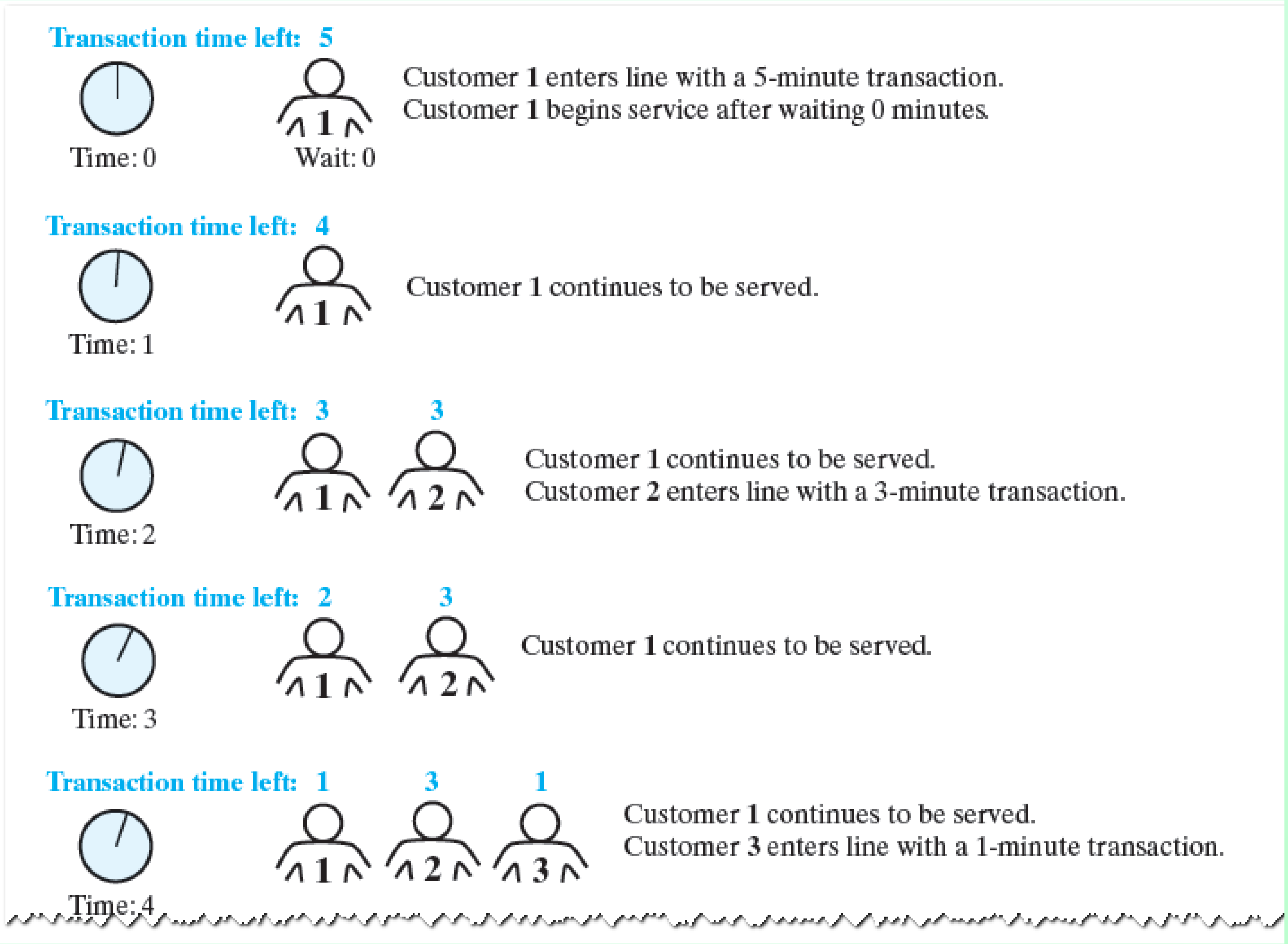
Figure 10-5 A diagram of the classes **`WaitLine`** and **`Customer`**

Algorithm for **`simulate`**



```
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++
    }
}
```



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

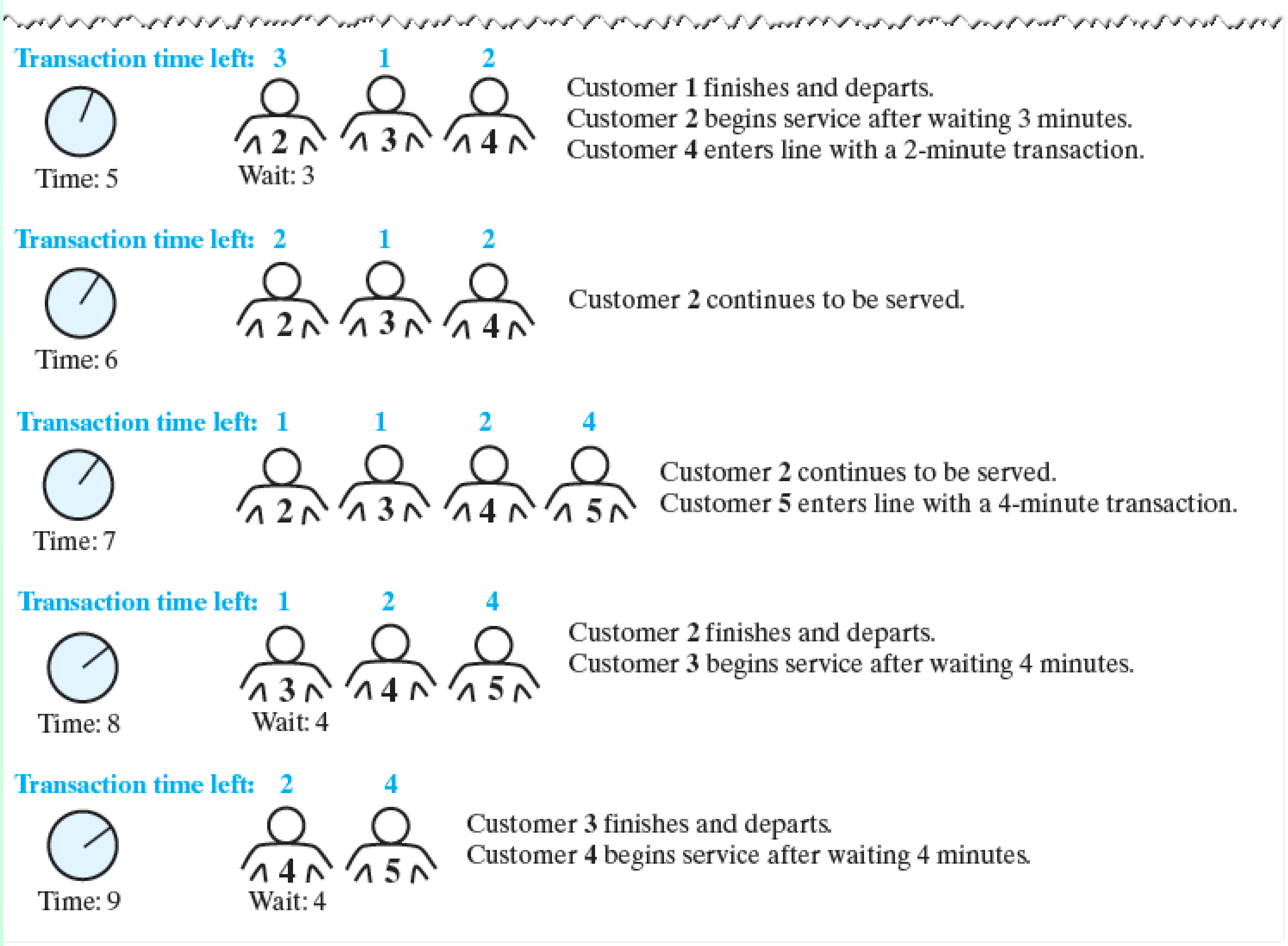
Customer 1 continues to be served.

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

Customer 1 continues to be served.

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

Figure 10-6 A simulated waiting line


Transaction time left: 3 1 2
Time: 5
2 3 4
Wait: 3

Customer 1 finishes and departs.
Customer 2 begins service after waiting 3 minutes.
Customer 4 enters line with a 2-minute transaction.

Transaction time left: 2 1 2
Time: 6
2 3 4

Customer 2 continues to be served.

Transaction time left: 1 1 2 4
Time: 7
2 3 4 5

Customer 2 continues to be served.
Customer 5 enters line with a 4-minute transaction.

Transaction time left: 1 2 4
Time: 8
3 4 5
Wait: 4

Customer 2 finishes and departs.
Customer 3 begins service after waiting 4 minutes.

Transaction time left: 2 4
Time: 9
4 5
Wait: 4

Customer 3 finishes and departs.
Customer 4 begins service after waiting 4 minutes.

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?

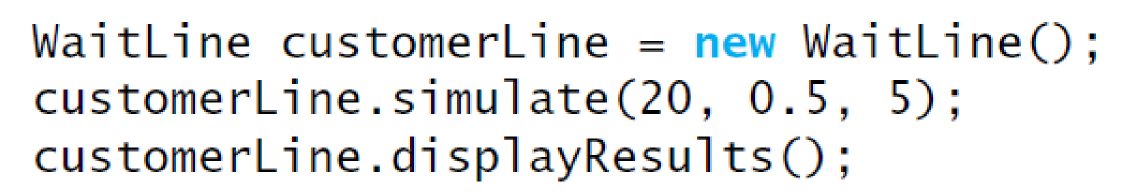
b. How long does Customer 5 wait before beginning the transaction?

2. **a.** 11.

b. 4.

Class **`WaitLine`**

- Implementation of class **`WaitLine`**
  Listing 10-2
- Statements

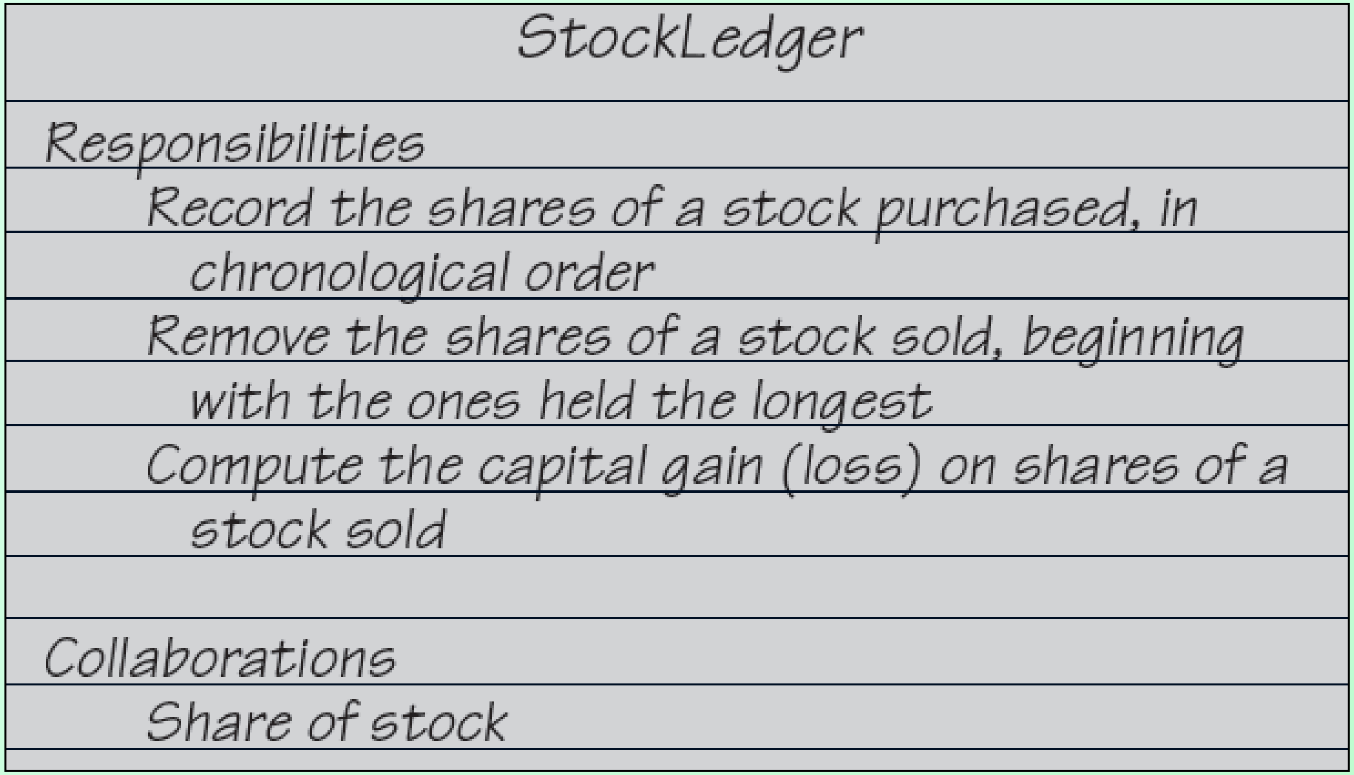


```
WaitLine customerLine = new WaitLine();
customerLine.simulate(20, 0.5, 5);
customerLine.displayResults();
```

  - 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, **`StockPurchase`**



StockLedger

Responsibilities

Record the shares of a stock purchased, in chronological order

Remove the shares of a stock sold, beginning with the ones held the longest

Compute the capital gain (loss) on shares of a stock sold

Collaborations

Share of stock

Figure 10-7 A CRC card for the class **`StockLedger`**

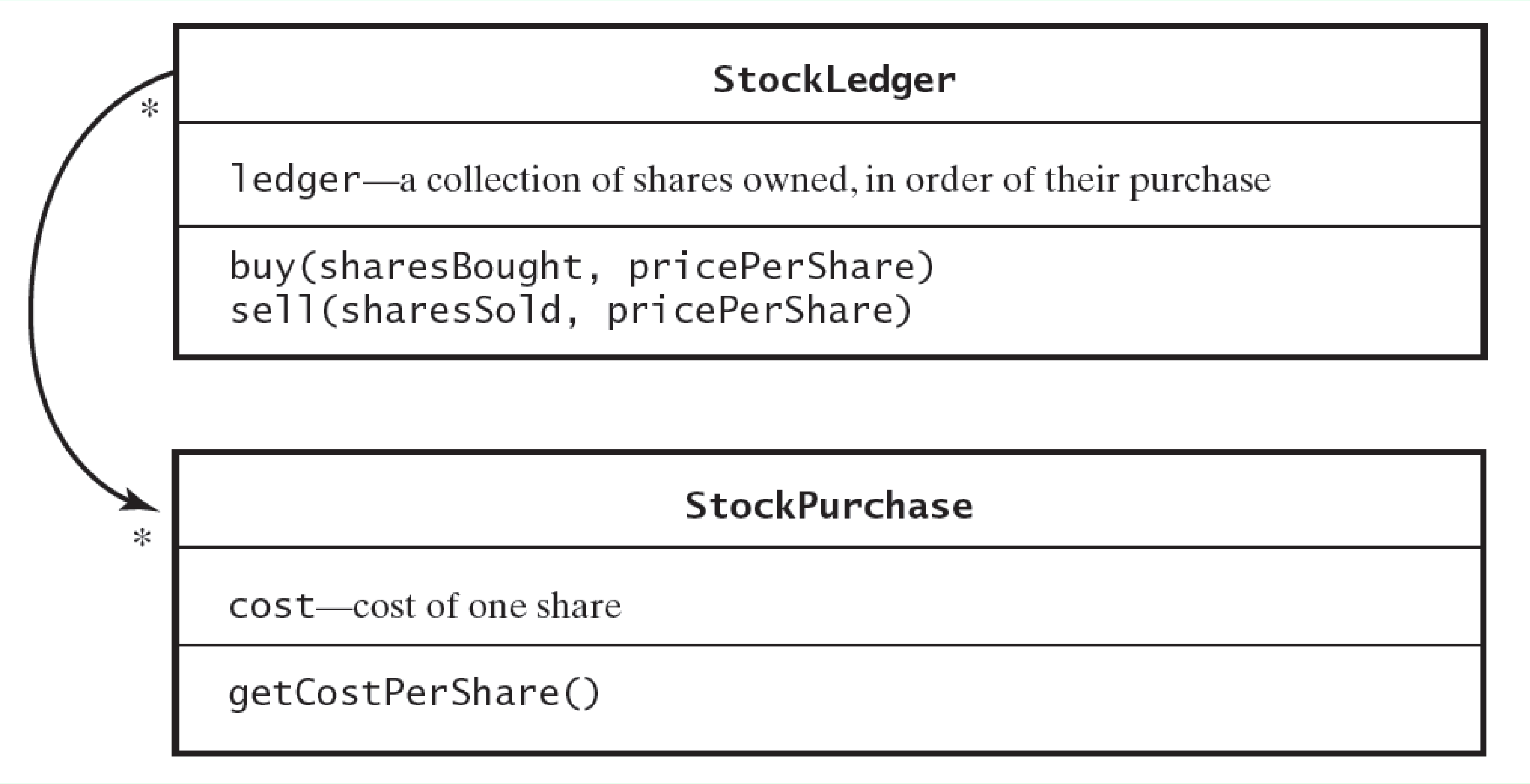

StockLedger
ledger—a collection of shares owned, in order of their purchase
buy(sharesBought, pricePerShare)
sell(sharesSold, pricePerShare)
*
*
StockPurchase
cost—cost of one share
getCostPerShare()

Figure 10-8 A diagram of the classes **`StockLedger`** and **`StockPurchase`**

Computing Capital Gain for Stock Sale

- View class implementation
  Listing 10-3

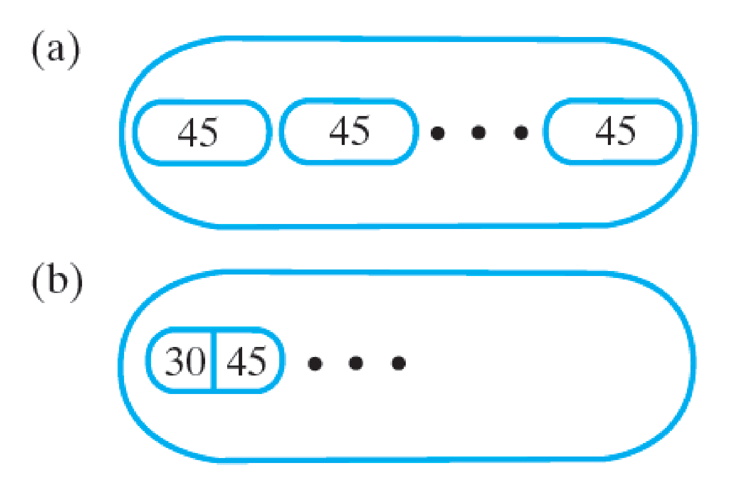


Figure 10-9 A queue of (a) individual shares of stock; (b) grouped shares

Java Class Library

- Interface java.util.`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

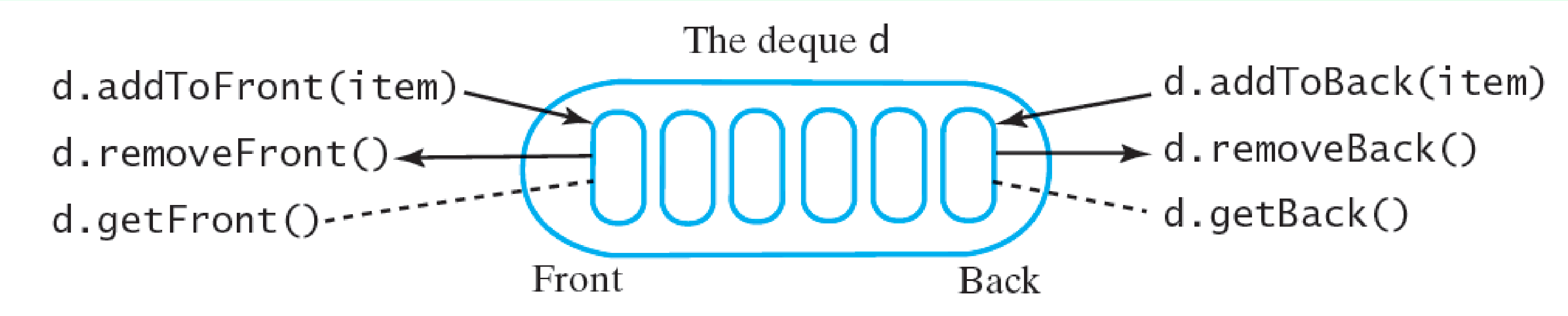


Figure 10-10 An instance *d* of a deque

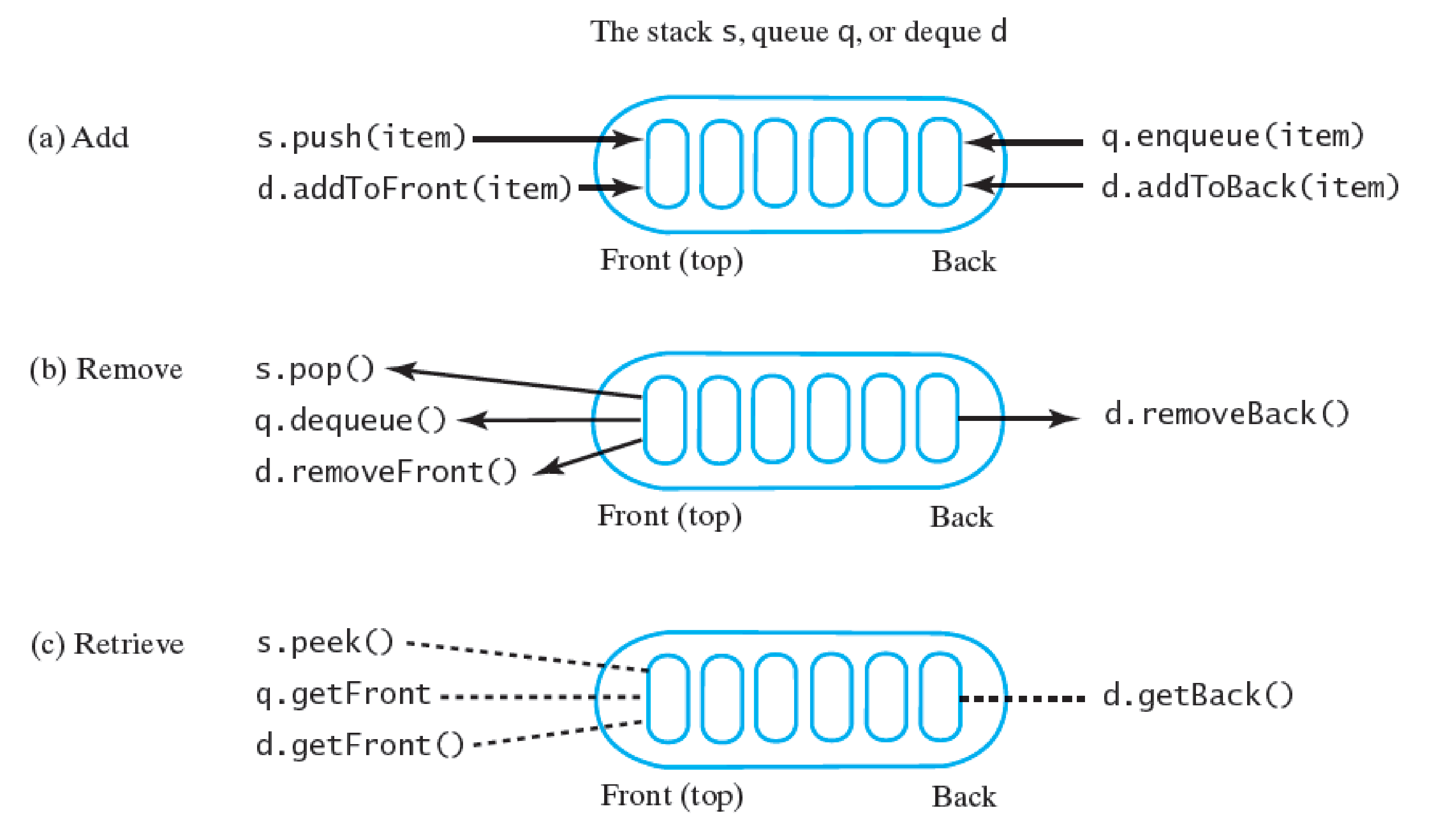


FIGURE 10-11 A comparison of operations for a stack s, a queue q, and a deque d: (a) add; (b) remove; (c) retrieve

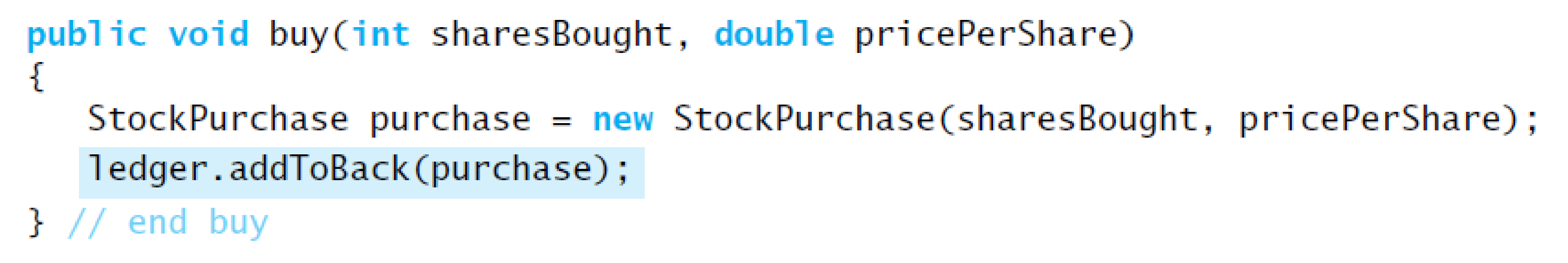
Question 3 After the following nine statements execute, what string is at the front of the deque and what string is at the back?

```
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());
```

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`**



```
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 java.util.**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

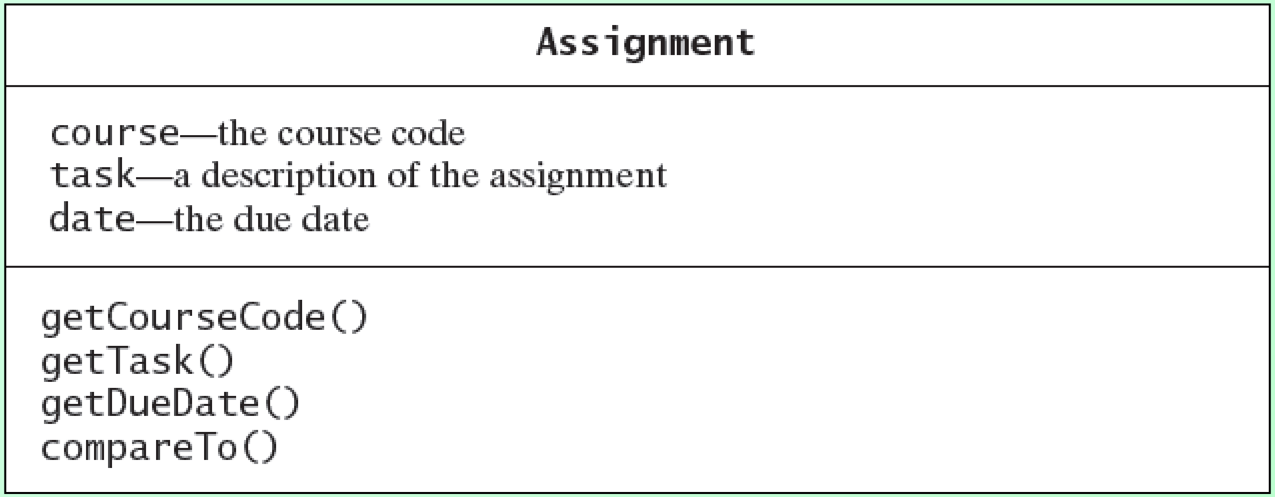
Question 4 After the following statements execute, what string is at the front of the priority queue and what string is at the back?

```
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



Assignment

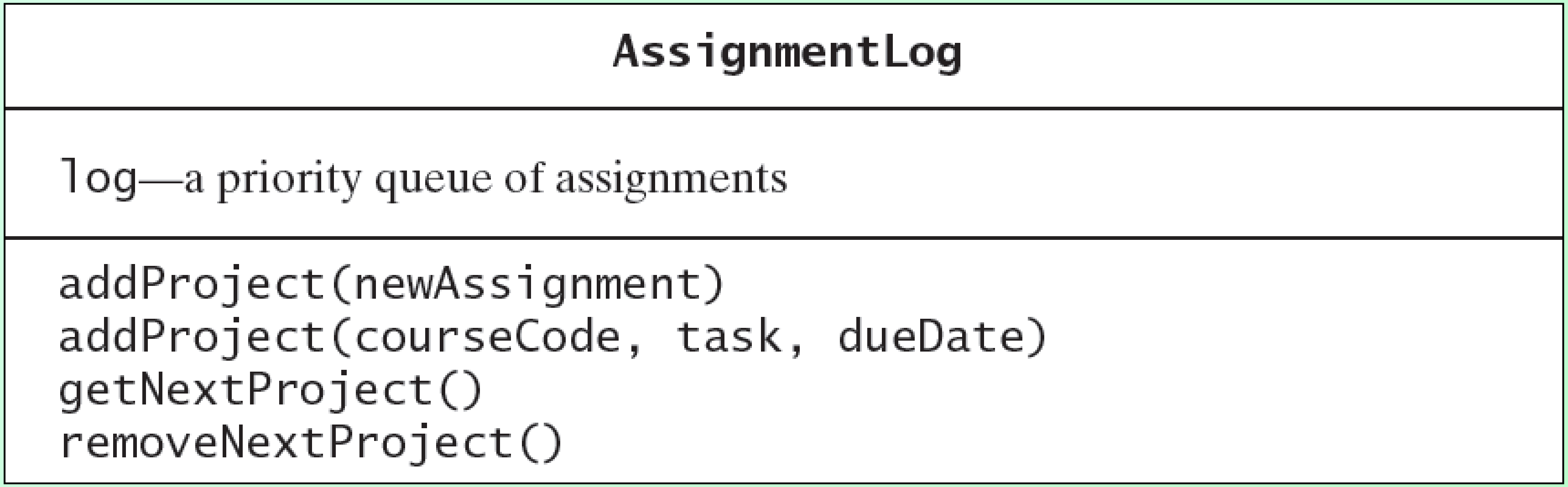
course—the course code
task—a description of the assignment
date—the due date

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

Figure 10-12 A diagram of the class **`Assignment`**

Tracking Your Assignments

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



AssignmentLog

`log`—a priority queue of assignments

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

Figure 10-13 A diagram of the class **`AssignmentLog`**

Java Class Library

- 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()`**

Lab4a StoreSim

- Kind of like the WaitLine example above
- Simpler in terms of data
- The Queue only holds integers
  - Representing arrival time for each customer

Lab4a StoreSim

- Each minute, customers Arrive with the following probability:
  - 50% of the time: 0 people
  - 25% of the time: 1 person
  - 25% of the time: 2 persons

How to Code It:

Generate random number 0,1,2, or 3

If 0 or 3, numArrivals = 0 (nobody came)

If 1, numArrivals = 1

If 2, numArrivals = 2

Serving

Minute: 0

- Queue: (empty)

  No customers served

Arrivals

Minute: 0

- Two customers arrive
- Queue: [0, 0]

Serving

Minute: 1

- Queue: [0]

  Serve customer:

  timeArrived: 0

  customersServed: 1

  waitTime: 1

  totalWaitTime: 1

Dequeue 0 and store in timeArrived

Arrivals

Minute: 1

- Two customers arrive
- Queue: [0, 1, 1]

Serving

Minute: 2

- Queue: [1, 1]

  Serve customer:

    timeArrived: 0

    customersServed: 2

    waitTime: 2

    totalWaitTime: 3

Dequeue 0 and store in timeArrived

Arrivals

Minute: 2

- One customer arrives
- Queue: [1, 1, 2]

Serving

Minute: 3

- Queue: [1, 2]

  Serve customer:

    timeArrived: 1

    customersServed: 3

    waitTime: 2

    totalWaitTime: 5

Dequeue 1 and store in timeArrived

Arrivals

Minute: 3

- Two customers arrive
- Queue: [1, 2, 3, 3]

Serving

Minute: 4

- Queue: [2, 3, 3]

  Serve customer:

  timeArrived: 1

  customersServed: 4

  waitTime: 3

  totalWaitTime: 8

Dequeue 1 and store in timeArrived

Arrivals

Minute: 4

- No customers arrive
- Queue: [2, 3, 3]

Serving

Minute: 5

- Queue: [3, 3]

  Serve customer:

  timeArrived: 2

  customersServed: 5

  waitTime: 3

  totalWaitTime: 11

Dequeue 2 and store in timeArrived

Arrivals

Minute: 5

- Two customers arrive
- Queue: [3, 3, 5, 5]

End

Chapter 10