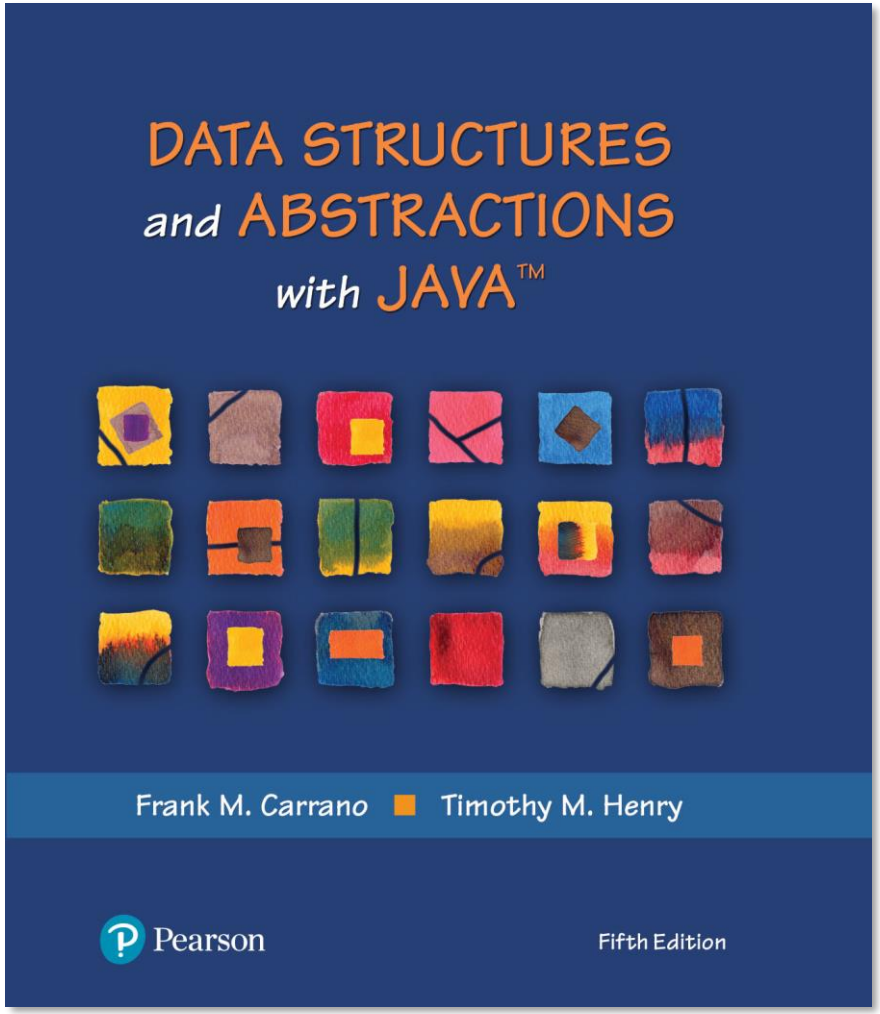
Data Structures and Abstractions with Java™

5th Edition



Chapter 7

Queues, Deques, and Priority Queues

The ADT Queue

- A queue is another name for a waiting line
- Used within operating systems and to simulate real-world events
  - Come into play whenever processes or events must wait
- Entries organized first-in, first-out

The ADT Queue

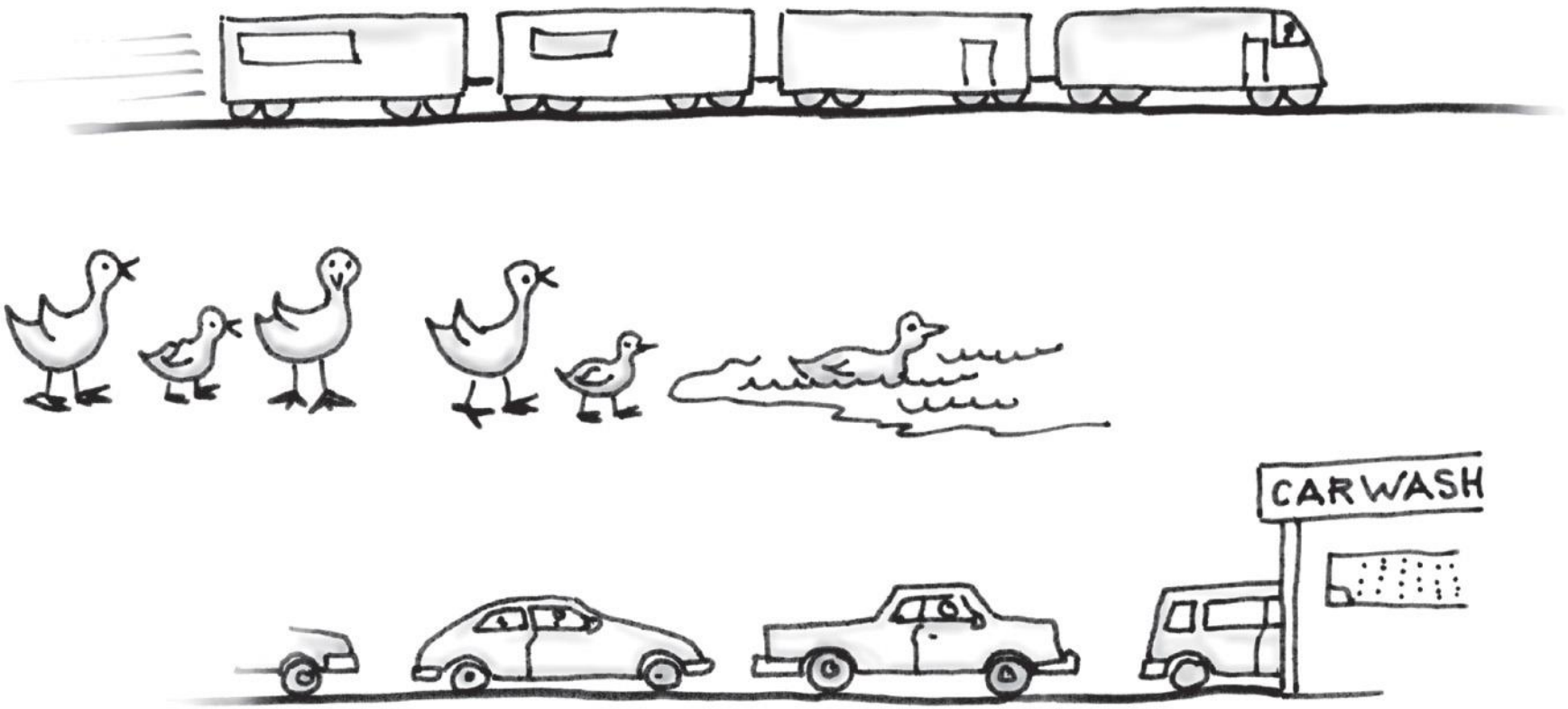


FIGURE 7-1 Some everyday queues

The ADT Queue

- Terminology
  - Item added first, or earliest, is at the front of the queue
  - Item added most recently is at the back of the queue
- Additions to a software queue must occur at its back
- Client can look at or remove only the entry at the front of the queue

The ADT Queue

- Data
  - A collection of objects in chronological order and having the same data type

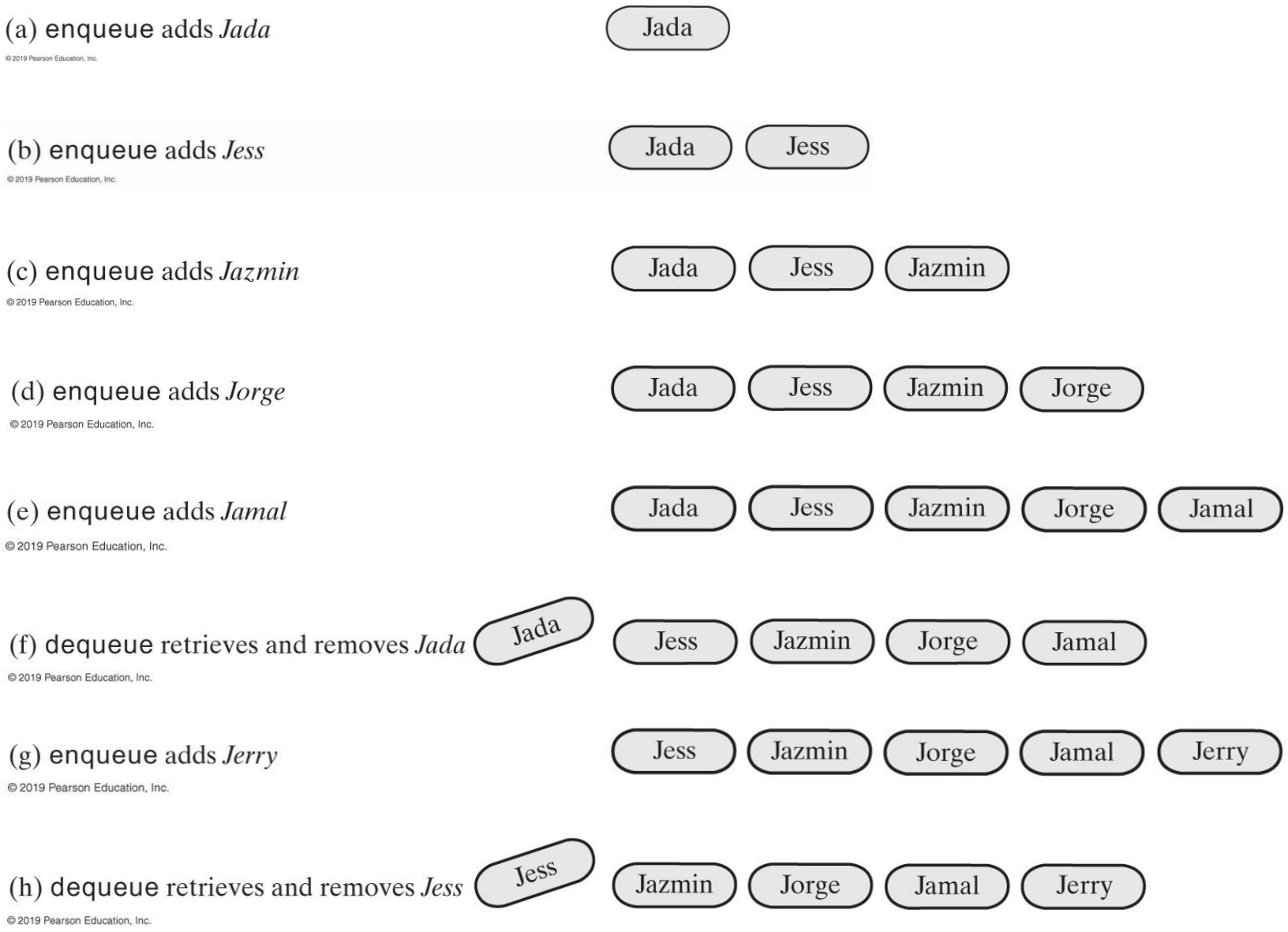
Pseudocode	UML	Description
enqueue(newEntry)	+enqueue(newEntry: integer): void	Task: Adds a new entry to the back of the queue. Input: newEntry is the new entry. Output: None.
dequeue()	+dequeue(): T	Task: Removes and returns the entry at the front of the queue. Input: None. Output: Returns the queue's front entry. Throws an exception if the queue is empty before the operation.
getFront()	+getFront(): T	Task: Retrieves the queue's front entry without changing the queue in any way. Input: None. Output: Returns the queue's front entry. Throws an exception if the queue is empty.
isEmpty()	+isEmpty(): boolean	Task: Detects whether the queue is empty. Input: None. Output: Returns true if the queue is empty.
clear()	+clear(): void	Task: Removes all entries from the queue. Input: None. Output: None.

The ADT Queue

```
/** An interface for the ADT queue.  */
public interface QueueInterface<T>
{
  /** Adds a new entry to the back of this queue.
      @param newEntry  An object to be added. */
  public void enqueue(T newEntry);

  /** Removes and returns the entry at the front of this queue.
      @return  The object at the front of the queue.
      @throws  EmptyQueueException if the queue is empty before the operation. */
  public T dequeue();

  /**  Retrieves the entry at the front of this queue.
      @return  The object at the front of the queue.
      @throws  EmptyQueueException if the queue is empty. */
  public T getFront();

  /** Detects whether this queue is empty.
      @return  True if the queue is empty, or false otherwise. */
  public boolean isEmpty();

  /** Removes all entries from this queue. */
  public void clear();
} // end QueueInterface
```

LISTING 7-1 An interface for the ADT queue

FIGURE 7-2 The effect of operations on a queue of strings



Simulating a Waiting Line

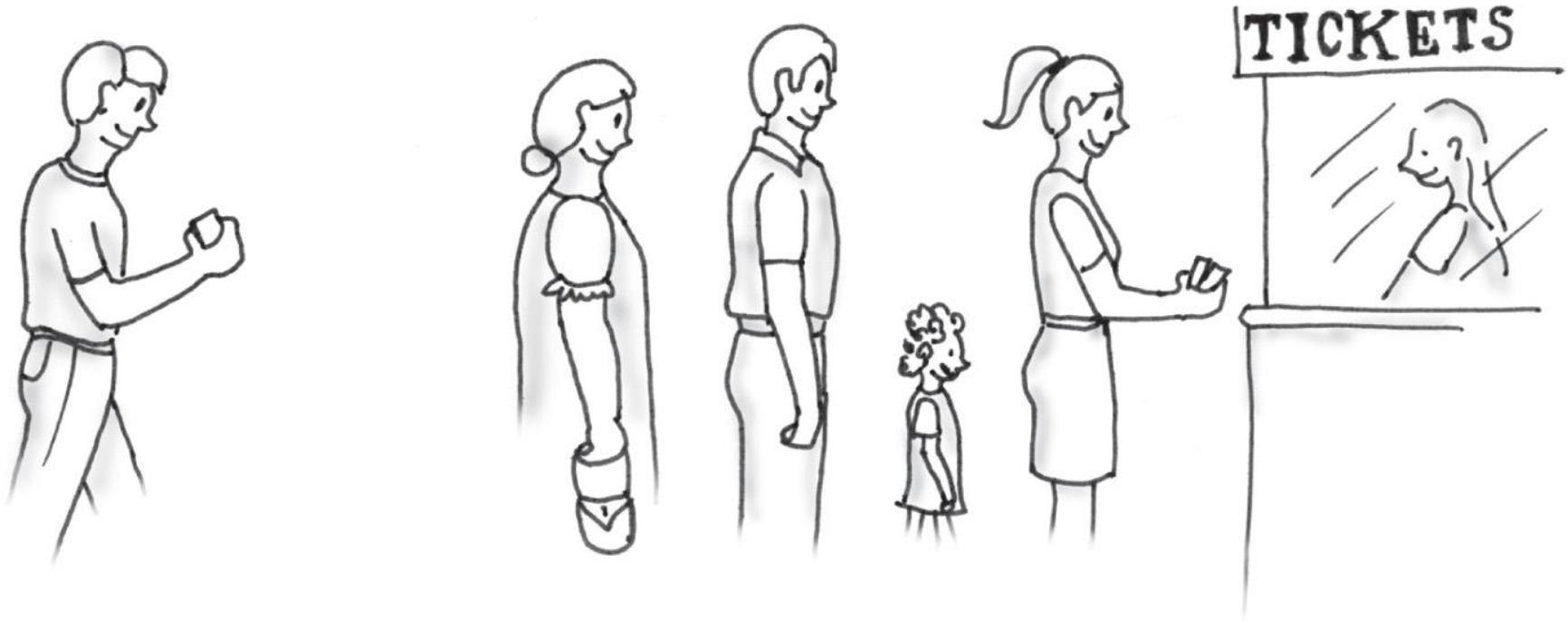


FIGURE 7-3 A line, or queue, of people

Simulating a Waiting Line

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 7-4 A CRC card for the class WaitLine

Simulating a Waiting Line

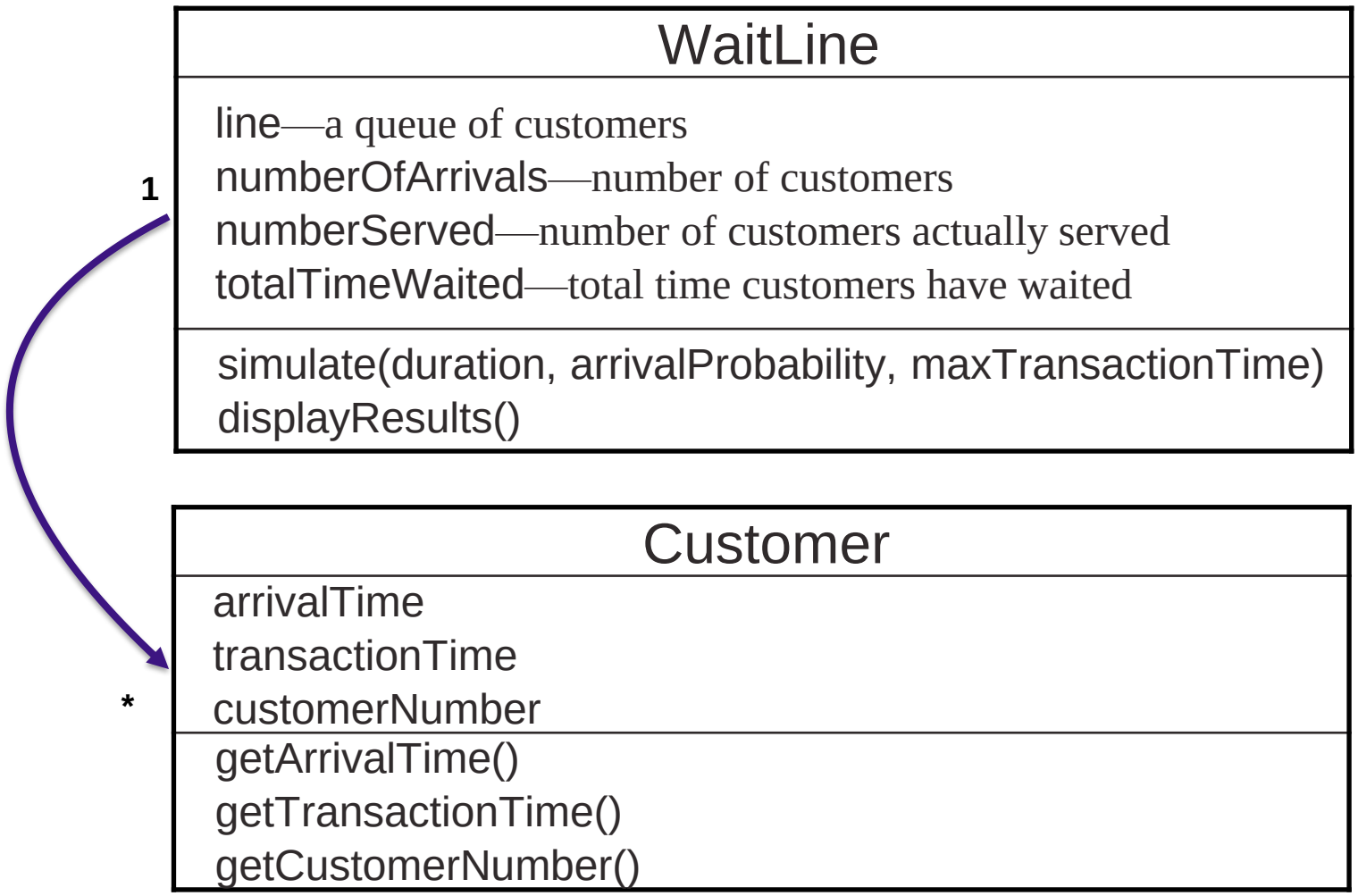


FIGURE 7-5 A diagram of the classes WaitLine and Customer

Simulating a Waiting Line

```
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++
   }
}
```

Algorithm for simulate

Simulating a Waiting Line (Part 1)

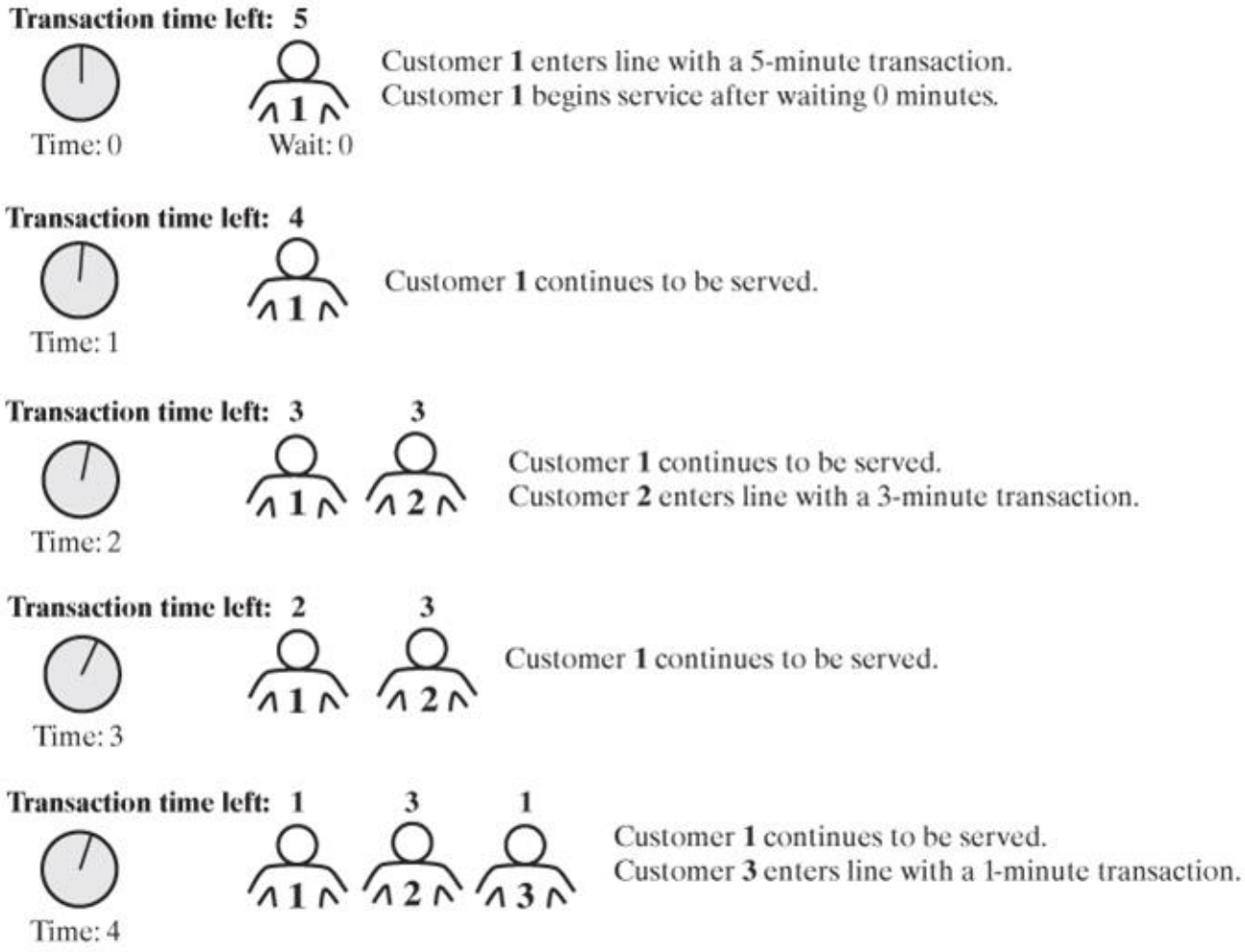


FIGURE 7-6 A simulated waiting line

Simulating a Waiting Line (Part 2)

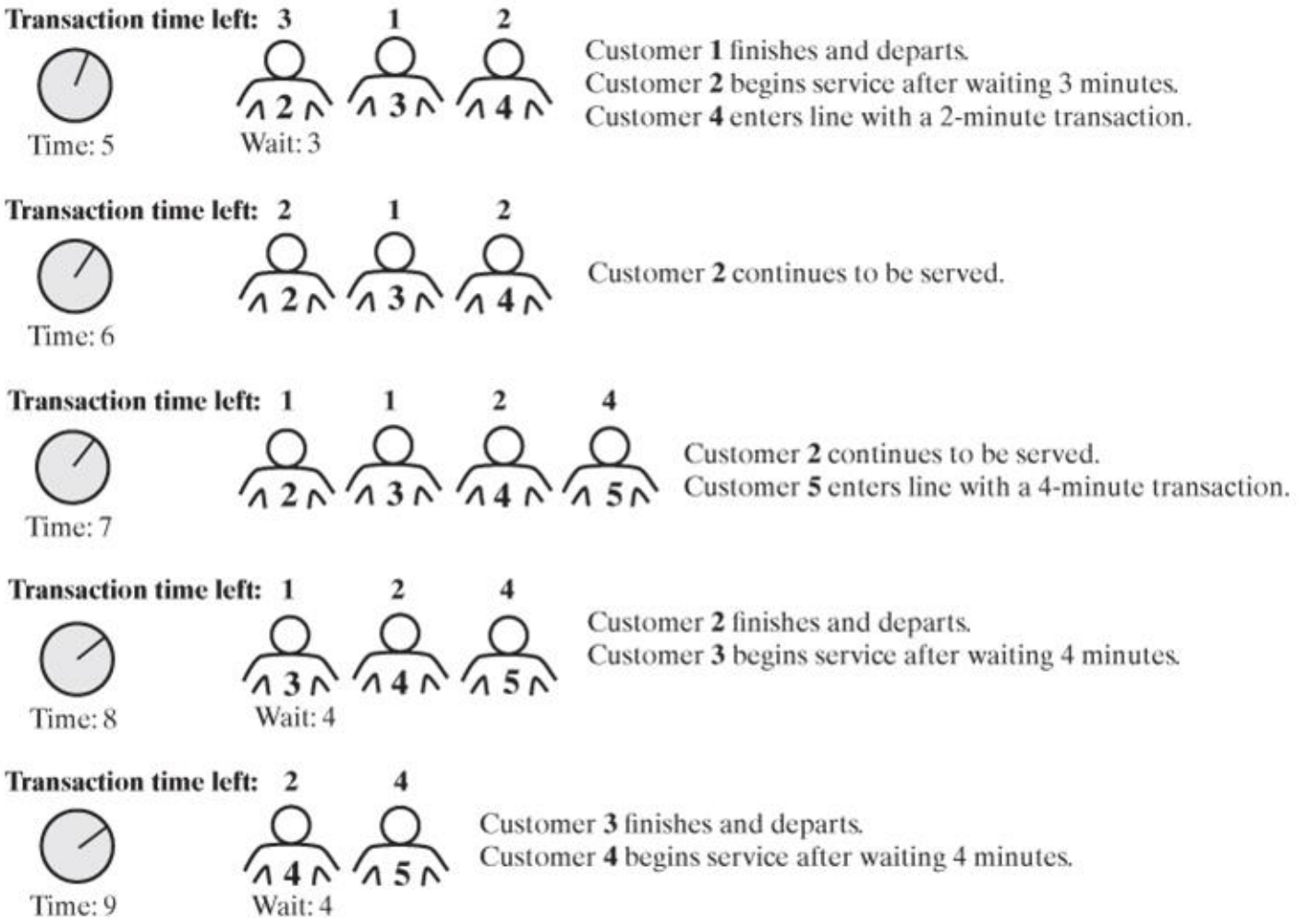


FIGURE 7-6 A simulated waiting line

Simulating a Waiting Line

```
/** Simulates a waiting line.  */
public class WaitLine
{
  private QueueInterface<Customer> line;
  private int numberOfArrivals;
  private int numberServed;
  private int totalTimeWaited;

  public WaitLine()
  {
    line = new LinkedQueue<>();
    reset();
  } // end default constructor

  /** Initializes the simulation. */
  public final void reset()
  {
    line.clear();
    numberOfArrivals = 0;
    numberServed = 0;
    totalTimeWaited = 0;
  } // end reset

  public void simulate(int duration, double arrivalProbability,
                       int maxTransactionTime)
  {  < implementation on next slide > }

  public void displayResults()
  {  < implementation on next slide >  }
} // end WaitLine
```

LISTING 7-2 The class WaitLine

Simulating a Waiting Line

```
public void simulate(int duration, double arrivalProbability, int maxTransactionTime)
{
  int transactionTimeLeft = 0;

  for (int clock = 0; clock < duration; clock++)
  {
    if (Math.random() < arrivalProbability)
    {
      numberOfArrivals++;
      int transactionTime = (int)(Math.random() * maxTransactionTime + 1);
      Customer nextArrival = new Customer(clock, transactionTime,
                                          numberOfArrivals);
      line.enqueue(nextArrival);
      System.out.println("Customer " + numberOfArrivals +
                         " enters line at time " + clock +
                         ". Transaction time is " + transactionTime);
    } // end if

    if (transactionTimeLeft > 0)
      transactionTimeLeft--;
    else if (!line.isEmpty())
    {
      Customer nextCustomer = line.dequeue();
      transactionTimeLeft = nextCustomer.getTransactionTime() - 1;
      int timeWaited = clock - nextCustomer.getArrivalTime();
      totalTimeWaited = totalTimeWaited + timeWaited;
      numberServed++;
      System.out.println("Customer " + nextCustomer.getCustomerNumber() +
                         " begins service at time " + clock +
                         ". Time waited is " + timeWaited);
    } // end if
  } // end for
} // end simulate
```

Simulating a Waiting Line

```
/** Displays summary results of the simulation. */
 public void displayResults()
 {
   System.out.println();
   System.out.println("Number served = " + numberServed);
   System.out.println("Total time waited = " + totalTimeWaited);
   double averageTimeWaited = ((double)totalTimeWaited) / numberServed;
   System.out.println("Average time waited = " + averageTimeWaited);
   int leftInLine = numberOfArrivals - numberServed;
   System.out.println("Number left in line = " + leftInLine);
 } // end displayResults
```

Computing the Capital Gain in a Sale of Stock

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 7-7 A CRC card for the class `StockLedger`

Computing the Capital Gain in a Sale of Stock

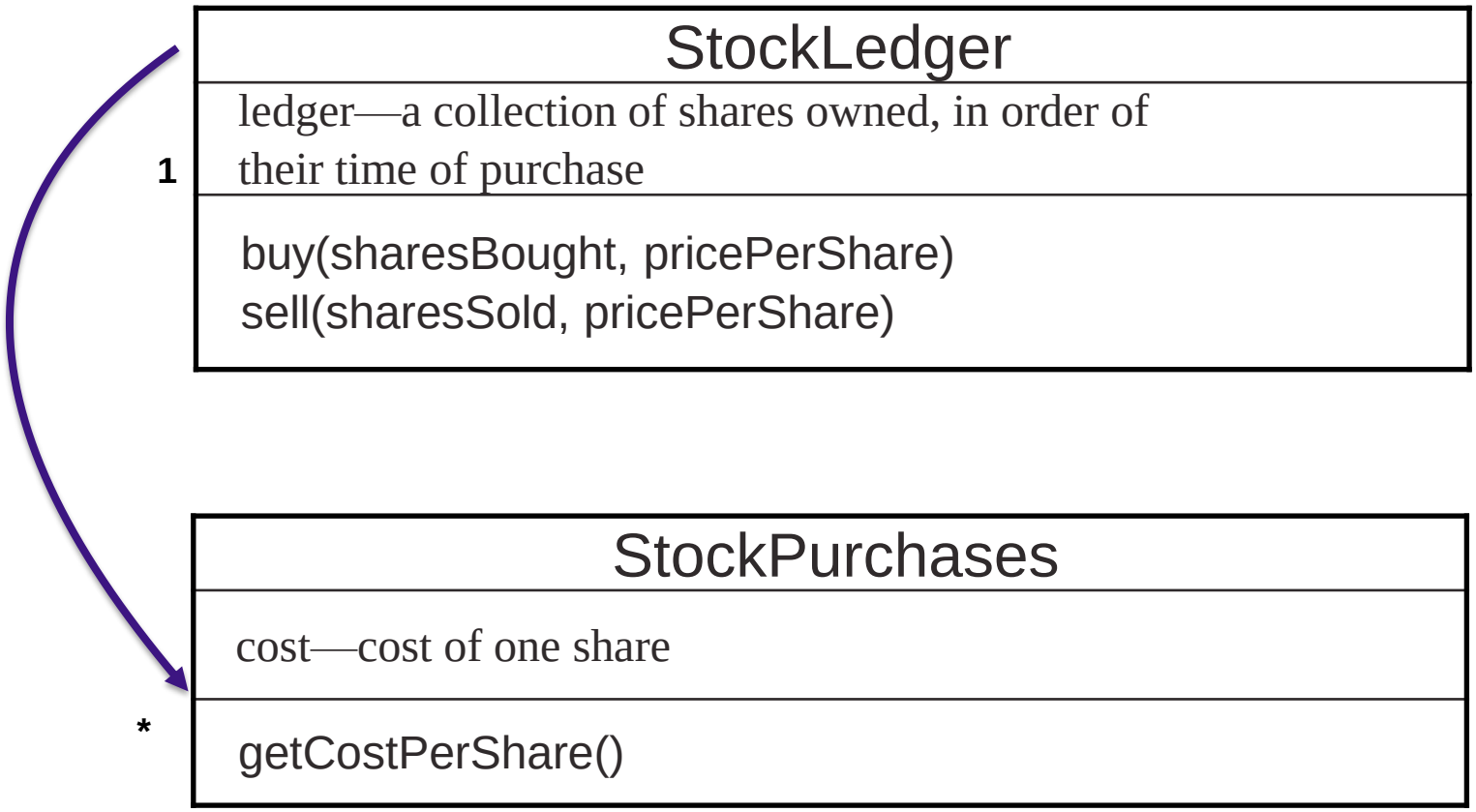


FIGURE 7-8 A diagram of the classes StockLedger and StockPurchase

Computing the Capital Gain in a Sale of Stock (Part 1)

```
public class StockLedger
{
 private QueueInterface<StockPurchase> ledger;

 public StockLedger()
 {
   ledger = new LinkedQueue<>();
 } // end default constructor

 /** Records a stock purchase in this ledger.
    @param sharesBought   The number of shares purchased.
    @param pricePerShare  The price per share. */
 public void buy(int sharesBought, double pricePerShare)
 {
  while (sharesBought > 0)
  {
    StockPurchase purchase = new StockPurchase(pricePerShare);
    ledger.enqueue(purchase);
    sharesBought--;
  } // end while
 } // end buy
```

LISTING 7-3 The class `StockLedger`

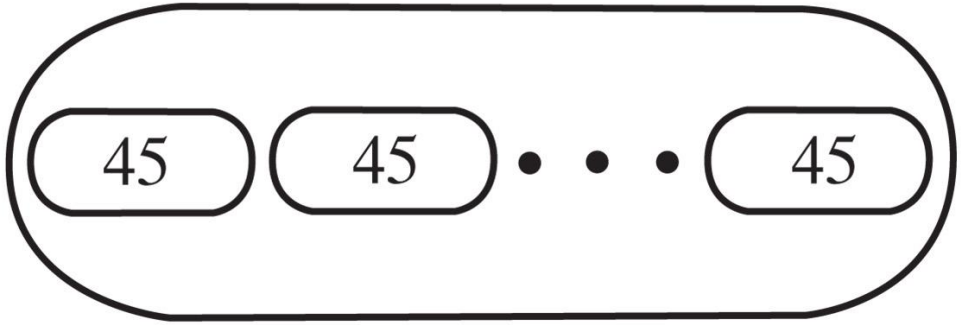
Computing the Capital Gain in a Sale of Stock (Part 2)

```
/** Removes from this ledger any shares that were sold
    and computes the capital gain or loss.
    @param sharesSold     The number of shares sold.
    @param pricePerShare  The price per share.
    @return  The capital gain (loss). */
 public double sell(int sharesSold, double pricePerShare)
 {
   double saleAmount = sharesSold * pricePerShare;
   double totalCost = 0;

   while (sharesSold > 0)
   {
     StockPurchase share = ledger.dequeue();
     double shareCost = share.getCostPerShare();
     totalCost = totalCost + shareCost;
     sharesSold--;
   } // end while

   return saleAmount - totalCost; // Gain or loss
 } // end sell
} // end StockLedger
```

LISTING 7-3 The class `StockLedger`

FIGURE 7-9 Two representations of stock shares in a queue

(a) Individual shares of stock in a queue



(b) Grouped shares of stock as objects in a queue

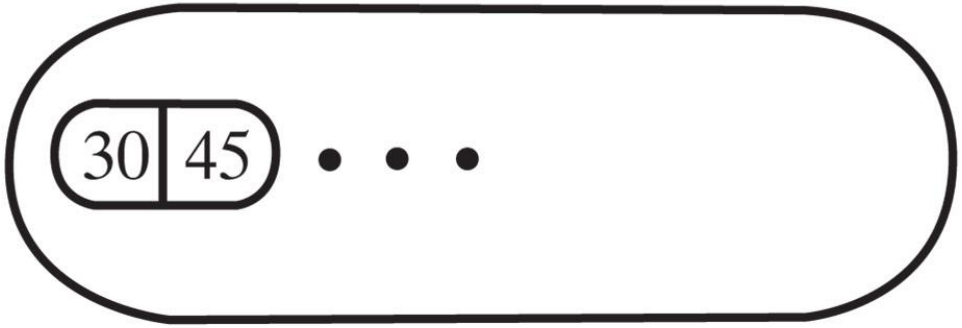


FIGURE 7-9 Two representations of stock shares in a queue

Java Class Library: The Interface `Queue`

- Methods provided
  - `add`
  - `offer`
  - `remove`
  - `poll`
  - `element`
  - `peek`
  - `isEmpty`
  - `size`

The ADT `Deque`

- A double ended queue
- `Deque` pronounced “deck”
- Has both queue-like operations and stack-like operations

The ADT Deque

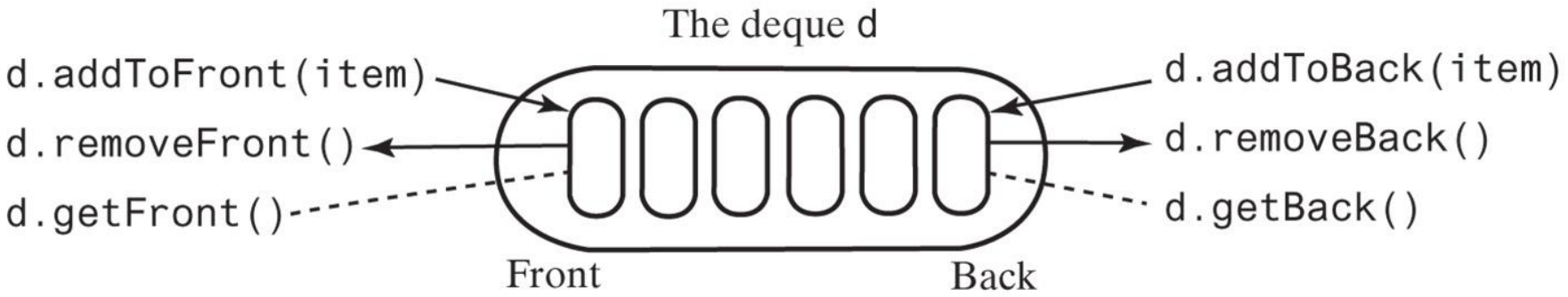


FIGURE 7-10 An instance d of a deque

FIGURE 7-11 A comparison of operations for a stack s, a queue q, and a deque d

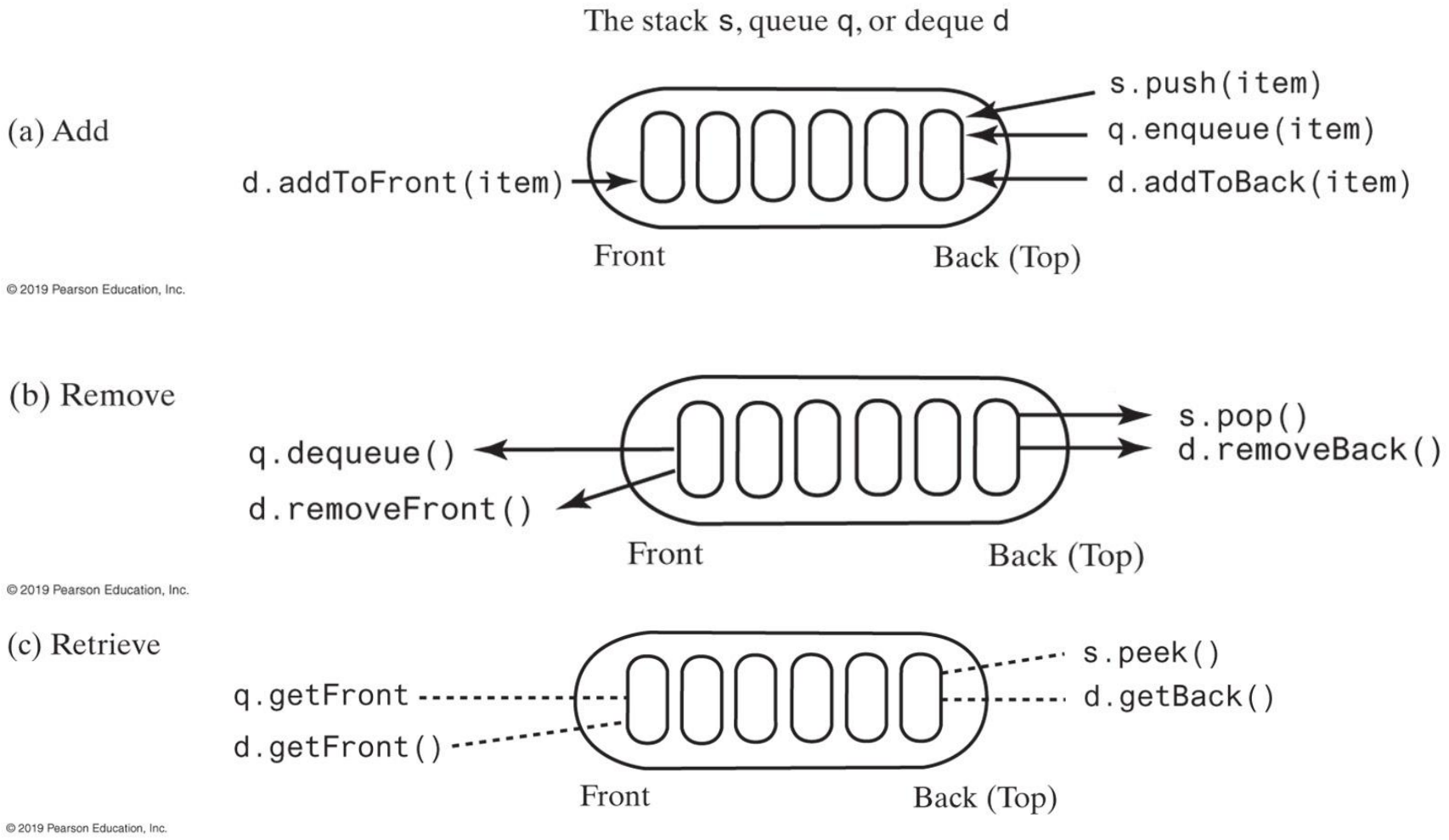


FIGURE 7-11 A comparison of operations for a stack s, a queue q, and a deque d

The ADT Deque

```
/** An interface for the ADT deque. */
public interface DequeInterface<T>
{
  /** Adds a new entry to the front/back of this deque.
     @param newEntry  An object to be added. */
  public void addToFront(T newEntry);
  public void addToBack(T newEntry);

  /** Removes and returns the front/back entry of this deque.
     @return  The object at the front/back of the deque.
     @throws  EmptyQueueException if the deque is empty before the
           operation. */
  public T removeFront();
  public T removeBack();

  /** Retrieves the front/back entry of this deque.
     @return  The object at the front/back of the deque.
     @throws  EmptyQueueException if the deque is empty. */
  public T getFront();
  public T getBack();

  /** Detects whether this deque is empty.
     @return  True if the deque is empty, or false otherwise. */
  public boolean isEmpty();

  /*  Removes all entries from this deque. */
  public void clear();
} // end DequeInterface
```

LISTING 7-4 An interface for the ADT deque

The ADT Deque

```
// Read a line
d = a new empty deque
while (not end of line)
{
   character = next character read
   if (character == ←)
      d.removeBack()
   else
      d.addToBack(character)
}
// Display the corrected line
while (!d.isEmpty())
   System.out.print(d.removeFront())
System.out.println()
```

Pseudocode that uses a deque to read and display a line of keyboard input

Computing the Capital Gain in a Sale of Stock

```
public double sell(int sharesSold, double pricePerShare)
{
  double saleAmount = sharesSold * pricePerShare;
  double totalCost = 0;

  while (sharesSold > 0)
  {
    StockPurchase transaction = ledger.removeFront();
    double shareCost = transaction.getCostPerShare();
    int numberOfShares = transaction.getNumberOfShares();

    if (numberOfShares > sharesSold)
    {
      totalCost = totalCost + sharesSold * shareCost;
      int numberToPutBack = numberOfShares - sharesSold;
      StockPurchase leftOver = new StockPurchase(numberToPutBack, shareCost);
      ledger.addToFront(leftOver); // Return leftover shares
      // Note: Loop will exit since sharesSold will be <= 0 later
    }
    else
      totalCost = totalCost + numberOfShares * shareCost;

    sharesSold = sharesSold - numberOfShares;
  } // end while

  return saleAmount - totalCost; // Gain or loss
} // end sell
```

Method buy creates an instance of `StockPurchase` and places it at the back of the deque

Java Class Library: The Interface Deque

- Methods provided
  - `addFirst, offerFirst`
  - `addLast, offerLast`
  - `removeFirst, pollFirst`
  - `removeLast, pollLast`
  - `getFirst, peekFirst`
  - `getLast, peekLast`
  - `isEmpty, clear, size`
  - `push, pop`

Java Class Library: The Class `ArrayDeque`

- Implements the interface **`Deque`**
- Constructors provided
  - **`ArrayDeque()`**
  - **`ArrayDeque(int initialCapacity)`**

ADT Priority Queue

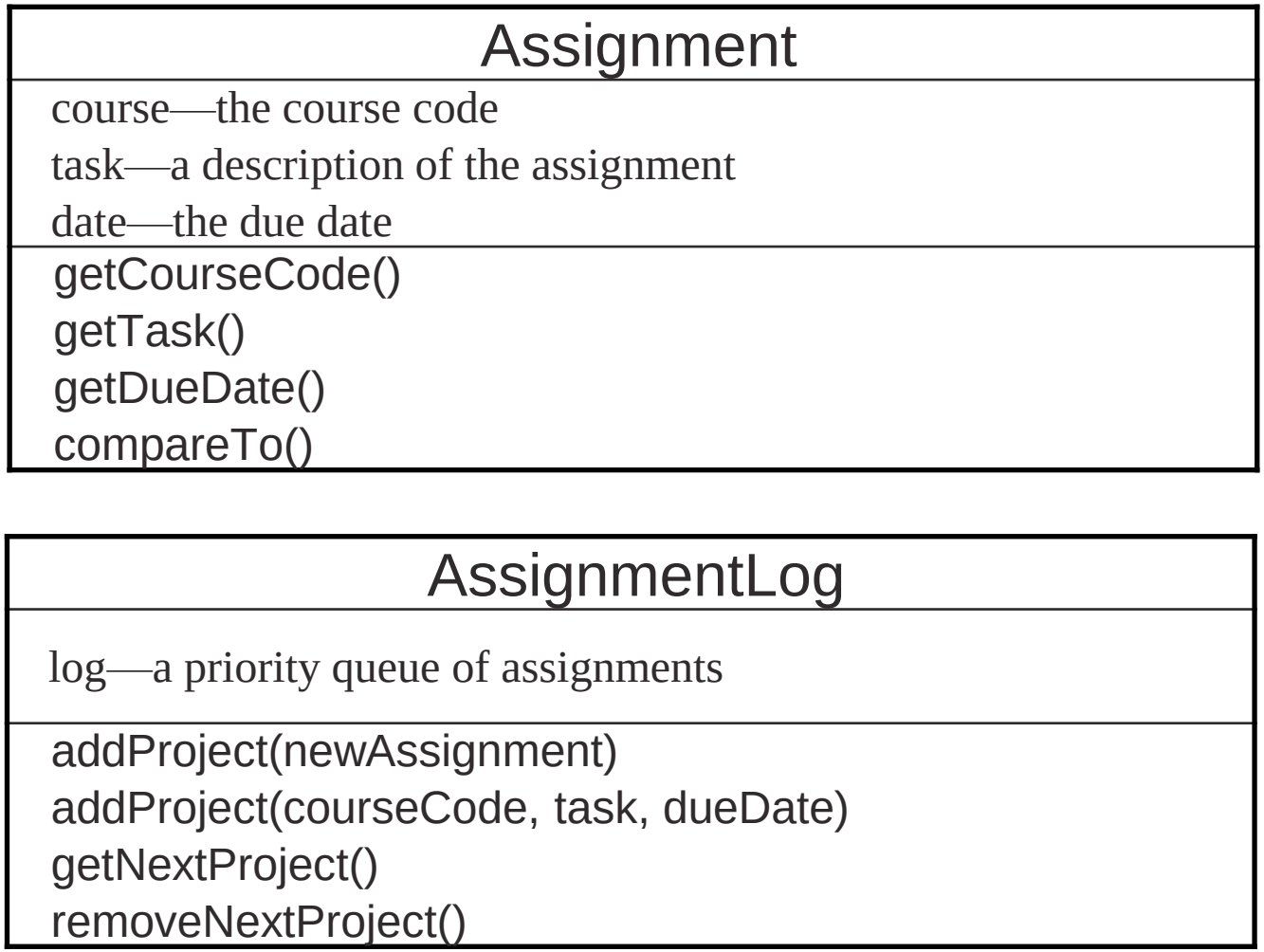
- Consider how a hospital assigns a priority to each patient that overrides time at which patient arrived.
- ADT priority queue organizes objects according to their priorities
- Definition of “priority” depends on nature of the items in the queue

ADT Priority Queue

```
/** An interface for the ADT priority queue. */
public interface PriorityQueueInterface<T extends Comparable<? super T>>
{
  /** Adds a new entry to this priority queue.
     @param newEntry  An object to be added. */
  public void add(T newEntry);

  /** Removes and returns the entry having the highest priority.
     @return  Either the object having the highest priority or, if the
              priority queue is empty before the operation, null. */
  public T remove();

  /** Retrieves the entry having the highest priority.
     @return  Either the object having the highest priority or, if the
              priority queue is empty, null. */
  public T peek();

  /** Detects whether this priority queue is empty.
     @return  True if the priority queue is empty, or false otherwise. */
  public boolean isEmpty();

  /** Gets the size of this priority queue.
     @return  The number of entries currently in the priority queue. */
  public int getSize();

  /** Removes all entries from this priority queue. */
  public void clear();
} // end PriorityQueueInterface
```

LISTING 7-5 An interface for the ADT priority queue

Tracking Your Assignments



FIGURES 7-12 & 7-13 UML diagrams of the class `Assignment` and `AssignmentLog`

Tracking Your Assignments

```
public class AssignmentLog
{
  private PriorityQueueInterface<Assignment> log;

  public AssignmentLog()
  {
    log = new PriorityQueue<>();
  } // end constructor

  public void addProject(Assignment newAssignment)
  {
    log.add(newAssignment);
  } // end addProject

  public void addProject(String courseCode, String task, Date dueDate)
  {
    Assignment newAssignment = new Assignment(courseCode, task, dueDate);
    addProject(newAssignment);
  } // end addProject

  public Assignment getNextProject()
  {
    return log.peek();
  } // end getNextProject

  public Assignment removeNextProject()
  {
    return log.remove();
  } // end removeNextProject
} // end AssignmentLog
```

LISTING 7-6 The class AssignmentLog

Java Class Library: The Class `PriorityQueue`

- Basic constructors and methods
- `PriorityQueue`
  - `add`
  - `offer`
  - `remove`
  - `poll`
  - `element`
  - `peek`
  - `isEmpty, clear, size`

End

Chapter 7