Exercises 4.40-4.42 and 4.54-4.55

The Club class is shown below:

```
/**
 * Store details of club memberships.
 *
 * @author (Artem Los)
 * @version (2014.09.29)
 */
import java.util.ArrayList;
import java.util.Iterator;

public class Club
{
    private ArrayList<Membership> memberships;

    public Club()
    {
        memberships = new ArrayList<Membership>();

    }

    /**
     * Add a new member to the club's list of members.
     * @param member The member object to be added.
     */
    public void join(Membership member)
    {
        memberships.add(member);
    }

    /**
     * @return The number of members (Membership objects) in
     *         the club.
     */
    public int numberOfMembers()
    {
        return memberships.size();
    }

    /**
     * Determine the number of members who joinded in the given month.
     * @param month The month we are interested in.
     * @return The number of members who joined in that month.
     */
    public int joinedInMonth(int month) throws IllegalArgumentException
    {
        int count = 0;

        if(month < 1 || month > 12) {
            throw new IllegalArgumentException(
                "Month " + month + " out of range. Must be in the range 1 ... 12");
        }

        for(Membership member : memberships )
        {
            if(member.getMonth() == month)
            {
                count++;
            }
        }

        return count;
    }

    /**
     * Remove from the club's collection all members who
     * joined in the given month, and return them stored
     * in a separate collection object.
```

```
67          * @param month The month of the membership.
68          * @param year The year of the membership.
69          * @return The members who joined in the given month and year
70          */
71        public ArrayList<Membership> purge(int month, int year)
72        {
73            if(month < 1 || month > 12) {
74                System.out.println("Month " + month + " out of range. Must be in the range 1
                      ... 12");
75                return new ArrayList<Membership>();
76            }
77
78            Iterator<Membership> it = memberships.iterator();
79
80            ArrayList<Membership> toReturn = new ArrayList<Membership>();
81
82            while(it.hasNext())
83            {
84                Membership m = it.next();
85
86                if(m.getMonth() == month && m.getYear()==year)
87                {
88                    toReturn.add(m);
89                    it.remove();
90                }
91            }
92
93            return toReturn;
94        }
95    }
```

Exercises 4.48-4.52

Answer for Exercise 4.50: The *getLot* method is entirely based on the array's size, thus removing an item will decrease the size of the array in such a way that the *lotNumber* will no longer correspond to the actual *lot* number.

```
Lot selectedLot = lots.get(lotNumber - 1);
```

If we instead create a for loop that will check individual lots and compare to the *lotNumber* requested, the size of the array (which works kind of like an internal counter) will no longer matter.

The Action class is shown below:

```
1   import java.util.ArrayList;
2   import java.util.Iterator;
3
4   public class Auction
5   {
6       // The list of Lots in this auction.
7       private ArrayList<Lot> lots;
8       // The number that will be given to the next lot entered
9       // into this auction.
10      private int nextLotNumber;
11
12      /**
13       * Create a new auction.
14       */
15      public Auction()
16      {
17          lots = new ArrayList<Lot>();
18          nextLotNumber = 1;
19      }
20
21
22      public void enterLot(String description)
23      {
24          lots.add(new Lot(nextLotNumber, description));
```

```
25          nextLotNumber++;
26      }
27
28
29      public void showLots()
30      {
31          for(Lot lot : lots) {
32              System.out.println(lot.toString());
33          }
34      }
35
36
37      public void makeABid(int lotNumber, Person bidder, long value)
38      {
39          Lot selectedLot = getLot(lotNumber);
40          if(selectedLot != null) {
41              Bid bid = new Bid(bidder, value);
42              boolean successful = selectedLot.bidFor(bid);
43              if(successful) {
44                  System.out.println("The␣bid␣for␣lot␣number␣" +
45                                     lotNumber + "␣was␣successful.");
46              }
47              else {
48                  // Report which bid is higher.
49                  Bid highestBid = selectedLot.getHighestBid();
50                  System.out.println("Lot␣number:␣" + lotNumber +
51                                     "␣already␣has␣a␣bid␣of:␣" +
52                                     highestBid.getValue());
53              }
54          }
55      }
56
57
58      public Lot getLot(int lotNumber)
59      {
60          if((lotNumber >= 1) && (lotNumber < nextLotNumber)) {
61
62
63              for(Lot lot : lots)
64              {
65                  if(lot.getNumber() == lotNumber)
66                  {
67                      return lot;
68                  }
69              }
70
71              return null;
72
73          }
74          else {
75              System.out.println("Lot␣number:␣" + lotNumber +
76                                 "␣does␣not␣exist.");
77              return null;
78          }
79      }
80
81      public void close()
82      {
83          for(Lot lot : lots)
84          {
85              Bid highestBid = lot.getHighestBid();
86
87              if(highestBid != null)
88              {
89                  System.out.println(lot.toString() + "␣by␣" + highestBid.getBidder().
                        getName());
90              }
91              else
92              {
93                  System.out.println(lot.toString() + "␣does␣not␣have␣any␣bidders");
```

```
            }
        }
    }

    public ArrayList<Lot> getUnsold()
    {
        ArrayList<Lot> unsoldLots = new ArrayList<Lot>();

        int count = 0;

        for(Lot lot : this.lots)
        {
            if(lot.getHighestBid() == null)
            {
                unsoldLots.add(this.lots.get(count));
            }
            count++;
        }

        return unsoldLots;
    }

    /**
     * Remove the lot with the given lot number
     * @param number The number of the lot to be removed.
     * @return The lot with the given number, or null if there is no such lot.
     */
    public Lot removeLot(int number)
    {
        if((number >= 1) && (number < nextLotNumber)) {

            Iterator<Lot> it = lots.iterator();

            while(it.hasNext())
            {
                Lot lot = it.next();

                if(lot.getNumber() == number)
                {
                    lots.remove(lot);
                    return lot;
                }
            }

            return null;

        }
        else {
            System.out.println("Lot␣number:␣" + number +
                               "␣does␣not␣exist.");
            return null;
        }
    }

}
```

Exercises 4.56-4.59

The **StockManager** class is illustrated below:

```
1  import java.util.ArrayList;
2
3  /**
4   * Manage the stock in a business.
5   * The stock is described by zero or more Products.
6   *
7   * @author (Artem Los)
8   * @version (2014.09.29)
9   */
10 public class StockManager
11 {
12     // A list of the products.
13     private ArrayList<Product> stock;
14
15     /**
16      * Initialise the stock manager.
17      */
18     public StockManager()
19     {
20         stock = new ArrayList<Product>();
21     }
22
23     /**
24      * Add a product to the list.
25      * @param item The item to be added.
26      */
27     public void addProduct(Product item)
28     {
29         stock.add(item);
30     }
31
32     /**
33      * Receive a delivery of a particular product.
34      * Increase the quantity of the product by the given amount.
35      * @param id The ID of the product.
36      * @param amount The amount to increase the quantity by.
37      */
38     public void delivery(int id, int amount)
39     {
40         Product product = findProduct(id);
41
42         if(product != null)
43         {
44             product.increaseQuantity(amount);
45         }
46     }
47
48     /**
49      * Try to find a product in the stock with the given id.
50      * @return The identified product, or null if there is none
51      *         with a matching ID.
52      */
53     public Product findProduct(int id)
54     {
55         for(Product product : stock)
56         {
57             if(product.getID() == id)
58             {
59                 return product;
60             }
61         }
62         return null;
63     }
64
65     /**
66      * Locate a product with the given ID, and return how
```

```
67          * many of this item are in stock. If the ID does not
68          * match any product, return zero.
69          * @param id The ID of the product.
70          * @return The quantity of the given product in stock.
71          */
72         public int numberInStock(int id)
73         {
74             Product product = findProduct(id);
75 
76             if(product != null)
77             {
78                 return product.getQuantity();
79             }
80 
81             return 0;
82         }
83 
84         /**
85          * Print details of all the products.
86          */
87         public void printProductDetails()
88         {
89             for(Product product : stock)
90             {
91                 System.out.println(product.toString());
92             }
93 
94         }
95 }
```