```
// doublyLinked.java
// demonstrates doubly-linked list
// to run this program: C>java DoublyLinkedApp
////////////////////////////////////////////////////////////////
class Link
   {
   public long dData;                // data item
   public Link next;                 // next link in list
   public Link previous;             // previous link in list
// -------------------------------------------------------------
   public Link(long d)               // constructor
      { dData = d; }
// -------------------------------------------------------------
   public void displayLink()         // display this link
      { System.out.print(dData + " "); }
// -------------------------------------------------------------
   }  // end class Link
////////////////////////////////////////////////////////////////
class DoublyLinkedList
   {
   private Link first;               // ref to first item
   private Link last;                // ref to last item
// -------------------------------------------------------------
   public DoublyLinkedList()         // constructor
      {
      first = null;                  // no items on list yet
      last = null;
      }
// -------------------------------------------------------------
   public boolean isEmpty()          // true if no links
      { return first==null; }
// -------------------------------------------------------------
   public void insertFirst(long dd)  // insert at front of list
      {
      Link newLink = new Link(dd);   // make new link

      if( isEmpty() )                // if empty list,
         last = newLink;             // newLink <-- last
      else
         first.previous = newLink;   // newLink <-- old first
      newLink.next = first;          // newLink --> old first
      first = newLink;               // first --> newLink
      }
// -------------------------------------------------------------
   public void insertLast(long dd)   // insert at end of list
      {
      Link newLink = new Link(dd);   // make new link
      if( isEmpty() )                // if empty list,
         first = newLink;            // first --> newLink
      else
         {
         last.next = newLink;        // old last --> newLink
         newLink.previous = last;    // old last <-- newLink
         }
      last = newLink;                // newLink <-- last
      }
// -------------------------------------------------------------
   public Link deleteFirst()         // delete first link
      {                              // (assumes non-empty list)
      Link temp = first;
      if(first.next == null)         // if only one item
         last = null;                // null <-- last
      else
         first.next.previous = null; // null <-- old next
      first = first.next;            // first --> old next
      return temp;
```

```
      }
// ------------------------------------------------------------
   public Link deleteLast()          // delete last link
      {                              // (assumes non-empty list)
      Link temp = last;
      if(first.next == null)         // if only one item
         first = null;               // first --> null
      else
         last.previous.next = null;  // old previous --> null
      last = last.previous;          // old previous <-- last
      return temp;
      }
// ------------------------------------------------------------
                                     // insert dd just after key
   public boolean insertAfter(long key, long dd)
      {                              // (assumes non-empty list)
      Link current = first;          // start at beginning
      while(current.dData != key)    // until match is found,
         {
         current = current.next;     // move to next link
         if(current == null)
            return false;            // didn't find it
         }
      Link newLink = new Link(dd);   // make new link

      if(current==last)              // if last link,
         {
         newLink.next = null;        // newLink --> null
         last = newLink;             // newLink <-- last
         }
      else                           // not last link,
         {
         newLink.next = current.next; // newLink --> old next
                                      // newLink <-- old next
         current.next.previous = newLink;
         }
      newLink.previous = current;    // old current <-- newLink
      current.next = newLink;        // old current --> newLink
      return true;                   // found it, did insertion
      }
// ------------------------------------------------------------
   public Link deleteKey(long key)   // delete item w/ given key
      {                              // (assumes non-empty list)
      Link current = first;          // start at beginning
      while(current.dData != key)    // until match is found,
         {
         current = current.next;     // move to next link
         if(current == null)
            return null;             // didn't find it
         }
      if(current==first)             // found it; first item?
         first = current.next;       // first --> old next
      else                           // not first
                                     // old previous --> old next
         current.previous.next = current.next;

      if(current==last)              // last item?
         last = current.previous;    // old previous <-- last
      else                           // not last
                                     // old previous <-- old next
         current.next.previous = current.previous;
      return current;                // return value
      }
// ------------------------------------------------------------
   public void displayForward()
      {
```

```
      System.out.print("List (first-->last): ");
      Link current = first;          // start at beginning
      while(current != null)         // until end of list,
         {
         current.displayLink();      // display data
         current = current.next;     // move to next link
         }
      System.out.println("");
      }
// -------------------------------------------------------------
   public void displayBackward()
      {
      System.out.print("List (last-->first): ");
      Link current = last;           // start at end
      while(current != null)         // until start of list,
         {
         current.displayLink();      // display data
         current = current.previous; // move to previous link
         }
      System.out.println("");
      }
// -------------------------------------------------------------
   }  // end class DoublyLinkedList
////////////////////////////////////////////////////////////////
class DoublyLinkedApp
   {
   public static void main(String[] args)
      {                              // make a new list
      DoublyLinkedList theList = new DoublyLinkedList();

      theList.insertFirst(22);      // insert at front
      theList.insertFirst(44);
      theList.insertFirst(66);

      theList.insertLast(11);       // insert at rear
      theList.insertLast(33);
      theList.insertLast(55);

      theList.displayForward();     // display list forward
      theList.displayBackward();    // display list backward

      theList.deleteFirst();        // delete first item
      theList.deleteLast();         // delete last item
      theList.deleteKey(11);        // delete item with key 11

      theList.displayForward();     // display list forward

      theList.insertAfter(22, 77);  // insert 77 after 22
      theList.insertAfter(33, 88);  // insert 88 after 33

      theList.displayForward();     // display list forward
      }  // end main()
   }  // end class DoublyLinkedApp
////////////////////////////////////////////////////////////////
```