```
1  package Assign_4;
2
3
4  import BasicIO.*;
5
6  /** This class creates a BasicForm driven application which reads a datafile of
   products and adds them
7    * to a list. The user enters a target amount they would like to "purchase" and
   the program recursively searches
8    * the list until a solution is found. It then adds the found items to a second
   list and displays that list
9    * to the user.
10   *
11   * @author S. Fenwick
12   *
13   * @version 1.0 (March. 2017)
   */
14
15 public class Knapsack{
16
17   public ASCIIDataFile file;
18   private BasicForm form;
19   private Node proList; //list of products
20   private Node selList; //list of products that have been selected
21   private int total; //the total of all product prices
22   private int finalTotal; //if request is greater than this total, there is not
   enough product to buy
23   private int request; //the number submitted by the user
24   private int search; //the result of subtracting the request from the price of the
   product
25   private int selListTotal; //the total value of the selected items
26
27
28   /* This constructor creates a BasicForm and reads in an ASCIIDataFile. It
   recursively searches the list
29    * looking for values that add up to what the user has requested.  */
30   public Knapsack(){
31
32     int button; //button pressed
33
34     form = new BasicForm("Browse", "Buy", "Quit");
35     proList = null;
36     selList = null;
37
38     loadProducts();
39     setUpForm();
40
41
42     for( ; ; ){
43       button = form.accept();
44       if(button == 2) break; //Quit
45       switch(button) {
46         case 0:{ //Browse
47           browse();
48           break;
49         }
50         case 1: { //Buy
51
52           Node p;
53           p = proList;
54
55           request = form.readInt("target");
56
57           if(request == 0){
58             form.writeString("display", "Nothing here is free.");
59             form.newLine("display");
60             form.writeString("display", "-------");
61             form.newLine("display");
62             form.writeInt("target", finalTotal);
63             break;
```

```
64            }
65
66            while( p != null ){
67
68              selList = buy(p, request);
69              if(selListTotal == request){
70                break;
71              }
72              selListTotal = 0;
73              selList = null;
74              p = p.next;
75
76            }
77
78            if(selList == null){
79              form.writeString("display", "No product selection to purchse.");
80              form.newLine("display");
81              form.writeString("display", "-------");
82              form.newLine("display");
83              form.writeInt("target", finalTotal);
84              break;
85            }
86
87            form.writeString("display", "Products Selected: ");
88            form.newLine("display");
89            print();
90            selListTotal = 0;
91            selList = null;
92            break;
93          }
94
95        }
96
97      }
98
99      form.close();
100
101   } //constructor
102
103   /* This method takes in the list from the constructor and searches through it
    looking for
104    * values that add up to the requested amount by the user.
105    *
106    * @param aNode the node being looked at
107    * @param find the value of (requeset - price of aNode) or what the program is
    searching for */
108
109   public Node buy(Node aNode, int find){
110
111     if( aNode == null ){ //Kick
112       return null;
113     }
114     else if ((find - aNode.item.getPrice()) < 0){ //Deeper
115       buy(aNode.next, search);
116       return null;
117     }
118     else if (finalTotal < request){ //Kick
119       form.writeString("display", "There is not enough product to match that
    request.");
120       form.newLine("display");
121       form.writeString("display", "--------");
122       form.newLine("display");
123       return null;
124     }
125     else if ((find - aNode.item.getPrice()) == 0){ //Kick
126       selListTotal = selListTotal + aNode.item.getPrice();
127       return new Node (aNode.item, null);
128     }
129     else{ //Deeper
130       search = find - aNode.item.getPrice();
```

```
131       selListTotal = selListTotal + aNode.item.getPrice();
132       return new Node (aNode.item , buy(aNode.next, search));
133     }
134
135   } //buy
136
137
138   /* This method browses the 'store' (datafile) and writes the file to the text
    area on the
139    * BasicForm. It calculates the total cost of all the products. */
140
141   public void browse(){
142
143     Node  p;
144
145     p = proList;
146
147     while ( p != null ) {
148
149       form.writeString("display", p.item.getName());
150       form.writeInt("display", p.item.getPrice());
151       form.newLine("display");
152       total = total + p.item.getPrice();
153       p = p.next;
154
155     };
156
157     form.writeInt("target", total);
158     form.writeString("display", "--------");
159     form.newLine("display");
160     finalTotal = total;
161     total = 0; //resetting total so total does not continually add
162
163   } //browse
164
165
166   /* This method prints the list of selected items onto the text area of the
    BasicForm. */
167
168   private void print(){
169
170     Node p;
171
172     p = selList;
173
174     while ( p != null ){
175
176       form.writeString("display", p.item.getName());
177       form.writeInt("display", p.item.getPrice());
178       form.newLine("display");
179       p = p.next;
180     }
181
182     form.writeString("display", "--------");
183     form.newLine("display");
184     form.writeInt("target", finalTotal);
185
186   } //print
187
188
189   /* This method adds a canvas and text field to the BasicForm 'form'. */
190
191   public void setUpForm(){
192
193     form.addTextArea("display", "Status", 25, 75 );
194     form.addTextField("target", "Target", 5, 10, 490);
195
196   } //setUpForm
197
198   /* This methods loads the products from the datafile. */
```

```
199
200   public void loadProducts(){
201
202     ASCIIDataFile file; //file of product info
203     Product aProduct;
204
205     file = new ASCIIDataFile();
206     for( ; ; ){
207       aProduct = new Product(file);
208       if(file.isEOF() ) break;
209       addList(aProduct);
210     }
211
212   } //loadProducts
213
214
215   /* This method adds the products to a linked list.
216    *
217    * @param aProduct the product being added  */
218
219   private void addList(Product aProduct){
220
221     Node p;
222     Node q;
223
224     q = null;
225     p = proList;
226     while( p != null ) {
227       q = p;
228       p = p.next;
229     };
230     if( q ==null ){
231       proList = new Node(aProduct, null);
232     }
233     else {
234       q.next = new Node(aProduct, null);
235     };
236   } //addList
237
238
239   public static void main (String args[]){ Knapsack k = new Knapsack();}
240
241 } //Knapsack
```