```
// joi/1/estore/EStore.java
//
//
// Copyright 2003 Bill Campbell and Ethan Bolker

/**
 * An EStore object simulates the behavior of a simple on line
 * shopping web site.
 *
 * It contains a Terminal object to model the customer's browser
 * and several Item objects a customer can add to her ShoppingCart.
 *
 * @version 1
 */

public class EStore
{
    private String storeName = "Virtual Minimal Minimall";

    // Use a Terminal object to communicate with customers.
    private Terminal browser = new Terminal();

    // The store stocks two kinds of Items.
    private Item widget = new Item(10); //  widgets cost $10
    private Item gadget = new Item(13); //  gadgets cost $13

    private String selectionList = "(gadget, widget, checkout)";

    /**
     * Visit this EStore.
     *
     * Loop allowing visitor to select items to add to her
     * ShoppingCart.
     */

    public void visit()
    {
        // Create a new, empty ShoppingCart.
        ShoppingCart basket = new ShoppingCart();

        // Print a friendly welcome message.
        browser.println( "Welcome to " + storeName );

        // Change to false when customer is ready to leave:
        boolean stillShopping = true;

        while ( stillShopping ) {
            Item nextPurchase = selectItem();
            if ( nextPurchase == null ) {
                stillShopping = false;
            }
            else {
                basket.add( nextPurchase );
            }
        }
        int numberPurchased = basket.getCount();
```



```
        int totalCost       = basket.getCost();
        browser.println("We are shipping " + numberPurchased + " Items");
        browser.println("and charging your account $" + totalCost);
        browser.println("Thank you for shopping at " + storeName);
    }

    // Discover what the customer wants to do next:
    //  send browser a message to get customer input
    //  examine response to make a choice
    //  if response makes no sense give customer another chance

    private Item selectItem()
    {
        String itemName =
            browser.readWord("Item " + selectionList + ":");

        if ( itemName.equals("widget")) {
            return widget;
        }
        else if ( itemName.equals("gadget")) {
            return gadget;
        }
        else if ( itemName.equals("checkout" )) {
            return null;
        }
        else {
            browser.println( "No item named " +
                             itemName + "; try again" );
            return selectItem();  // try again
        }
    }

    /**
     *  The EStore simulation program begins here when the user
     *  issues the command <code>java EStore</code>.
     */

    public static void main( String[] args )
    {
        // Print this to simulate delay while browser finds store
        System.out.println("connecting ...");

        // Create the EStore object.
        EStore webSite = new EStore();

        // Visit it.
        webSite.visit();
    }
} // end of class EStore
```

```
 1  // joi/1/estore/Item.java
 2  //
 3  //
 4  // Copyright 2003 Bill Campbell and Ethan Bolker
 5
 6  /**
 7   * An Item models an object that might be stocked in a store.
 8   * Each Item has a cost.
 9   *
10   * @version 1
11   */
12
13  public class Item
14  {
15      private int cost;
16
17      /**
18       *  Construct an Item object.
19       *
20       *  @param itemCost the cost of this Item.
21       */
22
23      public Item( int itemCost )
24      {
25          cost = itemCost;
26      }
27
28      /**
29       * How much does this Item cost?
30       *
31       * @return the cost.
32       */
33
34      public int getCost()
35      {
36          return cost;
37      }
38  }
```

```
 1  // joi/1/estore/ShoppingCart.java
 2  //
 3  //
 4  // Copyright 2003 Bill Campbell and Ethan Bolker
 5
 6  /**
 7   * A ShoppingCart keeps track of a customer's purchases.
 8   *
 9   * @see EStore
10   * @version 1
11   */
12
13  public class ShoppingCart
14  {
15      private int count; // number of Items in this ShoppingCart
16      private int cost;  // cost of Items in this ShoppingCart
17
18      /**
19       *  Construct a new empty ShoppingCart.
20       */
21
22      public ShoppingCart()
23      {
24          count = 0;
25          cost  = 0;
26      }
27
28      /**
29       * When this ShoppingCart is asked to add an Item to itself
30       * it updates its count field and then updates its cost
31       * field by sending the Item a getCost message.
32       *
33       * @param purchase the Item being added to this ShoppingCart.
34       */
35
36      public void add( Item purchase )
37      {
38          count++; // Java idiom for count = count + 1;
39          cost = cost + purchase.getCost();
40      }
41
42      /**
43       * What happens when this ShoppingCart is asked how many
44       * Items it contains.
45       *
46       * @return the count of Items.
47       */
48
49      public int getCount()
50      {
51          return count;
52      }
53
54      /**
55       * What happens when this ShoppingCart is asked the total
56       * cost of the Items it contains.
```



```
57       *
58       * @return the total cost.
59       */
60
61      public int getCost()
62      {
63          return cost;
64      }
65  }
```