```java
// 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();
```

```java
        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
```

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

/**
 * An Item models an object that might be stocked in a store.
 * Each Item has a cost.
 *
 * @version 1
 */

public class Item
{
    private int cost;

    /**
     *  Construct an Item object.
     *
     *  @param itemCost the cost of this Item.
     */

    public Item( int itemCost )
    {
        cost = itemCost;
    }

    /**
     * How much does this Item cost?
     *
     * @return the cost.
     */

    public int getCost()
    {
        return cost;
    }
}
```

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

/**
 * A ShoppingCart keeps track of a customer’s purchases.
 *
 * @see EStore
 * @version 1
 */

public class ShoppingCart
{
    private int count; // number of Items in this ShoppingCart
    private int cost;  // cost of Items in this ShoppingCart

    /**
     *  Construct a new empty ShoppingCart.
     */

    public ShoppingCart()
    {
        count = 0;
        cost  = 0;
    }

    /**
     * When this ShoppingCart is asked to add an Item to itself
     * it updates its count field and then updates its cost
     * field by sending the Item a getCost message.
     *
     * @param purchase the Item being added to this ShoppingCart.
     */

    public void add( Item purchase )
    {
        count++; // Java idiom for count = count + 1;
        cost = cost + purchase.getCost();
    }

    /**
     * What happens when this ShoppingCart is asked how many
     * Items it contains.
     *
     * @return the count of Items.
     */

    public int getCount()
    {
        return count;
    }

    /**
     * What happens when this ShoppingCart is asked the total
     * cost of the Items it contains.
```

```java
     *
     * @return the total cost.
     */

    public int getCost()
    {
        return cost;
    }
}
```