Programming Project # 1: Willy Loman’s Revenge
Date Due: Wednesday 3 February 2016

The Acme Sales Corporation would like to automate the order processing procedures used by its sales force. You have been hired to do the programming.

Acme has ten items for sale. The unit prices of each item are as follows:

<table>
<thead>
<tr>
<th>Item</th>
<th>Unit Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Toothpaste</td>
<td>$10.62</td>
</tr>
<tr>
<td>Fish sticks</td>
<td>14.89</td>
</tr>
<tr>
<td>Peanut butter</td>
<td>13.20</td>
</tr>
<tr>
<td>Zinc bushings</td>
<td>16.55</td>
</tr>
<tr>
<td>Axolotl</td>
<td>18.62</td>
</tr>
<tr>
<td>Shoe polish</td>
<td>9.47</td>
</tr>
<tr>
<td>Green eggs</td>
<td>6.58</td>
</tr>
<tr>
<td>Filets of mulberry</td>
<td>18.32</td>
</tr>
<tr>
<td>Doughnut molds</td>
<td>12.15</td>
</tr>
<tr>
<td>Cadmium syrup</td>
<td>3.98</td>
</tr>
</tbody>
</table>

Your program should ask the sales person to enter the number of each item sold. It should then print a neatly formatted table showing (for each item):

- the item number,
- the unit price of the item,
- the number of this item sold,
- the “extended amount” (i.e., the unit price multiplied by the number of items sold) for this item

It should then print the grand total of the order.

A sample run is found on the next page. Note that your program must not inquire about the unit prices of the sales items, but only the number of each item sold. The unit prices should be built into the program, but in a nice way (i.e., it should be easy to modify your program to reflect a change in the unit prices).

You are to turn in a typescript (produced by the script program or the photo program), containing

- a `cat` of your Java program code,
- the command `rm *.class`
- the `javac` line used to compile the program, and
- the results of one sample run, using the sample data below.

The reason for “`rm *.class`” is to force `javac` to actually compile something.

Notes:

1. You should do this project, along with all projects for this course, in a proper subdirectory of your `private` directory. See the Project 0 description, which may be found at

   [http://www.dsm.fordham.edu/~agw/client-server/projects/proj0.html](http://www.dsm.fordham.edu/~agw/client-server/projects/proj0.html)
for further discussion.

2. Use **Javadoc** for documentation. Put the resulting documentation tree in your

    ~/public_html/client-server/projects/proj1
directory. Go to

to see my documentation.

3. Use the **Scanner** class for input. Don’t forget to put

    ```java
    import java.util.*;
    ```

    near the beginning of the file.

4. Part of this problem involves getting the table of results to come out right. To do this you’ll need to address two problems.

   (a) The first is to print all monetary amounts as showing exactly two significant figures (the “cents” amount). Moreover, an amount that is less than one dollar should be printed with a leading zero (e.g., 0.58 instead of .58). You can do this by using the `java.text.DecimalFormat` class. First, make sure to put

       ```java
       import java.text.*;
       ```

       near the beginning of the file. Next, you need to put

       ```java
       static DecimalFormat moneyFormat = new DecimalFormat("0.00");
       ```

       into your `public class` definition (typically, near the beginning of same). Having done all this, you can do things such as

       ```java
       System.out.println("\$" + moneyFormat.format(grandTotal));
       ```

       and the like.

   (b) The second is that you will need to left-pad all numerical amounts in a given column (with blanks), so that they all have the same width. To do this, you’ll need to to something like

       ```java
       System.out.print(pad(Integer.toString(foo), 8));
       ```

       where `pad` is a programmer-defined function having the prototype

       ```java
       public static String pad(String str, int width);
       ```

       that returns the catenation of `width - str.length()` blanks with `str`. Note that to make things look right, you’ll need to do left-padding (for the numerical values) and right-padding (for the item names). You can either use two different functions (each padding in a different direction), or one function that takes an additional parameter (the direction of padding).

       Alternatively, you can use `java.util.Formatter`, as discussed in class.

5. Write your program so that it can easily handle a change in the inventory (e.g., additional items, fewer items, change in price). You can do this by using an array (or two) to represent the inventory.
Here’s the sample run. Please use the input data indicated!!

sobolev@dsm:proj1$ java Salesman
Enter the number sold for each item:

Toothpaste? 4  
Fish sticks? 8  
Peanut butter? 3  
Zinc bushings? 0  
Axolotl? 12  
Shoe polish? 7  
Green eggs? 2  
Filets of mulberry? 1  
Doughnut molds? 8  
Cadmium syrup? 3

<table>
<thead>
<tr>
<th>Item</th>
<th>Unit Price</th>
<th>Number Sold</th>
<th>Extended Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Toothpaste</td>
<td>10.62</td>
<td>4</td>
<td>42.48</td>
</tr>
<tr>
<td>Fish sticks</td>
<td>14.89</td>
<td>8</td>
<td>119.12</td>
</tr>
<tr>
<td>Peanut butter</td>
<td>13.20</td>
<td>3</td>
<td>39.60</td>
</tr>
<tr>
<td>Zinc bushings</td>
<td>16.55</td>
<td>0</td>
<td>0.00</td>
</tr>
<tr>
<td>Axolotl</td>
<td>18.62</td>
<td>12</td>
<td>223.44</td>
</tr>
<tr>
<td>Shoe polish</td>
<td>9.47</td>
<td>7</td>
<td>66.29</td>
</tr>
<tr>
<td>Green eggs</td>
<td>6.58</td>
<td>2</td>
<td>13.16</td>
</tr>
<tr>
<td>Filets of mulberry</td>
<td>18.32</td>
<td>1</td>
<td>18.32</td>
</tr>
<tr>
<td>Doughnut molds</td>
<td>12.15</td>
<td>8</td>
<td>97.20</td>
</tr>
<tr>
<td>Cadmium syrup</td>
<td>3.98</td>
<td>3</td>
<td>11.94</td>
</tr>
</tbody>
</table>

Order grand total: $631.55