```java
import components.map.Map;
import components.map.Map1L;
import components.simplereader.SimpleReader;
import components.simplereader.SimpleReader1L;
import components.simplewriter.SimpleWriter;
import components.simplewriter.SimpleWriter1L;

/**
 * Simple pizza order manager: inputs orders from a file and computes and
 * displays the total price for each order.
 *
 * @author Put your name here
 *
 */
public final class PizzaOrderManager {

    /**
     * Private constructor so this utility class cannot be instantiated.
     */
    private PizzaOrderManager() {
    }

    /**
     * Inputs a "menu" of words (items) and their prices from the given file and
     * stores them in the given {@code Map}.
     *
     * @param fileName
     *            the name of the input file
     * @param priceMap
     *            the word -> price map
     * @replaces priceMap
     * @requires <pre>
     * [file named fileName exists but is not open, and has the
     *  format of one "word" (unique in the file) and one price (in cents)
     *  per line, with word and price separated by ','; the "word" may
     *  contain whitespace but not ',']
     * </pre>
     * @ensures [priceMap contains word -> price mapping from file fileName]
     */
    private static void getPriceMap(String fileName,
            Map<String, Integer> priceMap) {
        assert fileName != null : "Violation of: fileName is not null";
        assert priceMap != null : "Violation of: priceMap is not null";

        SimpleReader file = new SimpleReader1L(fileName);

        while (!file.atEOS()) {
            String str = file.nextLine();

            String s1 = "";
            String s2 = "";

            int comma = str.indexOf(",");

            s1 = str.substring(0, comma);
            s2 = str.substring(comma + 1, str.length());

```

```
58              int price = Integer.parseInt(s2);
59              priceMap.add(s1, price);
60          }
61
62          file.close();
63      }
64
65      /**
66       * Input one pizza order and compute and return the total price.
67       *
68       * @param input
69       *            the input stream
70       * @param sizePriceMap
71       *            the size -> price map
72       * @param toppingPriceMap
73       *            the topping -> price map
74       * @return the total price (in cents)
75       * @updates input
76       * @requires <pre>
77       * input.is_open and
78       * [input.content begins with a pizza order consisting of a size
79       *  (something defined in sizePriceMap) on the first line, followed
80       *  by zero or more toppings (something defined in toppingPriceMap)
81       *  each on a separate line, followed by an empty line]
82       * </pre>
83       * @ensures <pre>
84       * input.is_open and
85       * #input.content = [one pizza order (as described
86       *              in the requires clause)] * input.content and
87       * getOneOrder = [total price (in cents) of that pizza order]
88       * </pre>
89       */
90      private static int getOneOrder(SimpleReader input,
91              Map<String, Integer> sizePriceMap,
92              Map<String, Integer> toppingPriceMap) {
93          assert input != null : "Violation of: input is not null";
94          assert sizePriceMap != null : "Violation of: sizePriceMap is not null";
95          assert toppingPriceMap != null : "Violation of: toppingPriceMap is not null";
96          assert input.isOpen() : "Violation of: input.is_open";
97
98          int total = 0;
99          String str = "temp";
100
101         while (!str.equals("")) {
102             str = input.nextLine();
103
104             if (sizePriceMap.hasKey(str)) {
105                 int price = sizePriceMap.value(str);
106                 total += price;
107             }
108
109             if (toppingPriceMap.hasKey(str)) {
110                 int extras = toppingPriceMap.value(str);
111                 total += extras;
112             }
113         }
114         return total;
```

```
115     }
116
117     /**
118      * Output the given price formatted in dollars and cents.
119      *
120      * @param output
121      *            the output stream
122      * @param price
123      *            the price to output
124      * @updates output
125      * @requires output.is_open = true and 0 <= price
126      * @ensures <pre>
127      * output.is_open and
128      * output.content = #output.content *
129      *  [display of price, where price is in cents but
130      *   display is formatted in dollars and cents]
131      * </pre>
132      */
133     private static void putPrice(SimpleWriter output, int price) {
134         assert output != null : "Violation of: output is not null";
135         assert output.isOpen() : "Violation of: output.is_open";
136         assert 0 <= price : "Violation of: 0 <= price";
137
138         output.println("$" + price / 100 + "." + price % 100 );
139
140     }
141
142     /**
143      * Main method.
144      *
145      * @param args
146      *            the command line arguments
147      */
148     public static void main(String[] args) {
149         SimpleReader in = new SimpleReader1L("orders.txt");
150         SimpleWriter out = new SimpleWriter1L();
151         Map<String, Integer> sizeMenu = new Map1L<String, Integer>();
152         Map<String, Integer> toppingMenu = new Map1L<String, Integer>();
153         int orderNumber = 1;
154         /*
155          * Get menus of sizes with prices and toppings with prices
156          */
157         getPriceMap("sizes.txt", sizeMenu);
158         getPriceMap("toppings.txt", toppingMenu);
159         /*
160          * Output heading for report of pizza orders
161          */
162         out.println();
163         out.println("Order");
164         out.println("Number Price");
165         out.println("------ ------");
166         /*
167          * Process orders, one at a time, from input file
168          */
169         while (!in.atEOS()) {
170             int price = getOneOrder(in, sizeMenu, toppingMenu);
171             out.print(orderNumber + "       ");
```

```
172             putPrice(out, price);
173             out.println();
174             orderNumber++;
175         }
176         out.println();
177         /*
178          * Close input and output streams
179          */
180         in.close();
181         out.close();
182     }
183 
184 }
```