```java
/**
package edu.ncsu.csc216.product_backlog.model.backlog;

import java.util.ArrayList;

/**
 * Class that maintains a list of Products, the active or
 * current Product, and handles events from the GUI
 * @author abbyfinan
 */

public class BacklogManager {
    /** Provides the instance of this class */
    private static BacklogManager singleton;
    /** Represents the current product being modified */
    private Product currentProduct;
    /** Represents the list of products */
    private ArrayList<Product> products;

    /**
     * Private constructor for BacklogManager class
     */
    private BacklogManager() {
        products = new ArrayList<Product>();
        currentProduct = null;
    }

    /**
     * Static method that handles getting an instance of a
class
     * @return returns singleton field
     */
    public static BacklogManager getInstance() {
        if (singleton == null) {
            singleton = new BacklogManager();
        }
        return singleton;
    }

    /**
     * Private method that checks for duplicate products
     * @param product the product being duplicated
     */
    private void isDuplicateProduct(String product) {
        for (int i = 0; i < products.size(); i++) {
```

```
54              if (product.equals(products.get(i).getProductName())) {
55                  throw new IllegalArgumentException();
56              }
57          }
58      }
59
60      /**
61       * Method that handles clearing the products
62       */
63      public void clearProducts() {
64          products = new ArrayList<Product>();
65          currentProduct = null;
66      }
67
68      /**
69       * Method that handles saving the data to a file
70       * @param fileName the filename of product
71       * @throws illegalArgumentException if unable to save the file
72       */
73      public void saveToFile String fileName) {
74          if (currentProduct == null || currentProduct.getTasks().size() == 0) {
75              throw new IllegalArgumentException("Unable to save file.");
76          }
77          ProductsWriter.writeProductsToFile(fileName, products);
78      }
79
80      /**
81       * Method that handles loading data from a file
82       * @param fileName the filename of a file
83       */
84      public void loadFromFile(String fileName) {
85          ArrayList<Product> product1 = ProductsReader.readProductsFile(fileName);
86          currentProduct = product1.get(0);
87          for (Product product : product1) {
88              products.add(product);
89          }
90      }
91
92      /**
93       * Method that handles loading products
```

```
94       * @param productName the name of a product
95       */
96      public void loadProduct(String productName) {
97          Product product2 = null;
98          for (int i = 0; i < products.size(); i++) {
99              if
  (productName.equals(products.get(i).getProductName())) {
100                 product2 = products.get(i);
101             }
102         }
103         if (product2 == null) {
104             throw new IllegalArgumentException("Invalid product
  name");
105         }
106         currentProduct = product2;
107     }
108
109     /**
110      * Method that handles getting a products name
111      * @return  the current products name
112      */
113     public String getProductName() {
114         if (currentProduct == null) {
115             return null;
116         }
117         return currentProduct.getProductName();
118     }
119
120     /**
121      * Method that handles deleting a task by the Id number
122      * @param id the Id of the task
123      */
124     public void deleteTaskById(int id) {
125         if (currentProduct != null) {
126             currentProduct.deleteTaskById(id);
127         }
128     }
129
130     /**
131      * Method that handles getting a task by the Id number
132      * @param id of the task
133      * @return current product's task Id
134      */
135     public Task getTaskById(int id) {
136         if (currentProduct == null) {
```

```
137             return null;
138         }
139         return currentProduct.getTaskById(id);
140     }
141
142     /**
143      * Method that handles deleting a product
144      */
145     public void deleteProduct() {
146         if (currentProduct == null) {
147             throw new IllegalArgumentException("No product
    selected.");
148         }
149         int idx = -1;
150         for (int i = 0; i < products.size(); i++) {
151             if
    (currentProduct.getProductName().compareTo(products.get(i).getP
    roductName()) == 0) {
152                 idx = i;
153                 break;
154             }
155         }
156         products.remove(idx);
157         try {
158             loadProduct(products.get(0).getProductName());
159         } catch (IndexOutOfBoundsException e) {
160             currentProduct = null;
161         }
162
163     }
164
165     /**
166      * Method that handles editing a product
167      * @param productName the name of the product
168      * @throw IllegalArgumentException if there is an invalid
    product name
169      * @throw IllegalArgumentException if there is an invalid
    product name
170      * @throw IllegalArgumentException if there is an invalid
    product name
171      */
172     public void editProduct(String productName) {
173         if (productName == null || "".equals(productName)) {
174             throw new IllegalArgumentException("Invalid product
    name.");
```

```
175         }
176         if (currentProduct != null) {
177             int idx = -1;
178             Product product2 = null;
179             for (int i = 0; i < products.size(); i++) {
180                 if
    (currentProduct.getProductName().compareTo(products.get(i).getP
    roductName()) == 0) {
181                     idx = i;
182                     product2 = products.get(i);
183                     break;
184                 }
185             }
186             if (product2 == null) {
187                 throw new IllegalArgumentException("Invalid
    product name.");
188             }
189             isDuplicateProduct(productName);
190             product2.getProductName();
191             products.remove(idx);
192             getProductName();
193             loadProduct(productName);
194         }
195         else {
196             throw new IllegalArgumentException("Invalid product
    name.");
197         }
198 
199     }
200 
201     /**
202      * Method that handles getting the product list
203      * @return list the list of products
204      */
205     public String[] getProductList() {
206         String[] list = new String[products.size()];
207         for (int i = 0; i < products.size(); i++) {
208             list[i] = products.get(i).getProductName();
209         }
210         return list;
211     }
212 
213     /**
214      * Method that handles getting a list of tasks as an array
215      * @return array the array of tasks
```

```
216      */
217     public String[][] getTasksAsArray() {
218         if (currentProduct == null) {
219             return null;
220         }
221         String[][] array = new
    String[currentProduct.getTasks().size()][4];
222         for (int i = 0; i < currentProduct.getTasks().size();
    i++) {
223             array[i][0] =
    Integer.toString(currentProduct.getTasks().get(i).getTaskId());
224             array[i][1] =
    currentProduct.getTasks().get(i).getStateName();
225             array[i][2] =
    currentProduct.getTasks().get(i).getTypeLongName();
226             array[i][3] =
    currentProduct.getTasks().get(i).getTitle();
227         }
228         return array;
229     }
230 
231     /**
232      * Method that handles adding a product
233      * @param productName the name of the product
234      */
235     public void addProduct String productName) {
236         products.add(new Product(productName));
237         loadProduct(productName);
238     }
239 
240     /**
241      * Method that handles executing a command
242      * @param id the id of the product
243      * @param c the command
244      */
245     public void executeCommand(int id, Command c) {
246         if (currentProduct != null) {
247             currentProduct.executeCommand(id, c);
248         }
249     }
250 
251     /**
252      * Method that handles adding a task to a product
253      * @param title the title of the task
254      * @param type the type of the task
```

```
255      * @param creator the creator of the task
256      * @param notes the notes left on the task
257      */
258     public void addTaskToProduct(String title, Type type,
    String creator, String notes) {
259         Task newTask = new Task(0, title, type, creator,
    notes);
260         currentProduct.addTask(newTask);
261     }
262 
263     /**
264      * Method that handles resetting the manager
265      */
266     protected void resetManager() {
267         singleton = null;
268         this.currentProduct = null;
269         this.products = new ArrayList<Product>();
270 
271     }
272 }
273 
```