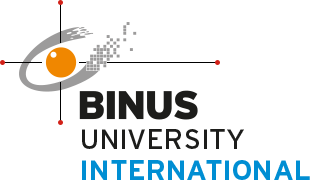
Odd Semester (2023)



**Assignment Cover Letter**

**(Individual Work)**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | |  | |  | |  |
| **Student Information**: **Surname** | | | | | **Given Names**    **Kevin Herman** | | **Student ID Number**  **2301891550** | |  |
| 1. | | **Otnieliem** |  | |  |
|  |  |  |
| **Course Code** | **: COMP6510** |  |  | | **Course Name** | | **: Programming Languages** | |  |
| **Class** | **: L2AC** |  |  | | **Name of Lecturer(s)** | | **: Jude Joseph Lamug Martinez** | |  |
|  |  |  |  | |  | |  | |
| **Major** | **: CS** |  |  | |  | |  | |  |
| **Title of Assignment**  (if any) | : **KevAndNic Cycles Inventory** | |  |  | |  | |  | |
| **Type of Assignment**    **Submission Pattern** | **: Final Project** |  |  | |  | |  | |  |
| **Due Date** | **: 20-06-20** |  |  | | **Submission Date** | | **: 20-06-20** | |  |

The assignment should meet the below requirements.

1. Assignment (hard copy) is required to be submitted on clean paper, and (soft copy) as per lecturer’s instructions.
2. Soft copy assignment also requires the signed (hardcopy) submission of this form, which automatically validates the softcopy submission.
3. The above information is complete and legible.
4. Compiled pages are firmly stapled.
5. Assignment has been copied (soft copy and hard copy) for each student ahead of the submission.

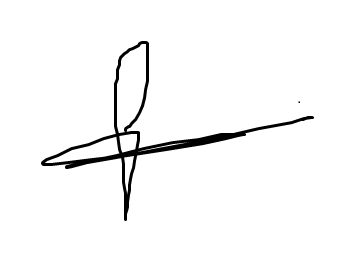
# Plagiarism/Cheating

BiNus International seriously regards all forms of plagiarism, cheating and collusion as academic offenses which may result in severe penalties, including loss/drop of marks, course/class discontinuity and other possible penalties executed by the university. Please refer to the related course syllabus for further information.

# Declaration of Originality

By signing this assignment, I understand, accept and consent to BiNus International terms and policy on plagiarism. Herewith I declare that the work contained in this assignment is my own work and has not been submitted for the use of assessment in another course or class, except where this has been notified and accepted in advance.

Signature of Student:



(Name of Student)

Kevin Herman Otnieliem

“Kev And Nic Inventory GUI Program”

Name : Kevin Herman Otnieliem

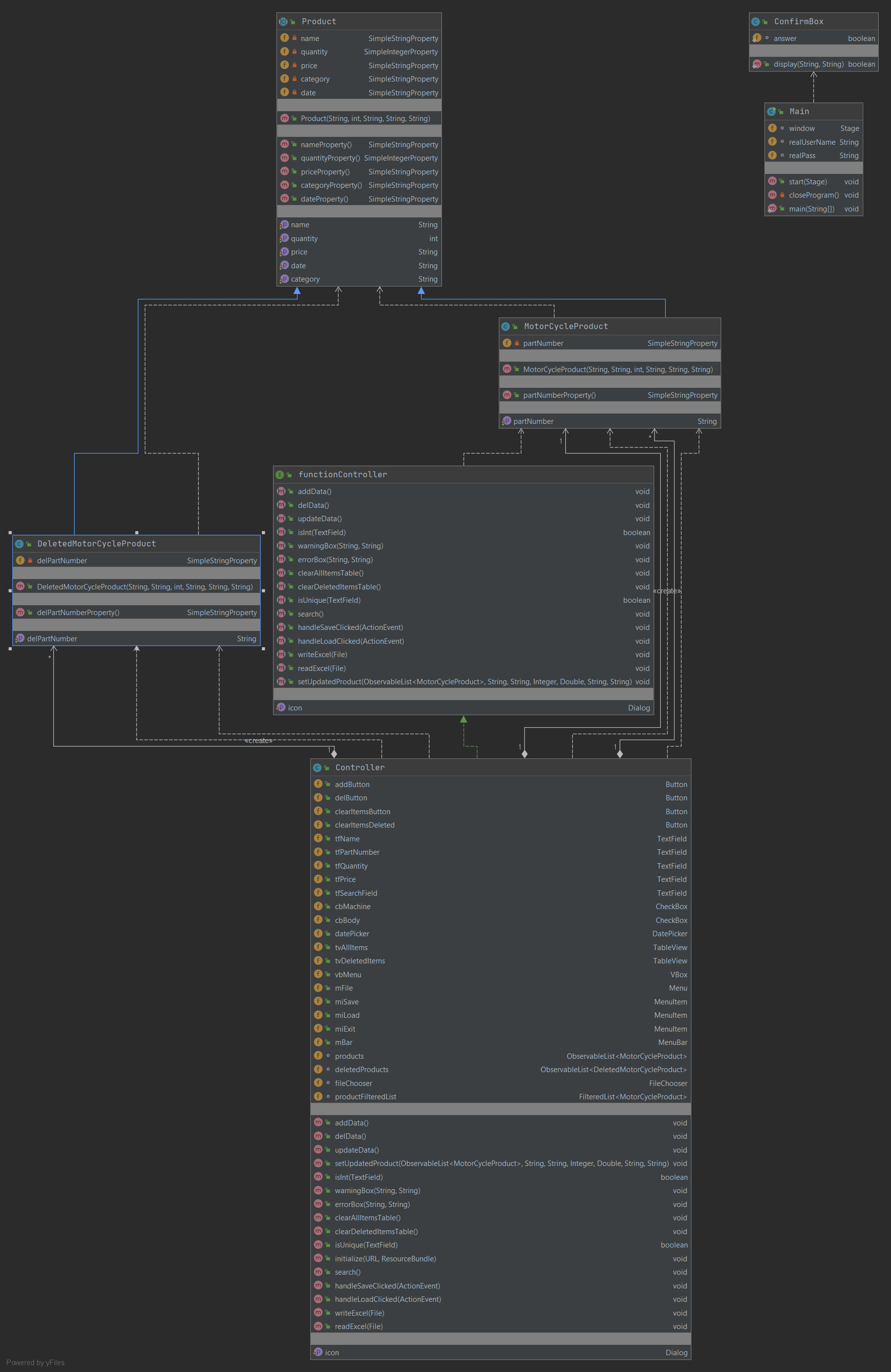
ID : 2301891550

1. **Description**

**Project specification:**

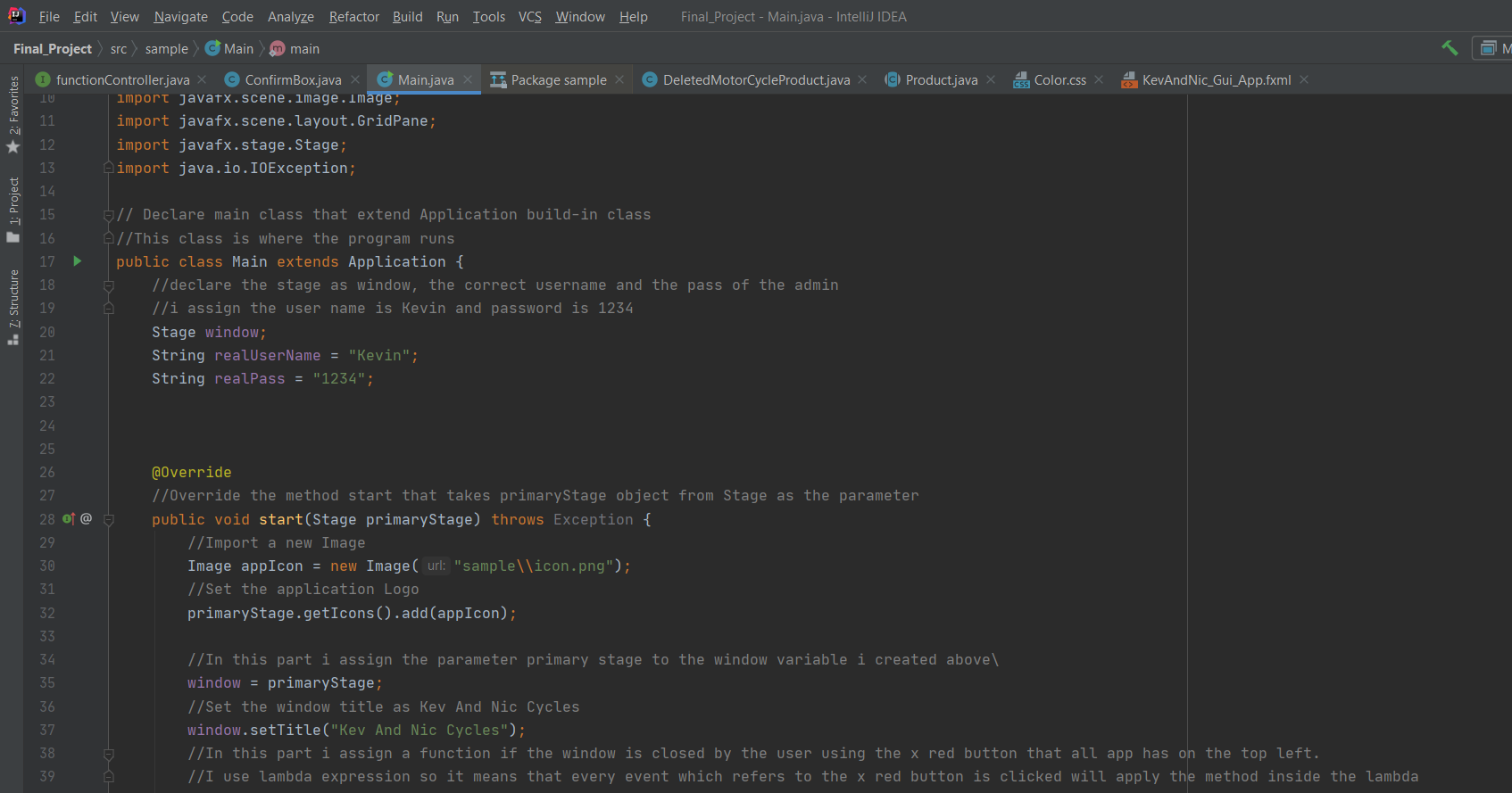
The purpose of making this program is to help my parents or other people on managing their stock of product in their company or store. The program is made using Java with Java FX. I manage the interface using Java FX graphical tool because the program is easy to use and work with. Also use CSS style sheet to make the application look interesting. And use an FXML to manage the GUI. I also use Scene Builder by Gluon to manage the GUI design and make the application looks nice and tidy. The program consist of 9 files : 5 Java class, 1 Abstract class, 1 Interface, 1 CSS StyleSheet and 1 FXML file.

1. **Solution Design**

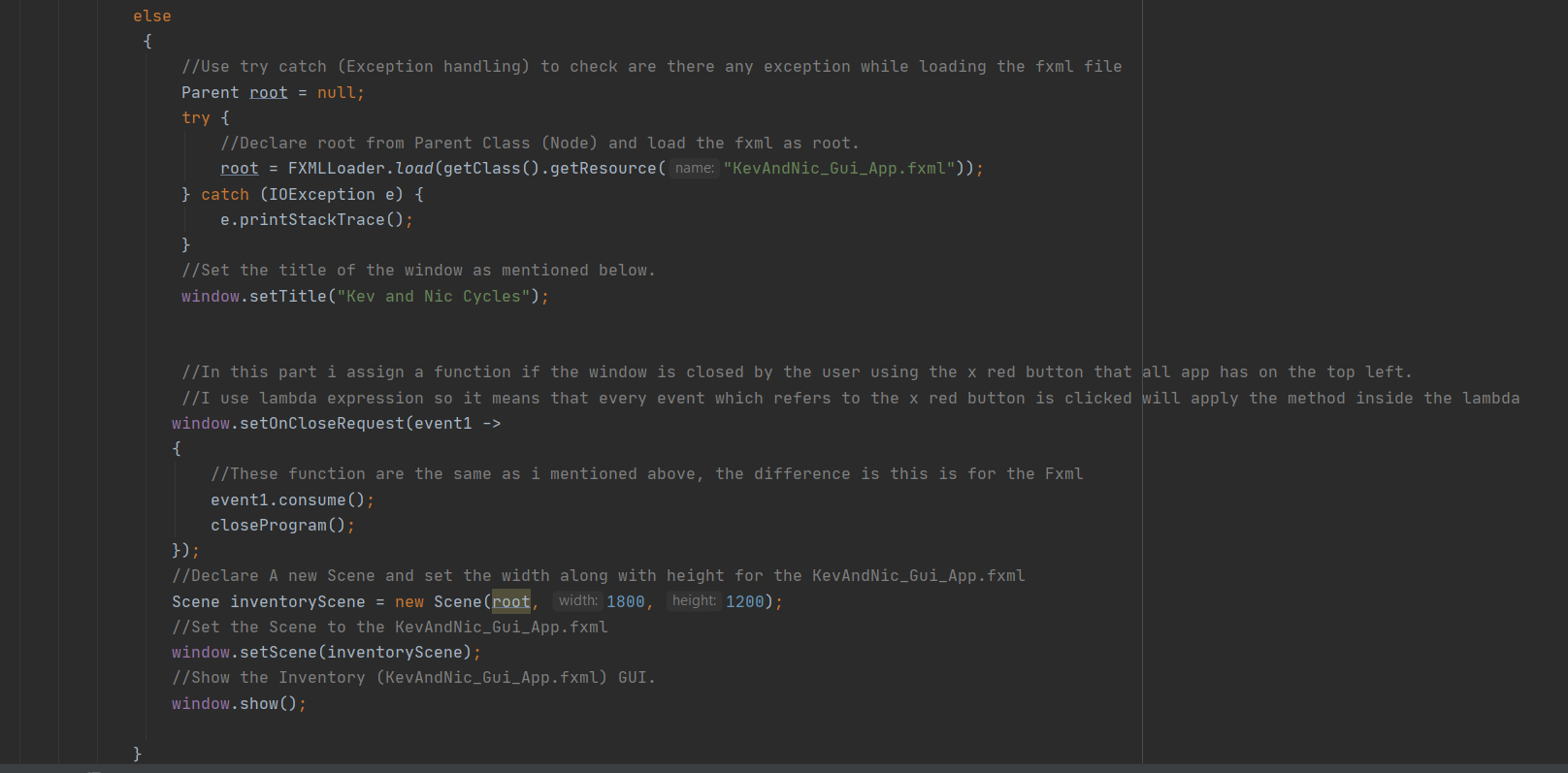


1. **Explanation**
   1. **Main.java**

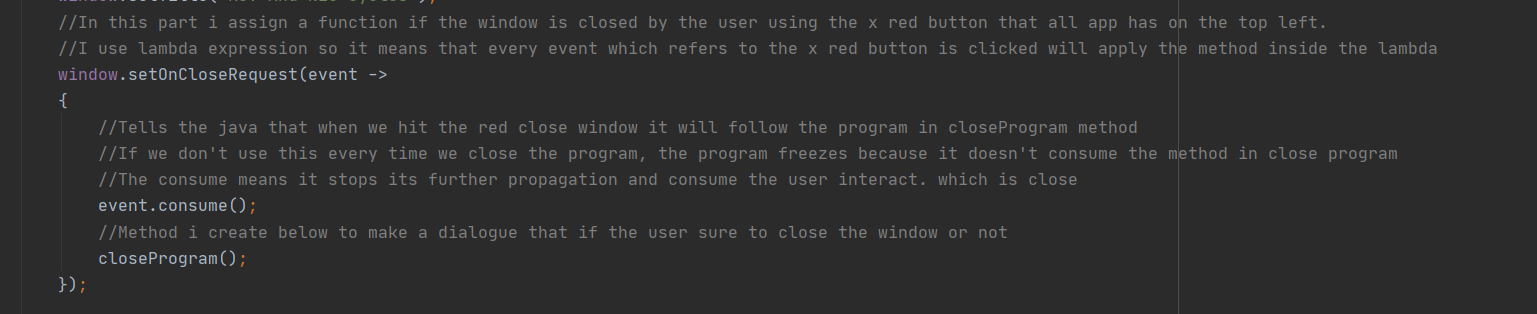
This file is where the program runs. This main class extends Application class to override the start method from Application parent class. The class contains the Login Scene where I create the scene by myself without using FXML and Scene Builder. And after the username and password has been filled correctly. The scene will load the FXML file named KevAndNic\_Gui\_App.fxml which is the main application where the user add, delete, and update products. The main purpose of this class is to manage switching scenes form login to the main GUI app.



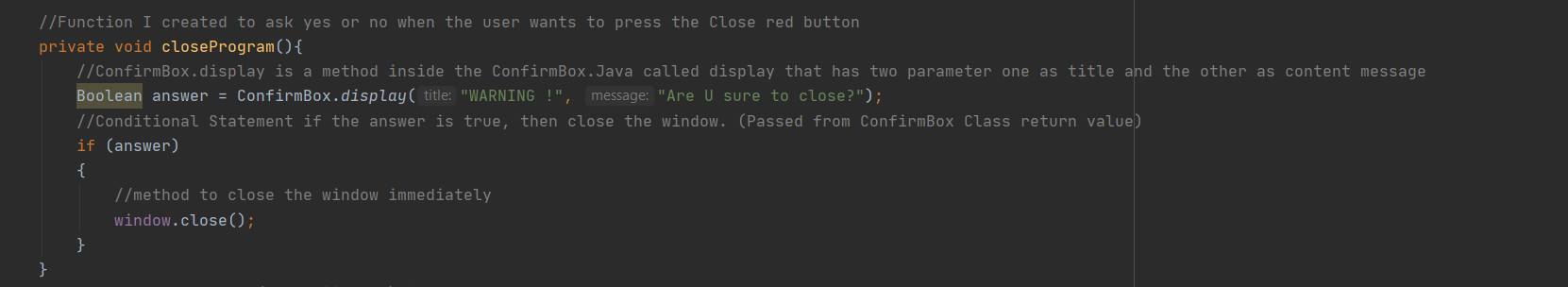
The screen shot above is the snippet on overriding the start function form Application Parent Class. And setting the icon of my application. Also, as we can see I declared username and password to access the application. In case only the owner of the business or company can use this application.



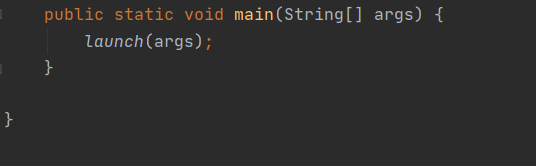
The screen shot above is the code to load the fxml. I used try catch to handle if there is an error while loading the fxml file.



One of the important methods that I implement is setting a dialogue whenever the user clicks the X red button on the top right. I used lambda Expression, means that every event which refers to clicking the x red button on the top left will apply the method inside the lambda expression. event.consume() Tells the java that when we hit the red close window it will follow the program in closeProgram method. If we don't use this every time we close the program, the program freezes because it doesn't consume the method in closeProgram() method. The consume means it stops its further propagation and consume the user interact, which is closing the app.



The closeProgram() method I created to show the yes or no dialogue whenever the user wants to close the program immediately. The method creates a new Boolean value named answer and assign the variable a method displays from ConfirmBox class.

****

Screen shot of a method to launch a standalone application. This method is typically called from the main method(). It must not be called more than one or an exception will be thrown.

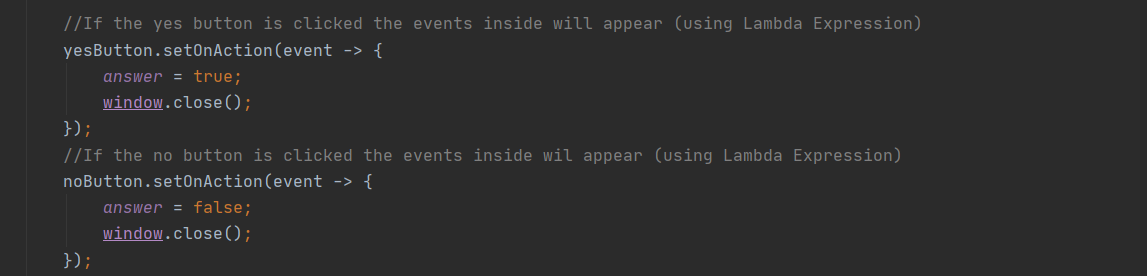
* 1. **ConfirmBox.java**

This Class purpose is just to create a new dialogue box to warn the user are they sure to exit the application whenever the user presses the x button on the top left side. This class has a Boolean function called display that returns true if the user chooses yes and false whenever the user choose no.

The important method that I implement is using the Application modality. The purpose of using the Application Modality is to define a modal window that blocks events from being delivered to any other application window other than the yes or no dialogue.

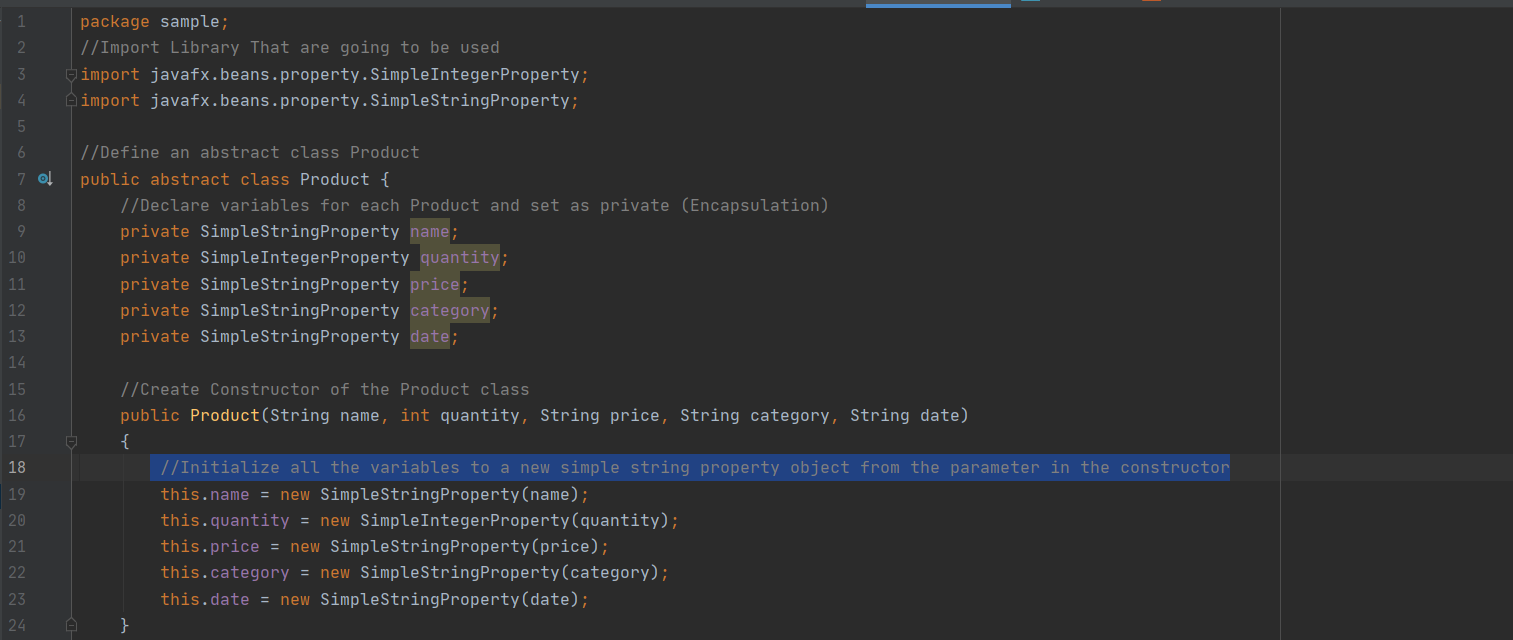


Another important method is using lambda expression anytime the yes or no button is clicked. So, whenever yes button is clicked the answer variable is set to true. And whenever no button is clicked



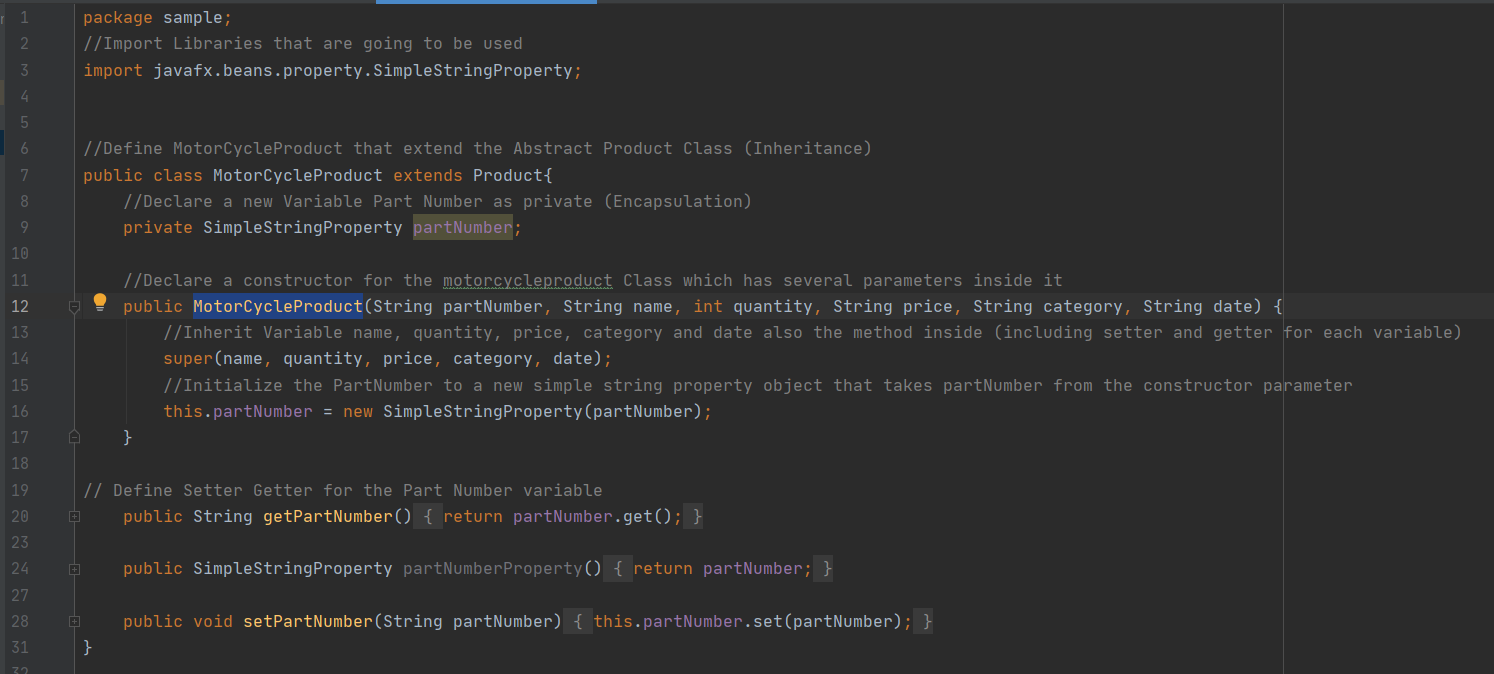
* 1. **Product.Java**

This file contains Product abstract class. This class is a parent class of MotorCycleProduct and DeletedMotorCycleProduct child class. This Product contains several attributes that a usual Product has such as name, quantity, price, category and date. The reason I create this abstract class is to implement the use of abstract class and inheritance. Also, in this class has a setter and getter of each variables. As you can see in the screen shot below, where I use Simple String Property and Simple Integer Property. In some circumstances it is mandatory to use JavaFX Property, like Product list rendered with tableView which is editable. To reflect the changes immediately in edited cell, the underlying bound field should be a property.



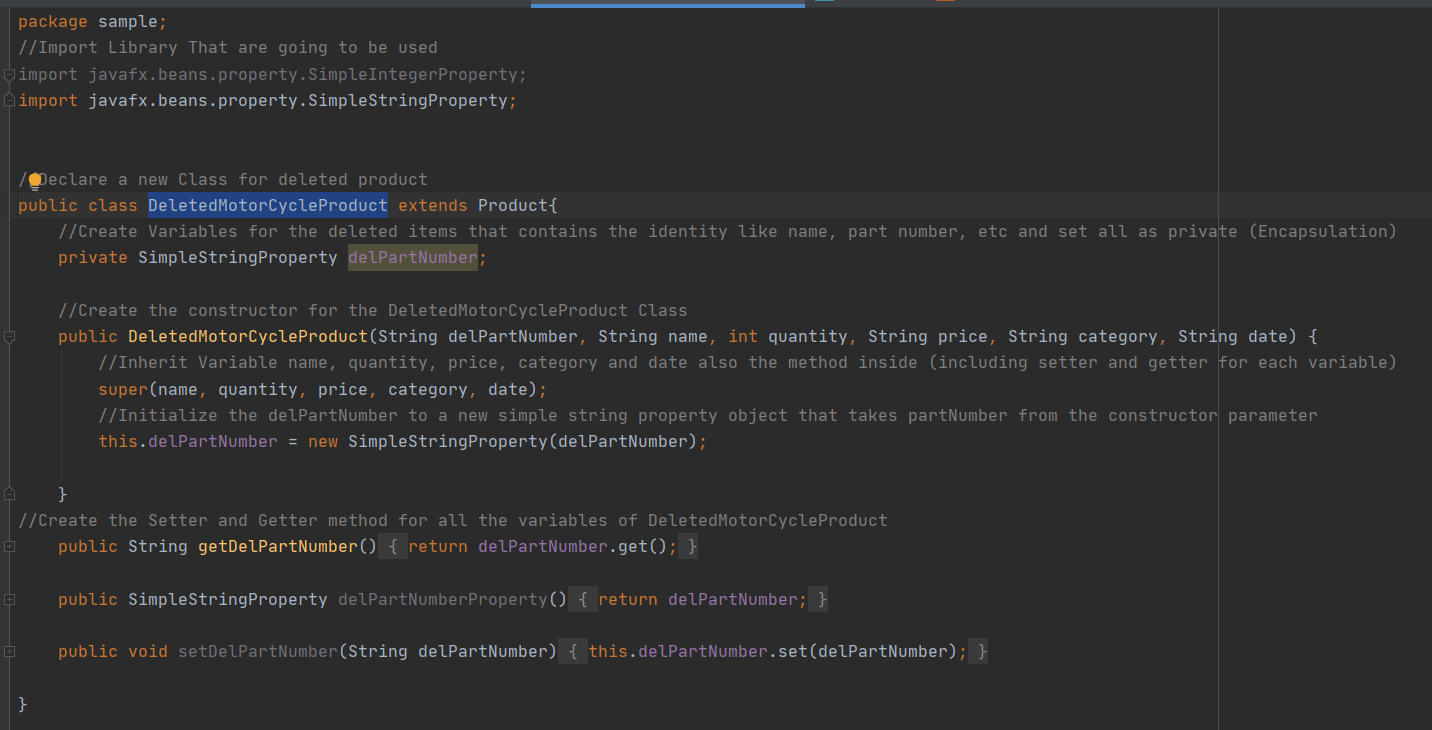
* 1. **MotorCycleProduct.java**

This file contains the MotorCycleProduct class that extends Product abstract class. This class inherits from the Product abstract class including the variable and the setter getter method. This class has a new variable partNumber along with the setter getter method. This class purpose is to add each product later to the observable array list containing all items.

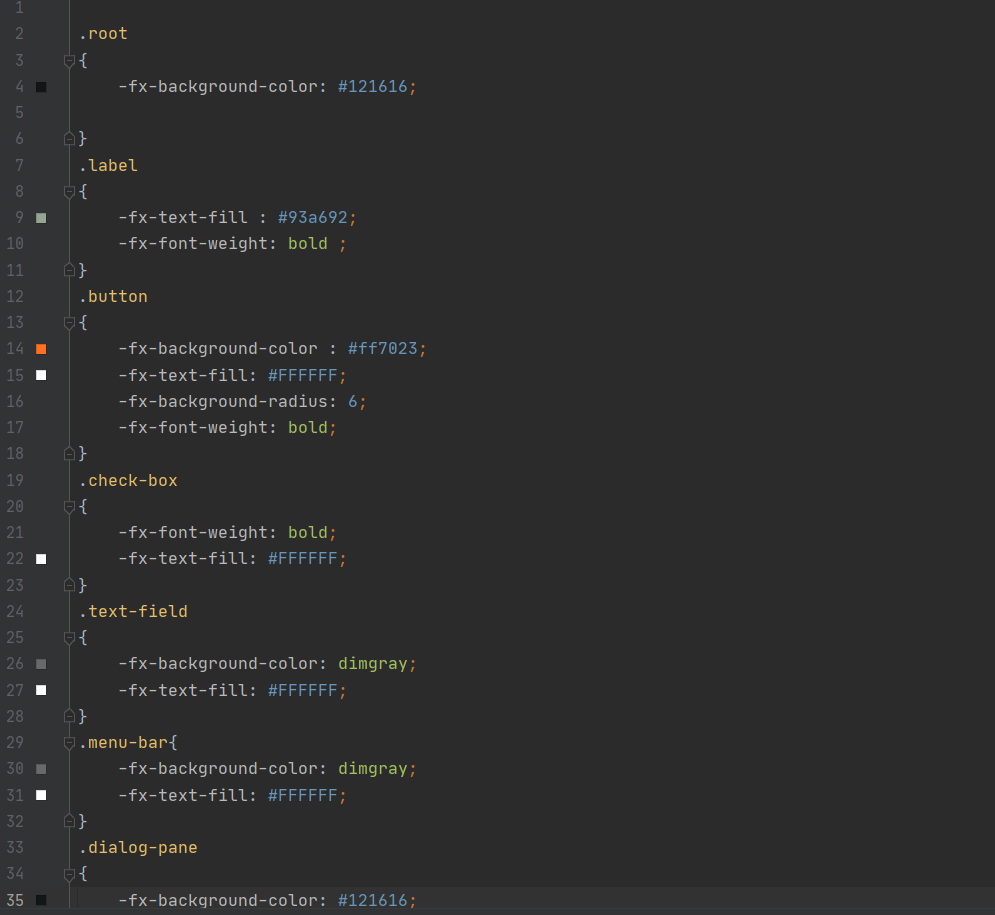


* 1. **DeletedMotorCycleProduct.java**

This file contains the DeletedMotorCycleProduct class that also extends Product abstract class. This class inherits from the Product abstract class including the variable and the setter getter method. This class has a new variable delPartNumber along with the setter getter method. This class purpose is to add each deleted product later to the observable array list containing deleted items.

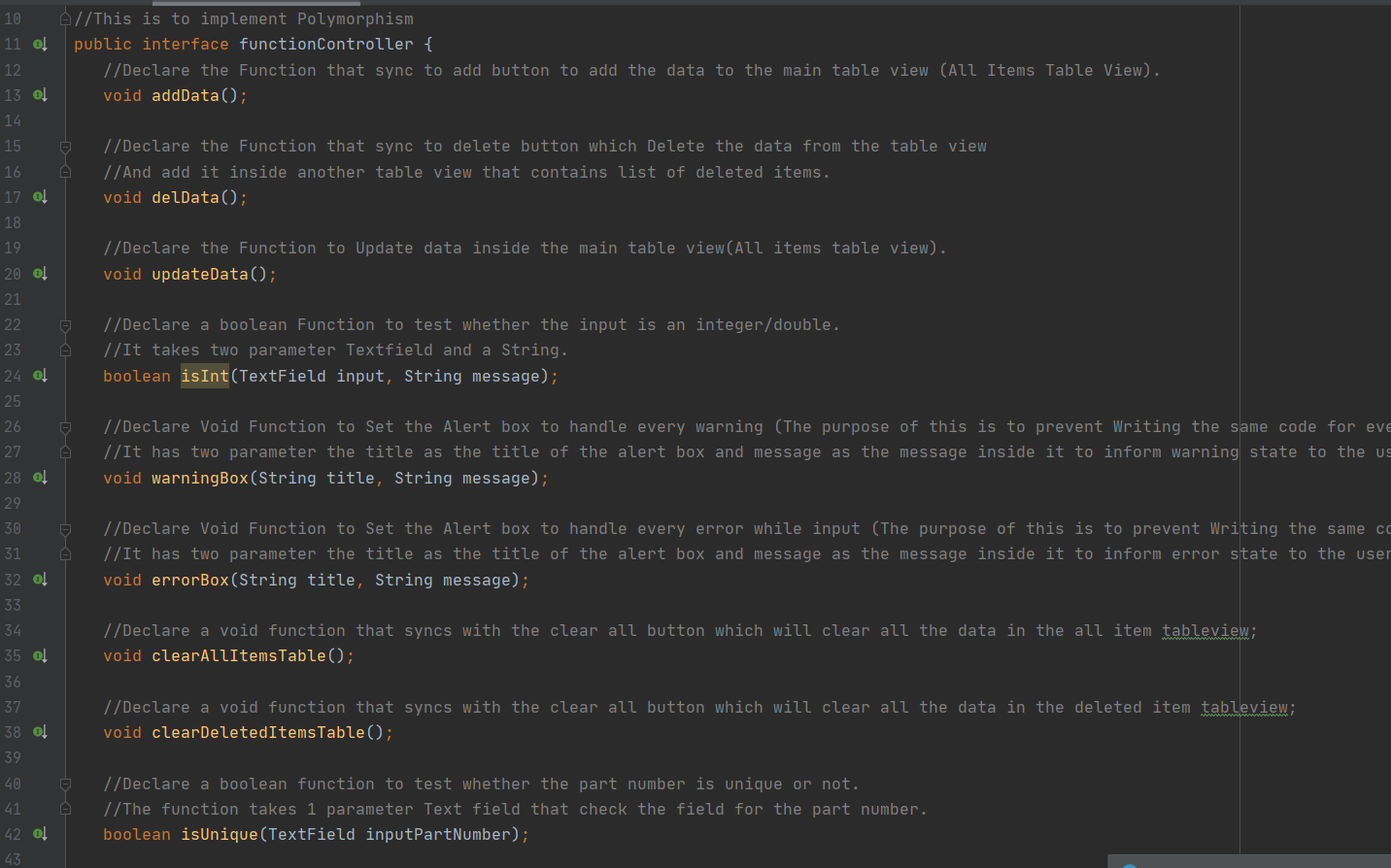


* 1. **Color.css**

This stylesheet file contains the design of my application. The purpose of this file is only to make my program look more interesting with mixed of color and fonts. As we can see from the screen shot below, the design of each elements inside the fxml such as label, button, check box, text field and many more are set here in this color.css file.

* 1. **functionController.java**

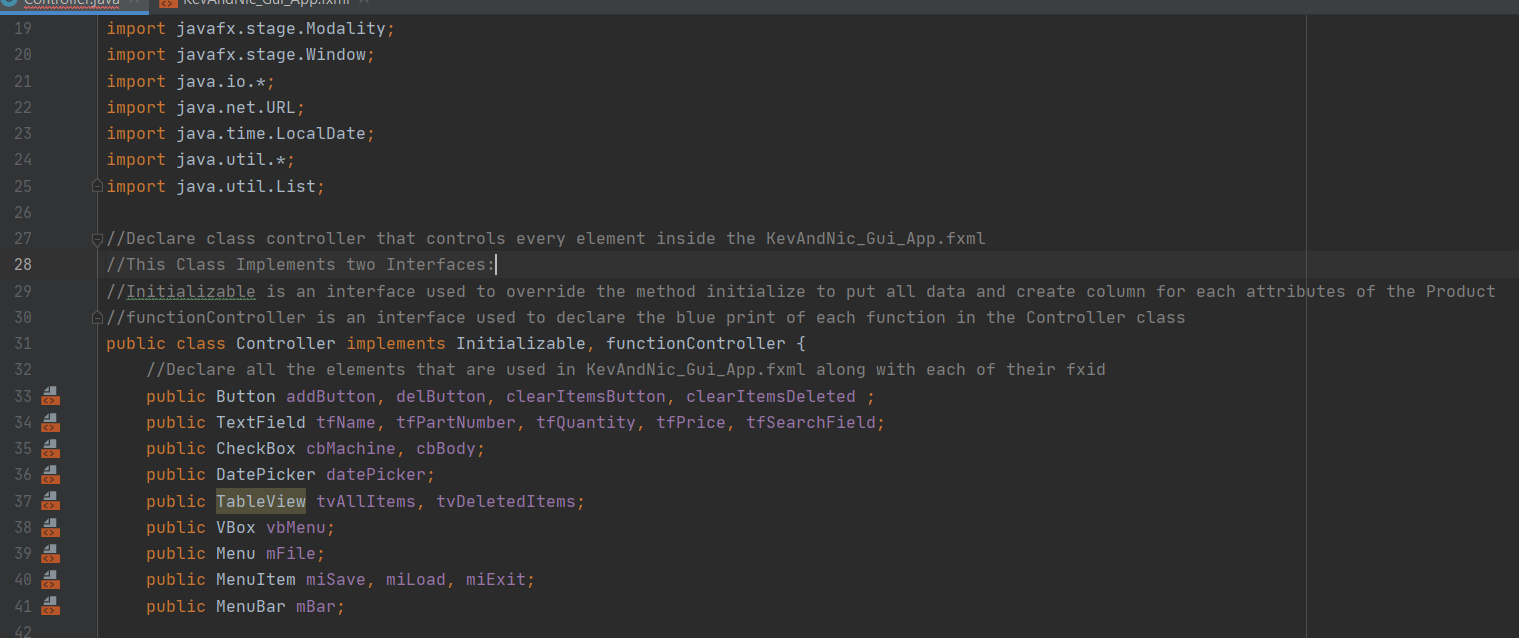
This file contains an interface named functionController. I created this interface as the blueprint of the Controller class. This interface contains the declaration of each function that are going to be used in Controller class. Screen shot of the interface class:

****

* 1. **Controller.java**

This file contains a Controller class that implements 2 interfaces. First is Initializable, another one is the functionController interface that I create previously. I implement initializable to override the the void function initialiaze. And functionController to override all the function I declared inside. I used text field for the user input on name, part number, quantity and price. For the date, I use a Datepicker and for the category I used check box.

First, I declare all the elements that are used in KevAndNic\_Gui\_App.fxml file along with each of their fxid. I have table view, buttons, textfields, and many more as shown below.

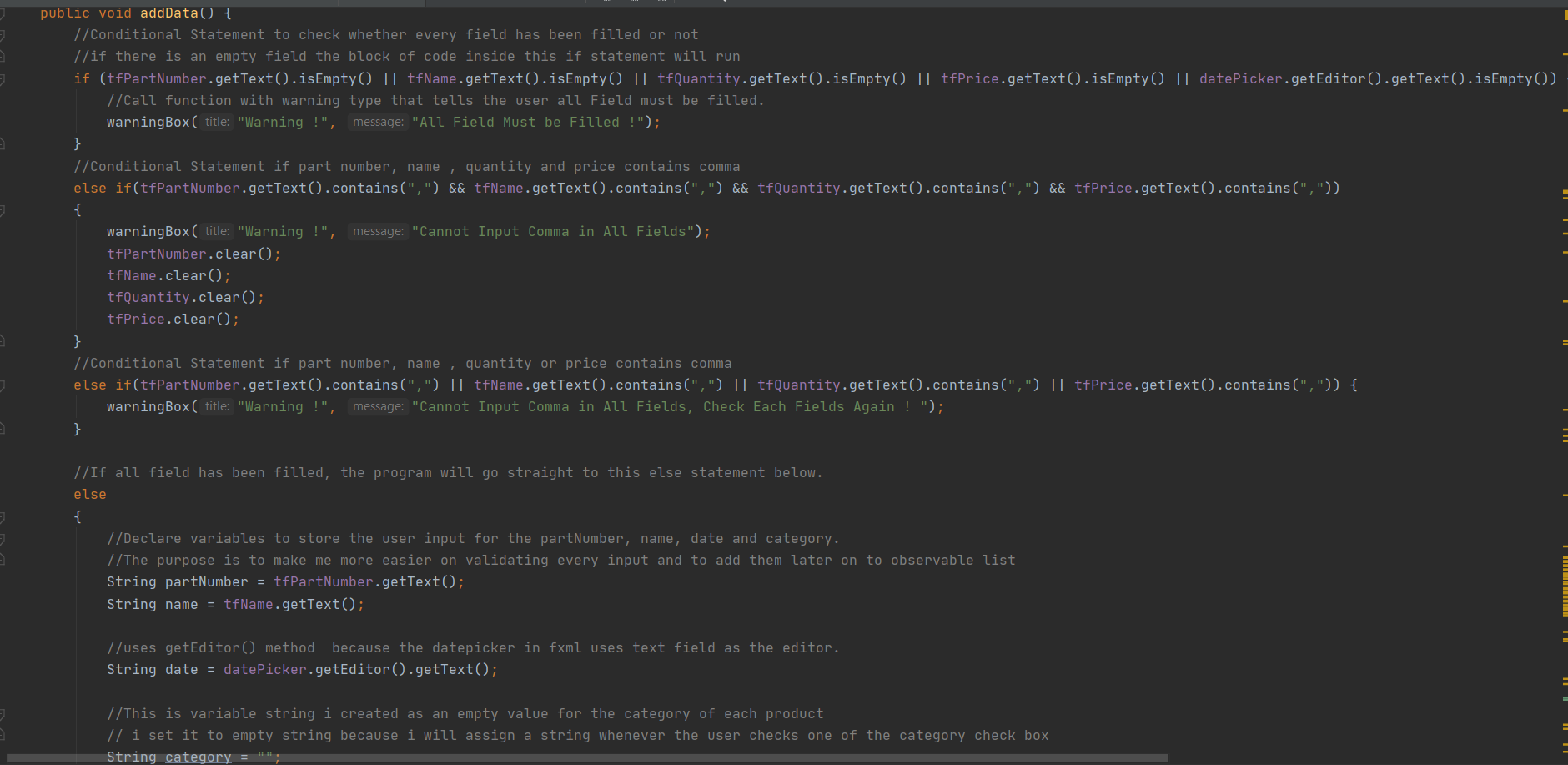


Also, I declare two observable array list named products and deleteProducts. products list contains all the items that the user has and deleteProducts list contains all the deleted items in case the user needs those informations. The reason I use simple string property instead of string is because in order to add an item inside the observable list it needs to be a property. Observable array list is easier to work with rather than just an Array list. In this file has a bunch of function. Three important function are addData, deleteData, and updateData. Those functions have been synced with each add button, delete button and update button in the FXML file. To add product, you must fill a form consist of identity of the product. To delete and update, you need to select one of the rows in the table view in order to delete or update products. If there is no product inside the table view, every time you press the delete or update a warning box will appear. The waring box tells the user that nothing to update or delete in the table.

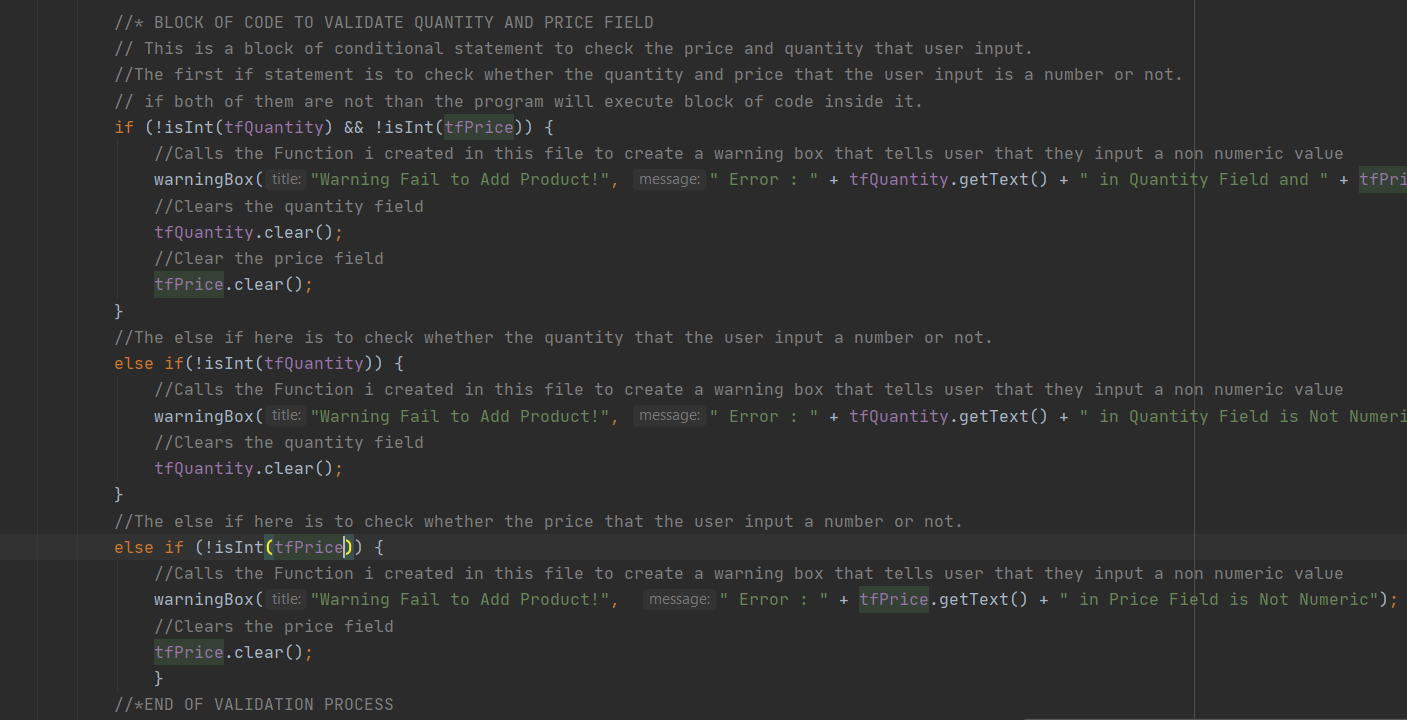
* addData() function.

This method is used to add product inside the product Observable List. This function is synchronized with the addButton I declared above. So, whenever I click the add button, this function/method will be called. But before adding the product inside the table, we need to validate each field. I start with validating whether all fields have been filled or not by the user. If not, the program will show a warning dialogue telling the user that all fields must be filled. Otherwise if all fields have been filled, the program will continue to validate the other fields.

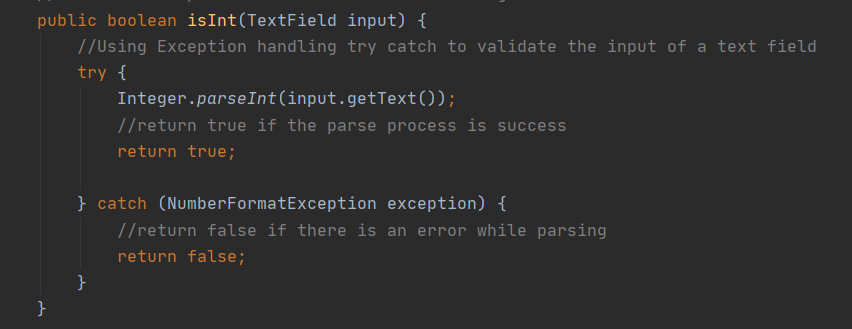
After all fields are filled, I declare variables to store the user input for the partNumber, name, date and category. I also did some validation so the user doesn’t input a comma. This is to prevent errors while saving and loading files. Because i use CSV (Comma Separated Values) files which uses commas as a delimiter. Here is a screenshot of validating input and storing user input to each of their variable.



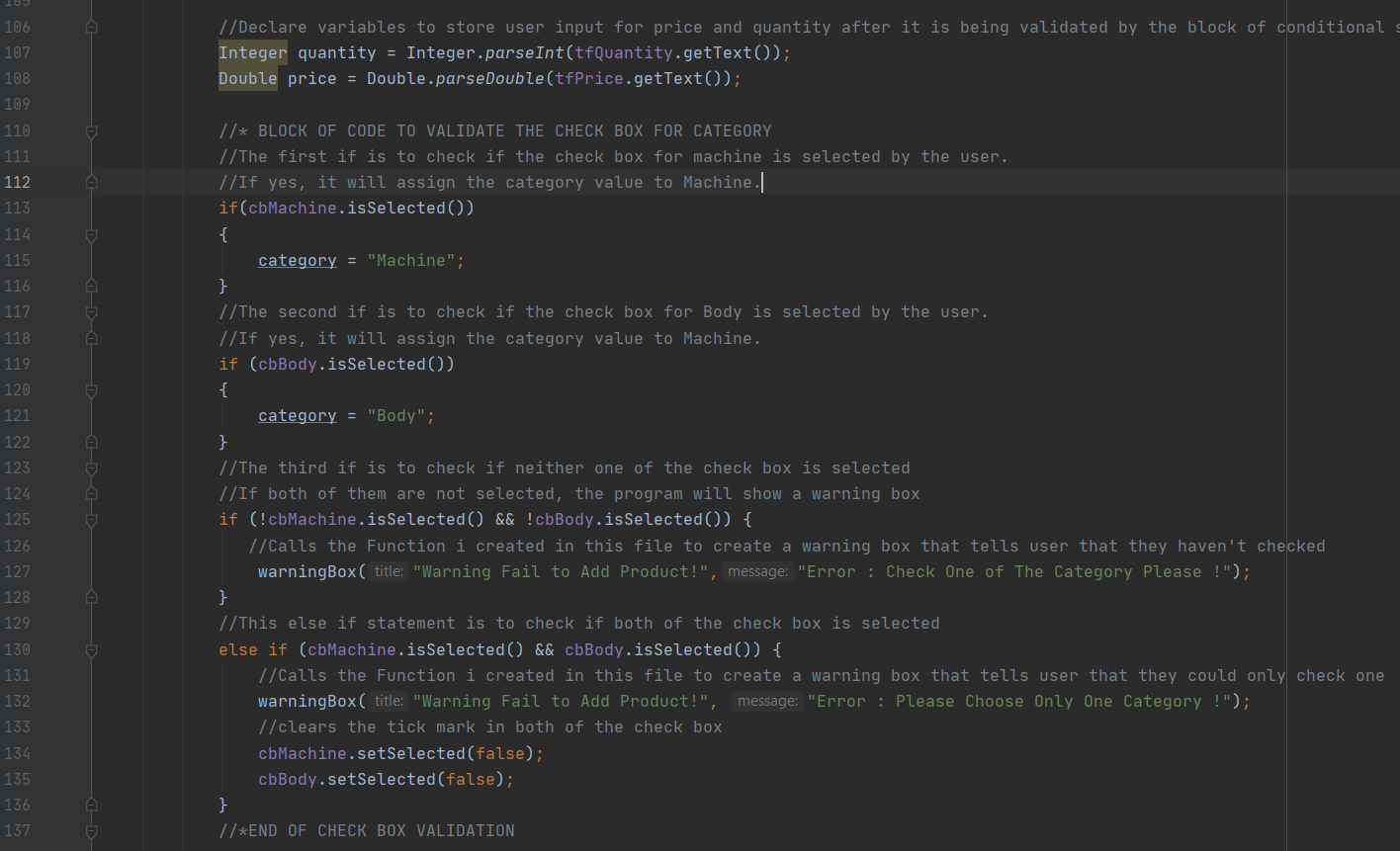
I also declare variables to store the quantity and price. But before assign the variable, I do some validation. The validation is to determine whether the input in price and quantity is numeric. If its not numeric, the program will show an alert box telling the user that they input a non-numeric value inside the price and quantity field. Otherwise if it’s a number, the program will continue to validate other inputs. Here is a screenshot on the validation process.



I used an isInt() boolean function I created to check whether user input in a text field is in a number format or not. I used an exception handling try catch. The try part is to parse the user input. The catch is used to handle if the input is a non-numeric value. The function takes one parameter which is the text field. Here is the function:

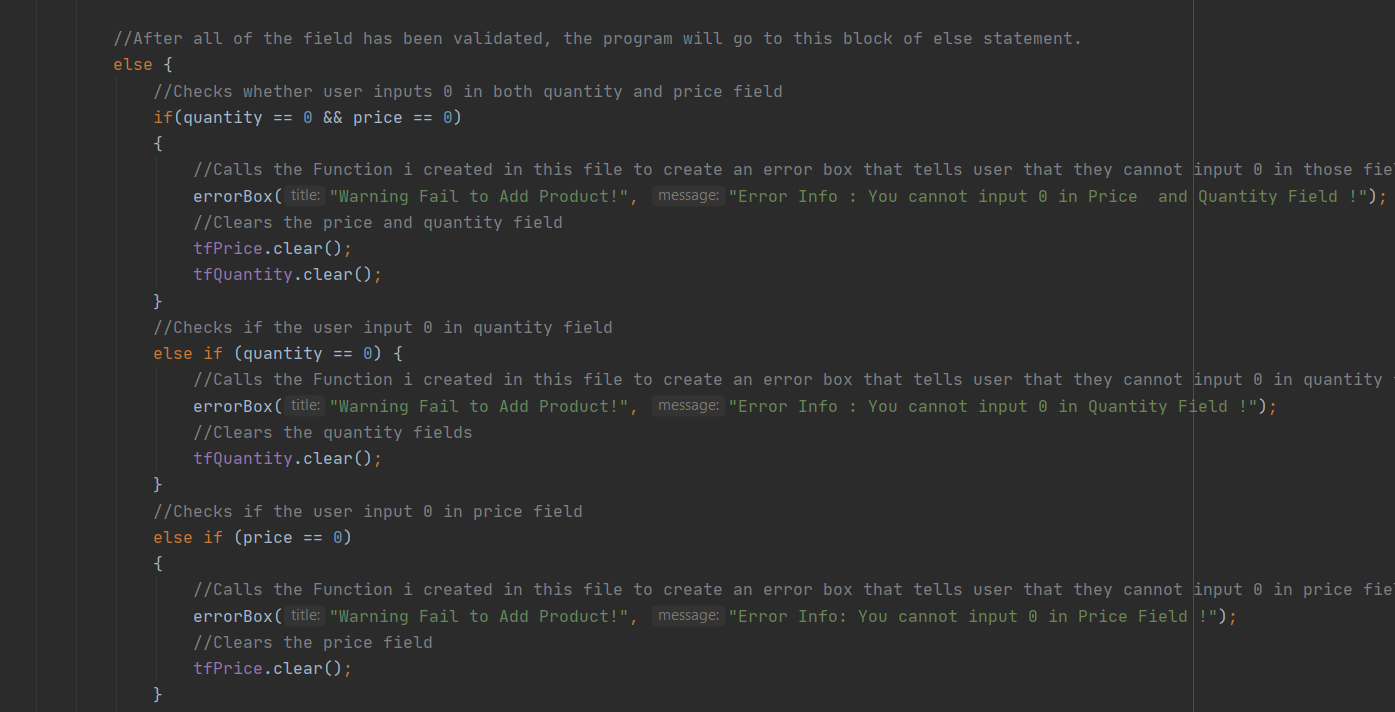


After I validate the input in price and quantity, I assign them to a new variable exactly same way as I do for the name, part number, category and date. The next thing I need to validate is the check box. Here is how I do it:

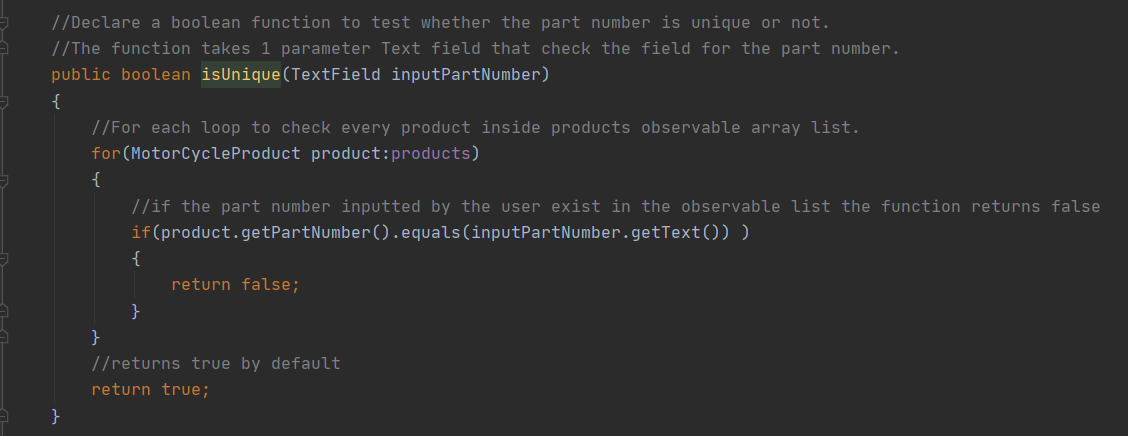


So, what I’m doing is check whether one of the check box either machine or body is selected using the isSelected() boolean method. If Machine is selected by the user, I assign the category value to “Machine” string. Otherwise, I assign the category value to “Body” string. If neither of the box is selected by the user, a warning box will show up and tells the user to select one of them. If the user selects both checkboxes, another warning box will show up telling the user to select only one category and clears then check boxes.

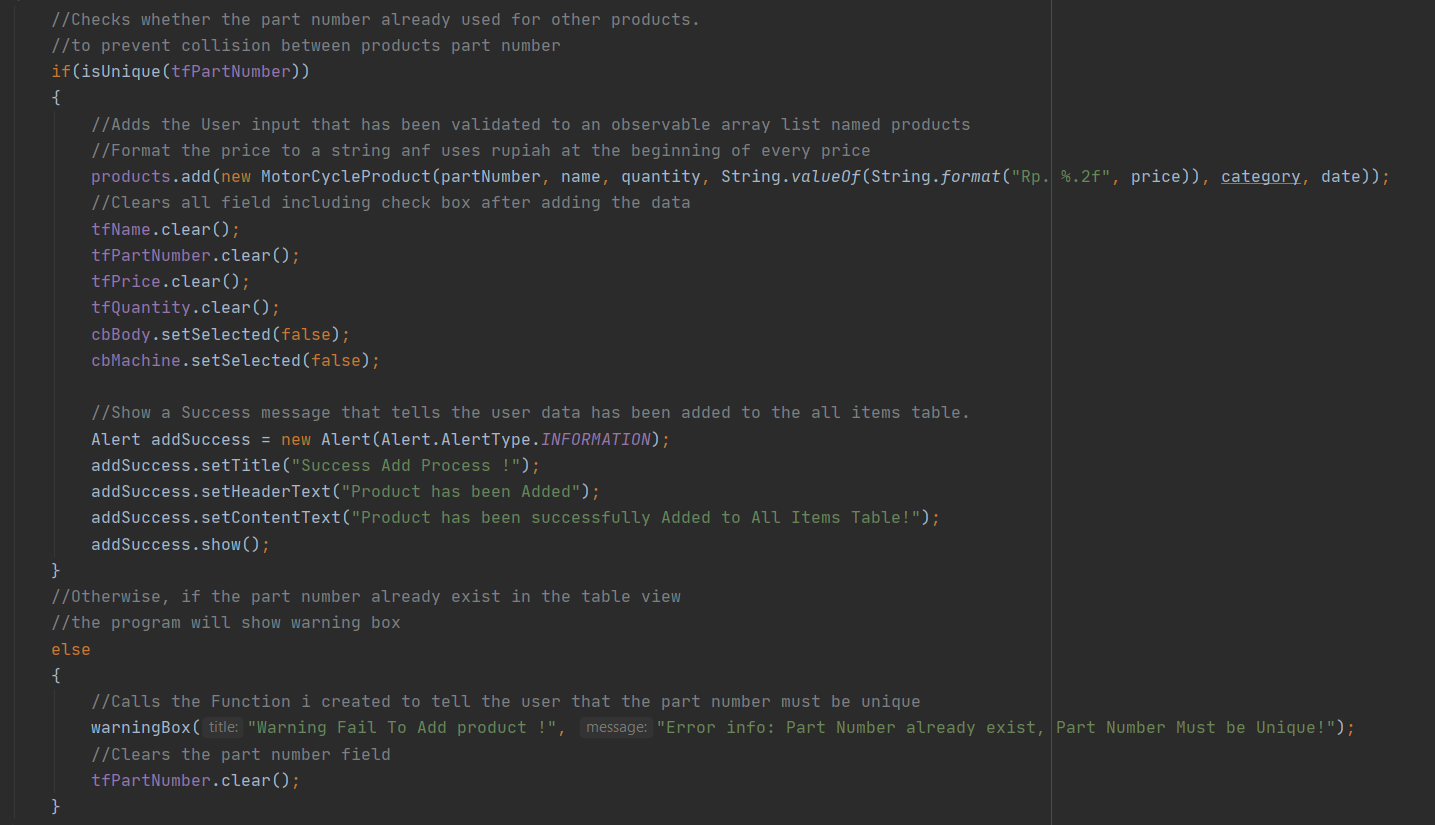
Next is to validate the variable quantity and price that contains the user input in quantity and price text field. This validation process is to check if the user inputs 0 in both fields. Because logically, input in price cannot be 0 unless its free, also input in quantity cannot be 0. Here is the validation process:



After that, I check the part number is it unique or not. I create a method isUnique() to check the part number input. The reason is to prevent collision between each product’s part number.

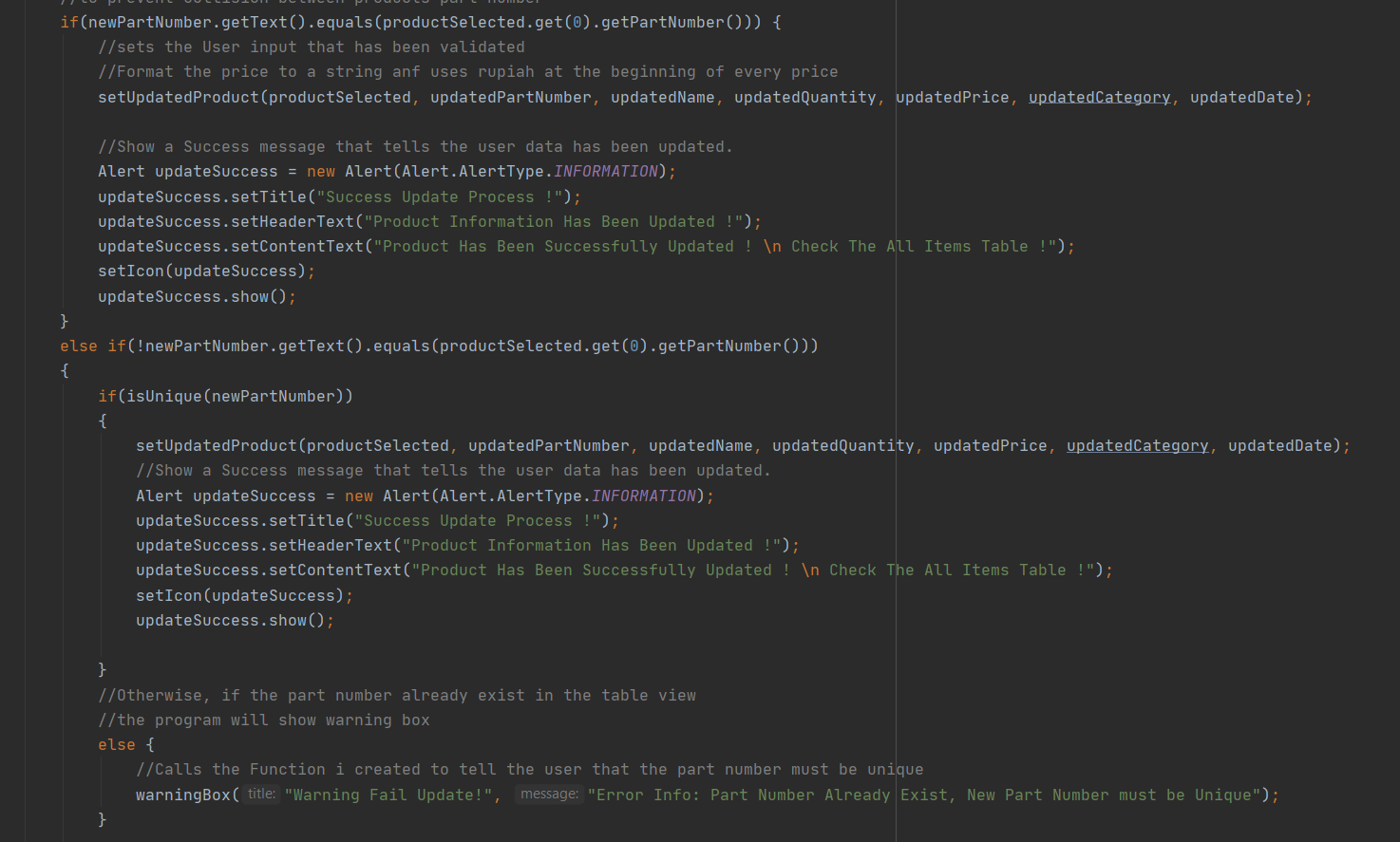


All user input has been validated, now its time to add all the variables that contain user inputs for identity of each product into an observable array list. To add it, I create new object from MotorCycleProduct class and fill all the parameters with all the variable for every identity of each product such as name, part number, quantity, price, category and date. Here is a screen shot on how I made it.

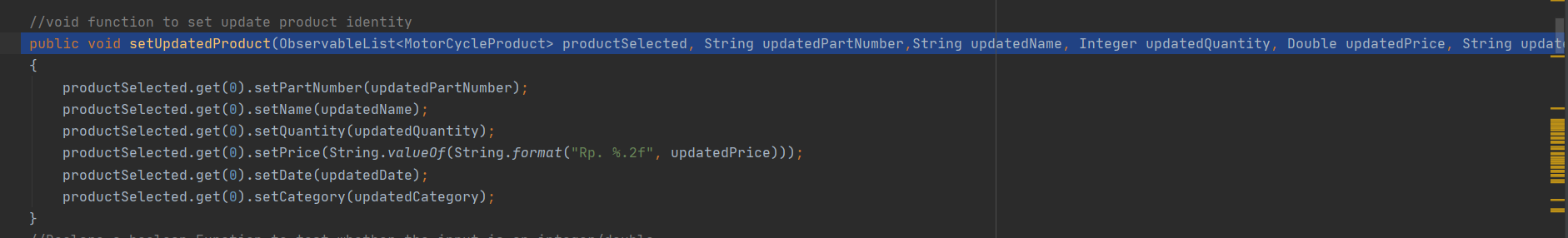


* updateData() function

This function is created to update an existing product inside the all items table view. So, I created a new update form where the user can input the new identity of product such as name, part number, quantity, price, category or date. If the user clicks the update button with no product inside the all items table view, the program will show a warning box says nothing to update. If there are products inside the table view and the user haven’t select on of them and clicks update, the program will show another warning box telling the user that you must select one product in order to update. The validation process of every input here is the same with addData() function. The difference is just before I validate the updated part number is unique or not, I check whether it is the same with the old part number. If it’s not, I check the new part number is unique or not.



I set each product identity using a function to set them called setUpdateProduct(). The reason I create function is to prevent duplicate code written in the program.

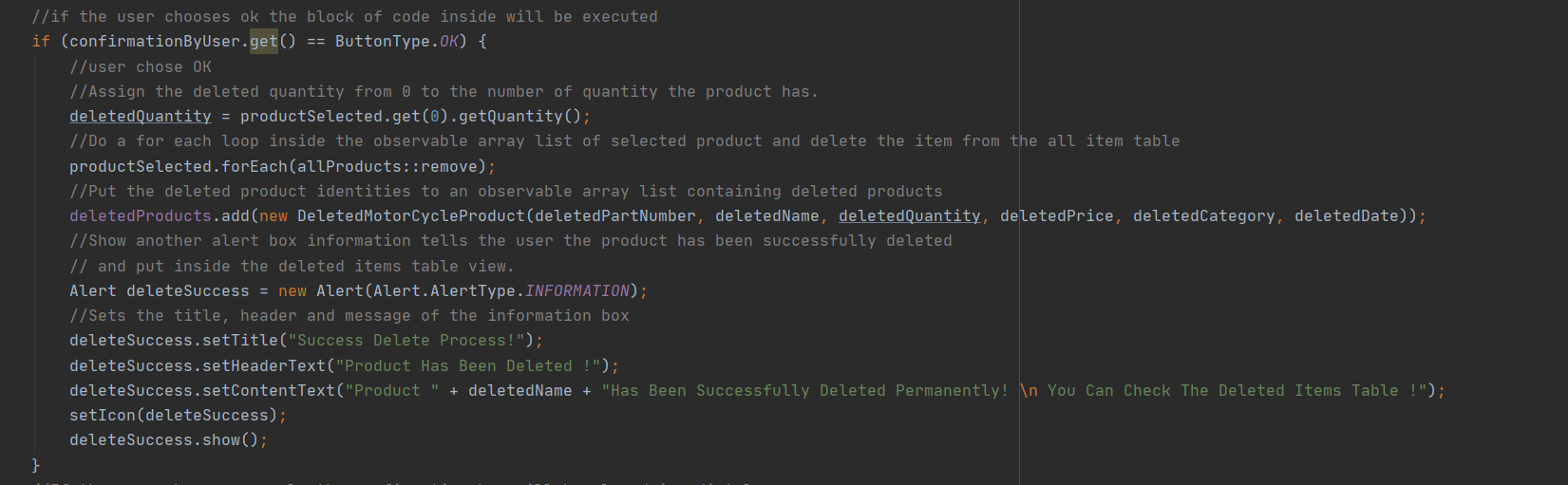


* delData() function

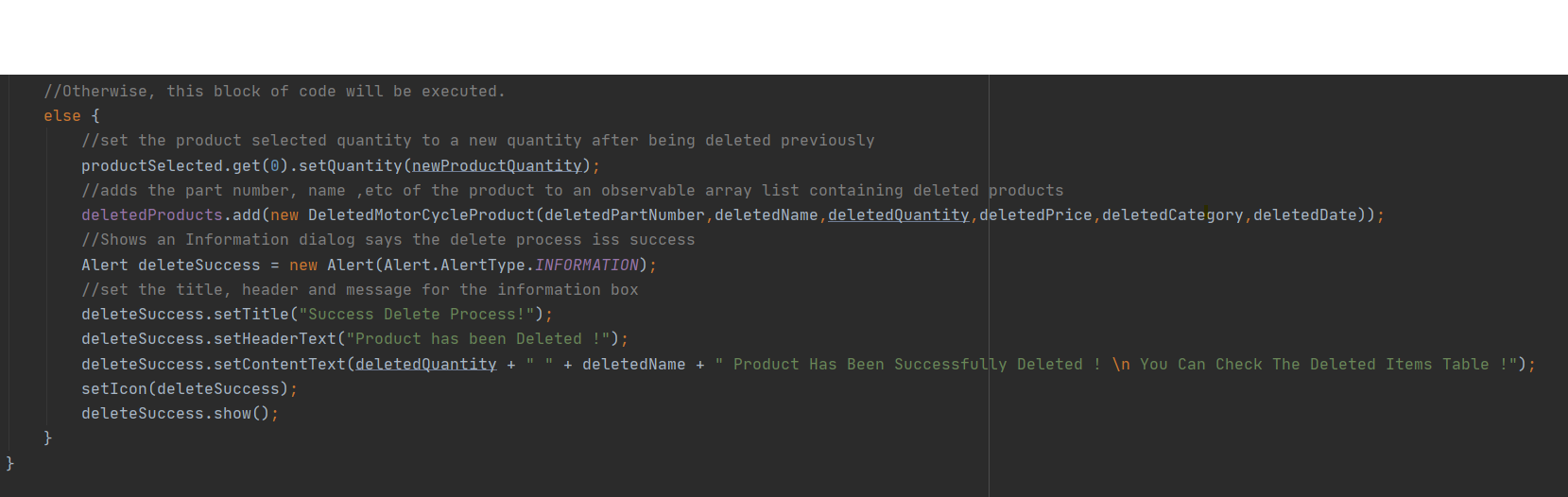
This function is created to delete an existing product inside the all items table view. If the user clicks the delete button with no product inside the all items table view, the program will show a warning box says nothing to delete. If there are products inside the table view and the user hasn’t select one of them and clicks update, the program will show another warning box telling the user that you must select one product in order to delete.

I created a new delete dialogue where the user can choose to delete all or delete by quantity. If the user chooses to delete all, the program will ask whether the user is sure to delete all product. If the user chooses to delete by quantity, the program will show a choice dialog. This choice dialog consists of a picker where the user picks how many products they wanted to delete.

Here is a nice idea I brought to this program. I add a new table view that displays the deleted products based on today’s date (assuming the user deletes it today). So, if the user chooses to delete all I store all the identity of the product inside observable array list containing list of deleted products. Here is the snippet of the function if the user chooses to delete all.



If the user chooses to delete by quantity, I store all the identity of the product inside observable array list containing list of deleted products but before that I set the new quantity. The new quantity is based on how much the initial quantity – with the how many products that the user wants to delete. Here is a snippet of the function id the user chooses to delete by quantity.



* setIcon() function

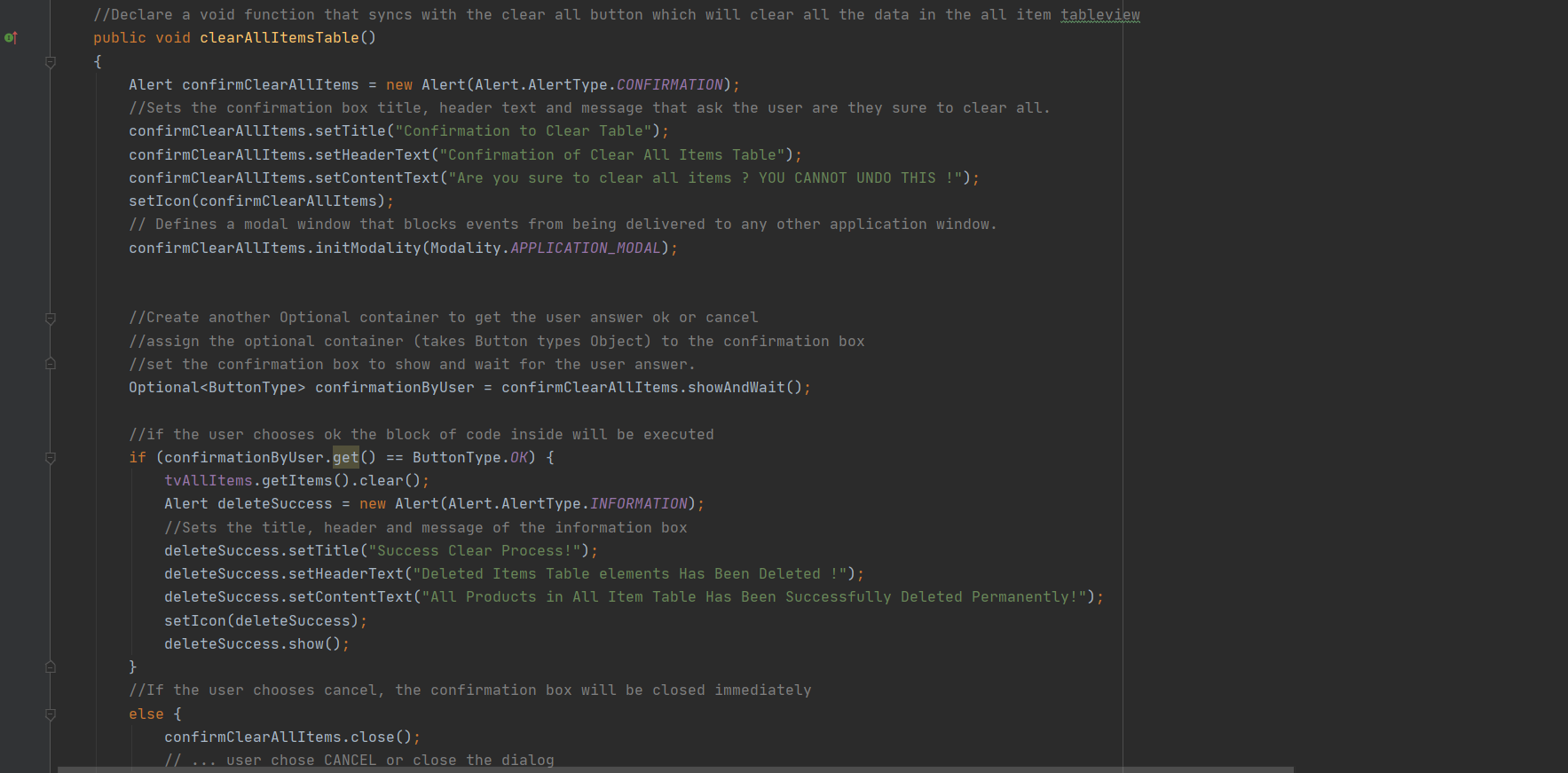
A void function to set the icon of each dialogue.

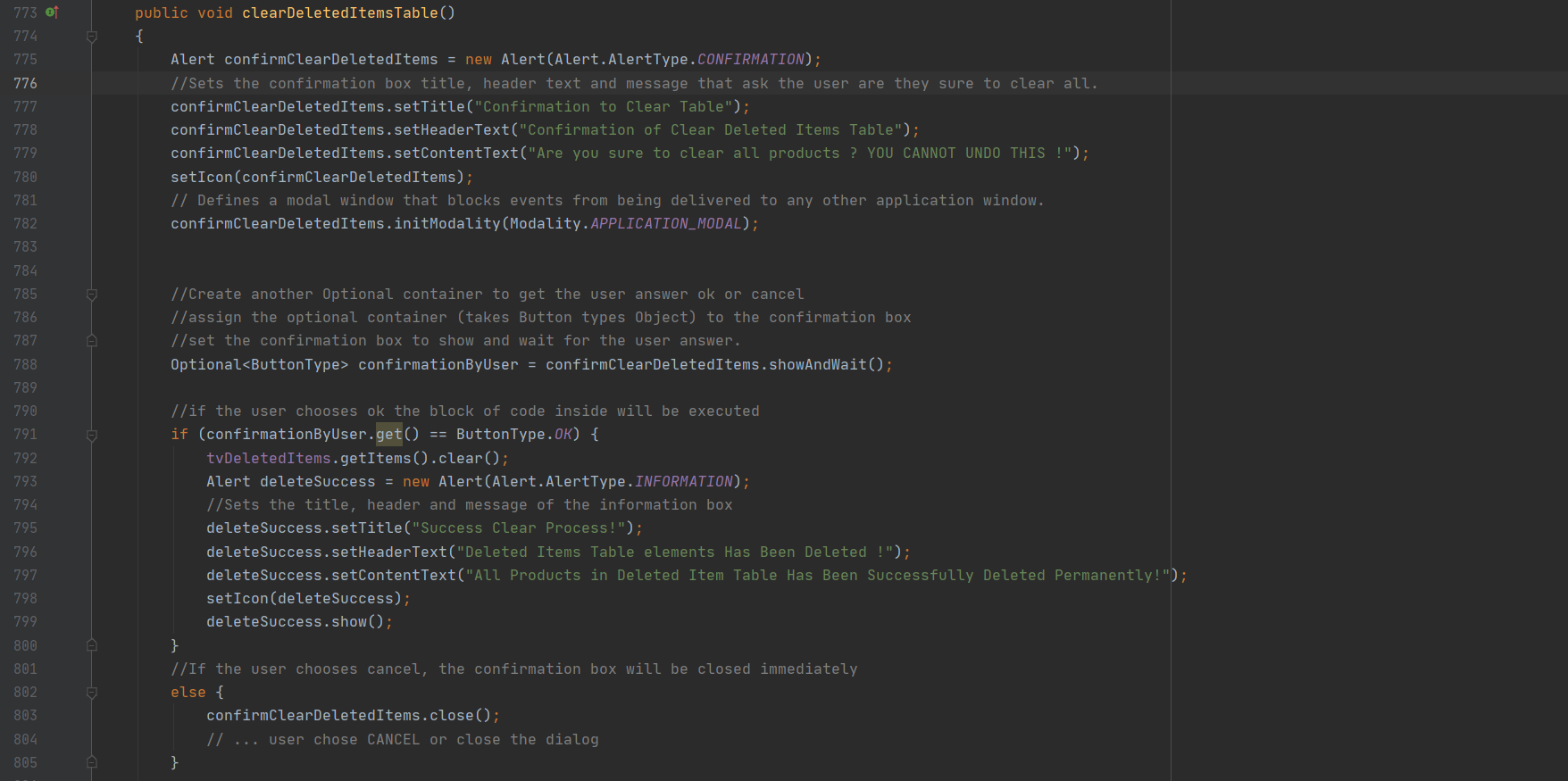
* warningBox() and errorBox() function

Void Function to Set the Alert box to handle every warning and error (The purpose of this function is to prevent writing the same code for every warning or error). It has two parameter the title as the title of the alert box and message as the message inside it to inform warning or error state to the user.

* clearAllItemsTable() and clearDeletedItemsTable() function

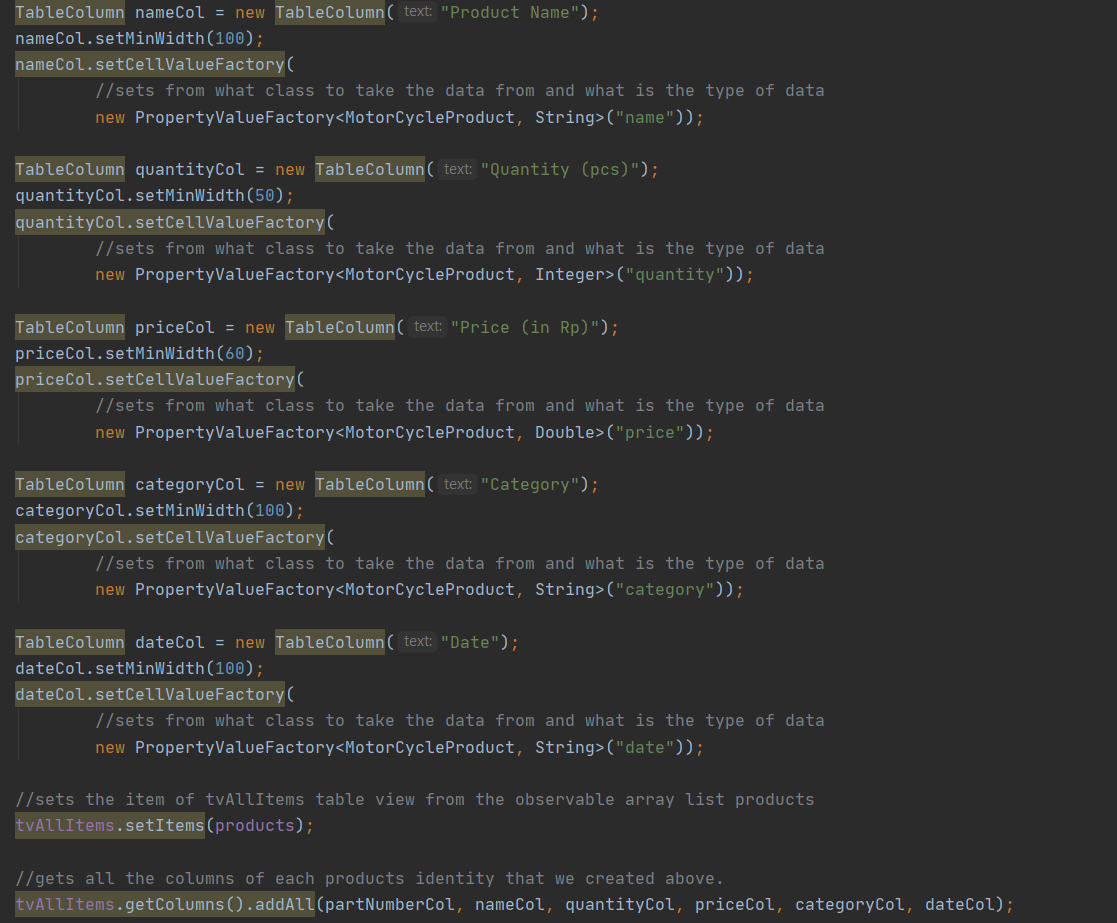
a void function that syncs with the clear all button which will clear all the data in the all item and deleted item tableview.





* initialize() function

Called to initialize a controller after its root element has been completely processed. Suppose you have a table view you want to populate with data. In this part as said before, I declare columns for each identity of every products and put it inside the table view. This function is basically to set every element not just table view but also date picker and many more. Here is a snippet of a code to set columns inside the table view.



* search() function

A void function for the search bar field to sort the table view based on the text inside. The search here is only to search products by name. I used FilteredList to filter the all the products. Then, wrap the FilteredList in a SortedList form. Bind the SortedList comparator to the TableView comparator. Otherwise, sorting the TableView would have no effect. Last, add sorted (and filtered) data to the all item table view.

* handleSaveClicked() function

A void function that sync to the menu item Save inside File menu to save the data of products. I used some try catch inside it and to handle the exception on saving the file. Also, I use FileChooser class to save files. I set the type of file each time user wants to save is csv (Comma Separated Value) file. Inside this function there is another function I created named writeExcel() to write all products inside the table view inside the csv files.

* writeExcel(File newFile) function

A void function that writes all the data of products to csv file or excel. This function uses BufferedWriter as a tool to write the files inside the csv file. This function takes a file object as the parameter.

* handleLoadClicked() function

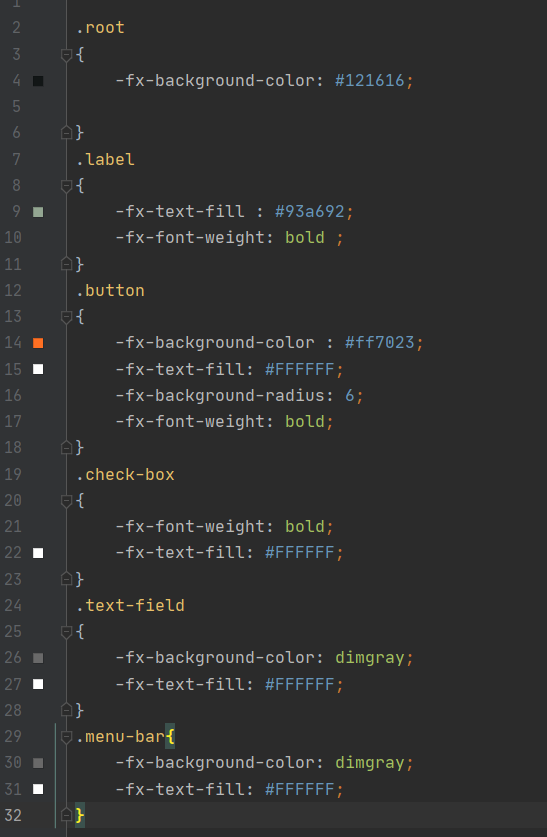
A void function that sync to the menu item load inside File menu too load/open the data of products. I used some try catch inside it and to handle the exception while opening the file. Also, I use FileChooser class to open files. I set the type of file each time user wants to open is csv (Comma Separated Value) file. Inside this function there is another function I created named readExcel() to read all products inside the csv file and put inside the table views.

* readExcel(File existedFile)

a void function that reads all the data of products from csv file or excel. This function uses BufferedReader as a tool read the csv file. This function takes File object as parameter.

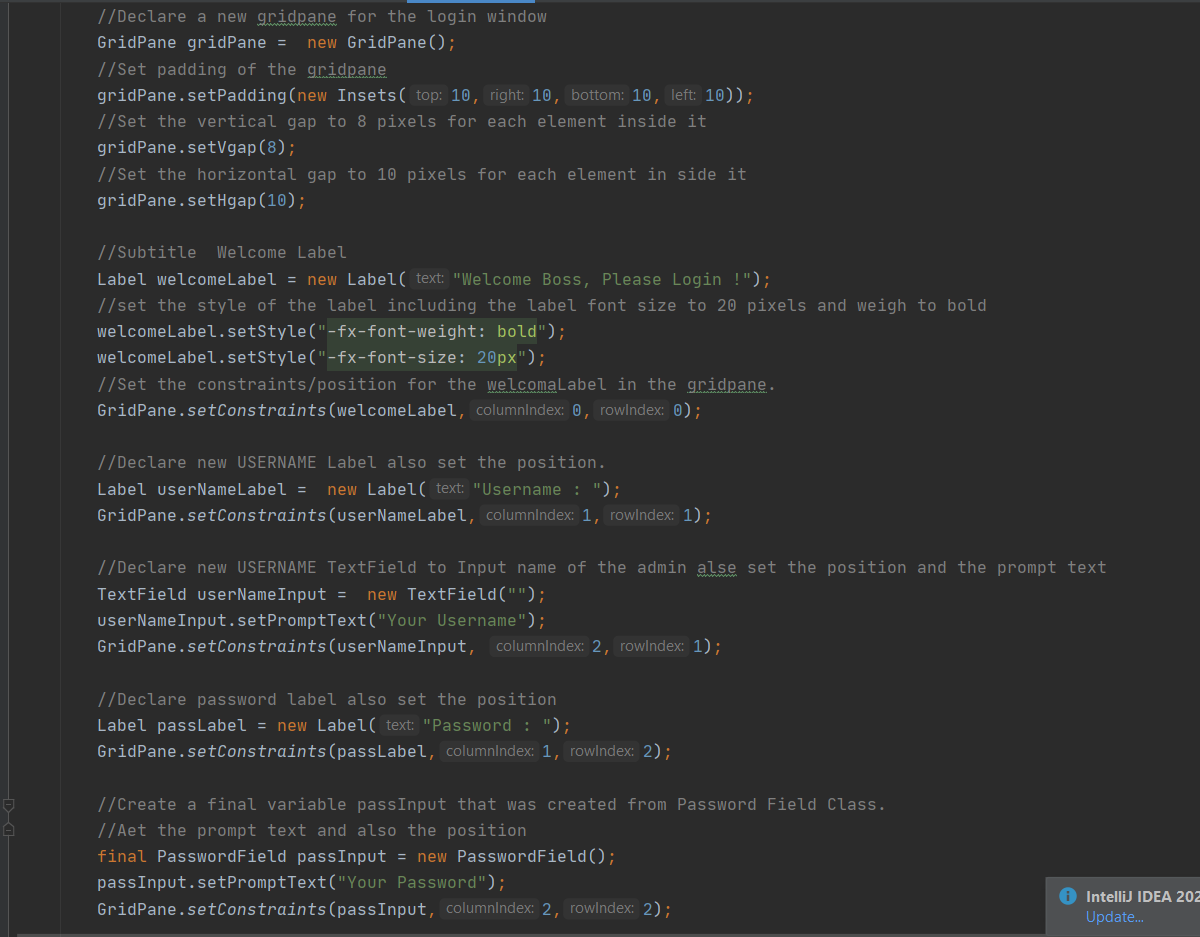
1. **Lessons That Have Been Learned**
   1. **The use of CSS Style Sheet**

I did some research and found out that java FX can work with CSS style sheet to design the GUI in FXML file.



* 1. **The way to create login scene without using Scene Builder**

I did some research to know how to connect two FXML files, but instead of using FXML for the login scene I create my own log in scene. I use Grid pane as my canvas to create the login scene.

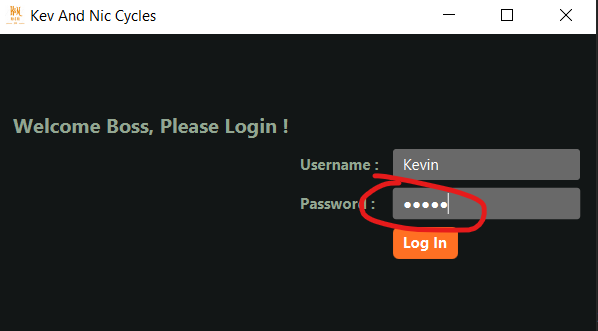


* 1. **The way to use event.consume() and create a dialogue each time the user wants to close the window.**

The consume means it stops its further propagation and consume the user interact, which is closing the app. I searched in the internet for the way to prevent user habit on clicking x instead of minimizing the window. So, I created a class ConfrimBox which contains a dialogue to ask the user are they sure to leave the program

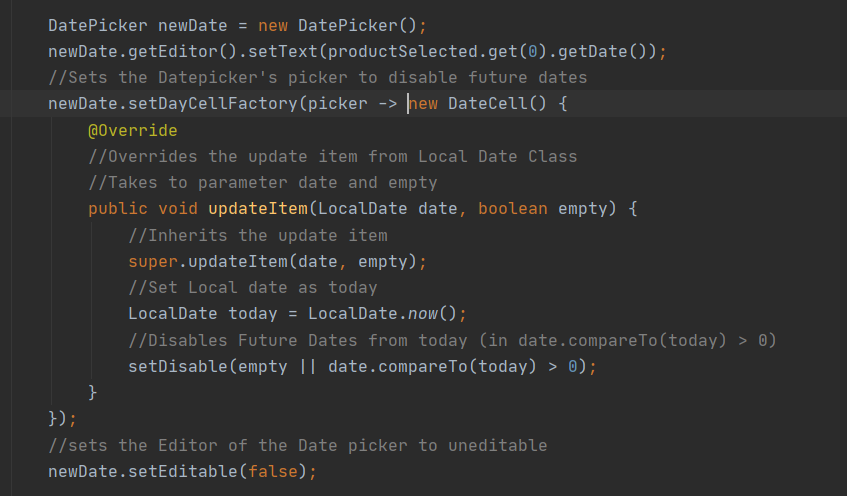
* 1. **The way to use Password Field**

Using a Password field instead of just a normal Text Field.

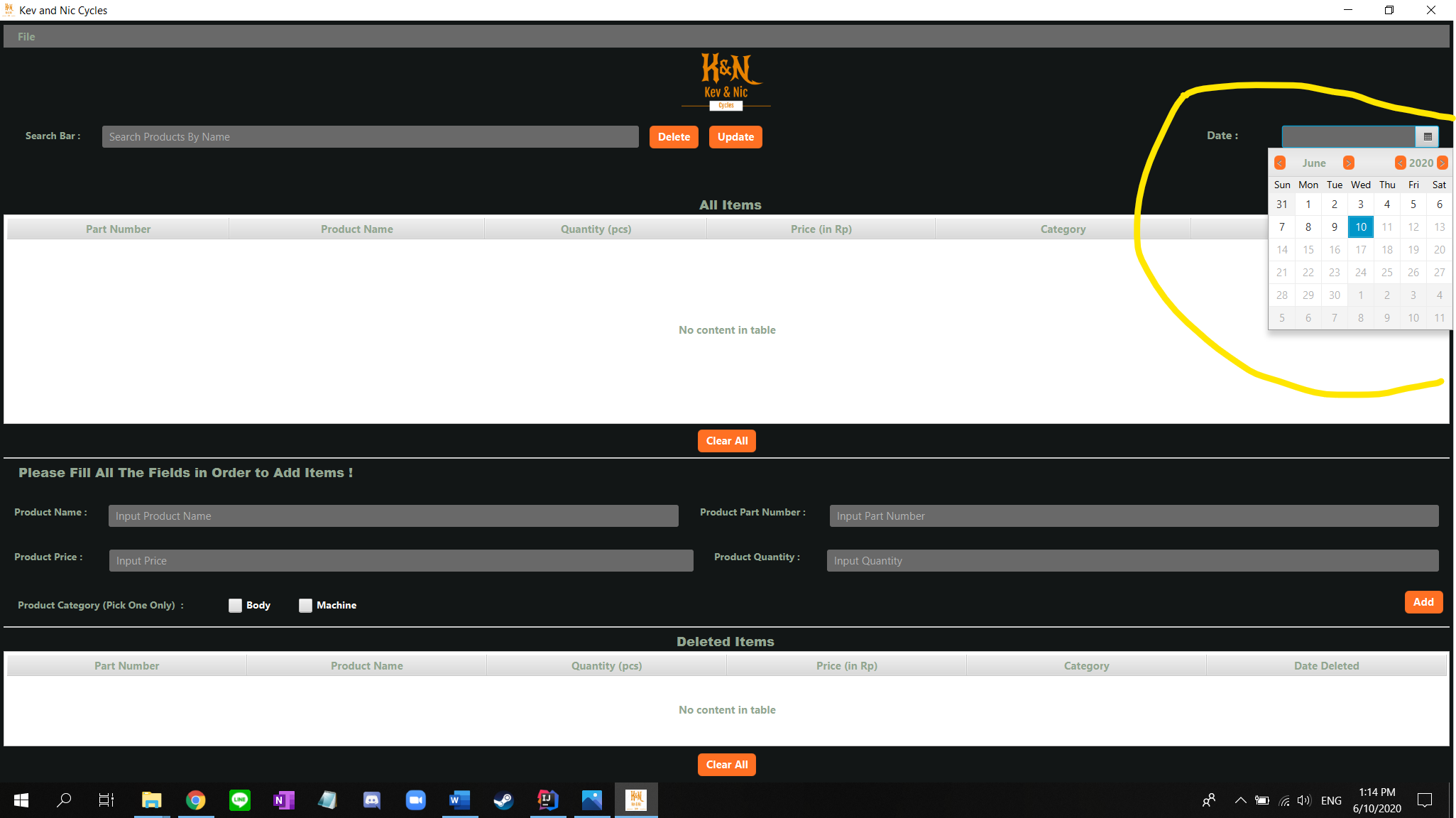


* 1. **The way to work with Date picker**

First date picker I use is in updateData() function. I assume that the user adds items in the present date or before, because the app is made to manage a new coming stock. So, its like impossible to set the date in the future. The thing that amazed me is that I can disable future dates starts from the day user opens the app.

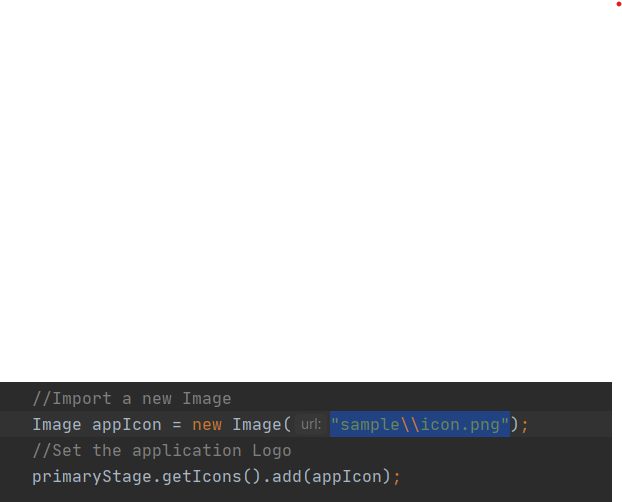


With this code, I learned that I can edit the date picker and inherit the updateItem() function from DateCell class. The code uses lambda expression saying everytime the picker is used by user the code inside will be executed. This function takes 2 parameters LocalDate object and Boolean variable. So, I set the local date as the day user opens the app and set disable days after the present day (which is > 0 means in the set disable method). I also set the Date picker field as uneditable. Here is what I meant by disabling future date cells.

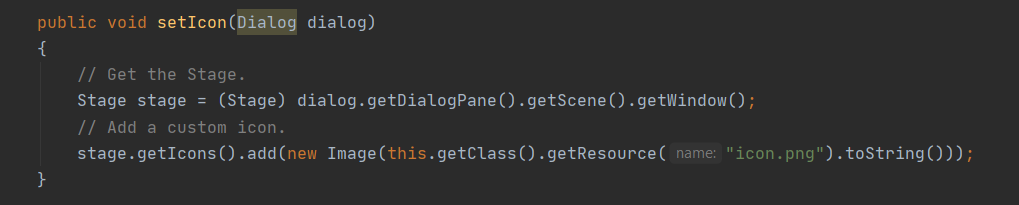


* 1. **The way to set icon for the app**

I did some research about how to add icons to the app. And it also took a long time for me to set the icon for my GUI app because the logo or image must be in the same directory. Eventually, I learned something new.



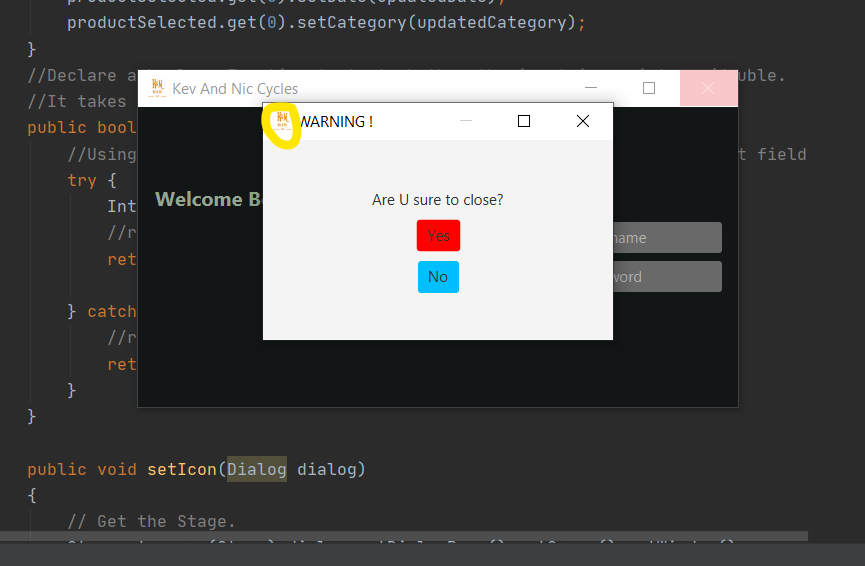
Code to set the app icon in main.java.

****

Function to set icon in every alert boxes and dialogs.

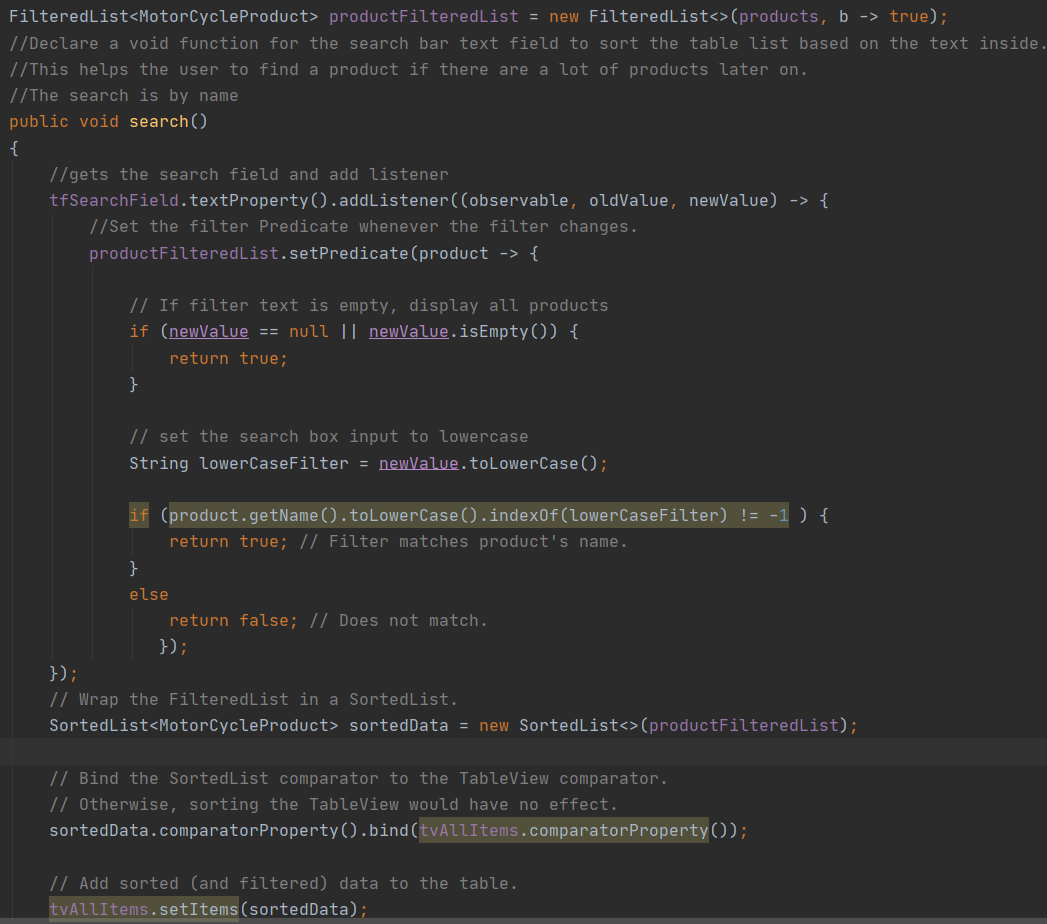


Screenshot of the icon that I implement for the app



Screenshot of the icon that I implement for the dialogue and alert box from setIcon() function.

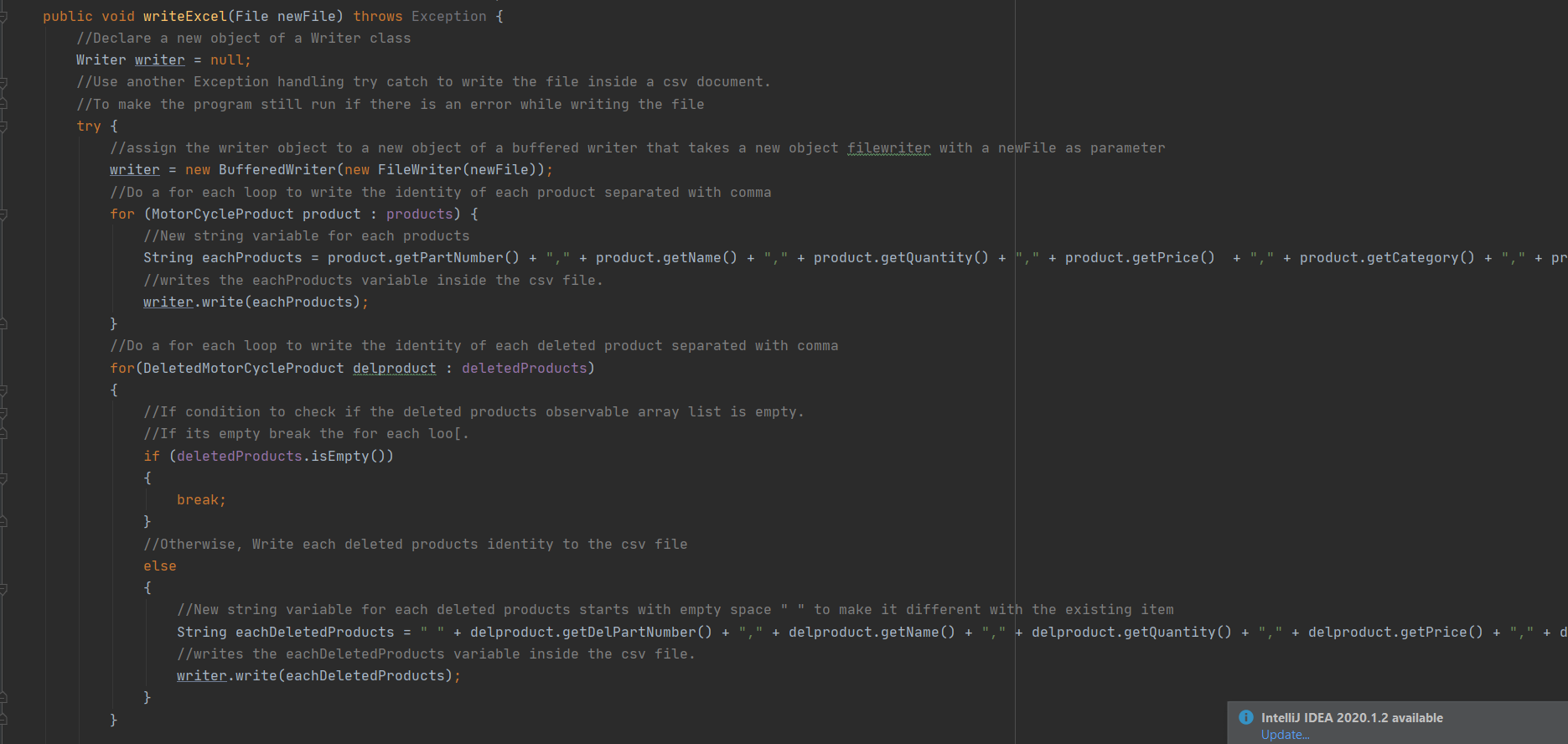
* 1. **Create a search box to search for an item by name.**



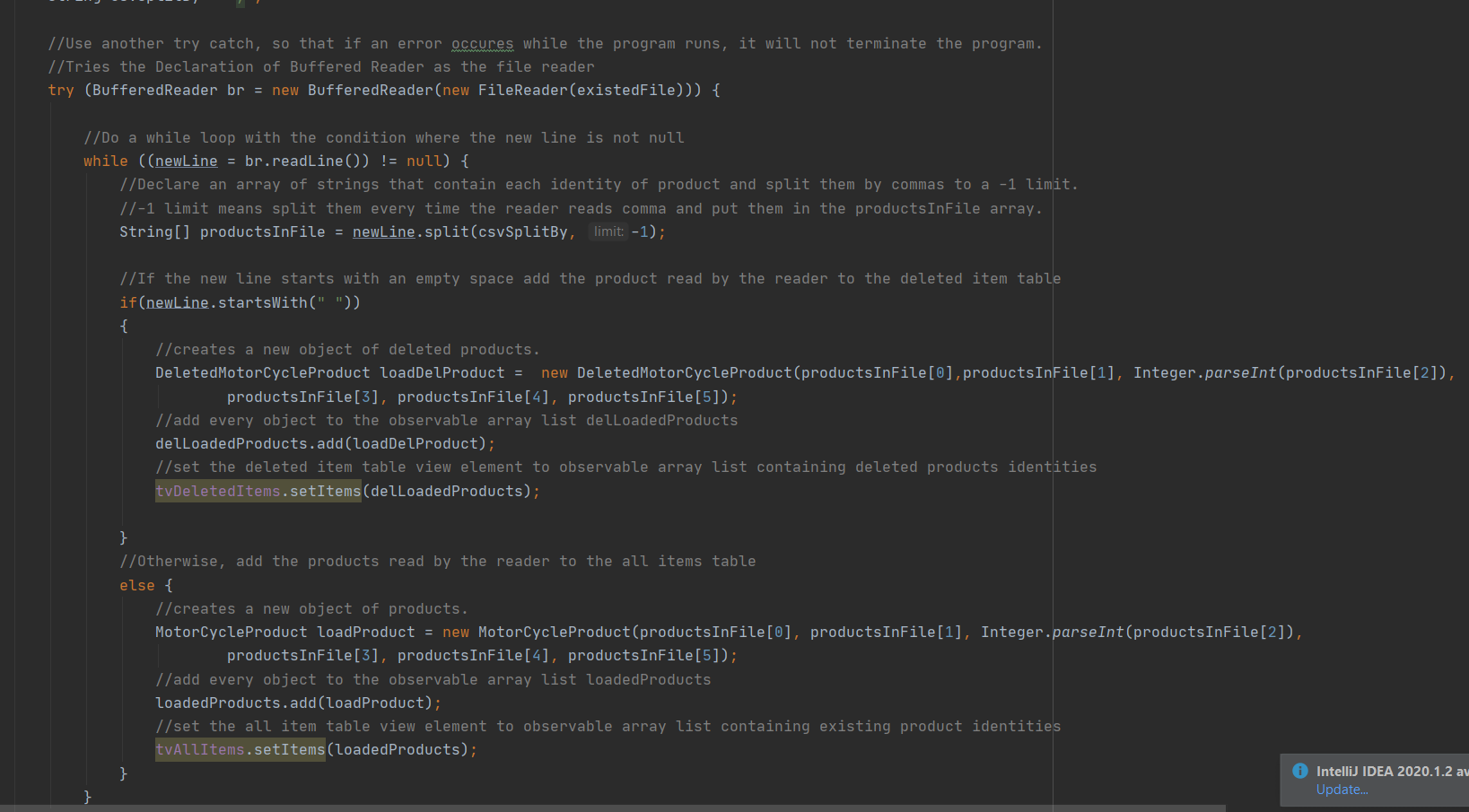
I also know how to create a search box to search for a product based on their names. How it works is whenever the search box is filled with a letter or a word, the all item table view will sort and search for a product name that matches with the word or letter name inside the table. If the there is a match, the table view will only display product that the user search for. Otherwise, if there is no match than the table view will display no content. In search() function, I used FilteredList to wrap the observable array list contains all products. And then, I get the search field and add a listener. Inside the listener, I set the filter list predicate whenever the filter changes. And some conditional statement to check if the search box is empty display all data. If it’s not empty, then the user input will be set to lower case and find each products name that matches. After that, wrap the filtered list in sorted list. Then, bind the sorted list comparator to the TableView comparator. Otherwise, sorting the TableView would have no effect. Finally, add sorted (and filtered data) to the table view.

* 1. **The way to save and open file**

After using this GUI application, of course users need to save their data so they can work with it again later. So, to save and open data I used CSV as the extension file. How it works is when saving the products identity, I used BufferedWriter as my file writer and each identity are separated with commas. and ends with enter (\n). When the user wants to open the file again, I used BufferedReader as my file reader. In the GUI app, I have 2 table views for all items and deleted items. The way to make the reader easy to determine between all items and deleted items is by adding an empty space when writing the deleted products inside the CSV document. In that way, the reader can easily determine which product belongs to all item table view and which product belongs to deleted item table view.



Screenshot of writeExcel() function



Screenshot of readExcel() function

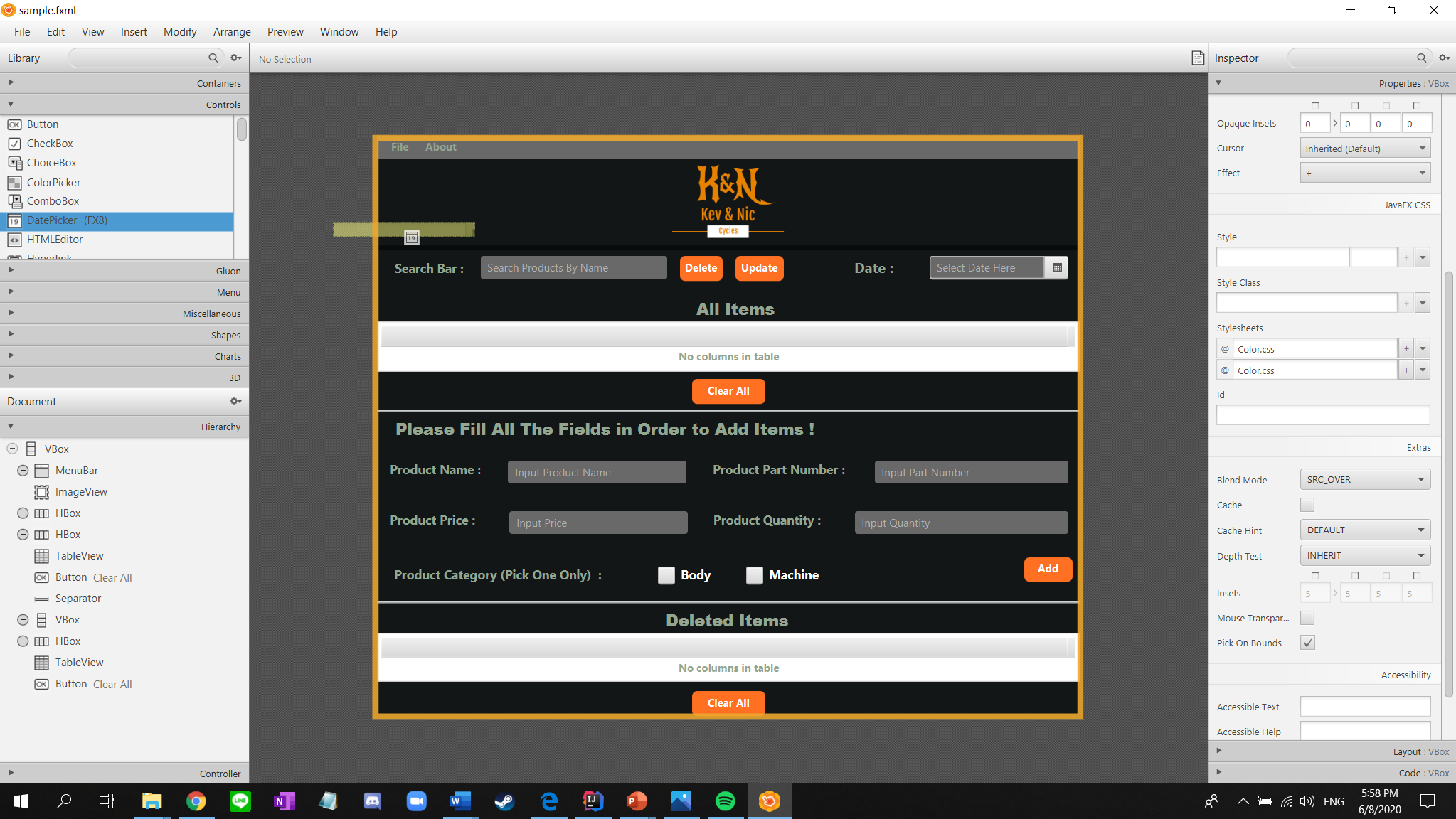
In each function, I used try catch in case there is an error while writing and reading the file. This prevents the program to terminate if there is an error. And to get identities of each product I implement a for each loop, to loop inside the observable list of products and deleted products. How I read the csv file is by creating variables for the delimiter of each product identity and the new line delimiter. Then to read one line I split them all into an array named productsInFile. Inside the split method there are 2 parameters one contains the delimiter another one contains the limit which is -1. Limit < 0 : In this case, the pattern will be applied as many times as possible, and the resulting array can be of any size.

* 1. **The way to work with File Chooser**

To work on saving and loading file, we need a file chooser to help us write and reading the file. File chooser helps us to set the intial directory where to read and write. I used FileChooser to set the directory and to show save and load data dialog.



**Using the Scene Builder by Gluon to work with the FXML file**

****

1. **Problems While Making GUI Application**

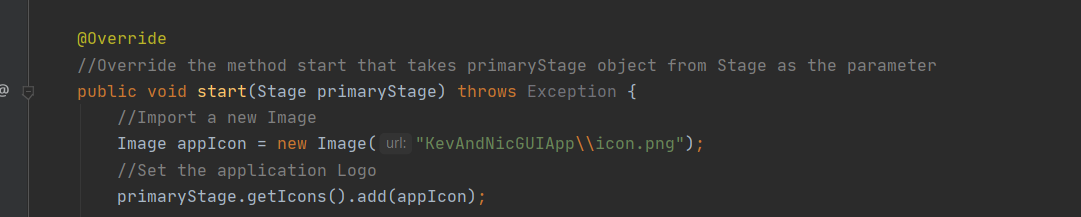
Making this program is not as easy as I imagine. I had to search for bunch of information and reference in the internet on how to implement Java FX using FXML. I also need to look for source code to deal with problems when connecting the FXML file with the controller class. I also need to make the GUI application to save and load files from a comma separated value (CSV) files. The problem while saving and loading file is to separate the all items table view element and the deleted items table view element. Another problem is when creating a search box for the all items table view. It’s a little bit complicated when trying to sync the search field with the searching code.

1. **Disclaimer Section**

There will be an error while tying to run the program from other devices. Because, I use a file chooser on working with saving and loading files. Also, in the program I set my initial directory based on my current directory in my device. Another technical problem maybe will occur while loading the icon of my app.



Screenshot of FileChooser Directory from initilaize function in Controller.Java



Screenshot of Setting the icon url from start function in Main.Jsva

1. **References**

* <https://stackoverflow.com> (website I used when I was trying to fix the errors)
* <https://code.makery.ch/blog/javafx-dialogs-official/>
* <https://noblecodemonkeys.com/javafx-menu-and-menu-items/#:~:text=Creating%20the%20Menu&text=In%20the%20FXML%2C%20simply%20use,heading%20is%20a%20%E2%80%9CMenuItem%E2%80%9D.>
* <https://noblecodemonkeys.com/javafx-filechooser-tutorial/>
* <https://community.oracle.com/message/10731570>
* <http://java-buddy.blogspot.com/2016/06/read-csv-run-in-background-thread-and.html>
* <http://tutorials.jenkov.com/javafx/menubar.html>
* <https://www.javatpoint.com/try-catch-block>
* <https://www.javatpoint.com/java-lambda-expressions>
* <http://www.java2s.com/Tutorials/Java/JavaFX/0470__JavaFX_PasswordField.htm>
* <https://www.youtube.com/watch?v=FLkOX4Eez6o&list=PL6gx4Cwl9DGBzfXLWLSYVy8EbTdpGbUIG>

Link to VIDEO DEMO : <https://www.youtube.com/watch?v=zm7shSmaggM&feature=youtu.be>