De La Salle University - Manila
Gokongwei College of Engineering

DONATION TRACKER

A Term-end Project in LBYCPA2 and DATSRAL

Gokongwei College of Engineering

De La Salle University

By:
Nacpil, Tristen Aaron D.

Neil Oliver Velasco

December 2023

Introduction

The Donation Tracker, painstakingly developed in Java, is a cutting-edge software solution designed to simplify and optimize the process of handling charitable contributions. This powerful tool enables organizations and individuals to precisely and efficiently track, record, and analyze donations.

The Donation Tracker utilizes the versatility of Java's programming capabilities to provide a user-friendly interface and smooth functionality. Its user-friendly interface enables users to easily submit and track financial contributions, offering a comprehensive overview of donation trends and patterns.

The accuracy, confidentiality, and scalability of data are prioritized in this Java-based donation management system.

Objectives

- Efficient Donation Tracking
- Promotes transparency
- Accessibility
- Donor engagement

Methodology

Flowchart:

Pseudocode:

In house class:

```
Class InHouse extends Part:

  private int machineID

  // Constructor
  InHouse(id, name, price, stock, min, max, machineId):
     Call Part constructor with parameters (id, name, price, stock, min, max)
     Set machineID to machineId

  // Setter method for machineID
  setMachineId(machineID):
     Set this.machineID to machineID
  // Getter method for machineID
  getMachineId():
     Return machineID
```

Inventory class:

```
Class Inventory:

  // Static variables
  private static final ObservableList<Part> allParts = createObservableListForAllParts();
  private static final ObservableList<Product> allProducts =
createObservableListForAllProducts();

  // Static methods

  // Method to create an observable list for all parts
  createObservableListForAllParts():
     Create an empty ObservableList of Part type
     Add new Outsourced instances to the list
     Add new InHouse instances to the list
     Return the created list

  // Method to create an observable list for all products
  createObservableListForAllProducts():
     Create an empty ObservableList of Product type
     Create ObservableLists for GuardianParts, CounselorParts, and SithParts
     Add new Product instances to the list with respective parts lists
     Return the created list

  // Method to add a part to the allParts list
  addPart(newPart):
     Add newPart to the allParts list

  // Method to add a product to the allProducts list
  addProduct(newProduct):
     Add newProduct to the allProducts list

  // Method to look up a part by partID
```

```
lookupPart(partID):
    Iterate through allParts
        If part's ID matches partID, return the part
    Return null if not found

// Method to look up a product by productID
lookupProduct(productID):
    Iterate through allProducts
        If product's ID matches productID, return the product
    Return null if not found

// Method to look up parts by partName
lookupPart(partName):
    Create an empty ObservableList for filtered parts
    Iterate through allParts
        If part's name matches partName, add it to the filtered list
    Return the filtered list

// Method to look up products by productName
lookupProduct(productName):
    Create an empty ObservableList for filtered products
    Iterate through allProducts
        If product's name matches productName, add it to the filtered list
    Return the filtered list

// Method to update a part at a specific index
updatePart(index, selectedPart):
    Set the part at index - 1 in allParts to selectedPart

// Method to update a product at a specific index
updateProduct(index, newProduct):
    Set the product at index in allProducts to newProduct

// Method to delete a part
deletePart(selectedPart):
    Get the ID of selectedPart
    Look up the part by ID
    Remove the part from allParts and return true if successful, false otherwise

// Method to delete a product
```

```
  deleteProduct(selectedProduct):
    Get the ID of selectedProduct
    Look up the product by ID
    Remove the product from allProducts and return true if successful, false otherwise

  // Method to get the list of all parts
  getAllParts():
    Return the allParts list

  // Method to get the list of all products
  getAllProducts():
    Return the allProducts list

  // Method to check if a string is numeric
  isNumeric(strNum):
    Check if strNum is not null
      Try to parse strNum to double
      If successful, return true; otherwise, catch NumberFormatException and return false
```

Outsourced class:

```
Class Outsourced extends Part:

  // Private attribute
  private String companyName

  // Constructor
  Outsourced(id, name, price, stock, min, max, companyName):
    Call Part constructor with parameters (id, name, price, stock, min, max)
    Set companyName to the provided companyName

  // Setter method for companyName
  setCompanyName(companyName):
    Set this.companyName to companyName

  // Getter method for companyName
  getCompanyName():
    Return companyName
```

Part class:

```
  // Private attributes
```

```
private int id
private String name
private double price
private int stock
private int min
private int max

// Constructor
Part(id, name, price, stock, min, max):
    Set this.id to id
    Set this.name to name
    Set this.price to price
    Set this.stock to stock
    Set this.min to min
    Set this.max to max

// Getter method for id
getId():
    Return id

// Setter method for id
setId(id):
    Set this.id to id

// Getter method for name
getName():
    Return name

// Setter method for name
setName(name):
    Set this.name to name

// Getter method for price
getPrice():
    Return price

// Setter method for price
setPrice(price):
    Set this.price to price

// Getter method for stock
getStock():
    Return stock
```

```
   // Setter method for stock
   setStock(stock):
      Set this.stock to stock

   // Getter method for min
   getMin():
      Return min

   // Setter method for min
   setMin(min):
      Set this.min to min

   // Getter method for max
   getMax():
      Return max

   // Setter method for max
   setMax(max):
      Set this.max to max


Add part class:
  // Private attributes
   private int id
   private String name
   private double price
   private int stock
   private int min
   private int max
   private ObservableList<Part> associatedParts

   // Constructor
   Product(id, name, price, stock, min, max, associatedParts):
      Set this.id to id
      Set this.name to name
      Set this.price to price
      Set this.stock to stock
      Set this.min to min
      Set this.max to max
      Set this.associatedParts to associatedParts

   // Getter method for id
   getId():
      Return id
```

```
// Setter method for id
setId(id):
    Set this.id to id

// Getter method for name
getName():
    Return name

// Setter method for name
setName(name):
    Set this.name to name

// Getter method for price
getPrice():
    Return price

// Setter method for price
setPrice(price):
    Set this.price to price

// Getter method for stock
getStock():
    Return stock

// Setter method for stock
setStock(stock):
    Set this.stock to stock

// Getter method for min
getMin():
    Return min

// Setter method for min
setMin(min):
    Set this.min to min

// Getter method for max
getMax():
    Return max

// Setter method for max
setMax(max):
    Set this.max to max
```

```
  // Method to add an associated part
  addAssociatedPart(part):
     Add part to the associatedParts list

  // Method to delete an associated part
  deleteAssociatedPart(selectedAssociatedPart):
     Get the ID of selectedAssociatedPart
     Iterate through associatedParts
        If part's ID matches ID, remove the part and set returnVal to true
     Return returnVal

  // Getter method for all associated parts
  getAllAssociatedParts():
     Return associatedParts

Add product class:
Class AddProductController implements Initializable:

  // FXML injected fields
  TextField textID
  TextField textName
  TextField textInventory
  TextField textPrice
  TextField textMin
  TextField textMax
  Button saveButton
  Button cancelButton
  TextField searchPartsBox
  TableView<Part> allPartsTable
  TableColumn<Part, Integer> allPartsPartIDCol
  TableColumn<Part, Integer> allPartsNameCol
  TableColumn<Part, String> allPartsInvCol
  TableColumn<Part, Integer> allPartsPriceCol
  TableView<Part> associatedPartsTable
  TableColumn<Part, Integer> asscPartsPartIDCol
  TableColumn<Part, String> asscPartsNameCol
  TableColumn<Part, String> assocPartsInvCol
  TableColumn<Part, Integer> assocPartsPriceCol
  Button addPartButton
  Button removeAssicatedPartButton

  // FXML method to handle cancel button press
  void setCancelButton(ActionEvent event):
        Show confirmation alert
        If user presses OK, navigate to MainScreen
        Otherwise, close the alert

    // FXML method to handle save button press
    void saveProductButtonPressed(ActionEvent event):
        Validate input data
        If data is valid:
            Show confirmation alert
            If user presses OK:
                Create a new Product with the provided data and associated parts
                Add the new Product to the Inventory
                Navigate to MainScreen

    // FXML method to handle add part button press
    void addPartButtonPressed():
        If no part is selected, show an error alert
        Otherwise, add the selected part to the associatedPartsTable

    // FXML method to update allPartsTable with all available parts
    void updateAllPartsTable():
        Set the items of allPartsTable to the list of all available parts in the Inventory

    // FXML method to handle remove associated part button press
    void setRemoveAssicatedPartButton():
        Show confirmation alert
        If user presses OK:
            Remove the selected associated part from the associatedPartsTable

    // FXML method to search for parts based on ID or name
    void searchParts():
        If the search box is empty, show all parts in allPartsTable
        Otherwise, try to search for parts based on input (handle both numeric and string searches)

    // FXML method to initialize the controller
    void initialize(URL url, ResourceBundle resourceBundle):
        Set cell value factories for columns in allPartsTable and associatedPartsTable
        Initialize allPartsTable with all available parts
        Set the next available ID for the new product in textID field


Mainscreen Class:
Class MainScreen implements Initializable:
```

```
// FXML injected fields
TextField partSearchBox
TextField productSearchBox
TableView<Part> partsTable
TableColumn<Part, Integer> partIdCol
TableColumn<Part, String> partNameCol
TableColumn<Part, Integer> partInventoryCol
TableColumn<Part, Integer> partPriceCol
TableView<Product> productTable
TableColumn<Product, Integer> productIdCol
TableColumn<Product, String> productNameCol
TableColumn<Product, Integer> productInventoryCol
TableColumn<Product, Integer> productPriceCol
Button partAddButton
Button partModifyButton
Button partDeleteButton
Button addProductButton
Button modifyProductButton
Button deleteProductButton
Button exitButton

// FXML method to search for parts based on ID or name
void searchParts():
   If the search box is empty, show all parts in partsTable
   Otherwise, try to search for parts based on input (handle both numeric and string searches)

// FXML method to search for products based on ID or name
void searchProducts():
   If the search box is empty, show all products in productTable
   Otherwise, try to search for products based on input (handle both numeric and string
searches)

// FXML method to add a new part
void addPartsAction(ActionEvent event):
   Navigate to AddPart view

// FXML method to modify a selected part
void modifyPartAction(ActionEvent event):
   If a part is selected, navigate to ModifyPart view with the selected part data
   Otherwise, show an error alert

// FXML method to delete a selected part
void deletePartsAction():
   If a part is selected, show confirmation alert
```

```
        If user presses OK, delete the selected part from Inventory and partsTable

    // FXML method to add a new product
    void addProductAction(ActionEvent event):
        Navigate to AddProduct view

    // FXML method to modify a selected product
    void modifyProductAction(ActionEvent event):
        If a product is selected, navigate to ModifyProduct view with the selected product data
        Otherwise, show an error alert

    // FXML method to delete a selected product
    void deleteProductAction():
        If a product is selected, show confirmation alert
        If user presses OK, delete the selected product from Inventory and productTable
        If the product has associated parts, show an error alert

    // FXML method to exit the application
    void exitButtonAction(ActionEvent event):
        Show confirmation alert
        If user presses OK, close the application
        Otherwise, close the alert

    // FXML method to update partsTable with all available parts
    void updateLightsaberParts():
        Set the items of partsTable to the list of all available parts in the Inventory

    // FXML method to update productTable with all available products
    void updateLightsaberProducts():
        Set the items of productTable to the list of all available products in the Inventory

    // FXML method to initialize the controller
    void initialize(URL url, ResourceBundle resourceBundle):
        Set cell value factories for columns in partsTable and productTable
        Initialize partsTable and productTable with all available parts and products, respectively
```

Modify part controller class:

```
Class ModifyPartController implements Initializable:

    // FXML injected fields
    RadioButton InHouseRadio
    RadioButton OutsourcedRadio
    TextField textID
    TextField textName
    TextField textInventory
    TextField textPrice
    TextField textMin
    TextField textMax
    TextField textDualPurpose
    Label dualPurpLabel
    Button saveButton
    Button cancelButton

    // Static field to hold the displayed part to be modified
    static Part displayedPart

    // Method to set up the view with data from the displayed part
    void setupView():
       Create a ToggleGroup for radio buttons
       Set text fields with data from displayedPart
       Set the selected radio button based on the type of displayedPart
       Update dualPurpLabel and textDualPurpose based on the selected radio button

    // FXML method to save modifications to the part
    void saveMod(ActionEvent event):
       Validate input values (numeric checks, min-max constraints)
       If validation passes, create a new part based on user input and update it in the Inventory
       Navigate back to MainScreen view

    // Method to display an alert for Machine ID input error
    void MachineIDError():
       Display an error alert for Machine ID input

    // Method to display an alert for Company Name input error
    void CompanyNameError():
       Display an error alert for Company Name input

    // FXML method to update UI based on radio button selection
    void updateRadioUI():
       Update dualPurpLabel based on the selected radio button

    // FXML method to cancel modification and return to MainScreen
    void setCancelButton(ActionEvent event) throws IOException:
       Display a confirmation alert
       If the user confirms, navigate back to MainScreen view
       Otherwise, close the alert
```

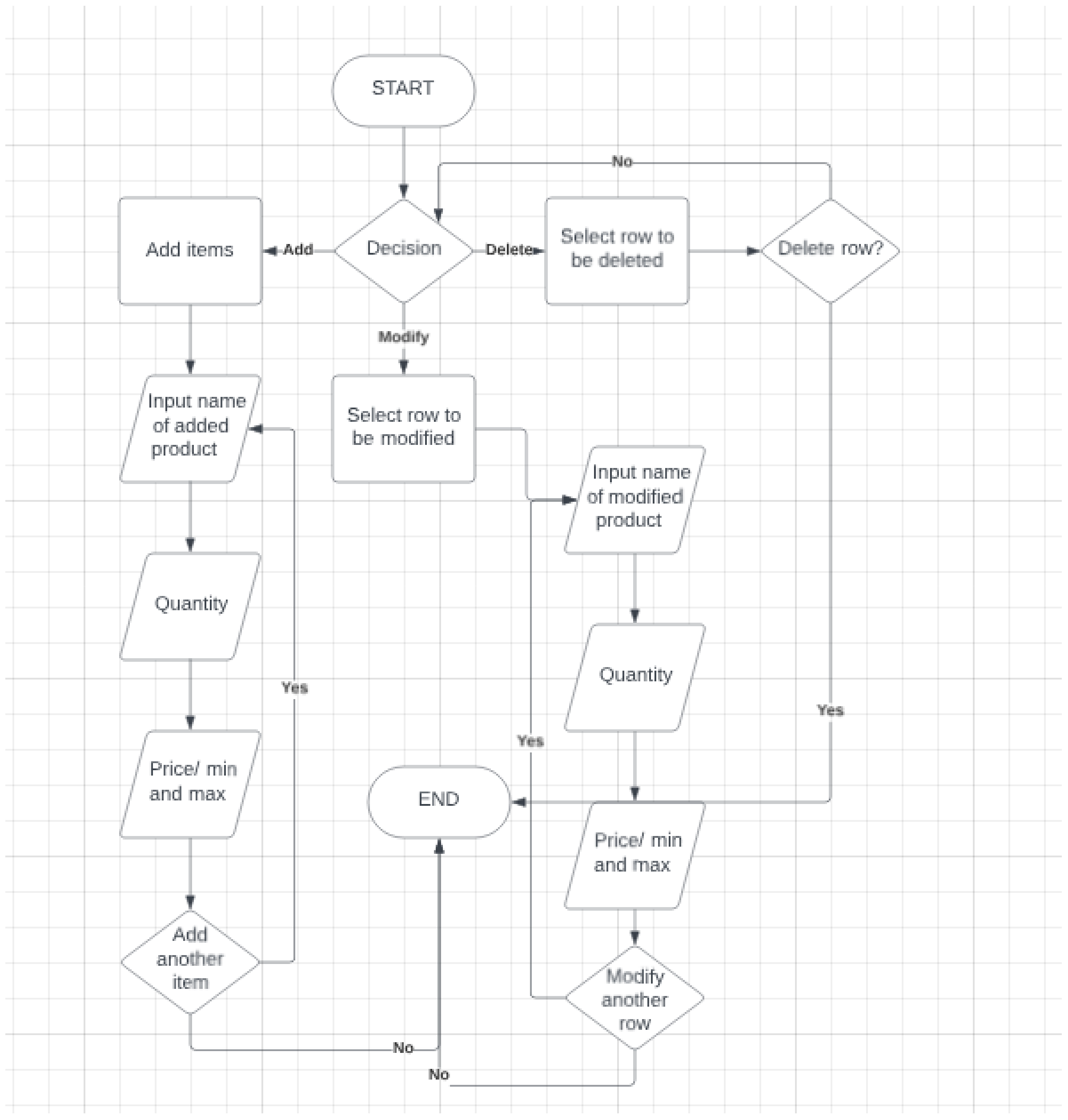
```
 // FXML method to initialize the controller
 void initialize(URL url, ResourceBundle resourceBundle):
    Call setupView() to populate the UI with displayed part data
```

Modify product controller class:

```
Class ModifyProductController implements Initializable:
   // Static variable to hold the currently displayed product
   Static Product displayedProduct

   // FXML-injected fields
   FXML TextField textID
   FXML TextField textName
   FXML TextField textInventory
   FXML TextField textPrice
   FXML TextField textMin
   FXML TextField textMax
   FXML Button saveButton
   FXML Button cancelButton
   FXML TextField searchPartsBox
   FXML TableView<Part> allPartsTable
   FXML TableColumn<Part, Integer> allPartsPartIDCol
   FXML TableColumn<Part, Integer> allPartsNameCol
   FXML TableColumn<Part, String> allPartsInvCol
   FXML TableColumn<Part, Integer> allPartsPriceCol
   FXML TableView<Part> associatedPartsTable
   FXML TableColumn<Part, Integer> asscPartsPartIDCol
   FXML TableColumn<Part, String> asscPartsNameCol
   FXML TableColumn<Part, String> assocPartsInvCol
   FXML TableColumn<Part, Integer> assocPartsPriceCol
   FXML Button addPartButton
   FXML Button removeAssicatedPartButton

   // Method to handle cancel button click event
   Method setCancelButton(ActionEvent event) throws IOException:
      Try
         Alert alert = ShowConfirmationDialog("Cancel?", "Are you sure you want to exit?",
"Press OK to exit the program. \nPress Cancel to stay on this screen.")
         If (alert result is OK)
            LoadMainScreen()
         Else
            CloseAlert()
      Catch (IOException E)
         PrintErrorMessage(E)
```

```
    // Method to handle save button click event
    Method saveProductButtonPressed(ActionEvent event) throws IOException,
NumberFormatException:
        If (Any required field is empty)
            ShowWarningAlert("Data Error", "Please enter valid data for every field")
        Else If (Any numeric field is not numeric)
            ShowWarningAlert("Value Error", "Min, Max, Inventory, and price should all be numeric")
        Else If (Min is greater than Max)
            ShowWarningAlert("Min Max Error", "Product Mins cannot be greater than Maxs")
        Else If (Inventory is not between Min and Max)
            ShowWarningAlert("Inventory Error", "Inventory should be between the min and max")
        Else
            Try
                Create a new Product from the entered data and associated parts
                Update the product in the inventory
                LoadMainScreen()
            Catch (IOException E)
                PrintErrorMessage(E)
            Catch (NumberFormatException E)
                ShowErrorAlert("Type Error", "Please format your inputs correctly")

    // Method to handle add part button click event
    Method addPartButtonPressed():
        If (No part is selected in the available parts table)
            ShowErrorAlert("No Selection", "Please select a part to add")
        Else
            Add the selected part to the associated parts table

    // Method to handle remove associated part button click event
    Method setRemoveAssociatedPartButton():
        Get the selected part from the associated parts table
        ShowConfirmationDialog("Remove Part Association?", "Remove Association for part: " +
selectedItem.getName() + "?")
        If (User confirms)
            Remove the part from the associated parts table
        Else
            CloseAlert()

    // Method to update the available parts table
    Method updateAllPartsTable():
        Set the items of the available parts table to all parts in the inventory

    // Method to update the associated parts table
    Method updateAssociatedPartsTable():
        Set the items of the associated parts table to all associated parts of the displayed product

    // Method to search for parts based on user input
    Method searchParts():
        If (Search box is empty)
            Set the items of the available parts table to all parts in the inventory
        Else
            Try
                Look up the part by ID and set the items of the available parts table accordingly
                If (No part is found)
                    ShowErrorAlert("No Parts Found", "Please Search Again")
            Catch (NumberFormatException e)
                Look up the part by name and set the items of the available parts table accordingly

    // Initialization method when the FXML is loaded
    Method initialize(URL url, ResourceBundle resourceBundle):
        Set up the columns of the available parts table
        Update the available parts table
        Set the text fields with information from the displayed product
        Set up the columns of the associated parts table
        Update the associated parts table


Main Class:
Class Main extends Application:
    // Method to start the JavaFX application
    Method start(Stage stage) throws IOException:
        Try
            // Create an FXMLLoader for loading the MainScreen.fxml file
            FXMLLoader fxmlLoader =
CreateFXMLLoader(MainScreen.class.getResource("MainScreen.fxml"))

            // Load the FXML file and get the controller instance
            Parent root = fxmlLoader.load()
            MainScreen mainScreen = fxmlLoader.getController()

            // Create a new scene with the loaded FXML content
            Scene scene = CreateScene(root)

            // Set the title, scene, and show the stage
            SetStageProperties(stage, "Inventory Management System", scene)
        Catch (IOException e)
            PrintStackTrace(e)
    // Method to launch the JavaFX application
    Method main(String[] args):
        Launch the JavaFX application with the given arguments

// Helper methods

// Method to create an FXMLLoader for loading FXML files
Method CreateFXMLLoader(URL resource):
    Return a new instance of FXMLLoader with the specified resource

// Method to create a new scene with the specified root
Method CreateScene(Parent root):
    Return a new instance of Scene with the specified root

// Method to set stage properties
Method SetStageProperties(Stage stage, String title, Scene scene):
    Set the title, scene, and show the stage
```

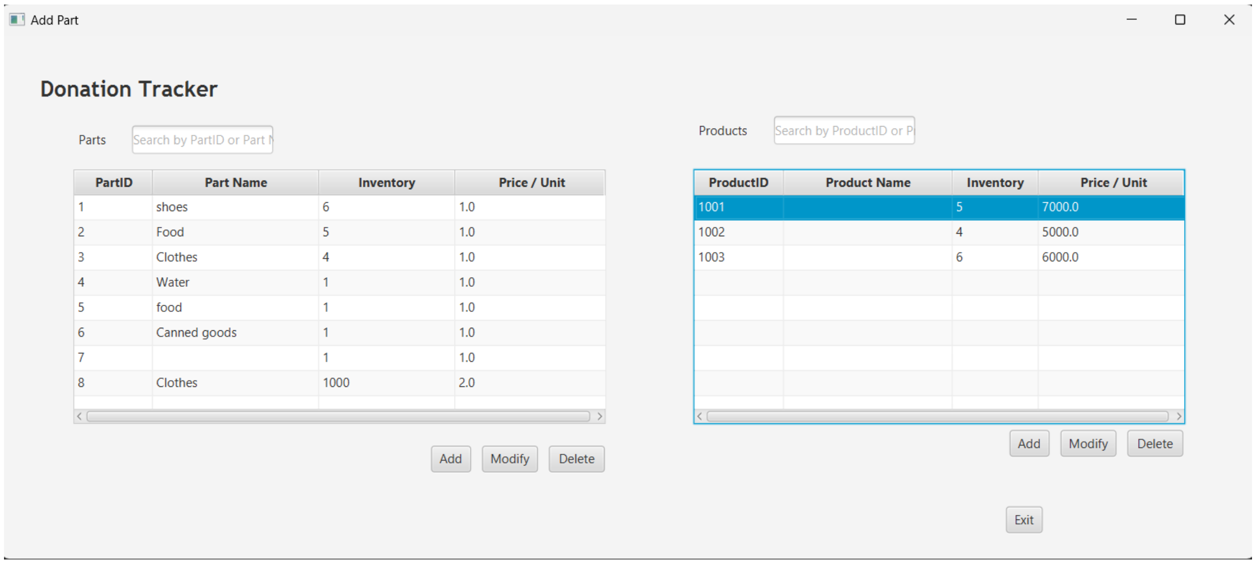
Flowchart



Hierarchy Charts for Modular Programming

Results and Discussion:

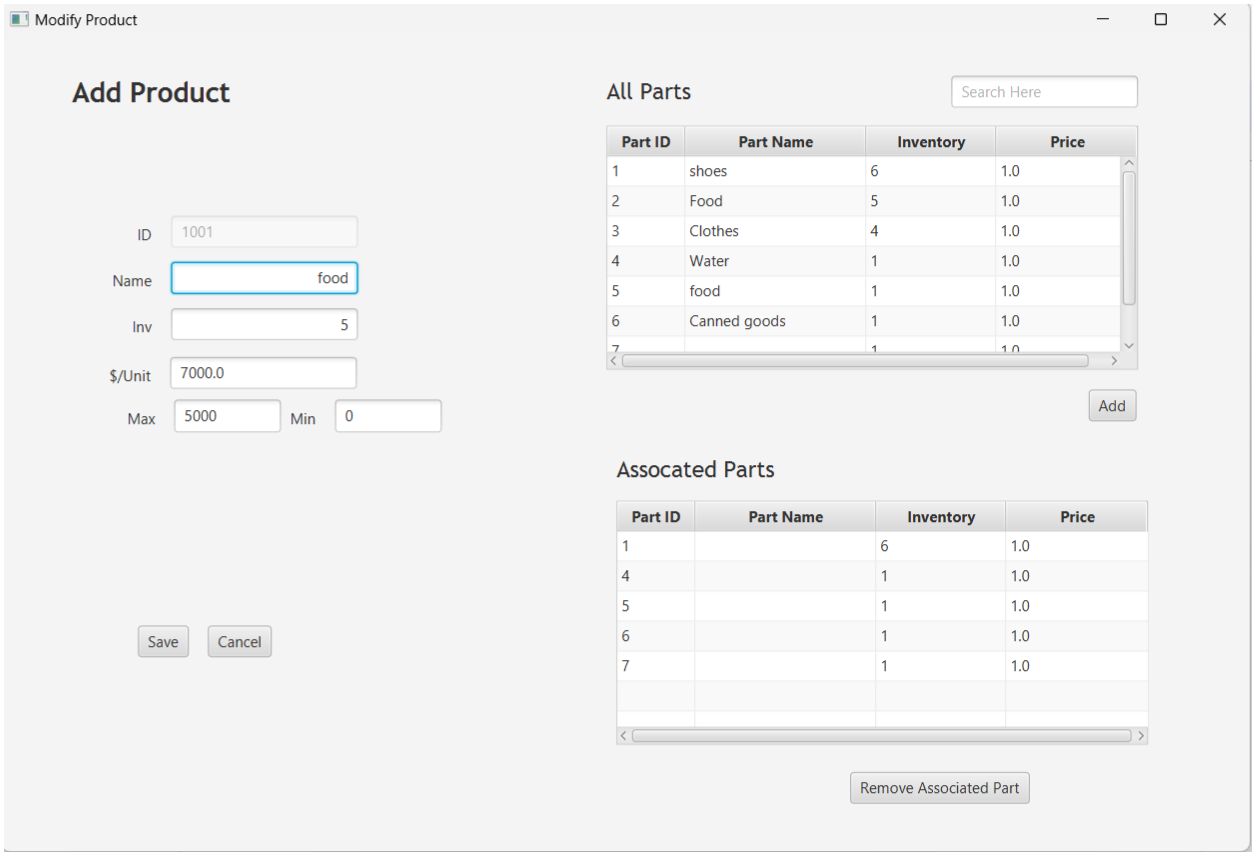
Sample Test Runs:



PartID	Part Name	Inventory	Price / Unit
1	shoes	6	1.0
2	Food	5	1.0
3	Clothes	4	1.0
4	Water	1	1.0
5	food	1	1.0
6	Canned goods	1	1.0
7		1	1.0
8	Clothes	1000	2.0

ProductID	Product Name	Inventory	Price / Unit
1001		5	7000.0
1002		4	5000.0
1003		6	6000.0

Different windows/screenshots of the program


Modify Product

Add Product

ID	1001
Name	food
Inv	5
$/Unit	7000.0
Max	5000
Min	0

Save Cancel

All Parts

Search Here

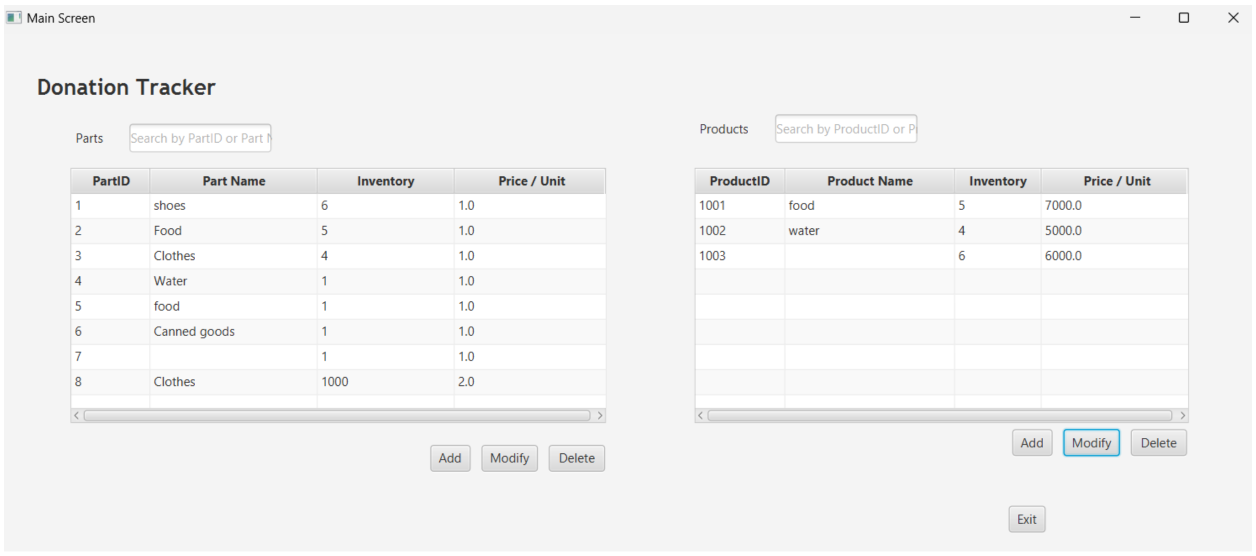
Part ID	Part Name	Inventory	Price
1	shoes	6	1.0
2	Food	5	1.0
3	Clothes	4	1.0
4	Water	1	1.0
5	food	1	1.0
6	Canned goods	1	1.0
7		1	1.0

Add

Assocated Parts

Part ID	Part Name	Inventory	Price
1		6	1.0
4		1	1.0
5		1	1.0
6		1	1.0
7		1	1.0

Remove Associated Part


Main Screen

Donation Tracker

Parts Search by PartID or Part N

PartID	Part Name	Inventory	Price / Unit
1	shoes	6	1.0
2	Food	5	1.0
3	Clothes	4	1.0
4	Water	1	1.0
5	food	1	1.0
6	Canned goods	1	1.0
7		1	1.0
8	Clothes	1000	2.0

Add Modify Delete

Products Search by ProductID or P

ProductID	Product Name	Inventory	Price / Unit
1001	food	5	7000.0
1002	water	4	5000.0
1003		6	6000.0

Add Modify Delete

Exit

Conclusion

To summarize, the Donation Tracker in Java is an essential tool for effective and impactful philanthropy. It satisfies the different demands of both organizations and individual contributors by streamlining donation management, improving transparency, and encouraging donor interaction. The application's user-friendly interface, integration possibilities, and commitment to continual improvement distinguish it as a dynamic tool capable of adapting to the changing landscape of charity initiatives.

Appendices

Code:

In house class:

```java
package lightsaberInventory.Model;


public class InHouse extends Part {

   private int machineID;


   public InHouse(int id, String name, double price, int stock, int min, int
max, int machineId) {
       super(id, name, price, stock, min, max);
       this.machineID = machineId;
   };



   public void setMachineId(int machineID) {
       this.machineID = machineID;
   }


   public int getMachineId() {
       return machineID;
   };


}
```

Inventory class:

```java
package lightsaberInventory.Model;
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;



public class Inventory {

   private static final ObservableList<Part> allParts =
FXCollections.observableArrayList(
           new Outsourced(1, "", 1, 6, 0 , 5000, ""),
           new Outsourced(2, "", 1, 5, 0, 5000, ""),
           new Outsourced(3, "", 1, 4, 0, 500, ""),
           new InHouse(4, "", 1, 1, 0, 5000,2),
           new InHouse(5, "", 1, 1, 0, 5000, 1),
           new InHouse(6, "", 1, 1, 0, 5000, 2),
           new InHouse(7, "", 1, 1, 0, 5000, 1)

   );


   private static ObservableList<Part> GuardianParts =
FXCollections.observableArrayList(
           lookupPart(1),
           lookupPart(4),
           lookupPart(5),
           lookupPart(6),
           lookupPart(7)
   );


   private static ObservableList<Part> CounselorParts =
FXCollections.observableArrayList(
           lookupPart(2),
           lookupPart(4),
           lookupPart(5),
           lookupPart(6),
           lookupPart(7)
   );

   private static ObservableList<Part> SithParts =
FXCollections.observableArrayList(
           lookupPart(3),
           lookupPart(4),
           lookupPart(5),
           lookupPart(6),
           lookupPart(7)
   );


   private static ObservableList<Product> allProducts =
FXCollections.observableArrayList(
           new Product(1001, "", 7000, 5, 0, 5000, GuardianParts),
           new Product(1002, "", 5000, 4,0,5000, CounselorParts),
           new Product(1003, "", 6000, 6, 0, 5000, SithParts)
   );
```

```java
    public static void addPart(Part newPart) {
        allParts.add(newPart);
    };


    public static void addProduct(Product newProduct) {
        allProducts.add(newProduct);
    };


    public static Part lookupPart(int partID) {
        for(Part part : allParts) {
            if (part.getId() == partID) {
                return part;
            }
        }
        return null;
    };


    public static Product lookupProduct(int productID){
        for(Product prod  : allProducts) {
            if (prod.getId() == productID) {
                return prod;
            }
        }
        return null;
    };

    public static ObservableList<Part> lookupPart(String partName){
        ObservableList<Part> filteredPartsList =
FXCollections.observableArrayList();
        for (Part p : allParts) {
            if (partName.compareTo(p.getName()) == 0) {
                filteredPartsList.add(p);
            }
        }
        return filteredPartsList;

    };


    public static ObservableList<Product> lookupProduct(String productName) {
        ObservableList<Product> filteredProductList =
FXCollections.observableArrayList();
        for (Product p : allProducts) {
            if (productName.compareTo(p.getName()) == 0) {
                filteredProductList.add(p);
            }
        }
        return filteredProductList;

    };


    public static void updatePart(int index, Part selectedPart) {
        allParts.set(index - 1, selectedPart);
    };


    public static void updateProduct(int index, Product newProduct){
        allProducts.set(index, newProduct);
    }




    public static boolean deletePart(Part selectedPart) {
        int id = selectedPart.getId();
        Part lookupPart = lookupPart(id);

        return allParts.remove(lookupPart);
    };


    public static boolean deleteProduct(Product selectedProduct) {
        int id = selectedProduct.getId();
        Product lookupProduct = lookupProduct(id);
        return allProducts.remove(lookupProduct);
    }


    public static ObservableList<Part> getAllParts() {
        return allParts;
    }

    public static ObservableList<Product> getAllProducts() {
        return allProducts;
    }



    public static boolean isNumeric(String strNum) {
        if (strNum == null) {
            return false;
        }
        try {
            double d = Double.parseDouble(strNum);
        } catch (NumberFormatException nfe) {
            return false;
        }
        return true;
    }


}
```

Outsourced class:

```
package lightsaberInventory.Model;



public class Outsourced extends Part {

    private String companyName;


    public Outsourced(int id, String name, double price, int stock, int min, int
max, String companyName) {
        super(id, name, price, stock, min, max);
        this.companyName = companyName;
    };


    public void setCompanyName(String companyName) {
        this.companyName = companyName;
    }

    public String getCompanyName() {
        return companyName;
    };


}
```

Part class:

```
package lightsaberInventory.Model;



public abstract class Part {
    private int id;
    private String name;
    private double price;
    private int stock;
```

```java
    private int min;
    private int max;
    public Part(int id, String name, double price, int stock, int min, int max)
{
        this.id = id;
        this.name = name;
        this.price = price;
        this.stock = stock;
        this.min = min;
        this.max = max;
    }

    /**
     * @return the id
     */
    public int getId() {
        return id;
    }

    /**
     * @param id the id to set
     */
    public void setId(int id) {
        this.id = id;
    }

    /**
     * @return the name
     */
    public String getName() {
        return name;
    }

    /**
     * @param name the name to set
     */
    public void setName(String name) {
        this.name = name;
    }

    /**
     * @return the price
     */
    public double getPrice() {
        return price;
    }

    /**
     * @param price the price to set
     */
    public void setPrice(double price) {
        this.price = price;
    }

    /**
     * @return the stock
     */
    public int getStock() {
        return stock;
    }

    /**
     * @param stock the stock to set
     */
    public void setStock(int stock) {
        this.stock = stock;
    }

    /**
     * @return the min
     */
    public int getMin() {
        return min;
    }

    /**
     * @param min the min to set
     */
    public void setMin(int min) {
        this.min = min;
    }

    /**
     * @return the max
     */
    public int getMax() {
        return max;
    }

    /**
     * @param max the max to set
     */
    public void setMax(int max) {
        this.max = max;
    }

}
```

Product Class:
```java
package lightsaberInventory.Model;

import javafx.collections.ObservableList;

/** The Product class defines methods and attributes for Products. */

public class Product {

   private int id;
   private String name;
   private double price;
   private int stock;
   private int min;
   private int max;
   private ObservableList<Part> associatedParts;


   public Product(int id, String name, double price, int stock, int min, int
max, ObservableList<Part> associatedParts) {
       this.id = id;
       this.name = name;
       this.price = price;
       this.stock = stock;
       this.min = min;
       this.max = max;
       this.associatedParts = associatedParts;
   }



   public int getId() {
       return id;
   }


   public void setId(int id) {
       this.id = id;
   }

   public String getName() {
       return name;
   }


   public void setName(String name) {
       this.name = name;
   }
```

```java
public double getPrice() {
    return price;
}


public void setPrice(double price) {
    this.price = price;
}

public int getStock() {
    return stock;
}

public void setStock(int stock) {
    this.stock = stock;
}


public int getMin() {
    return min;
}


public void setMin(int min) {
    this.min = min;
}


public int getMax() {
    return max;
}


public void setMax(int max) {
    this.max = max;
}


public void addAssociatedPart(Part part) {
    associatedParts.add(part);
};



public boolean deleteAssociatedPart(Part selectedAssociatedPart){
    int id = selectedAssociatedPart.getId();
    boolean returnVal = false;
    for(int i = 0; i < associatedParts.size(); i++) {
           if (associatedParts.get(i).getId() == id) {
               associatedParts.remove(i);
               returnVal = true;
           }
       }
       return returnVal;
   };



   public ObservableList<Part> getAllAssociatedParts() {
       return associatedParts;
   }


}
```

Add part class:

```
package lightsaberInventory.View;

import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.fxml.Initializable;
import javafx.scene.Node;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.stage.Stage;
import lightsaberInventory.Model.InHouse;
import lightsaberInventory.Model.Inventory;
import lightsaberInventory.Model.Outsourced;
import lightsaberInventory.Model.Part;

import java.io.IOException;
import java.net.URL;
import java.util.ResourceBundle;


public class AddPartController implements Initializable {

   @FXML
   RadioButton InHouseRadio;

   @FXML
   RadioButton OutsourcedRadio;

   @FXML
```

```
TextField textID;

@FXML
TextField textName;

@FXML
TextField textInventory;

@FXML
TextField textPrice;

@FXML
TextField textMin;

@FXML
TextField textMax;

@FXML
TextField textDualPurpose;

@FXML
Label dualPurpLabel;

@FXML
Button addButton;

@FXML
Button cancelButton;



@FXML
public void updateRadioUI() {
    ToggleGroup Tgroup = new ToggleGroup();
    InHouseRadio.setToggleGroup(Tgroup);
    OutsourcedRadio.setToggleGroup(Tgroup);

    if (InHouseRadio.isSelected()) {
        dualPurpLabel.setText("Machine ID");
    }
    else if (OutsourcedRadio.isSelected()) {
        dualPurpLabel.setText("Company Name");
    }

};



@FXML
```

```
   void setCancelButton(ActionEvent event) throws IOException {

       try {
           Alert alert = new Alert(Alert.AlertType.CONFIRMATION);
           alert.setTitle("Cancel?");
           alert.setHeaderText("Are you sure you want to exit?");
           alert.setContentText("Press OK to exit the program. \nPress Cancel
to stay on this screen.");
           alert.showAndWait();
           if (alert.getResult() == ButtonType.OK) {

               FXMLLoader loader = new
FXMLLoader(getClass().getResource("MainScreen.fxml"));
               Parent addPartScreen = loader.load();
               Scene addPartScene = new Scene(addPartScreen);
               Stage winAddPart =
(Stage)((Node)event.getSource()).getScene().getWindow();
               winAddPart.setTitle("Add Part");
               winAddPart.setScene(addPartScene);
               winAddPart.show();
           }
           else {
               alert.close();
           }
       }
       catch (IOException E) {
           System.out.println(E.getLocalizedMessage());
       }

   };



   @FXML
   public void addPartButtonPressed(ActionEvent event) throws IOException,
NumberFormatException {
       if (textName.getText().isEmpty()
               || textInventory.getText().isEmpty()
               || textDualPurpose.getText().isEmpty()
               || textMin.getText().isEmpty()
               || textMax.getText().isEmpty()
               || textPrice.getText().isEmpty()) {

           System.out.println("Data Empty");
           Alert alert = new Alert(Alert.AlertType.WARNING);
           alert.setTitle("Data Error");
           alert.setHeaderText("Please enter valid data for every field");
           alert.showAndWait();
       }
```

```
        else if (!Inventory.isNumeric(textMax.getText())
                || !Inventory.isNumeric(textMin.getText())
                || !Inventory.isNumeric(textPrice.getText())
                || !Inventory.isNumeric(textInventory.getText())) {


            Alert alert = new Alert(Alert.AlertType.WARNING);
            alert.setTitle("Value Error");
            alert.setHeaderText("Min, Max, Inventory, and price should all be
numeric");
            alert.showAndWait();

        }

        else if (Integer.parseInt(textMin.getText()) >
Integer.parseInt(textMax.getText())) {
            System.out.println("Min Max Error");
            Alert alert = new Alert(Alert.AlertType.WARNING);
            alert.setTitle("Min Max Error");
            alert.setHeaderText("Part Mins cannot be greater than Maxs");
            alert.showAndWait();
        }

        else if (Integer.parseInt(textMin.getText()) >
Integer.parseInt(textInventory.getText()) ||
Integer.parseInt(textInventory.getText()) >
Integer.parseInt(textMax.getText())) {
            System.out.println("Inventory Error");
            Alert alert = new Alert(Alert.AlertType.WARNING);
            alert.setTitle("Inventory Error");
            alert.setHeaderText("Inventory should be between the min and max");
            alert.showAndWait();
        }


        else {

            System.out.println(!Inventory.isNumeric(textInventory.getText()));

            System.out.println("Data not empty");
            try {
                int id = Integer.parseInt(textID.getText());
                String name = textName.getText();
                int invText = Integer.parseInt(textInventory.getText());
                double price = Double.parseDouble(textPrice.getText());
                int min = Integer.parseInt(textMin.getText());
                int max = Integer.parseInt(textMax.getText());
                String dualText = textDualPurpose.getText();
```

```java
                Alert alert = new Alert(Alert.AlertType.CONFIRMATION);
                alert.setTitle("Add Part");
                alert.setHeaderText("Would like to save this part to the
inventory? ");
                alert.showAndWait();


                if (alert.getResult() == ButtonType.OK)  {


                    if (InHouseRadio.isSelected()) {
                        InHouse inhousePart = new InHouse(id, name, price,
invText, min, max, Integer.parseInt(dualText));
                        inhousePart.setMachineId(Integer.parseInt(dualText));
                        System.out.println(inhousePart.getMachineId());
                        Inventory.addPart(inhousePart);

                    }
                    else if (OutsourcedRadio.isSelected()) {
                        System.out.println("Saving Outsourced");
                        Outsourced outsourcedPart = new Outsourced(id, name,
price, invText, min, max, dualText);
                        outsourcedPart.setCompanyName(dualText);
                        System.out.println(outsourcedPart.getCompanyName());
                        Inventory.addPart(outsourcedPart);
                    }


                    FXMLLoader loader = new
FXMLLoader(getClass().getResource("MainScreen.fxml"));
                    Parent root = loader.load();
                    Scene scene = new Scene(root);
                    Stage mainScreen =
(Stage)((Node)event.getSource()).getScene().getWindow();
                    mainScreen.setScene(scene);
                    mainScreen.show();

                }
                else {
                    alert.close();
                }
            }
            catch (IOException E) {
                Alert alert = new Alert(Alert.AlertType.ERROR);
                alert.setTitle("Type Error");
                alert.setHeaderText(E.getLocalizedMessage());
                alert.showAndWait();
            }
```

```
           catch (NumberFormatException E) {
               Alert alert = new Alert(Alert.AlertType.ERROR);
               alert.setTitle("Type Error");
               alert.setHeaderText("Please format your inputs like the 
following:" +
                       "\nName: String" +
                       "\nPrice: Double" +
                       "\nMin, Max, Inventory: Integer" +
                       "\nMachine ID: Number " +
                       "\nCompany Name: String");
               alert.showAndWait();
           }


       }




   }




   @Override
   public void initialize(URL url, ResourceBundle resourceBundle) {
       InHouseRadio.setSelected(true);

       Part lastPart = 
Inventory.getAllParts().get(Inventory.getAllParts().size() - 1);
       int lastID = lastPart.getId();
       textID.setText(String.valueOf(++lastID));

   }
}
```

Add product class

```
package lightsaberInventory.View;

import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.fxml.Initializable;
import javafx.scene.Node;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.control.cell.PropertyValueFactory;
import javafx.stage.Stage;
```

```java
import lightsaberInventory.Model.*;

import java.io.IOException;
import java.net.URL;
import java.util.ResourceBundle;


public class AddProductController implements Initializable {


   @FXML
   TextField textID;

   @FXML
   TextField textName;

   @FXML
   TextField textInventory;

   @FXML
   TextField textPrice;

   @FXML
   TextField textMin;

   @FXML
   TextField textMax;

   @FXML
   Button saveButton;

   @FXML
   Button cancelButton;

   @FXML
   TextField searchPartsBox;

   @FXML
   TableView<Part> allPartsTable;

   @FXML
   TableColumn<Part, Integer> allPartsPartIDCol;

   @FXML
   TableColumn<Part, Integer> allPartsNameCol;

   @FXML
   TableColumn<Part,String>allPartsInvCol;
   @FXML
   TableColumn<Part, Integer> allPartsPriceCol;

   @FXML
   TableView<Part> associatedPartsTable;

   @FXML
   TableColumn<Part, Integer> asscPartsPartIDCol;

   @FXML
   TableColumn<Part, String> asscPartsNameCol;

   @FXML
   TableColumn<Part,String>assocPartsInvCol;

   @FXML
   TableColumn<Part, Integer> assocPartsPriceCol;

   @FXML
   Button addPartButton;

   @FXML
   Button removeAssicatedPartButton;


   /** Displays MainScreen and does not save new product. The new product
instance is abandoned.
    * @throws IOException failed to read the file */
   @FXML
   void setCancelButton(ActionEvent event) throws IOException {

       try {
           Alert alert = new Alert(Alert.AlertType.CONFIRMATION);
           alert.setTitle("Cancel?");
           alert.setHeaderText("Are you sure you want to exit?");
           alert.setContentText("Press OK to exit the program. \nPress Cancel
to stay on this screen.");
           alert.showAndWait();
           if (alert.getResult() == ButtonType.OK) {

               FXMLLoader loader = new
FXMLLoader(getClass().getResource("MainScreen.fxml"));
               Parent addPartScreen = loader.load();
               Scene addPartScene = new Scene(addPartScreen);
               Stage winAddPart =
(Stage)((Node)event.getSource()).getScene().getWindow();
               winAddPart.setTitle("Add Part");
               winAddPart.setScene(addPartScene);
               winAddPart.show();
```

```
            }
            else {
                alert.close();
            }
        }
        catch (IOException E) {
            System.out.println(E.getLocalizedMessage());
        }

    };


    /** Saves new product and displays MainScreen. The new product instance is
added to allProducts if it passes error checks.
     * Checks include, but are not limited to: Min is less than Inventory is
less than Max, all textFields are filled in.
     * @param event An ActionEvent to help scene transition
     * @throws IOException failed to read the file
     * @throws NumberFormatException inputted  a string in a field designed for
integers */
    @FXML
    public void saveProductButtonPressed(ActionEvent event) throws IOException,
NumberFormatException {
        if (textName.getText().isEmpty()
                || textInventory.getText().isEmpty()
                || textMin.getText().isEmpty()
                || textMax.getText().isEmpty()
                || textPrice.getText().isEmpty()) {

            System.out.println("Data Empty");
            Alert alert = new Alert(Alert.AlertType.WARNING);
            alert.setTitle("Data Error");
            alert.setHeaderText("Please enter valid data for every field");
            alert.showAndWait();
        }

        else if (!Inventory.isNumeric(textMax.getText())
                || !Inventory.isNumeric(textMin.getText())
                || !Inventory.isNumeric(textPrice.getText())
                || !Inventory.isNumeric(textInventory.getText())) {


            Alert alert = new Alert(Alert.AlertType.WARNING);
            alert.setTitle("Value Error");
            alert.setHeaderText("Min, Max, Inventory, and price should all be
numeric");
            alert.showAndWait();

        }
```

```
       else if (Integer.parseInt(textMin.getText()) >
Integer.parseInt(textMax.getText())) {
           System.out.println("Min Max Error");
           Alert alert = new Alert(Alert.AlertType.WARNING);
           alert.setTitle("Min Max Error");
           alert.setHeaderText("Product Mins cannot be greater than Maxs");
           alert.showAndWait();
       }
       else if (Integer.parseInt(textMin.getText()) >
Integer.parseInt(textInventory.getText()) ||
Integer.parseInt(textInventory.getText()) >
Integer.parseInt(textMax.getText())) {
           System.out.println("Inventory Error");
           Alert alert = new Alert(Alert.AlertType.WARNING);
           alert.setTitle("Inventory Error");
           alert.setHeaderText("Inventory should be between the min and max");
           alert.showAndWait();
       }

       else {
           System.out.println("Data not empty");
           try {
               int id = Integer.parseInt(textID.getText());
               String name = textName.getText();
               int invText = Integer.parseInt(textInventory.getText());
               double price = Double.parseDouble(textPrice.getText());
               int min = Integer.parseInt(textMin.getText());
               int max = Integer.parseInt(textMax.getText());

               Alert alert = new Alert(Alert.AlertType.CONFIRMATION);
               alert.setTitle("Add Product");
               alert.setHeaderText("Would like to save this Product to the
inventory? ");
               alert.showAndWait();


               if (alert.getResult() == ButtonType.OK)  {
                   Product newProduct = new Product(id, name, price, invText,
min, max, associatedPartsTable.getItems());
                   System.out.println(newProduct);
                   Inventory.addProduct(newProduct);

                   FXMLLoader loader = new
FXMLLoader(getClass().getResource("MainScreen.fxml"));
                   Parent root = loader.load();
                   Scene scene = new Scene(root);
                   Stage mainScreen =
(Stage)((Node)event.getSource()).getScene().getWindow();
                   mainScreen.setScene(scene);
                   mainScreen.show();
               }
               else {
                   alert.close();
               }
           }
           catch (IOException E) {
               System.out.println(E.getLocalizedMessage());
           }
           catch ( NumberFormatException E) {
               Alert alert = new Alert(Alert.AlertType.ERROR);
               alert.setTitle("Type Error");
               alert.setHeaderText("Please format your inputs like the
following:" +
                       "\nName: String" +
                       "\nPrice: Double" +
                       "\nMin, Max, Inventory: Integer");
               alert.showAndWait();
           }


       }

   }

   /** Associates a selected part with the product. If no part is selected
display an alert error */
   @FXML
   public void addPartButtonPressed() {
       if (allPartsTable.getSelectionModel().getSelectedItem() == null) {
           Alert alert = new Alert(Alert.AlertType.ERROR);
           alert.setTitle("No Selection");
           alert.setHeaderText("Please select a part to add");
           alert.showAndWait();
       }

       else {
           Part selectedItem =
allPartsTable.getSelectionModel().getSelectedItem();
           associatedPartsTable.getItems().add(selectedItem);
       }

   }
```

```
    /** Loads the all parts for the allPartsTable.  Helper function do to any
necessary data population after scene is loaded. */
    @FXML
    public void updateAllPartsTable() {
        allPartsTable.setItems(Inventory.getAllParts());
    }

    /** Disassociates a selected part with the product. Confirmation alert is
shown. */
    @FXML
    public void setRemoveAssicatedPartButton() {
        Part selectedItem =
associatedPartsTable.getSelectionModel().getSelectedItem();

        Alert alert = new Alert(Alert.AlertType.CONFIRMATION);
        alert.setTitle("Remove Part Association?");
        alert.setHeaderText("Remove Association for part: " +
selectedItem.getName() + "?");
        alert.showAndWait();

        if (alert.getResult() == ButtonType.OK) {
            associatedPartsTable.getItems().remove(selectedItem);
        }
        else {
            alert.close();
        }

    }


    /** Search function applied to the parts search bar. Filters the
allPartsTable with parts matching inputted ID or name */
    @FXML
    public void searchParts() {
        if (searchPartsBox.getText().trim().isEmpty()) {
            allPartsTable.setItems(Inventory.getAllParts());
        }

        else {
            try {
                Part returnedPart =
Inventory.lookupPart(Integer.parseInt(searchPartsBox.getText()));

                if (returnedPart == null) {
                    Alert alert = new Alert(Alert.AlertType.ERROR);
                    alert.setTitle("No Parts Found");
                    alert.setHeaderText("Please Search Again");
                    alert.showAndWait();
                }
                else {
                    ObservableList<Part> filteredPartsList =
FXCollections.observableArrayList();
                    filteredPartsList.add(returnedPart);
                    allPartsTable.setItems(filteredPartsList);
                }
            }

            catch (NumberFormatException e) {
                System.out.println("Number Format Exception");

allPartsTable.setItems(Inventory.lookupPart(searchPartsBox.getText().trim()));
            }
        }

    }

    /** Loads the allPartsTable and assocPartsTable with columns names. Helper
function do to any necessary data population after scene is loaded. */
    @Override
    public void initialize(URL url, ResourceBundle resourceBundle) {

        allPartsPartIDCol.setCellValueFactory(new PropertyValueFactory<>("id"));
        allPartsNameCol.setCellValueFactory(new PropertyValueFactory<>("name"));
        allPartsInvCol.setCellValueFactory(new PropertyValueFactory<>("stock"));
        allPartsPriceCol.setCellValueFactory(new
PropertyValueFactory<>("price"));
        updateAllPartsTable();


        asscPartsPartIDCol.setCellValueFactory(new
PropertyValueFactory<>("id"));
        asscPartsNameCol.setCellValueFactory(new
PropertyValueFactory<>("name"));
        assocPartsInvCol.setCellValueFactory(new
PropertyValueFactory<>("stock"));
        assocPartsPriceCol.setCellValueFactory(new
PropertyValueFactory<>("price"));

        Product lastProduct =
Inventory.getAllProducts().get(Inventory.getAllProducts().size() - 1);
        int lastID = lastProduct.getId();
        textID.setText(String.valueOf(++lastID));

    }
}
```

Mainscreen class

```
package lightsaberInventory.View;
```

```java
import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.fxml.Initializable;
import javafx.scene.Node;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.control.cell.PropertyValueFactory;
import javafx.stage.Stage;
import lightsaberInventory.Model.Inventory;
import lightsaberInventory.Model.Part;
import lightsaberInventory.Model.Product;

import java.io.IOException;
import java.net.URL;
import java.util.ResourceBundle;

public class MainScreen implements Initializable {

   @FXML
   TextField partSearchBox;

   @FXML
   TextField productSearchBox;

   @FXML
   TableView<Part> partsTable = new TableView<Part>(Inventory.getAllParts());

   @FXML
   TableColumn<Part, Integer> partIdCol;

   @FXML
   TableColumn<Part, String> partNameCol;

   @FXML
   TableColumn<Part, Integer> partInventoryCol;

   @FXML
   TableColumn<Part,Integer> partPriceCol;

   @FXML
   TableView<Product> productTable = new
TableView<Product>(Inventory.getAllProducts());

   @FXML
   TableColumn<Product, Integer> productIdCol;

   @FXML
   TableColumn<Product, String> productNameCol;

   @FXML
   TableColumn<Product, Integer> productInventoryCol;

   @FXML
   TableColumn<Product, Integer> productPriceCol;


   /** Search function applied to the Parts search bar. Filters the partsTable
with parts matching inputted ID or name */
   @FXML
   public void searchParts() {
       if (partSearchBox.getText().trim().isEmpty()) {
           partsTable.setItems(Inventory.getAllParts());
       }

       else {
           try {
               Part returnedPart =
Inventory.lookupPart(Integer.parseInt(partSearchBox.getText()));

               if (returnedPart == null) {
                   Alert alert = new Alert(Alert.AlertType.ERROR);
                   alert.setTitle("No Parts Found");
                   alert.setHeaderText("Please Search Again");
                   alert.showAndWait();
               }
               else {
                   ObservableList<Part> filteredPartsList =
FXCollections.observableArrayList();
                   filteredPartsList.add(returnedPart);
                   partsTable.setItems(filteredPartsList);
               }

           }

           catch (NumberFormatException e) {
               System.out.println("Number Format Exception");

partsTable.setItems(Inventory.lookupPart(partSearchBox.getText().trim()));
           }


       }
```

```
    }

    /** Search function applied to the Products search bar. Filters the
productTable with parts matching inputted ID or name */
    @FXML
    public void searchProducts() {
        if (productSearchBox.getText().trim().isEmpty()) {
            productTable.setItems(Inventory.getAllProducts());
        }

        else {
            try {
                Product returnedProduct =
Inventory.lookupProduct(Integer.parseInt(productSearchBox.getText()));
                if (returnedProduct == null) {
                    Alert alert = new Alert(Alert.AlertType.ERROR);
                    alert.setTitle("No Products Found");
                    alert.setHeaderText("Please Search Again");
                    alert.showAndWait();
                }
                else {
                    ObservableList<Product> filteredProductList =
FXCollections.observableArrayList();
                    filteredProductList.add(returnedProduct);
                    productTable.setItems(filteredProductList);
                }

            }

            catch (NumberFormatException e) {
                System.out.println("Number Format Exception");

productTable.setItems(Inventory.lookupProduct(productSearchBox.getText().trim()
));
            }
        }

    }


    @FXML
    Button partAddButton;

    @FXML
    Button partModifyButton;

    @FXML
    Button partDeleteButton;
```

```
   /** Displays the addPart view. Allows users to add a part the part table.
    @param screenAddParts An ActionEvent to help scene transition
    @throws IOException failed to read the file*/
   @FXML
   public void addPartsAction (ActionEvent screenAddParts) throws IOException {
       try {
           FXMLLoader loader = new
FXMLLoader(getClass().getResource("AddPart.fxml"));
           Parent addPartScreen = loader.load();
           Scene addPartScene = new Scene(addPartScreen);
           Stage winAddPart =
(Stage)((Node)screenAddParts.getSource()).getScene().getWindow();
           winAddPart.setTitle("Add Part");
           winAddPart.setScene(addPartScene);
           winAddPart.show();
       }
       catch (IOException e) {
           System.out.println(e.getLocalizedMessage());
       }
   }

   /** Displays the modifyPart view. Allows users to modify a selected part
from part table.
    @param event An ActionEvent to help scene transition
    @throws IOException failed to read the file*/
   @FXML
   public void modifyPartAction (ActionEvent event) throws IOException {

       if (partsTable.getSelectionModel().getSelectedItem() == null) {
           Alert alert = new Alert(Alert.AlertType.ERROR);
           alert.setTitle("No Selection");
           alert.setHeaderText("Please select a part to modify");
           alert.showAndWait();

       }
       else {

           Part selectedItem =
partsTable.getSelectionModel().getSelectedItem();
           System.out.println(selectedItem.getName());

           ModifyPartController.displayedPart = selectedItem;


           try {
           FXMLLoader loader = new
FXMLLoader(getClass().getResource("ModifyPart.fxml"));
           Parent addPartScreen = loader.load();
           Scene addPartScene = new Scene(addPartScreen);
```

```
            Stage winAddPart =
(Stage)((Node)event.getSource()).getScene().getWindow();
            winAddPart.setTitle("Modify Part");
            winAddPart.setScene(addPartScene);
            winAddPart.show();
            }
            catch (IOException e) {
                System.out.println(e.getLocalizedMessage());
            }
        }



    }

    /** Deletes a part from the partTable. Allows users to delete a selected
part from part table. */
    @FXML
    public void deletePartsAction() {
        if (partsTable.getSelectionModel().getSelectedItem() == null) {
            Alert alert = new Alert(Alert.AlertType.ERROR);
            alert.setTitle("No Selection");
            alert.setHeaderText("Please select a part to modify");
            alert.showAndWait();

        }

        else {
            Part selectedItem =
partsTable.getSelectionModel().getSelectedItem();
            Alert alert = new Alert(Alert.AlertType.CONFIRMATION);
            alert.setTitle("Delete Part?");
            alert.setHeaderText("Delete " + selectedItem.getName() + "?");
            alert.showAndWait();

            if (alert.getResult() == ButtonType.OK)  {
                Inventory.deletePart(selectedItem);
                partsTable.getItems().remove(selectedItem);


            }
            else {
                alert.close();
            }

        }

    }
```

```
   // Product Section

   /** Displays the addProduct view. Allows users to add a product the product
table.
    @param screenAddProducts An ActionEvent to help scene transition
    @throws IOException failed to read the file */
   @FXML
   public void addProductAction (ActionEvent screenAddProducts) throws
IOException {
       try {
           FXMLLoader loader = new
FXMLLoader(getClass().getResource("AddProduct.fxml"));
           Parent addPartScreen = loader.load();
           Scene addPartScene = new Scene(addPartScreen);
           Stage winAddPart =
(Stage)((Node)screenAddProducts.getSource()).getScene().getWindow();
           winAddPart.setTitle("Add Part");
           winAddPart.setScene(addPartScene);
           winAddPart.show();
       }
       catch (IOException e) { assert true; }
   }

   /** Displays the modifyProduct view. Allows users to modify a selected
product from product table.
    @param event An ActionEvent to help scene transition */
   @FXML
   public void modifyProductAction(ActionEvent event) {
       if (productTable.getSelectionModel().getSelectedItem() == null) {
           Alert alert = new Alert(Alert.AlertType.ERROR);
           alert.setTitle("No Selection");
           alert.setHeaderText("Please select a product to modify");
           alert.showAndWait();

       }
       else {

           Product selectedItem =
productTable.getSelectionModel().getSelectedItem();
           System.out.println(selectedItem.getName());

           ModifyProductController.displayedProduct = selectedItem;

           try {
               FXMLLoader loader = new
FXMLLoader(getClass().getResource("ModifyProduct.fxml"));
               Parent addPartScreen = loader.load();
               Scene addPartScene = new Scene(addPartScreen);
```

```
                Stage winAddPart =
(Stage)((Node)event.getSource()).getScene().getWindow();
                winAddPart.setTitle("Modify Product");
                winAddPart.setScene(addPartScene);
                winAddPart.show();
            }
            catch (IOException e) {
                System.out.println(e.getLocalizedMessage());
            }
        }

    }

    /** Deletes a product from the productTable. If no associated parts exist,
deletes a selected product from product table. */
    @FXML
    public void deleteProductAction() {
        if (productTable.getSelectionModel().getSelectedItem() == null) {
            Alert alert = new Alert(Alert.AlertType.ERROR);
            alert.setTitle("No Selection");
            alert.setHeaderText("Please select a part to modify");
            alert.showAndWait();

        }
        else {
            Product selectedItem =
productTable.getSelectionModel().getSelectedItem();

            Alert alert = new Alert(Alert.AlertType.CONFIRMATION);
            alert.setTitle("Delete Product?");
            alert.setHeaderText("Delete " + selectedItem.getName() + "?");
            alert.showAndWait();

            if (alert.getResult() == ButtonType.OK)   {

                if (!selectedItem.getAllAssociatedParts().isEmpty()) {
                    Alert newAlert = new Alert(Alert.AlertType.ERROR);
                    newAlert.setTitle("Delete Error");
                    newAlert.setHeaderText("Products cant be delete if they have
associated parts");
                    newAlert.showAndWait();
                }
                else {
                    Inventory.deleteProduct(selectedItem);
                    productTable.getItems().remove(selectedItem);
                }

            }
            else {
                alert.close();
            }




        }
    }



    /** Populates the partsTable with parts. Gets parts list from Inventory. */
    @FXML
    public void updateLightsaberParts() {
        partsTable.setItems(Inventory.getAllParts());
    }

    /** Populates the productsTable with products. Gets product list from
 Inventory. */
    @FXML
    public void updateLightsaberProducts() {
        productTable.setItems(Inventory.getAllProducts());
    };

    /** Terminates the app. If users confirms, Lightsaber Inventory is
 terminated.
     * @param event An ActionEvent to help scene transition */
    @FXML
    public void exitButtonAction (ActionEvent event) {
        Alert alert = new Alert(Alert.AlertType.CONFIRMATION);
        alert.setTitle("Exit Lightsaber Inventory");
        alert.setHeaderText("Are you sure you want to exit?");
        alert.setContentText("Press OK to exit the program. \nPress Cancel to
 stay on this screen.");
        alert.showAndWait();
        if (alert.getResult() == ButtonType.OK) {
            Stage winMainScreen =
 (Stage)((Node)event.getSource()).getScene().getWindow();
            winMainScreen.close();
        }
        else {
            alert.close();
        }
    }




    /** Loads the partsTable and productTable with columns names.  Helper
 function do to any necessary data population after scene is loaded. */
```

```
   @Override
   public void initialize(URL url, ResourceBundle resourceBundle) {

       partIdCol.setCellValueFactory(new PropertyValueFactory<>("id"));
       partNameCol.setCellValueFactory(new PropertyValueFactory<>("name"));
       partInventoryCol.setCellValueFactory(new
PropertyValueFactory<>("stock"));
       partPriceCol.setCellValueFactory(new PropertyValueFactory<>("price"));
       updateLightsaberParts();

       productIdCol.setCellValueFactory(new PropertyValueFactory<>("id"));
       productNameCol.setCellValueFactory(new PropertyValueFactory<>("name"));
       productInventoryCol.setCellValueFactory(new
PropertyValueFactory<>("stock"));
       productPriceCol.setCellValueFactory(new
PropertyValueFactory<>("price"));
       updateLightsaberProducts();

   }
}
```

Modify part controller class:

```
package lightsaberInventory.View;

import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.fxml.Initializable;
import javafx.scene.Node;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.stage.Stage;
import lightsaberInventory.Model.InHouse;
import lightsaberInventory.Model.Inventory;
import lightsaberInventory.Model.Outsourced;
import lightsaberInventory.Model.Part;

import java.io.IOException;
import java.net.URL;
import java.util.ResourceBundle;


/** ModifyPart controller of LightSaber Inventory application. */
public class ModifyPartController  implements Initializable {

   public static Part displayedPart;
```

```
    @FXML
    RadioButton InHouseRadio;

    @FXML
    RadioButton OutsourcedRadio;

    @FXML
    TextField textID;

    @FXML
    TextField textName;

    @FXML
    TextField textInventory;

    @FXML
    TextField textPrice;

    @FXML
    TextField textMin;

    @FXML
    TextField textMax;

    @FXML
    TextField textDualPurpose;

    @FXML
    Label dualPurpLabel;

    @FXML
    Button saveButton;

    @FXML
    Button cancelButton;



    public void setupView() {
        ToggleGroup Tgroup = new ToggleGroup();
        InHouseRadio.setToggleGroup(Tgroup);
        OutsourcedRadio.setToggleGroup(Tgroup);

        textID.setText(String.valueOf(displayedPart.getId()));
        textName.setText(displayedPart.getName());
        textInventory.setText(String.valueOf(displayedPart.getStock()));
        textPrice.setText(String.valueOf(displayedPart.getPrice()));
        textMax.setText(String.valueOf(displayedPart.getMax()));
        textMin.setText(String.valueOf(displayedPart.getMin()));
        if (displayedPart instanceof InHouse) {
            InHouseRadio.setSelected(true);
            dualPurpLabel.setText("Machine ID");
            textDualPurpose.setText(Integer.toString(((InHouse)
displayedPart).getMachineId()));
        }
        else {
            OutsourcedRadio.setSelected(true);
            dualPurpLabel.setText("Company Name");
            textDualPurpose.setText(((Outsourced)
displayedPart).getCompanyName());
        }

    }



    @FXML
    public void saveMod(ActionEvent event) throws IOException,
NumberFormatException {

        if (!Inventory.isNumeric(textMax.getText())
                || !Inventory.isNumeric(textMin.getText())
                || !Inventory.isNumeric(textPrice.getText())
                || !Inventory.isNumeric(textInventory.getText())) {

            Alert alert = new Alert(Alert.AlertType.WARNING);
            alert.setTitle("Value Error");
            alert.setHeaderText("Min, Max, Inventory, and price should all be
numeric");
            alert.showAndWait();

        }

        else if (Integer.parseInt(textMin.getText()) >
Integer.parseInt(textMax.getText())) {
            System.out.println("Min Max Error");
            Alert alert = new Alert(Alert.AlertType.WARNING);
            alert.setTitle("Min Max Error");
            alert.setHeaderText("Part Mins cannot be greater than Maxs");
            alert.showAndWait();
        }
        else if (Integer.parseInt(textMin.getText()) >
Integer.parseInt(textInventory.getText()) ||
Integer.parseInt(textInventory.getText()) >
Integer.parseInt(textMax.getText())) {
            System.out.println("Inventory Error");
            Alert alert = new Alert(Alert.AlertType.WARNING);
            alert.setTitle("Inventory Error");
            alert.setHeaderText("Inventory should be between the min and max");
            alert.showAndWait();
        }
        else {
            try {
                int index = displayedPart.getId();

                String name = textName.getText();
                int invText = Integer.parseInt(textInventory.getText());
                double price = (Double.parseDouble(textPrice.getText()));
                int min = Integer.parseInt(textMin.getText());
                int max = Integer.parseInt(textMax.getText());
                String dualText = textDualPurpose.getText();

                if (InHouseRadio.isSelected()) {

                    if (!Inventory.isNumeric(dualText)) {
                        MachineIDError();
                    }
                    else {
                        InHouse inhousePart = new InHouse(index, name, price,
invText, min, max, Integer.parseInt(dualText));
                        Inventory.updatePart(index, inhousePart);

                        FXMLLoader loader = new
FXMLLoader(getClass().getResource("MainScreen.fxml"));
                        Parent addPartScreen = loader.load();
                        Scene addPartScene = new Scene(addPartScreen);
                        Stage winAddPart =
(Stage)((Node)event.getSource()).getScene().getWindow();
                        winAddPart.setTitle("Add Part");
                        winAddPart.setScene(addPartScene);
                        winAddPart.show();
                    }
                }

                else if (OutsourcedRadio.isSelected()) {

                    if (Inventory.isNumeric(dualText)) {
                        CompanyNameError();
                    }
                    else {
                        Outsourced outsourcedPart = new Outsourced(index, name,
price, invText, min, max, dualText);
                        Inventory.updatePart(index, outsourcedPart);

                        FXMLLoader loader = new
FXMLLoader(getClass().getResource("MainScreen.fxml"));
```

```
                       Parent addPartScreen = loader.load();
                       Scene addPartScene = new Scene(addPartScreen);
                       Stage winAddPart =
(Stage)((Node)event.getSource()).getScene().getWindow();
                       winAddPart.setTitle("Add Part");
                       winAddPart.setScene(addPartScene);
                       winAddPart.show();
                   }

               }


           }
           catch (IOException E) {
               System.out.println(E.getLocalizedMessage());
           }
           catch ( NumberFormatException E) {
               Alert alert = new Alert(Alert.AlertType.ERROR);
               alert.setTitle("Type Error");
               alert.setHeaderText("Please format your inputs like the
following:" +
                       "\nName: String" +
                       "\nPrice: Double" +
                       "\nMin, Max, Inventory: Integer" +
                       "\nMachine ID: Number " +
                       "\nCompany Name: String");
               alert.showAndWait();
           }

       }


   }

   /** Displays an alert error based on the machineID TextField. */
   public void MachineIDError() {
       System.out.println("Machine Id Error");
       Alert alert = new Alert(Alert.AlertType.ERROR);
       alert.setTitle("Machine Id Error");
       alert.setHeaderText("Machine ID should be an Integer");
       alert.showAndWait();
   }

   /** Displays an alert error based on the companyName TextField. */
   public void CompanyNameError() {
       System.out.println("Company Name Error");
       Alert alert = new Alert(Alert.AlertType.ERROR);
       alert.setTitle("Company Name Error");
       alert.setHeaderText("Company Name should be an String");
```

```
        alert.showAndWait();
    }


    /** Updates the UI of based on Radio Buttons. Changes the dualPurpLabel
based on selection. */
    @FXML
    public void updateRadioUI() {
        if (InHouseRadio.isSelected()) {
            dualPurpLabel.setText("Machine ID");
        }
        else if (OutsourcedRadio.isSelected()) {
            dualPurpLabel.setText("Company Name");
        }

    };


    /** Displays MainScreen and does not save part changes. The modifications
to the part instance are abandoned.
     * @throws IOException failed to read the file */
    @FXML
    void setCancelButton(ActionEvent event) throws IOException {

        try {
            Alert alert = new Alert(Alert.AlertType.CONFIRMATION);
            alert.setTitle("Cancel?");
            alert.setHeaderText("Are you sure you want to exit?");
            alert.setContentText("Press OK to discard edits.");
            alert.showAndWait();
            if (alert.getResult() == ButtonType.OK) {

                FXMLLoader loader = new
FXMLLoader(getClass().getResource("MainScreen.fxml"));
                Parent addPartScreen = loader.load();
                Scene addPartScene = new Scene(addPartScreen);
                Stage winAddPart =
(Stage)((Node)event.getSource()).getScene().getWindow();
                winAddPart.setTitle("Add Part");
                winAddPart.setScene(addPartScene);
                winAddPart.show();
            }
            else {
                alert.close();
            }
        }
        catch (IOException E) {
            System.out.println(E.getLocalizedMessage());
        }
```

```
   }


   /** Helper function do to any necessary data population after scene is
loaded. */
   @Override
   public void initialize(URL url, ResourceBundle resourceBundle) {
       setupView();
   }

}
```

Modify product controller class:

```
package lightsaberInventory.View;

import javafx.collections.FXCollections;
import javafx.collections.ObservableList;
import javafx.event.ActionEvent;
import javafx.fxml.FXML;
import javafx.fxml.FXMLLoader;
import javafx.fxml.Initializable;
import javafx.scene.Node;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.control.cell.PropertyValueFactory;
import javafx.stage.Stage;
import lightsaberInventory.Model.*;

import java.io.IOException;
import java.net.URL;
import java.util.ResourceBundle;



public class ModifyProductController implements Initializable {

   static public Product displayedProduct;

   @FXML
   TextField textID;

   @FXML
   TextField textName;

   @FXML
   TextField textInventory;
```

```
@FXML
TextField textPrice;

@FXML
TextField textMin;

@FXML
TextField textMax;

@FXML
Button saveButton;

@FXML
Button cancelButton;

@FXML
TextField searchPartsBox;

@FXML
TableView<Part> allPartsTable;

@FXML
TableColumn<Part, Integer> allPartsPartIDCol;

@FXML
TableColumn<Part, Integer> allPartsNameCol;

@FXML
TableColumn<Part,String>allPartsInvCol;

@FXML
TableColumn<Part, Integer> allPartsPriceCol;

@FXML
TableView<Part> associatedPartsTable;

@FXML
TableColumn<Part, Integer> asscPartsPartIDCol;

@FXML
TableColumn<Part, String> asscPartsNameCol;

@FXML
TableColumn<Part,String>assocPartsInvCol;

@FXML
TableColumn<Part, Integer> assocPartsPriceCol;
```

```
   @FXML
   Button addPartButton;

   @FXML
   Button removeAssicatedPartButton;



   @FXML
   void setCancelButton(ActionEvent event) throws IOException {

       try {
           Alert alert = new Alert(Alert.AlertType.CONFIRMATION);
           alert.setTitle("Cancel?");
           alert.setHeaderText("Are you sure you want to exit?");
           alert.setContentText("Press OK to exit the program. \nPress Cancel
to stay on this screen.");
           alert.showAndWait();
           if (alert.getResult() == ButtonType.OK) {

               FXMLLoader loader = new
FXMLLoader(getClass().getResource("MainScreen.fxml"));
               Parent addPartScreen = loader.load();
               Scene addPartScene = new Scene(addPartScreen);
               Stage winAddPart =
(Stage)((Node)event.getSource()).getScene().getWindow();
               winAddPart.setTitle("Add Part");
               winAddPart.setScene(addPartScene);
               winAddPart.show();
           }
           else {
               alert.close();
           }
       }
       catch (IOException E) {
           System.out.println(E.getLocalizedMessage());
       }

   };


   @FXML
   public void saveProductButtonPressed(ActionEvent event) throws IOException,
NumberFormatException {
       if (textName.getText().isEmpty()
               || textInventory.getText().isEmpty()
               || textMin.getText().isEmpty()
               || textMax.getText().isEmpty()
               || textPrice.getText().isEmpty()) {
```

```
            System.out.println("Data Empty");
            Alert alert = new Alert(Alert.AlertType.WARNING);
            alert.setTitle("Data Error");
            alert.setHeaderText("Please enter valid data for every field");
            alert.showAndWait();
        }

        else if (!Inventory.isNumeric(textMax.getText())
                || !Inventory.isNumeric(textMin.getText())
                || !Inventory.isNumeric(textPrice.getText())
                || !Inventory.isNumeric(textInventory.getText())) {


            Alert alert = new Alert(Alert.AlertType.WARNING);
            alert.setTitle("Value Error");
            alert.setHeaderText("Min, Max, Inventory, and price should all be
numeric");
            alert.showAndWait();

        }

        else if (Integer.parseInt(textMin.getText()) >
Integer.parseInt(textMax.getText())) {
            System.out.println("Min Max Error");
            Alert alert = new Alert(Alert.AlertType.WARNING);
            alert.setTitle("Min Max Error");
            alert.setHeaderText("Product Mins cannot be greater than Maxs");
            alert.showAndWait();
        }

        else if (Integer.parseInt(textMin.getText()) >
Integer.parseInt(textInventory.getText()) ||
Integer.parseInt(textInventory.getText()) >
Integer.parseInt(textMax.getText())) {
            System.out.println("Inventory Error");
            Alert alert = new Alert(Alert.AlertType.WARNING);
            alert.setTitle("Inventory Error");
            alert.setHeaderText("Inventory should be between the min and max");
            alert.showAndWait();
        }

        else {
            System.out.println("Data not empty");
            try {
                int index = displayedProduct.getId();

                String name = textName.getText();
                int invText = Integer.parseInt(textInventory.getText());
                double price = (Double.parseDouble(textPrice.getText()));
                int min = Integer.parseInt(textMin.getText());
                int max = Integer.parseInt(textMax.getText());

                Product newProduct = new Product(index, name, price, invText,
min, max, associatedPartsTable.getItems());
                System.out.println(newProduct);
                Inventory.updateProduct(index - 1001, newProduct);

                FXMLLoader loader = new
FXMLLoader(getClass().getResource("MainScreen.fxml"));
                Parent addPartScreen = loader.load();
                Scene addPartScene = new Scene(addPartScreen);
                Stage winAddPart =
(Stage)((Node)event.getSource()).getScene().getWindow();
                winAddPart.setTitle("Main Screen");
                winAddPart.setScene(addPartScene);
                winAddPart.show();
            }
            catch (IOException E) {
                System.out.println(E.getLocalizedMessage());
            }
            catch ( NumberFormatException E) {
                Alert alert = new Alert(Alert.AlertType.ERROR);
                alert.setTitle("Type Error");
                alert.setHeaderText("Please format your inputs like the
following:" +
                        "\nName: String" +
                        "\nPrice: Double" +
                        "\nMin, Max, Inventory: Integer");
                alert.showAndWait();
            }



        }

    }



    @FXML
    public void addPartButtonPressed() {
        if (allPartsTable.getSelectionModel().getSelectedItem() == null) {
            Alert alert = new Alert(Alert.AlertType.ERROR);
            alert.setTitle("No Selection");
            alert.setHeaderText("Please select a part to add");
            alert.showAndWait();
        }
        else {
            Part selectedItem =
allPartsTable.getSelectionModel().getSelectedItem();
            associatedPartsTable.getItems().add(selectedItem);
        }
    }



    @FXML
    public void setRemoveAssociatedPartButton() {
        Part selectedItem =
associatedPartsTable.getSelectionModel().getSelectedItem();

        Alert alert = new Alert(Alert.AlertType.CONFIRMATION);
        alert.setTitle("Remove Part Association?");
        alert.setHeaderText("Remove Association for part: " +
selectedItem.getName() + "?");
        alert.showAndWait();

        if (alert.getResult() == ButtonType.OK) {
            associatedPartsTable.getItems().remove(selectedItem);
        }
        else {
            alert.close();
        }
    }



    @FXML
    public void updateAllPartsTable() {
        allPartsTable.setItems(Inventory.getAllParts());
    }




    @FXML
    public void  updateAssociatedPartsTable() {
        associatedPartsTable.setItems(displayedProduct.getAllAssociatedParts());
    }

    @FXML
    public void searchParts() {
        if (searchPartsBox.getText().trim().isEmpty()) {
            allPartsTable.setItems(Inventory.getAllParts());
        }
        else {
            try {
                Part returnedPart =
Inventory.lookupPart(Integer.parseInt(searchPartsBox.getText()));

                if (returnedPart == null) {
                    Alert alert = new Alert(Alert.AlertType.ERROR);
                    alert.setTitle("No Parts Found");
                    alert.setHeaderText("Please Search Again");
                    alert.showAndWait();
                }
                else {
                    ObservableList<Part> filteredPartsList =
FXCollections.observableArrayList();
                    filteredPartsList.add(returnedPart);
                    allPartsTable.setItems(filteredPartsList);
                }
            }

            catch (NumberFormatException e) {
                System.out.println("Number Format Exception");

allPartsTable.setItems(Inventory.lookupPart(searchPartsBox.getText().trim()));
            }
        }

    }



    /** Loads the allPartsTable and assocPartsTable with columns names. Helper
function do to any necessary data population after scene is loaded. */
    @Override
    public void initialize(URL url, ResourceBundle resourceBundle) {

        allPartsPartIDCol.setCellValueFactory(new PropertyValueFactory<>("id"));
        allPartsNameCol.setCellValueFactory(new PropertyValueFactory<>("name"));
        allPartsInvCol.setCellValueFactory(new PropertyValueFactory<>("stock"));
        allPartsPriceCol.setCellValueFactory(new
PropertyValueFactory<>("price"));
        updateAllPartsTable();

        textID.setText(String.valueOf(displayedProduct.getId()));
        textName.setText(displayedProduct.getName());
        textInventory.setText(String.valueOf(displayedProduct.getStock()));
        textPrice.setText(String.valueOf((displayedProduct.getPrice())));
        textMax.setText(String.valueOf(displayedProduct.getMax()));
        textMin.setText(String.valueOf(displayedProduct.getMin()));
```

```
       asscPartsPartIDCol.setCellValueFactory(new
PropertyValueFactory<>("id"));
       asscPartsNameCol.setCellValueFactory(new
PropertyValueFactory<>("name"));
       assocPartsInvCol.setCellValueFactory(new
PropertyValueFactory<>("stock"));
       assocPartsPriceCol.setCellValueFactory(new
PropertyValueFactory<>("price"));
       updateAssociatedPartsTable();

   }
}
```

Main class:

```
package lightsaberInventory;
import java.io.IOException;
import javafx.application.Application;
import javafx.fxml.FXMLLoader;
import javafx.scene.Parent;
import javafx.scene.Scene;
import javafx.stage.Stage;
import lightsaberInventory.View.MainScreen;


public class Main extends Application {

   @Override
   public void start(Stage stage) throws IOException {
       try {
           FXMLLoader fxmlLoader = new
FXMLLoader(MainScreen.class.getResource("MainScreen.fxml"));

           Parent root = fxmlLoader.load();
           MainScreen mainScreen = fxmlLoader.getController();
           Scene scene = new Scene(root);

           stage.setTitle("Inventory Management System");
           stage.setScene(scene);
           stage.show();
       }
       catch (IOException e) {
           e.printStackTrace();
       }
   }


   public static void main(String[] args) {
       launch(args);
```

```
    }
}
```