Report of Inventory Management

System Application Solution

Therese M. Parks

Western Governors University

**Table of Contents**

[Summary 3](#_Toc22324112)

[Review of Other Work 9](#_Toc22324113)

[Changes to the Project Environment 13](#_Toc22324114)

[Methodology 15](#_Toc22324115)

[Project Goals and Objectives 17](#_Toc22324116)

[Project Timeline 21](#_Toc22324117)

[Unanticipated Requirements 23](#_Toc22324118)

[Conclusions 23](#_Toc22324119)

[Project Deliverables 26](#_Toc22324120)

[References 35](#_Toc22324121)

[Appendix A 37](#_Toc22324122)

[Appendix B 38](#_Toc22324123)

[Appendix C 42](#_Toc22324124)

[Appendix D 43](#_Toc22324125)

[Appendix E 46](#_Toc22324126)

# Summary

Safely Home Security started as a small business selling only one kind of door lock. They had since grown into a company of twenty employees, selling a large variety of customized home security systems. However, as sales increased*, mistakes* in the***inventory*** count occurred frequently, as employees made frequent data entry mistakes because of the rush in sales calls. As a result, Safely Home Security’s employees were *accidently selling out of stock items* and having to apologize to customers for these mistakes. They were using a manual system of adjusting shared spreadsheets to keep track of inventory. The company had a *database*, that an employee manually updated at the end of every day with the results from these spreadsheets.

The proposed solution to the problem of these frequently occurring *mistakes* made while *inventory data* was *manually adjusted* by Safely Home Security employees, was the implementation of a custom *inventory management system desktop application* which would connect to the existing database. An inventory management system desktop applicationwould be able to help employees keep track of inventory in *real time* so that the inventory data values could be automatically adjusted and up to date. *Simple Modern Software* was more than happy to design and create the new inventory management system desktop application for *Safely Home Security*. The following is a summary which includes the logical overview of the project with details that describe the actual development of the entire project. The intention here was to provide a description of the flow of the project, including all the major aspects that were accomplished.

**Firstly**, Simple Modern Software **planned** the project. Simple Modern Software audited the current inventory system from start to finish. This process started at the customer’s phone call, proceeded to the employee editing the spreadsheets, and finished when adjustments were made at the end of the day in the database.

The main purpose of this planning stage was to identify where the errors were taking place. The findings were that employees had differing opinions of when to update the spreadsheets. The main problem, which was leading to data entry mistakes was **inconsistent timing** in the methods which the employees were using. There were three different methods beings used: *first*, some employees were adjusting spreadsheets *as* they were making the sale with the customer on the phone. *Second*, some employees were waiting until right *after* the phone call to adjust. *Third*, some employees made a handwritten sticky note reminding them to adjust the spreadsheets at the end of their shifts for the day. Once a complete understanding was developed of precisely how and when these problems were occurring, Simply Modern Software took notes and planned out how the desktop application would prevent these problems.

Another purpose of this stage was to see what was already working well in the inventory process so that these implementations could be carried as much as possible into the design of the application. The findings for what was working well in the old process, was that a few employees were adjusting the spreadsheets *as* they made the sale with the customer on the phone. This method was closest to the plan of keeping inventory data up-to-date and in real time. Notes were taken stating that this part of the process would be utilized and enhanced by the use the desktop application. Employees would all be trained to use the application to adjust inventory data *as* the sale was being made, and *not afterwards* in any way.

**Secondly**, Simple Modern Software **designed** the project. Plans were made to align structures and variables between the existing database and the new application. It was noted here that the database contained certain tables and variables which the application would have to contain as well. The database had five tables to hold and manage inventory data. These were: 1. A table for ***in-house******parts***, which contained variables for a *part ID*, the part *name*, the part *price*, whether it was *in stock*, the *min* value, the *max* value, and the *machine ID* ; 2. A table for ***outsourced parts***, which contained variables for a *part ID*, the part *name*, the part *price*, whether it was *in stock*, the *min* value, the *max* value, and the *company name*; 3. A table for ***products*** which contained variables for the *product ID,* the *name*, the *price*, whether it was *in stock*, the *min* value, and the *max* value (Western Governors University, 2019). A *UML Class Diagram* was created and arranged along with planning notes using a simple Microsoft Word processing document. This was done by first studying the database structure as it is now, and then mirroring that design by the inventory application.

**User Interface Designed**

The tables for the in-house parts, the outsourced part, and the product were placed into the *UML Class Diagram* along with their variables. Relationships were drawn between them and notes were taken showing how the application would cause these three tables to seamlessly work together as five classes *(See Figure A1, Appendix A).* Also, during this stage, the user interface was designed according to these charts and diagrams. This was done by choosing features and an ascetical design that would make the interface easy for the employees to use. The application was designed to make it difficult for them to make data entry mistakes, by creating error messages, reminders, and notifications.

A GUI mockup was designed using Scene Builder, which is a visual layout tool that lets users quickly design JavaFX application user interfaces *without coding (See Appendix B to see GUI Mockup)*. The user interface was *designed* with the following requirements: It would be a JavaFX application with a graphical user interface (GUI) based on the pre-designed GUI mock-up. Code would be written to display each of the following screens in the GUI:

A ***main screen***showing controls for “Add”, “Modify”, “Delete”, “Search” for parts and products, and “Exit”; *lists* for parts and products; *text boxes* for searching for parts and products, and *title labels* for parts, products, and the application title (Western Governors University, 2019).

An ***add part screen***, showing controls for radio buttons for “In-House” and “Outsourced” parts; *buttons* for “Save” and “Cancel”; *text fields* for ID, name, inventory level, price, max and min values, and company or machine ID; and *labels* for ID, name, inventory level, price/cost, max and min values, the application title, and company name or machine ID (Western Governors University, 2019).

A ***modify part screen***, with fields that populate with pre-saved data, showing controls for *radio buttons* for “In-House” an “Outsourced” parts, *buttons* for “Save and “Cancel”; *text fields* for ID, name, inventory level, price, max and min values, the application title, and company name or machine ID; and *labels* for ID, name, inventory level , price, max and min values, the application title, and the company name or machine ID (Western Governors University, 2019).

An ***add* *product screen****,* showing controls for *buttons* for “Save”, “Cancel”, “Add” part, and “Delete” part; *text fields* for ID, name, inventory level, price, and max and min values; *labels,* for ID, name, inventory level, price, max and min values, and the application; a *list* for associated parts and their products; and a “Search” *button* and a *text field* with an associated list for displaying the results of the search. (Western Governors University, 2019).

A ***modify* *product screen****,* showing controls for *buttons* for “Save”, “Cancel”, “Add” part, and “Delete” part; *text fields* for ID, name, inventory level, price, and max and min values; *labels,* for ID, name, inventory level, price, max and min values, and the application; a *list* for associated parts and their products; and a “Search” *button* and a *text field* with an associated list for displaying the results of the search (Western Governors University, 2019).

The purpose for the planning and aligning of variables and structures between the database and the application was for the inventory management application to connect seamlessly with the current database. When the two were aligned, then data manipulation became straightforward and easy.

Originally, Simple Modern Software had planned to develop the error messages, reminders, and notifications during this *designing* phase, but it was decided to move this part of development to the *third* phase when the project would be *built*. The reason for this, was that it would be more convenient to see the application in action and get a feel of the actual flow of the application, in order to *accurately* place controls and error messages in the right places and in the right *context*.

**Thirdly**, Simple Modern Software **built** the project.The application was built according to the *UML Class Diagram*, notes, and GUI mockup previously created. These planning materials were meticulously followed in order to meet the design requirements. Error messages, reminders, and notifications were then created in order to help make the application easy to use and to prevent user mistakes. The code was written using the NetBeans IDE, which is an open-source integrated environment for developing with Java, (the chosen language for this application) and other programming languages.

Code was written to create the class structure provided by the *UML Class Diagram* to create appropriate classes and instance variables with the following criteria: Five classes with the 16 associated instance variables; variables were only accessible through getter methods; and variables were modifiable through setter methods (Western Governors University, 2019).

Code was written to add the following functionalities to the main screen, using the methods provided in the *UML Class Diagram*: to redirect the user to the “Add Part”, “Modify Part”, “Add Product”, or “Modify Product” screens; to delete a selected part or product and display matching result; to search for a part or product and display matching results, and to exit the main screen (Western Governors University, 2019).

Code was written to add the following functionalities to the part screens, using methods provide in the *UML Class Diagram:* First,for the “Add Part” screen: to *select* “In-House” or “Outsourced”; to *enter* name, inventory level, price, max and min values, and company name or machine ID; and, to *save* the data and then redirect to the main screen. Second, for the “Modify Part” screen to *select* “In-House” or “Outsourced”; to *modify* or change data values; to *save* modifications to the data and then redirect to the main screen; to *cancel* or *exit* out of this screen and go back to the main screen (Western Governors University, 2019).

Code was written to add the following functionalities to the product screens, using the methods provided in the attached *UML Class Diagram.* First, for the “Add Product” screen, code was written to add the functionalities to *enter* name, inventory level, price, max and min values, and company name or machine ID; to *save* the data and then redirect to the main screen; to associate one or more parts with a product; to remove or disassociate a part from a product; to cancel or exit out of this screen and go back to the main screen. Second, for the “Modify Product” screen code was written to add the functionalities to modify or change data values; to save modifications to the data and then redirect to the main screen; to associate one or more parts with a product; to remove or disassociate a part from a product; to cancel or exit out of this screen and go back to the main screen (Western Governors University, 2019).

Code was written to implement exception controls with custom error messages for each of the following sets. For the first set, code was written to implement exception controls with custom error messages for entering an inventory value greater than the maximum value for a part or product, or lower than the minimum value for a part or product. For the second set, code was written to implement exception controls with custom error messages for preventing the user from deleting a product that has a part assigned to it; including a confirm dialogue for all “Delete” and “Cancel’ buttons; ensuring that the price of a product cannot be less than the cost of parts; ensuring that a product must have a name, price, and inventor level (default 0) (Western Governors University, 2019).

Simple Modern Software installed the application and verified proper connectivity to the database server for all employee computers. Since there were only 20 computers, Simple Modern Software installed the Java application from a flash drives onto each computer. The command line was used to execute the java command to test the connection with the database. The application was opened on each computer and tested. Each installation worked as it should.

Once, proper connectivity was verified, employees started using it for a test period.

# Review of Other Work

This section presents a review of other works done by a third party that were relevant and directly related to the completed project. Four works have been chosen for this review which will present Safely Home Security with information to provide an understanding of how the proposed solution, the *inventory management system application* is the *best* solution for the company at this time. The four works summarized in this section are the titled: *Why Choose Customized Over Out-Of-The-Box Inventory Management; The Dangers of Using Excel for Inventory Management;* *10 Ways Inventory Management Software Increases Sales and Profit; and, Improve Your Inventory Management to Increase Sales.*

**Relation of Artifacts to Project Development**

The **first** work, titled *Why Choose Customized Over Out-Of-The-Box Inventory Management*, is an explanation by Clear Spider about the benefits and reasons for having an inventory management system customized to meet the needs of a business. Two of these reasons are: first, they provide *easy integrations,* which means that a customized system can mold to the needs of the business including any needed software integrations; second, a customized inventory system *suits your business,* which means customized software is easier to use, navigation is quicker for new users and training is less strenuous (Clear Spider, 2019).

Since a customized inventory management system is easy integrations is one of the benefits of a customized inventory management system, it will be easy to seamlessly connect the new desktop application to the existing inventory database server. It is beneficial to realize that a customized inventory management system *suits your business*, because the employees will find the new system easy to use. This is because it provides quick navigation so that they could very easily become used to using the new application, and there won’t be too much of a learning curve to slow down the sales process. The application will rather aid in the facilitation of the process rather than hinder it in any way.

The **second** work, titled *The Dangers of Using Excel for Inventory Management,* explains how using spreadsheets for inventory management is okay when a business is just starting out, but as it grows, the spreadsheet system can negatively impact a business if relied on too much in circumstances where other tools could work better. (Homby, 2015). The author, Homby, specifically explains how using spreadsheets can lead to information not being updated on time, skewed reporting in inventory management*, human error occurring when manually keeping track of stock items and updated information*, and the potential for lost data when sharing spreadsheets (2015).

This work is helpful because it offers information to help Simply Safe Security to understand the enormity of its current predicament and the danger that the company would be in, if it should remain as it is, without implementing an inventory management system. Using spreadsheets to track inventory was a great implementation for the company when it was still in its *beginning stages*, but now that the company has grown and is selling many more products to many more customers, the current spreadsheet inventory system has become more of a hindrance than a help to the company. The errors that Homby writes about, including *human error occurring when manually keeping track of stock items and updated information,* which is the main problem right now at Safely Home Security, will be eradicated by the implementation of the proposed solution.

The **third** work, *10 Ways Inventory Management Software Increases Sales and Profit,* offers explanations of the many benefits that implementing an inventory management application would come with. Some of these benefits highlight the goals of the proposed project. One main explanation is that it provides *real-time data access,* which means that inventory management software makes it a lot easier to keep an updated overview of stocking levels. (Intelligence Node, 2015).

Having real-time data access will enable Safely Home Security to sell items to customers that are in -stock and reduce customer complaints, as well as free up time spent on dealing with customer complaints so that employees can have more phone calls, and more happy customers throughout each day. This is the benefit that implementing the proposed solution will produce.

The **fourth** work, *Improve Your Inventory Management to Increase Sales,* highlights some major benefits that implementing the new inventory management application would provide. It states that by “using an automated inventory management system, your business will be more productive and efficient, earning you more sales and a greater profit margin” (Musaoglu, 2018).

The earning of more sales and a greater profit margin, on account of the implementing an inventory management system is good because it directly counterattacks the danger of losing customers. In other words, it would counterattack in a very beneficial way, recent bad reviews and complaints that Safely Home Security has been receiving, on account of errors in the current spreadsheet system.

**Additional Supporting Works**

Three additional works have been chosen for this review which present Safely Home Security with support for the implementation. The three additional works summarized in this section are the titled: *Catastrophic Inventory Mistakes by Huge Brands and How to Avoid Them, Common Challenges in Real-Time Inventory Management,* and *the 9 steps you need to solve your inventory problems.*

The **first additional** work*, Catastrophic Inventory Mistakes by Huge Brands and How to Avoid Them,* gave some helpful information and motives behind the importance of having an inventory system put in that is customized to fit a business’s needs. Among other big brand companies, *Nike* lost millions of dollars over an inventory system that did not give what it was promised to give. This is because the system was not custom built just for the company (Audral, 2019). The article provided other informative and eye-opening facts to be aware of when choosing the best inventory management solution for a business. This article gave Simple Modern Software *confirmation* that building a customized inventory system was really the best solution and we were on the right track.

The **second additional** work, *Common Challenges in Real-Time Inventory Management,* gave some helpful important information about how a company should stay in control of their inventory if real time. Upgrading old procedures and implementing new technology and software would make a company more efficient in managing inventory (Product Fulfillment Services, 2018). This article *supported* Simple Modern Software’s plan to implement the new inventory management system, which is utilizing the latest technology and to dispose of the old spreadsheet system.

The **third additional** work, *the 9 steps you need to solve your inventory problems,* highlighted some best practices to follow when developing an inventory solution. Among other things, this article emphasized the need to clearly define the problem. (Roy, 2017) This article helped to encourage Simple Modern Software to really *emphasize*d *the need* to find the *actual problem* with data entry mistakes and helped them to decide to give the problem-searching enough time and attention before trying to build the solution.

# Changes to the Project Environment

**Previous Project Environment**

Before the implementation of the Inventory Management System Desktop Application, Safely Home Security had a high-ranking trademark reputation for their personalized customer phone support. They were proud to have this interaction versus having the customers go through the difficult process of putting together a security solution and processing an order online.

However, as sales were increasing, mistakes in the inventory count were occurring frequently, as employees made frequent data entry mistakes because of the rush in sales calls. Safely Home Security’s employees had been accidently selling out of stock items and having to apologize to customers for these mistakes. They were using a manual system of adjusting shared spreadsheets to keep track of inventory. They had a database, that they updated at the end of every day with the results from these spreadsheets. The company had recently received several complaints and bad reviews, which could lead to the loss of customers.

Before the project began, the overall environment in the office was filled with *frustration* among the employees as they answered one phone call after another, without knowing whether it would be a prospective customer, or a customer calling with a complaint about a late shipment. During the rush and frustration, employees either completely forgot to adjust inventory data on the spreadsheets or entered it in the wrong way.

A server was used to manage and store inventory data, but it would need to have someone enter information into it manually at the end of every day. However, after the implementation, the inventory management system desktop application automatically updates the server after each data manipulation.

**Post Project Environment**

*After the implementation* of the project, Safely Home Security still held its high-ranking trademark reputation for their personalized customer phone support. Inventory continued to be consistently up-to-date and in real time. There have been no late shipments directly due to mistakes in the inventory system, and there have been no bad reviews or complaints about matters originating from mistakes in the inventory system.

After the completion of the project there was a generally *positive* and *cordial attitude* among the employees. The new inventory management system was instrumental in allowing the employees to experience being a part of the company’s season of growth, because the implementation fits right into the growing company with immense benefits.

Positively, employees finally reached the point where they were happy to be involved in a new and better way to keep track of data as they built customized security solutions for customers. All employees were using the new inventory management application to enter data *as* they were on the phone with the customer, instead of afterwards. They were receiving more *positive reviews* and have less angry customers calling in to complain.

The manual adjustment of data has no longer been needed and there have been no inventory management spreadsheets because they have been discarded for good and the application has completely replaced them.

# Methodology

The methodology which the completed project closely followed was the SDLC methodology. SDLC stands for *Software Development Life Cycle* and is a process that produces software with the highest quality and lowest cost in the shortest time. SDLC presented a plan detailing the development, alteration, maintenance, and replacement of a software system. It was made up of distinct stages including planning, design, building, testing, and deployment (Stackify, 2017). During the implementation of the *inventory management system desktop application*, the project went through the following SDLC stages:

**Plan (included auditing)**

The first stage of the SDLC methodology was the **planning** stage. Simple Modern Software *audited* the current inventory system from the start to finish to identify where the errors were taking place. This was done by following a process that started at the customer’s phone call, proceeded to the employee editing the spreadsheet, and finished when adjustments were made at the end of the day in the database. After a period of seven days, Simply Modern Software sat behind or beside employees and observed them for a period of seven more days while taking notes and looking for patterns. Plans were made that aligned structures and variables between the existing database and the new application.

**Design**

The second stage of the SDLC methodology was the **designing** stage. Simple Modern Software *designed* the Inventory Management Application to connect with the current database, by aligning structures and variables between the two, and making the interface easy for the employees to use, and difficult for them to make data entry mistakes. A *UML Class Diagram* was created and arranged along with planning notes using a simple Microsoft Word processing document. Relationships were drawn between them and notes were taken showing how the application would cause these five tables to seamlessly work together.

Also, during this stage, the user interface was designed according to these charts and diagrams. A GUI mock-up was designed using Scene Builder, which is a visual layout tool that lets users quickly design JavaFX application user interfaces *without coding*.

**Build**

The third stage of the SDLC methodology was the **building** stage. Simple Modern Software *built* the Application based on the designed plans and connected it to the existing database, debugging the code as needed. The application was built according to the *UML Class Diagram*, notes, and GUI mockup previously created. These planning materials were meticulously followed in order to meet the design requirements. Error messages, reminders, and notifications were then created in order to help make the application easy to use and to prevent user mistakes. The code was written using the NetBeans IDE, which is an open-source integrated environment for developing with Java, (the chosen language for this application) and other programming languages.

**Deploy (includes testing)**

The fourth stage of the SDLC methodology was the **deploying** stage. Simple Modern Software verified proper connectivity on the devices used by the developers to create the application to the server and ran extensive test to insure proper data manipulation and connection. Once, proper connectivity was verified, Simple Modern Software *installed/deployed* the application on all employee computers and allowed them to start using it for a test period. The old spreadsheets were removed from the system and discarded.

# Project Goals and Objectives

This section provides a detailed explanation of the goal and objectives for the project.

**Goal, Objectives and Deliverables Table**

In this section, a table is presented showing the overreaching primary goal encompassing the supportive objectives, which in turn are made up of deliverables. Following the table are descriptions of the Primary Goal, objectives, and deliverables.

|  |  |  |
| --- | --- | --- |
| Primary Goal | Supporting objectives | Deliverables enabling the project objectives |
| Solve the problem of frequent *mistakes* being made in the *manual* adjustment of *inventory data.* | 1. Audit current inventory system | 1.a. Inspect current system |
| 1.b. Audit phone call/sales procedure including use of spreadsheets |
| 2. Design the inventory application | 2.a. Design how the application will relate to the database, draw out diagrams and charts |
| 2.b. Design the user interface |
| 3. Build the application | 3.a. Use pre-planned material to build application using Java and FXML |
| 3.b. Connect the application to the database for the first time |
| 4. Install application on employee computers | 4.a. Install application on all employee computers and verify connectivity with the database, test and adjust |
| 4.b. Discard old spreadsheets which are now rendered useless |

**Description of Project Goal, Objectives, and Deliverables**

The **primary goal** of this project was to solve the problem of frequent *mistakes* being made in the *manual adjustment of inventory data* by Safely Home Security employees. To do this, Simple Modern Software provided an inventory management desktop application system*,* which connects to the existing database. An inventory management system desktop application is now helping employees keep track of inventory in *real time* and the inventory data are automatically adjusted and up to date. The goal was broken down into *four goals*, each with a precise measurement of success. The primary goal was met by the successful completion of each objective referenced below:

**Objective 1**: **Audit Current Inventory System**

The current inventory system was audited from the start to finish to identify where the errors are taking place. This process started at the customer’s phone call, proceeded to the employee editing the spreadsheet and finished when adjustments were made at the end of the day in the database. The findings here were that employees had differing opinions of when to update the spreadsheets. The main problem, which was leading to data entry mistakes was **inconsistent timing** in the methods which the employees were using.

The findings for what was working well in the old process, was that a few employees were adjusting the spreadsheets *as* they made the sale with the customer on the phone. This method was closest to the plan of keeping inventory data up-to-date and in real time. Notes were taken stating that this part of the process would be utilized and enhanced using the desktop application. Employees would all be trained to use the application to adjust inventory data *as* the sale was being made, and *not afterwards* in any way.

*Objective 1 Successfully Met*

This objective was *successfully* met because the main problem was found, and a constructive decision was that a current method would be utilized and improved by the application.

**Objective 2: Design the Inventory Application**

The Inventory Management Application was designed to connect with the current database, buy aligning structures and variables between the two, and making the interface easy for the employees to use, and difficult for them to make data entry mistakes. Plans were made that aligned structures and variables between the existing database and the new application. A *UML Class Diagram* was created and arranged along with planning notes using a simple Microsoft Word processing document. Also, during this stage, the user interface was designed according to these charts and diagrams. A GUI mock-up was designed using Scene Builder, which is a visual layout tool that lets users quickly design JavaFX application user interfaces *without coding.*

*Objective 2 Successfully Met*

This objective was *successfully* met because a UML diagram was created that provided the structure for a GUI mock-up to be created as well. Also, the project planning was completed before the building phase began.

**Objective 3: Build the Application**

The application was built based on the design plans and connect it to the existing database, debugging the code as needed. Prepared planning materials were meticulously followed in order to meet the design requirements. Error messages, reminders, and notifications were then created in order to help make the application easy to use and to prevent user mistakes. The application was successfully connected to the database and worked as expected during the building and testing phase.

*Objective 3 Successfully Met*

This objective was *successful* because the application was built according to the *UML Class Diagram*, notes, and GUI mockup previously created. Also, success was determined by the fact that the application was connecting to the database and successfully manipulating data as expected.

**Objective 4: Install Application on Employee Computers**

During this stage, Simple Modern Software installed the application on all employee computers and verified connectivity to the database server for all user computers. They were allowed use it for a testing period, after which the old spreadsheets were discarded.

*Objective 2 Successfully Met*

The successful installation of the application and its connection with the database on all employee computers launched the last step to throw away the old spreadsheets. The spreadsheets were successfully discarded as planned.

# Project Timeline

In this section, compare the projected and actual timelines of the milestones or deliverables of the project and explain why the differences occurred. Explain the reasons for each deviation of the actual time frame from the estimated time frame.

**Project Timeline with Milestones**

|  |  |  |  |
| --- | --- | --- | --- |
| **Description of milestone/ deliverable** | **Projected (PD)**  **and *Actual (AD)***  **Duration (days)** | **Projected (PSD) and Actual (ASD) start date** | **Projected (PED) and Actual (AED) end date** |
|  |  |  |  |
|  |  |  |  |
| Inspect the current system | **PD** 7 days  **AD 7 days** | **PSD/ASD**  May 4, 2019 | **PED/AED**  May 11, 2019 |
| Audit phone call/sales procedure including use of spreadsheets | **PD** 7 days  **AD 7 days** | **PSD/ASD**  May 12, 2019 | **PED/AED**  May 19, 2019 |
| Design how the application will relate to the database, draw out diagrams and charts | **PD** 2 days  **AD 2 days** | **PSD/ASD**  May 21, 2019 | **PED/AED**  May 23, 2019 |
| Design the user interface | **PD** 2 days  **AD 2 days** | **PSD/ASD**  May 25, 2019 | **PED/AED**  May 28, 2019 |
| Use pre-planned material to build application using Java and FXML | **PD** 30 days  **AD 30 days** | **PSD/ASD**  June 2, 2019 | **PED/AED**  June 30, 2019 |
| Connect application to the database for the first time | **PD** 2 days  **AD 2 days** | ***PSD*** *July 2,* 2019  **ASD June 9, 2019** | **PED** *July 4,* 2019  **ASD June 9, 2019** |
| Install application on all employee computers and verify connectivity with the database, test and adjust | **PD** 30 days  **AD 30 days** | **PSD/ASD**  *July* 6, 2019 | **PED/AED**  *August 3*, 2019 |
| Discard old spreadsheets | **PD** 3 days  **AD 3 days** | **PSD/ASD**  August 5, 2019 | **PED/AED**  August 8, 2019 |

All the Project Deliverables were completed within or under their prospected time frames as planned. The only change was that of the deliverable for *connecting the application to the database for the first time,*which was planned to be from the start date: *January 2, 2020* to the end date: *January 4, 2020.* The deliverable was completed earlier than planned by being consolidated into the deliverable: *Use pre-planned material to build application using Java and FXML*, which was planned to begin on *December 2, 2019,* and last until *December 31, 2019.* The reason for the change was that the connection to the database was needed during the building stage so that the developers could work with inventory data to perform necessary tests as they were building the application. In this case, the deliverable whose date changed, was completed sooner than expected. All the other dates for the other deliverables remained the same.

# Unanticipated Requirements

There were no unanticipated requirements other than that the deliverable *for connecting the application to the database for the first time*was needed to complete the previous deliverable which was to *use pre-planned material to build the application using Java and FXML*. If the deliverables were not consolidated, as they were, the application would have to come to a halt because developers would not be able to perform preliminary tests that involved the flow of the user interface of the application. Since the deliverables were consolidated, tests were performed as needed, and everything ran smoothly from that point on as planned.

# Conclusions

Three basic metrics will help Safely Home Security to assess the project’s success and effectiveness, now that it is complete. These are the *frequency of data entry mistakes*, the *selling of out-of-stock items*, and *customer satisfaction*. A decrease or increase in any of these within the expected percentage ranges would mean that the project was very successful and effective.

**Actual Project Accomplishments**

First, the **frequency in data entry mistakes** was measured. This measurement would determine whether the employees were still making data entry mistakes while using the new inventory management system desktop application, and if so, how many? This was measured at the end of every workweek for a period of six months. Before the application was implemented, the expectation was that the frequency in data entry mistakes made by the employees would *dramatically decrease* using the application instead of the spreadsheets. A decrease of 80-100% in the amount of data entry mistakes by employees would be considered successful. The success of this objective contributed to the fact that there is currently a 99% decrease in the amount of data entry mistakes by employees, which is very *successful*.

Second, the **selling of out-of-stock items** was measured. This measurement would determine whether the data in the inventory database was being updated properly and whether customers’ orders were late because they were mistakenly sold out-of-stock items. Before the application was implemented, the expectation was that inventory data manipulations would take place in *real-time* and is always up-to-date and accurate, so problems like this should be rare. The expectation was that the selling of out-of-stock itemswill *dramatically decrease*. This was measured at the end of every month for a period of six months. A decrease of 90-100% in the accidental selling of out-of-stock itemswill be considered successful. The success of this objective contributed to the fact that there is currently a 99.5% decrease accidental selling of out-of-stock itemsdue to inventory mistakes**,** which is very *successful*.

Third, **customer satisfaction** was measured. This measurement would determine whether customers have generally satisfied experiences with the company since it has implemented its new inventory management system application. This was measured on the last workday of every month for six months. It is expected that customers would have less complaints to offer and more satisfied reviews to leave for Safely Home Security. An increase of 10-20% in customer satisfaction will be considered successful. The success of this objective contributed to the fact that there is currently a 25% increase in customer satisfaction very *successful*.

**Potential Impacts on Completed Project Environment**

The potential effects of the completed project are that data entry mistakes will continue to be low in number, which means the selling of out-of-stock items will be kept to a bare minimum. As a result, Safely Home Security’s future should see that customers can expect to receive their shipments on time, customers will leave more positive reviews and less negative reviews, and there will be fewer customer complaints. Employees should continue to remain cordial and generally high-spirited in the office as they enjoy the many benefits that the new inventory management desktop application is bringing to the company, the employees, and the customers. The application-database connection will make sure that inventory data is always in real-time and the company will feel an overall sense of security because of this.

**Project Considered a Success**

Overall, the ***primary goal*** of this project, to solve the problem of frequent *mistakes* being made in the *manual adjustment of inventory data* by Safely Home Security employees was successfully met. The encompassed goals: *to audit current inventory system*, *to design the inventory application,* *to build the application*, and to *install application on employee computers*, were successfully met. Also, the expected outcomes of the project were all successfully met. Since the primary goal, the encompassed goals, and the expected project outcomes were all met, it can be finally concluded that the **implementation was a success**.

# Project Deliverables

This section includes explanations and details of the project key deliverables. The deliverables provide a detailed logical explanation of what the project provided to substantiate the work and completion of such. A table is presented here showing the overreaching primary goal encompassing the supportive objectives, which in turn are made up of deliverables. Following the table, the project’s key deliverables will be explained and detailed.

|  |  |  |
| --- | --- | --- |
| **Primary Goal** | Supporting objectives | Deliverables enabling the project objectives |
| **Solve the problem of frequent *mistakes* being made in the *manual* adjustment of *inventory data.*** | 1. Audit current inventory system | 1.a. Inspect current system |
| 1.b. Audit phone call/sales procedure including use of spreadsheets |
| 2. Design the inventory application | 2.a. Design how the application will relate to the database, draw out diagrams and charts |
| 2.b. Design the user interface |
| 3. Build the application | 3.a. Use pre-planned material to build application using Java and FXML |
| 3.b. Connect the application to the database for the first time |
| 4. Install application on employee computers | 4.a. Install application on all employee computers and verify connectivity with the database, test and adjust |
| 4.b. Discard old spreadsheets which are now rendered useless |

**1. a. Inspected Current System**

Simple Modern Software audited the current inventory system from start to finish. This process started at the customer’s phone call, proceeded to the employee editing the spreadsheets, and finished when adjustments were made at the end of the day in the database. Simply Modern Software was physically present at the Safely Home Security office for a period of seven days performing a general observation of the current system, asking questions and taking notes. Simply Modern Software consolidated their finding at the end of seven days. The findings here were that the mistakes were originating from the employees’ data entry, and not when the data was being entered not the database at the end of the day, by a trained server manager.

**1.b. Audited Phone Call/Sales Procedure Including Use of Spreadsheets**

After a period of seven days, Simply Modern Software sat behind or beside employees and observed them for a period of seven days while taking notes and looking for patterns. Simply Modern Software consolidated their finding at the end of seven days. The findings were that employees had differing opinions of when to update the spreadsheets. The main problem, which was leading to data entry mistakes was *inconsistent timing* in the methods which the employees were using. There were *three* different methods beings used: first, some employees were adjusting spreadsheets *as* they were making the sale with the customer on the phone. Second, some employees were waiting until right *after* the phone call to adjust. Thirdly, some employees made a handwritten sticky note reminding them to adjust the spreadsheets at the end of their shifts for the day. Once a complete understanding was developed of precisely how and when these problems were occurring, Simply Modern Software took notes and planned out how the desktop application would prevent these problems.

Another purpose of this stage was to see what was already working well in the inventory process so that these implementations could be carried as much as possible into the design of the application. The findings for what was working well in the old process, was that a few employees were adjusting the spreadsheets *as* they made the sale with the customer on the phone. This method was closest to the plan of keeping inventory data up-to-date and in real time. Notes were taken stating that this part of the process would be utilized and enhanced using the desktop application. Employees would all be trained to use the application to adjust inventory data *as* the sale was being made, and *not afterwards* in any way.

**2.a. Designed How Application Would Relate to Database/Drew Diagrams and Charts**

Plans were made that aligned structures and variables between the existing database and the new application.

It was noted here that the database contained certain tables and variables which the application would have to contain as well. The database had five tables to hold and manage inventory data. These were: 1. A table for *in-house parts*, which contained variables for a *part ID*, the part *name*, the part *price*, whether it was *in stock*, the *min* value, the *max* value, and the *machine ID* ; 2. A table for *outsourced parts*, which contained variables for a *part ID*, the part *name*, the part *price*, whether it was *in stock*, the *min* value, the *max* value, and the *company name*; 3. A table for *products* which contained variables for the *product ID,* the *name*, the *price*, whether it was *in stock*, the *min* value, and the *max* value (Western Governors University, 2019).

A *UML Class Diagram* was created and arranged along with planning notes using a simple Microsoft Word processing document *(See Figure A1*, *Appendix A).* This was done by first studying the database structure as it is now, and then mirroring that design by the inventory application. The tables for the in-house parts, the outsourced part, and the product were placed into the *UML Class Diagram* along with their variables. Relationships were drawn between them and notes were taken showing how the application would cause these three tables to seamlessly work together.

**2.b. Designed the User Interface**

Also, during this stage, the user interface was designed according to these charts and diagrams. This was done by choosing features and an ascetical design that would make the interface easy for the employees to use. The application was designed to make it difficult for them to make data entry mistakes, by creating error messages, reminders, and notifications.

A GUI mockup was designed using Scene Builder, which is a visual layout tool that lets users quickly design JavaFX application user interfaces *without coding (See Appendix B to see GUI mockup)*. The user interface was *designed* with the following requirements: It would be a JavaFX application with a graphical user interface (GUI) based on the pre-designed GUI mock-up. Code would be written to display each of the following screens in the GUI:

A *main screen* showing controls for “Add”, “Modify”, “Delete”, “Search” for parts and products, and “Exit”; *lists* for parts and products (*See Figure B1, Appendix B*); *text boxes* for searching for parts and products, and *title labels* for parts, products, and the application title (Western Governors University , 2019).

An *add part screen ,* showing controls for radio buttons for “In-House and “Outsourced “parts; *buttons* for “Save” and “Cancel”; *text fields* for ID, name, inventory level, price, max and min values, and company or machine ID; and *labels* for ID, name, inventory level, price/cost, max and min values, the application title, and company name or machine ID *(See Figures B2 and B3 Appendix B),*

A *modify part screen*, with fields that populate with pre-saved data, showing controls for *radio buttons* for “In-House” , an “Outsourced” parts, *buttons* for “Save and “Cancel” *(See Figures B4 and B5, Appendix B);* *text fields* for ID, name, inventory level, price, max and min values, the application title, and company name or machine ID; and *labels* for ID, name, inventory level , price, max and min values, the application title, and the company name or machine ID (Western Governors University, 2019).

An *add* *product screen,* showing controls for *buttons* for “Save”, “Cancel”, “Add” part, and “Delete” part *(See Figure B6, Appendix B); text fields* for ID, name, inventory level, price, and max and min values; *labels,* for ID, name, inventory level, price, max and min values, and the application; a *list* for associated parts and their products; and a “Search” *button* and a *text field* with an associated list for displaying the results of the search (Western Governors University, 2019).

A *modify* *product screen,* showing controls for *buttons* for “Save”, “Cancel”, “Add” part, and “Delete” part *(See Figure B7, Appendix B); text fields* for ID, name, inventory level, price, and max and min values; *labels,* for ID, name, inventory level, price, max and min values, and the application; a *list* for associated parts and their products; and a “Search” *button* and a *text field* with an associated list for displaying the results of the search (Western Governors University, 2019).

Originally, Simple Modern Software had planned to develop the error messages, reminders, and notifications during this *designing* phase, but it was decided to move this part of development to the *third* phase when the project would be *built*. The reason for this, was that it would be more convenient to see the application in action and get a feel of the actual flow of the application, in order to *accurately* place controls and error messages in the right places and in the right *context* (Western Governors University, 2019).

**3.a. Used Pre-planned Material to Build Application Using Java and FXML**

The application was built according to the *UML Class Diagram*, notes, and GUI mockup previously created. These planning materials were meticulously followed in order to meet the design requirements. Error messages, reminders, and notifications were then created in order to help make the application easy to use and to prevent user mistakes. The code was written using the NetBeans IDE, which is an open-source integrated environment for developing with Java, (the chosen language for this application) and other programming languages.

Code was written to create the class structure provided by the *UML Class Diagram* to create appropriate classes and instance variables with the following criteria: Five with the 16 associated instance variables classes *(See Figure C1, Appendix C);* variables were only accessible through getter methods; and variables were modifiable through setter methods *(See Figure D1, Appendix D).*

Code was written to add the following functionalities to the main screen, using the methods provided in the *UML Class Diagram*: to redirect the user to the “Add Part, “Modify Part”, “Add Product”, or “Modify Product” screens*”(See Figure D2, Appendix D)*; to delete a selected part or product and display matching result; to search for a part or product and display matching results, and to exit the main screen (Western Governors University, 2019).

Code was written to add the following functionalities to the part screens, using methods provide in the *UML Class Diagram:* First,for the “Add Part” screen: to *select* “In-House” or “Outsourced”; to *enter* name, inventory level, price, max and min values, and company name or machine ID; and, to *save* the data and then redirect to the main screen. Second, for the “Modify Part” screen to *select* “In-House” or “Outsourced”; to *modify* or change data values; to *save* modifications to the data and then redirect to the main screen; to *cancel* or *exit* out of this screen and go back to the main screen (Western Governors University, 2019).

Code was written to add the following functionalities to the product screens, using the methods provided in the attached *UML Class Diagram.* First, for the “Add Product” screen, code was written to add the functionalities to *enter* name, inventory level, price, max and min values, and company name or machine ID; to *save* the data and then redirect to the main screen; to associate one or more parts with a product; to remove or disassociate a part from a product; to cancel or exit out of this screen and go back to the main screen. Second, for the “Modify Product” screen code was written to add the functionalities to modify or change data values; to save modifications to the data and then redirect to the main screen; to associate one or more parts with a product; to remove or disassociate a part from a product; to cancel or exit out of this screen and go back to the main screen (Western Governors University, 2019).

**Example of Confirming Cancel Message for “Add Part” Screen**

Code was written to implement exception controls with custom error messages for each of the following sets. For the first set, code was written to implement exception controls with custom error messages for entering an inventory value greater than the maximum value for a part or product, or lower than the minimum value for a part or product (*See Figure D3, Appendix D).* For the second set, code was written to implement exception controls with custom error messages for preventing the user from deleting a product that has a part assigned to it; including a confirm dialogue for all “Delete” and “Cancel’ buttons *(See E1 and E2, Appendix E);* ensuring that the price of a product cannot be less than the cost of parts; ensuring that a product must have a name, price, and inventor level (default 0) (Western Governors University, 2019).

**3.b. Connected the Application to the Database for the First Time**

Simple Modern Software verified proper connectivity on the devices used by the developers to create the application to the server and ran extensive test to insure proper data manipulation and connection. Connection variables were created in the main connection class of the application. The values for these variables included authentication data needed for the application to connect to the database which included values for the connection driver, the server URL, and name, as well as a password *(See Figure D4, Appendix D).* When the application ran, the code was executed, and these variables were used by the application to establish connectivity successfully.

**4.a. Installed Application On all Employee Computers and Verify Connectivity with the Database, Test and Adjust**

Simple Modern Software installed the application and verified proper connectivity to the database server for all employee computers. Since there were only 20 computers, Simple Modern Software installed the Java application from a flash drives onto each computer. The command line was used to execute the java command to test the connection with the database. The application was opened on each computer and tested. Each installation worked as it should.

Once, proper connectivity was verified, employees started using it for a test period. The old spreadsheets were removed from the system and discarded. Each employee was able to use the application to manipulate inventory data without any problems. It was important to allow the employees to use the application for a testing period, in order to ensure the proper use of the application, and to adjust if any problems were to arise during this time. The old spreadsheets were removed in order to prevent any confusion and to prevent the employees from regressing to the use of the spreadsheets. This is because the new inventory management system automatically updated the database per individual manipulation rather than all at once like the old method.

# References

Audral. (2019). *Catastrophic inventory mistakes by huge brands and how to avoid them.* Retrieved from <https://www.assetpanda.com/catastrophic-inventory-mistakes-by-huge-brands-and-how-to-avoid-them/>.

Clear Spider. (2019). *Why choose customized over out-of-the-box inventory management.* **(**Retrieved from <https://www.clearspider.com/blog-customized-inventory-management/>.

Homby, S. (2015). *The Dangers of using Excel for inventory management.* Retrieved from

https://www.bluelinkerp.com/blog/2016/01/08/the-dangers-of-using-excel-for-inventory-management/.

Intelligence Node. (2019). *10 ways inventory management software increases sales and profit.* Retrieved from <https://www.intelligencenode.com/blog/10-ways-inventory-management-software-increases-sales-profit/>

Musaoglu, E. (2018). *Improve your inventory management to increase sales.* Retrieved from

<https://www.logiwa.com/blog/online-sales-inventory-management>

Product Fulfillment Services. (2018). *Common challenges in real-time inventory mangement.* Retrieved from <https://www.apsfulfillment.com/product-fulfillment-services/common-challenges-in-real-time-inventory-management/>

Roy, E. (2017). *The 9 steps you need to solve your inventory problems.* Retrieved from <http://www.tradeready.ca/2017/topics/supply-chain-management/9-steps-need-solve-inventory-problems/>

Stackify. (2017). *What is SDLC? Understand the software development life cycle. Retrieved from* <https://stackify.com/what-is-sdlc/#targetText=Definition%20of%20SDLC&targetText=SDLC%20includes%20a%20detailed%20plan,spiral%20model%2C%20and%20Agile%20model.>

Western Governors University. (2019). *GYP task 1.* Retrieved from <https://www.taskstream.com/Main/main_frame.asp>

Western Governors University. (2019). *GUI mock up.* Retrieved from

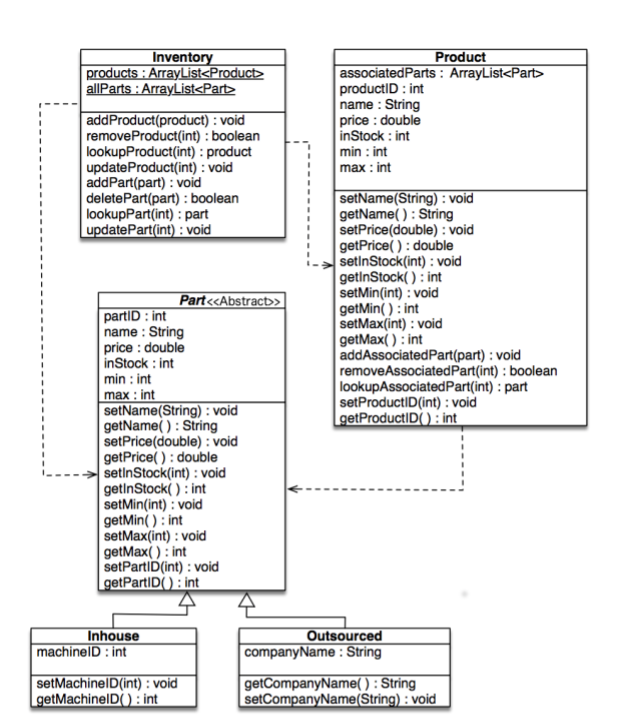
<https://www.taskstream.com/Main/main_frame.asp>

Western Governors University. (2019). *UML class diagram.* Retrieved from

<https://www.taskstream.com/Main/main_frame.asp>

# Appendix A

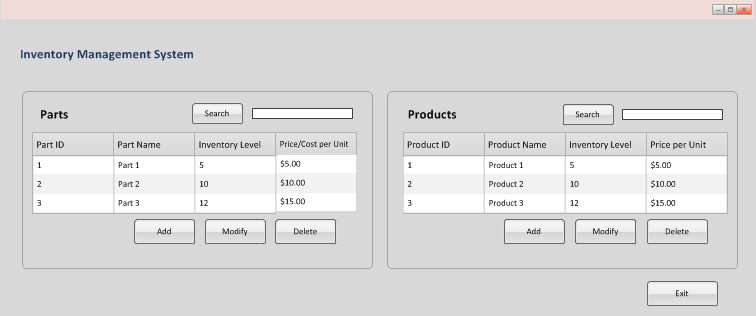
Class Diagram

Figure A1. UML Class Diagram

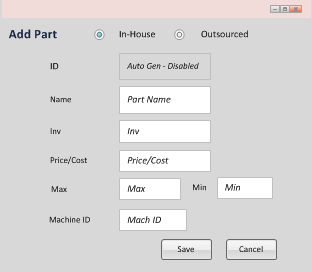
UML Class Diagram. Adapted from *Western Governors University*, by *Western Governors University*. 2019. Retrieved from <https://www.taskstream.com/Main/main_frame.asp>.

# Appendix B

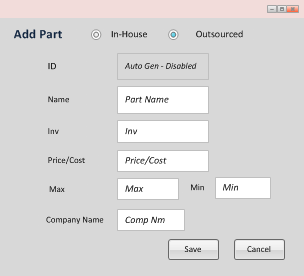
GUI Mockup

 Figure B1. GUI Mockup of *Main Screen*

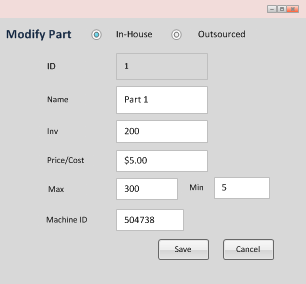
GUI Mock Up. Adapted from Western Governors University, by Western Governors University. 2019. Retrieved from <https://www.taskstream.com/Main/main_frame.asp>

 Figure B2. GUI Mockup of *Add In-House Part* Scree

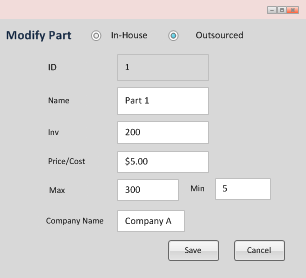
GUI Mock Up. Adapted from Western Governors University, by Western Governors University. 2019. Retrieved from <https://www.taskstream.com/Main/main_frame.asp>

 Figure B3. GUI Mockup of *Add Outsourced Part* Screen

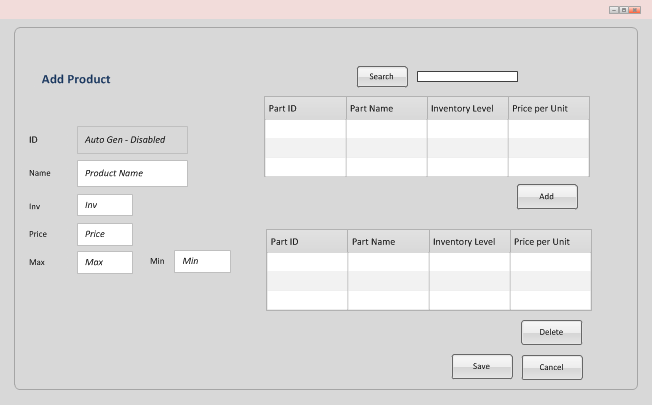
GUI Mock Up. Adapted from Western Governors University, by Western Governors University. 2019. Retrieved from <https://www.taskstream.com/Main/main_frame.asp>

Figure B4. GUI Mockup of *Modify In-House Part* Screen

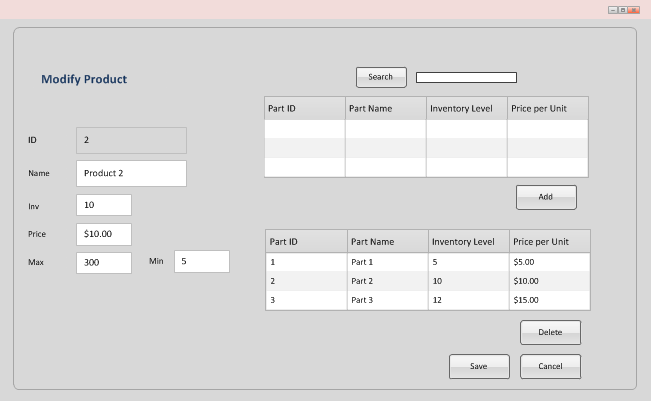
GUI Mock Up. Adapted from Western Governors University, by Western Governors University. 2019. Retrieved from <https://www.taskstream.com/Main/main_frame.asp>

Figure B5. GUI Mockup of Modify Outsourced Part Screen

GUI Mock Up. Adapted from Western Governors University, by Western Governors University. 2019. Retrieved from <https://www.taskstream.com/Main/main_frame.asp>

Figure B6. GUI Mockup of Add Product Screen

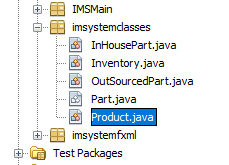
GUI Mock Up. Adapted from Western Governors University, by Western Governors University. 2019. Retrieved from <https://www.taskstream.com/Main/main_frame.asp>

Figure B7. GUI Mockup of Modify Product Screen

GUI Mock Up. Adapted from Western Governors University, by Western Governors University. 2019. Retrieved from <https://www.taskstream.com/Main/main_frame.asp>

# Appendix C

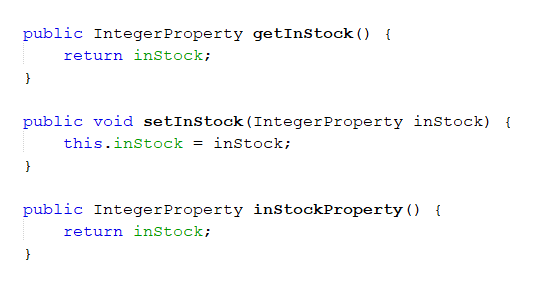
NetBeans Class Structure from Project

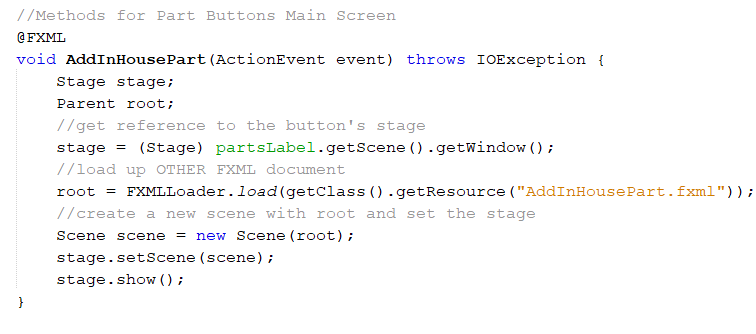
Figure C1. NetBeans Class Structure of 5 Classes that Mirror Database Structure

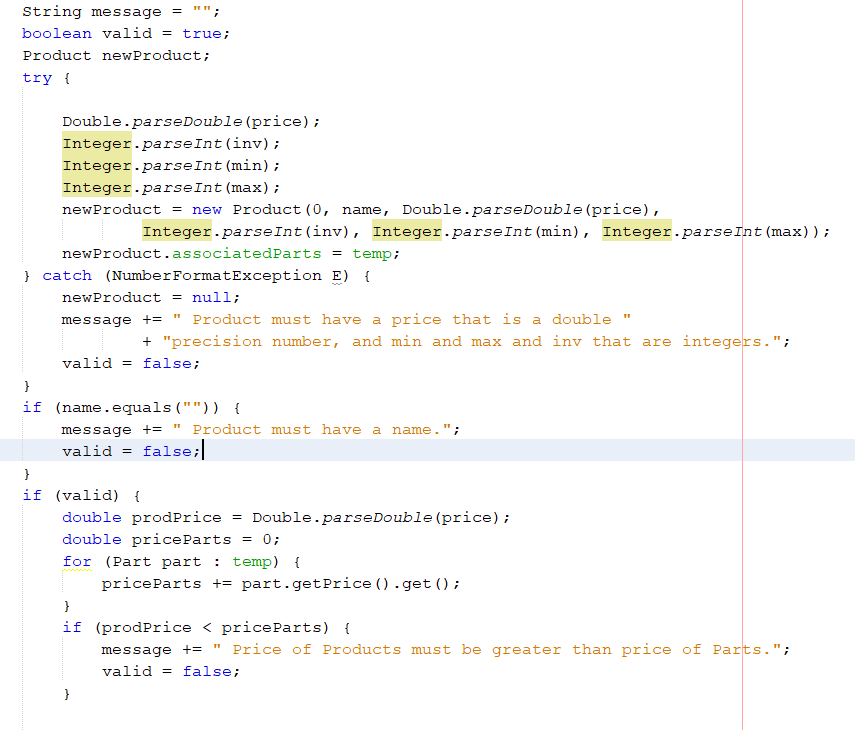
# Appendix D

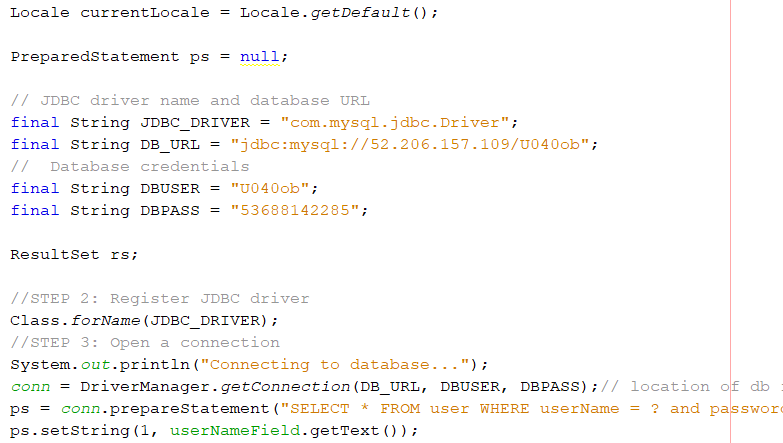
Java/FXML Code Snippets from Project

Figure D1. Example Code of Getter and Setter Methods from IMS Project



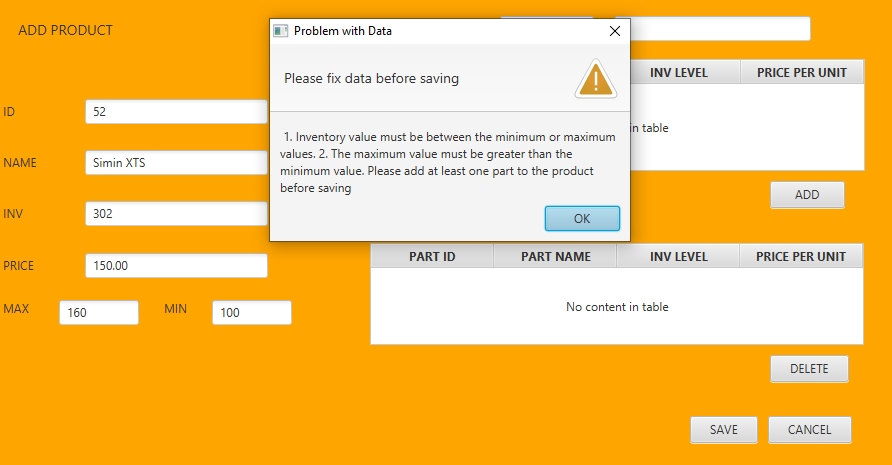
Figure D2. Code that adds functionality to button on Main screen to navigate to Add Part screen

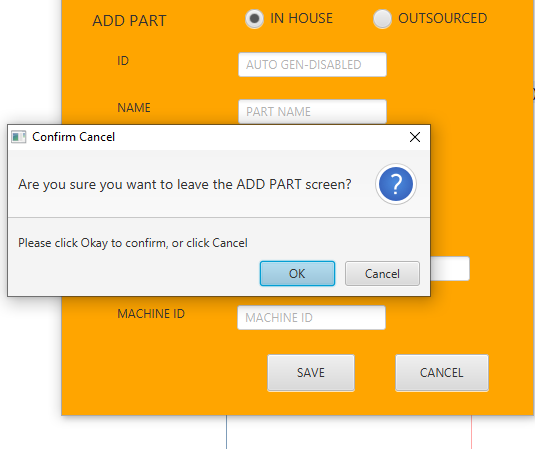
Figure D3 Example Code for Error messages as exception controls in Project

 Figure D4. Java code that enables application’s connection to database

# Appendix E

User Interface Examples of Error messages and Exception Controls from Actual Application

Figure E1. Example of Error Messages for “Add Product” Screen

Figure E2. Example of Confirming Cancel on “Add Part” screen.